

Comprehensive, hands-on training that solves real world problems

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Automation with Ansible

Student Workbook (ROLE)

© 2016 Red Hat, Inc.

DO407-A2.0-en-1-20160804

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

AUTOMATION WITH ANSIBLE

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Ansible 2.0 D0407
Automation with Ansible
Edition 1 20160804 20160804

Authors: Chen Chang, George Hacker, Razique Mahroua, Adolfo Vazquez, Snehangshu Karmakar
Editor: Forrest Taylor, Steven Bonneville

Copyright © 2016 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2016 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed please e-mail training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, Hibernate, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Ivan Makfinsky, Allison Hrivnak, Derek Carter,

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Document Conventions ix
Notes and Warnings ... ix

Introduction xi
Automation with Ansible ... xi
Orientation to the Classroom Environment ... xii
Internationalization ... xiv

1. Introducing Ansible 1
Overview of Ansible Architecture ... 2
Quiz: Ansible Architecture ... 6
Overview of Ansible Deployments ... 8
Quiz: Ansible Deployments ... 10
Describing Ansible Inventory ... 12
Quiz: Ansible Inventory ... 15
Quiz: Introducing Ansible ... 17
Summary ... 19

2. Deploying Ansible 21
Installing Ansible ... 22
Guided Exercise: Installing Ansible ... 29
Managing Ansible Configuration Files ... 31
Guided Exercise: Managing Ansible Configuration Files ... 36
Running Ad Hoc Commands ... 39
Guided Exercise: Running Ad Hoc Commands ... 44
Managing Dynamic Inventory ... 48
Guided Exercise: Managing Dynamic Inventory ... 51
Lab: Deploying Ansible ... 54
Summary ... 60

3. Implementing Playbooks 61
Writing YAML Files ... 62
Guided Exercise: Writing YAML Files ... 68
Implementing Modules ... 71
Guided Exercise: Implementing Modules ... 78
Implementing Ansible Playbooks ... 81
Guided Exercise: Implementing Ansible Playbooks ... 94
Lab: Implementing Playbooks ... 101
Summary ... 109

4. Managing Variables and Inclusions 111
Managing Variables ... 112
Guided Exercise: Managing Variables ... 122
Managing Facts ... 126
Guided Exercise: Managing Facts ... 134
Managing Inclusions ... 139
Guided Exercise: Managing Inclusions ... 148
Lab: Managing Variables and Inclusions ... 153
Summary ... 162

5. Implementing Task Control 163
Constructing Flow Control ... 164
Guided Exercise: Constructing Flow Control ... 171
Implementing Handlers ... 175

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Handlers 178
Implementing Tags 182
Guided Exercise: Implementing Tags 189
Handling Errors 194
Guided Exercise: Handling Errors 202
Lab: Implementing Task Control 208
Summary 222

6. Implementing Jinja2 Templates **223**
Describing Jinja2 Templates 224
Quiz: Describing Jinja2 Templates 227
Implementing Jinja2 Templates 229
Guided Exercise: Implementing Jinja2 Templates 231
Lab: Implementing Jinja2 Templates 234
Summary 239

7. Implementing Roles **241**
Describing Role Structure 242
Quiz: Describing Role Structure 246
Creating Roles 248
Guided Exercise: Creating Roles 251
Deploying Roles with Ansible Galaxy 258
Guided Exercise: Deploying Roles with Ansible Galaxy 263
Lab: Implementing Roles 268
Summary 279

8. Optimizing Ansible **281**
Configuring Connection Types 282
Guided Exercise: Configuring Connection Types 286
Configuring Delegation 290
Guided Exercise: Configuring Delegation 295
Configuring Parallelism 301
Guided Exercise: Configuring Parallelism 305
Lab: Optimizing Ansible 310
Summary 322

9. Implementing Ansible Vault **323**
Configuring Ansible Vault 324
Guided Exercise: Configuring Ansible Vault 327
Executing with Ansible Vault 330
Guided Exercise: Executing with Ansible Vault 335
Lab: Implementing Ansible Vault 339
Summary 350

10. Troubleshooting Ansible **351**
Troubleshooting Playbooks 352
Guided Exercise: Troubleshooting Playbooks 355
Troubleshooting Ansible Managed Hosts 361
Guided Exercise: Troubleshooting Ansible Managed Hosts 364
Lab: Troubleshooting Ansible 367
Summary 376

11. Implementing Ansible Tower **377**
Describing Ansible Tower 378

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Describing Ansible Tower 381
Deploying Ansible Tower 383
Guided Exercise: Deploying Ansible Tower 388
Configuring Users in Ansible Tower 393
Guided Exercise: Configuring Users in Ansible Tower 399
Managing Hosts with Ansible Tower 401
Guided Exercise: Managing Hosts with Ansible Tower 406
Managing Jobs in Ansible Tower 409
Guided Exercise: Managing Jobs in Ansible Tower 415
Lab: Implementing Ansible Tower 418
Summary 430

12. Implementing Ansible in a DevOps Environment 431
Provisioning Vagrant Machines 432
Guided Exercise: Provisioning Vagrant Machines 438
Deploying Vagrant in a DevOps Environment 441
Guided Exercise: Deploying Vagrant in a DevOps Environment 446
Lab: Implementing Ansible in a DevOps Environment 450
Summary 456

13. Comprehensive Review: Automation with Ansible 457
Comprehensive Review 458
Lab: Deploying Ansible 461
Lab: Deploying Ansible Tower and Executing Jobs 466
Lab: Creating Roles and using Dynamic Inventory 479
Lab: Optimizing Ansible 495

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Introduction

Automation with Ansible

Automation with Ansible (DO407) is designed for system administrators who are intending to use Ansible for automation, configuration, and management. Students will learn how to install and configure Ansible. Students will also create and run playbooks to configure systems, and learn to manage inventories. Students will manage encryption for Ansible with Ansible Vault, and deploy Ansible Tower and use Tower to manage systems. Students will use Ansible in a DevOps environment with Vagrant.

Objectives

- Automate system administration tasks on managed hosts with Ansible.
- Learn how to write Ansible playbooks to standardize task execution.
- Centrally manage playbooks and schedule recurring execution through a web interface with Ansible Tower.

Audience

- System and cloud administrators needing to automate cloud provisioning, configuration management, application deployment, intra-service orchestration, and other IT needs.

Prerequisites

- Red Hat Certified System Administrator (RHCSA in Red Hat Enterprise Linux) certification or equivalent experience.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Orientation to the Classroom Environment

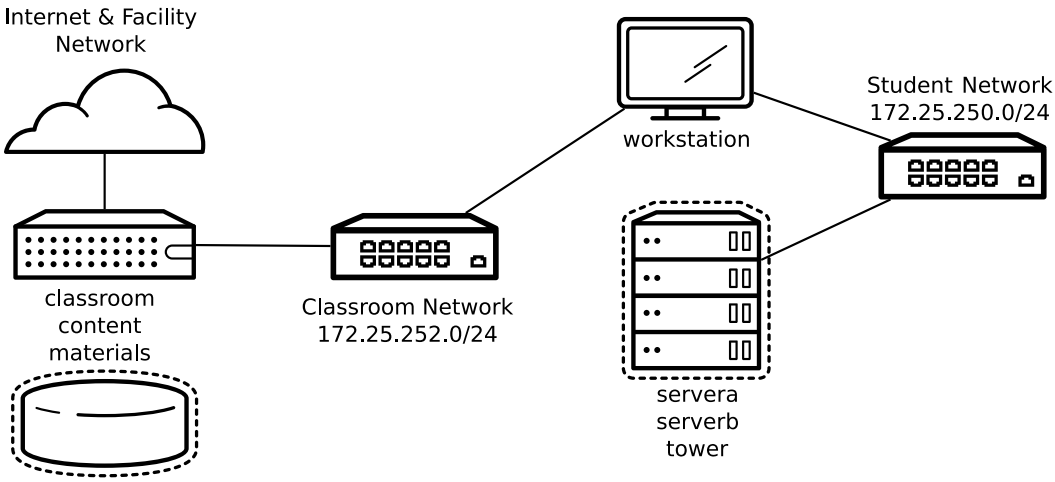


Figure 0.1: Classroom Environment

In this course, the main computer system used for hands-on learning activities is **`workstation`**. Three other machines will also be used by students for these activities. These are **`servera`**, **`serverb`**, and **`tower`**. All four of these systems are in the **`lab.example.com`** DNS domain.

All student computer systems have a standard user account, *student*, which has the password *student*. The *root* password on all student systems is *redhat*.

Classroom Machines

Machine name	IP addresses	Role
workstation.lab.example.com	172.25.250.254	Graphical workstation used to run most Ansible management commands
servera.lab.example.com	172.25.250.10	Host managed with Ansible
serverb.lab.example.com	172.25.250.11	Host managed with Ansible
tower.lab.example.com	172.25.250.12	Host used for Ansible Tower and Vagrant

One additional function of **`workstation`** is that it acts as a router between the network that connects the student machines and the classroom network. If **`workstation`** is down, other student machines will only be able to access systems on the student network.

There are several systems in the classroom that provide supporting services. Two servers, **`content.example.com`** and **`materials.example.com`** are sources for software and lab materials used in hands-on activities. Information on how to use these servers will be provided in the instructions for those activities.

Controlling your station

The top of the console describes the state of your machine.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Machine States

State	Description
none	Your machine has not yet been started. When started, your machine will boot into a newly initialized state (the desk will have been reset).
starting	Your machine is in the process of booting.
running	Your machine is running and available (or, when booting, soon will be.)
stopping	Your machine is in the process of shutting down.
stopped	Your machine is completely shut down. Upon starting, your machine will boot into the same state as when it was shut down (the disk will have been preserved).
impaired	A network connection to your machine cannot be made. Typically this state is reached when a student has corrupted networking or firewall rules. If the condition persists after a machine reset, or is intermittent, please open a support case.

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

Machine Actions

Action	Description
Start Station	Start ("power on") the machine.
Stop Station	Stop ("power off") the machine, preserving the contents of its disk.
Reset Station	Stop ("power off") the machine, resetting the disk to its initial state. **Caution: Any work generated on the disk will be lost**.
Refresh	Refresh the page will re-probe the machine state.
Increase Timer	Adds 15 minutes to the timer for each click.

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 60 minutes when your machine is started.

The timer operates as a "dead man's switch," which decrements as your machine is running. If the timer is winding down to 0, you may choose to increase the timer.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
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.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 1

INTRODUCING ANSIBLE

Overview	
Goal	Describe the terminology and architecture of Ansible.
Objectives	• Describe Ansible concepts, reference architecture, and use cases. • Describe Ansible deployments and orchestration methods. • Describe Ansible inventory concepts.
Sections	• Overview of Ansible Architecture (and Quiz) • Overview of Ansible Deployments (and Quiz) • Describing Ansible Inventory (and Quiz)
Quiz	• Quiz: Introducing Ansible

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Overview of Ansible Architecture

Objectives

After completing this section, students should be able to:

- Describe Ansible concepts, reference architecture, and use cases.

What is Ansible?

Ansible is an open source configuration management and orchestration utility. It can automate and standardize the configuration of remote hosts and virtual machines. Its orchestration functionality allows Ansible to coordinate the launch and graceful shutdown of multitiered applications. Because of this, Ansible can perform rolling updates of multiple systems in a way that results in zero downtime.

Instead of writing custom, individualized scripts, system administrators create high-level *plays* in Ansible. A play performs a series of *tasks* on the host, or group of hosts, specified in the play. A file that contains one or more plays is called a *playbook*.

Ansible's architecture is agentless. Work is pushed to remote hosts when Ansible executes. *Modules* are the programs that perform the actual work of the tasks of a play. Ansible is immediately useful because it comes with hundreds of *core modules* that perform useful system administrative work.

Ansible was originally written by Michael DeHaan, the creator of the Cobbler provisioning application. Ansible has been widely adopted, because it is simple to use for system administrators. Developers ease into using Ansible because it is built on Python. Ansible is supported by DevOps tools, such as Vagrant and Jenkins.

There are many things that Ansible can not do. Ansible can not audit changes made locally by other users on a system. For example, who made a change to a file? The following list provides some other examples of items that Ansible can not perform.

- Ansible can add packages to an installation, but it does not perform the initial minimal installation of the system. Every system can start with a minimal installation, either via Kickstart or a base cloud starter image, then use Ansible for further configuration.
- Although Ansible can remediate configuration drift, it does not monitor for it.
- Ansible does not track what changes are made to files on the system, nor does it track what user or process made those changes. These types of changes are best tracked with a version control system or the Linux Auditing System.

Ansible concepts and architecture

There are two types of machines in the Ansible architecture: the *control node* and *managed hosts*. Ansible software is installed on the control node and all of its components are maintained on it. The managed hosts are listed in a *host inventory*, a text file on the control node that includes a list of managed host names or IP addresses.

System administrators log into the control node and launch Ansible, providing it with a *playbook* and a target host to manage. Instead of a single system to control, a group of hosts or a wildcard

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

can be specified. Ansible uses SSH as a network transport to communicate with the managed hosts. The modules referenced in the playbook are copied to the managed hosts. Then they are executed, in order, with the arguments specified in the playbook. Ansible users can write their own *custom modules*, if needed, but the core modules that come with Ansible can perform most system administration tasks.

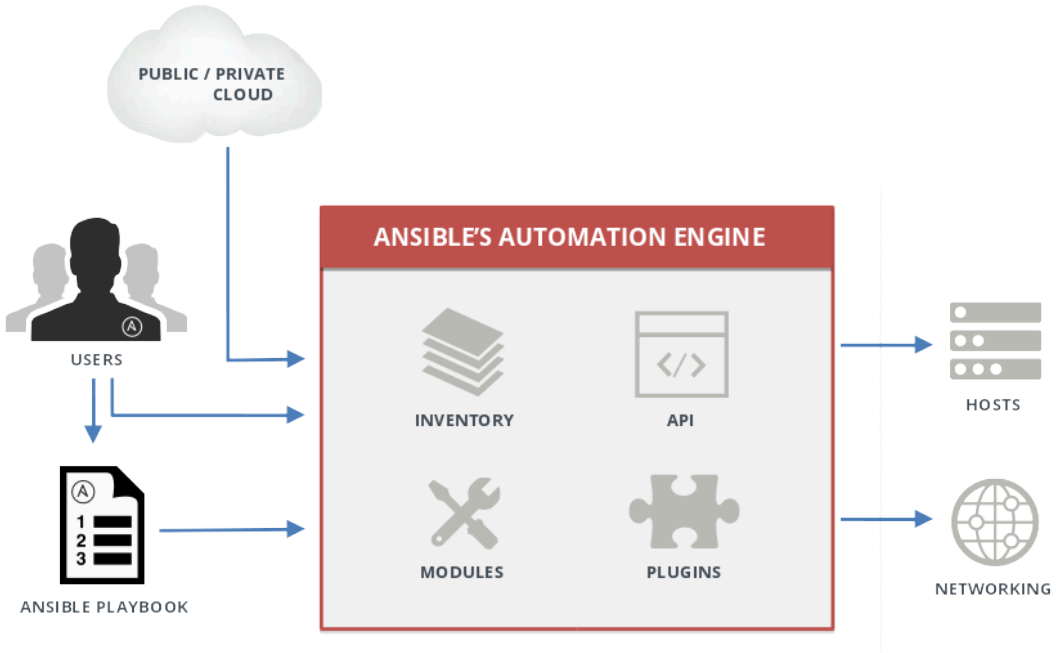


Figure 1.1: Ansible architecture

The following table lists and describes the Ansible components that are maintained on the control node.

Ansible control node components

Component	Description
Ansible configuration	Ansible has configuration settings that defines how it behaves. These settings include such things as the remote user to execute commands, and passwords to provide when executing remote commands with **sudo**. Default configuration values can be overridden by environment variables or values defined in configuration files.
Host inventory	The Ansible host inventory defines which configuration groups hosts belong to. The inventory can define how Ansible communicates with a managed host and it also defines host and group variable values.
Core modules	Core modules are the modules that come bundled with Ansible, There are over 400 core modules.
Custom modules	Users can extend Ansible's functionality by writing their own modules and adding them to the Ansible library. Modules are typically written in Python, but they can be written in any interpreted programming language (shell, Ruby, Python, etc.).
Playbooks	Ansible playbooks are files written in YAML syntax that define the modules, with arguments, to apply to managed nodes. They declare the tasks that need to be performed.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Component	Description
Connection plugins	Plugins that enable communication with managed hosts or cloud providers. These include native SSH, paramiko SSH, and local. Paramiko is a Python implementation of OpenSSH for Red Hat Enterprise Linux 6 that provides the **`ControlPersist`** setting that improves Ansible performance.
Plugins	Extensions that enhance Ansible's functionality. Examples include e-mail notifications and logging.

Role and requirements for the control node

System administrators log in and initiate all Ansible operations from the control node. Ansible software is installed on the control node. Ansible configuration files are also maintained on the control node. Other names for the control node include Ansible host and control machine, but this course will consistently use the term "control node" for the machine that serves this role.

A machine acting as a control node must have Python 2.6 or 2.7 installed. This includes Linux, OS/X, and any BSD based Unix system. For Red Hat Enterprise Linux, the *ansible* package and its dependencies must be installed. Red Hat Enterprise Linux 6 or 7 will run Ansible software. Windows is not supported for the control node at this time.

Role and requirements for the managed hosts

A managed host is a system that Ansible logs into, installs modules, and executes remote commands to perform configuration tasks. Other names for a managed host include managed node and remote node. This course will consistently use the term "managed host" for machines managed by Ansible.

Ansible uses SSH to communicate with managed hosts, so SSH must be installed and configured to allow incoming connections. Managed hosts must have Python 2.4 or later installed to run Ansible, which includes Red Hat Enterprise Linux 5, 6, and 7 hosts.

The *python-simplejson* package must also be installed on Red Hat Enterprise Linux 5 managed hosts. It is not required on Red Hat Enterprise Linux 6 and 7 managed hosts, since Python 2.5 (and newer versions) provide its functionality by default.



Note

Modules can have their own, unique software requirements. Playbooks must be written so that prerequisite software is installed before a module is called that uses it.

Ansible use cases

When administrators think of Ansible, one thing that comes to mind is configuration management. Ansible can deploy and manipulate the configuration files of a remote host. The files can be static or created on the fly, using templates.

Ansible can also be used as a multi-node deployment tool. Playbooks can define the applications that are installed and configured on remote machines. A playbook can be applied to multiple machines, building them all in a consistent manner. Also, multi-node applications can be orchestrated by Ansible's rules.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Remote task execution can also be performed by Ansible. This can be demonstrated by specifying ad hoc commands on the command line, causing Ansible to execute them on remote hosts.



References

Installation — Ansible Documentation
http://docs.ansible.com/ansible/intro_installation.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Ansible Architecture

Choose the correct answer to the following questions:

1. Which of the following programming languages is Ansible built on?

   a. C++
   b. Perl
   c. Python
   d. Ruby

2. Which of the following terms best describes Ansible's architecture?

   a. Agentless
   b. Client/Server
   c. Event-driven
   d. Stateless

3. What is the network protocol that Ansible uses, by default, to communicate with managed nodes?

   a. HTTP
   b. HTTPS
   c. Paramiko
   d. SNMP
   e. SSH

4. Which of the following files define the actions Ansible performs on managed nodes?

   a. Configuration file
   b. Host inventory
   c. Manifest
   d. Playbook
   e. Script

5. What syntax is used to define Ansible playbooks?

   a. Bash
   b. Perl
   c. Python
   d. YAML

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. Which of the following programming languages is Ansible built on?

   a. C++
   b. Perl
   c. Python
   d. Ruby

2. Which of the following terms best describes Ansible's architecture?

   a. Agentless
   b. Client/Server
   c. Event-driven
   d. Stateless

3. What is the network protocol that Ansible uses, by default, to communicate with managed nodes?

   a. HTTP
   b. HTTPS
   c. Paramiko
   d. SNMP
   e. SSH

4. Which of the following files define the actions Ansible performs on managed nodes?

   a. Configuration file
   b. Host inventory
   c. Manifest
   d. Playbook
   e. Script

5. What syntax is used to define Ansible playbooks?

   a. Bash
   b. Perl
   c. Python
   d. YAML

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Overview of Ansible Deployments

Objectives

After completing this section, students should be able to:

- Describe Ansible deployments and orchestration methods.

Ansible deployments

One of Ansible's strengths is in how it simplifies the configuration of software on servers. When Ansible accesses managed hosts, it can discover the version of RHEL running on the remote server. The installed applications and applied software subscriptions can be compared to determine if the host is properly entitled. Ansible playbooks can be used to consistently build development, test, and production servers. Kickstart can get bare-metal servers running enough to let Ansible take over and build them further. They can be provisioned to a corporate baseline standard, or they can be built for a specific role within the datacenter.

Similarly, Ansible can discover JBoss EAP versions and reconcile subscriptions. Since Ansible supports managed hosts running Windows, JBoss products can be deployed consistently, regardless of the target machines' operating systems. Ansible can be used to take the next step and deploy and manage JBoss applications. All of the JBoss EAP configurations may be centrally stored in or managed by the Ansible project on the control node.

Ansible can also be used to supplement the functionality provided by Red Hat Satellite. It can deploy Satellite agents to existing servers in a datacenter. As discussed previously, Ansible can discover and manage software subscriptions on Satellite clients. It can also perform post-install configuration of hosts provisioned by Red Hat Satellite.

Ansible orchestration methods

Ansible is commonly used to finish provisioning application servers. For example, a playbook can be written to perform the following steps on a newly installed base system:

1. Configure software repositories.
2. Install the application.
3. Tune configuration files. Optionally download content from a version control system.
4. Open required service ports in the firewall.
5. Start relevant services.
6. Test the application and confirm it is functioning.

Orchestrating zero-downtime rolling upgrades

Ansible is also simple tool to use for updating applications in parallel. For example, a playbook can be developed to execute the following steps on application servers:

1. Stop system and application monitoring.
2. Remove the server from load balancing.
3. Stop relevant services.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4. Deploy, or update, the application.
5. Start relevant services.
6. Confirm the services are available and add the server back to load balancing.
7. Start system and application monitoring.

The **serial** keyword can be used to limit the number of hosts that the playbook runs on at once. Once the subset of servers have been deployed and are functioning properly, Ansible will move on to another batch of servers in the target group. By default, Ansible will try to apply a playbook to the target managed servers in parallel, with the exact number of parallel processes to spawn controlled by the **forks** directive in the applicable **ansible.cfg** configuration file.

Ansible connection plugins

Connection plugins allow Ansible to communicate with managed hosts and cloud providers. The preferred connection plugin for newer versions of Ansible is the native SSH plugin, **ssh**. It is connection method that Ansible uses when OpenSSH on the control node supports the **ControlPersist** option. Although Ansible can be configured to use passwords for SSH authentication, the most common practice is to use SSH user keys to get access to managed hosts.

Another connection plugin used for Linux applications is the **local** connection plugin. It can be used to manage the Ansible control node locally, without having to use SSH. This connection method is typically used when writing Ansible playbooks that interface with cloud services or some other API. It can also be used when Ansible is invoked locally by a **cron** job.

The **paramiko** Ansible connection plugin is used on Red Hat Enterprise Linux 6 machines. Paramiko SSH is a Python-based OpenSSH implementation that implements persistent SSH connections. It was a connection solution for older systems, where versions of OpenSSH did not implement the **ControlPersist** connection setting. The **ControlPersist** connection setting allows for persistent SSH connections, a feature that improves Ansible performance by eliminating SSH connection overhead, when multiple SSH commands are executed in succession.

The **winrm** Ansible connection plugin allows Microsoft Windows machines to be managed hosts. The **pywinrm** Python module must be installed on the Linux control node to support this connection plugin.

Ansible 2 introduced the **docker** connection plugin. The Docker host serves as the Ansible control node, and the containers act as managed hosts. When this connection plugin is used, each container does not have to enable an SSH server to allow Ansible communication.



References

Ansible — Use Case: Application Deployment
https://www.ansible.com/application-deployment

Integration: Ansible & Red Hat
https://www.ansible.com/red-hat

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Ansible Deployments

Choose the correct answer to the following questions:

1. Which of the following is ***not*** a deployment task suitable for Ansible?

    a. Deploy JBoss consistently over different operating systems.
    b. Deploy Red Hat Satellite agents to existing servers in a datacenter.
    c. Discover the operating system version and software subscription status of Red Hat Enterprise Linux servers.
    d. Monitor the state of the system to ensure it does not experience configuration drift.
    e. Manage the software development life cycle of OpenShift Enterprise applications.

2. Which of the following Ansible keywords facilitates zero-downtime rolling updates to occur by limiting the number of managed hosts a playbook can run on in parallel?

    a. **`accelerate`**
    b. **`gather_subset`**
    c. **`handlers`**
    d. **`serial`**
    e. **`tasks`**

3. The **`paramiko`** Ansible connection plugin is used to communicate with which type of managed host?

    a. Docker containers
    b. Red Hat Enterprise Linux 6 servers
    c. Red Hat Enterprise Linux 7 servers
    d. Windows servers

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. Which of the following is ***not*** a deployment task suitable for Ansible?

   a. Deploy JBoss consistently over different operating systems.
   b. Deploy Red Hat Satellite agents to existing servers in a datacenter.
   c. Discover the operating system version and software subscription status of Red Hat Enterprise Linux servers.
   d. Monitor the state of the system to ensure it does not experience configuration drift.
   e. Manage the software development life cycle of OpenShift Enterprise applications.

2. Which of the following Ansible keywords facilitates zero-downtime rolling updates to occur by limiting the number of managed hosts a playbook can run on in parallel?

   a. `accelerate`
   b. `gather_subset`
   c. `handlers`
   d. `serial`
   e. `tasks`

3. The **`paramiko`** Ansible connection plugin is used to communicate with which type of managed host?

   a. Docker containers
   b. Red Hat Enterprise Linux 6 servers
   c. Red Hat Enterprise Linux 7 servers
   d. Windows servers

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Describing Ansible Inventory

Objectives

After completing this section, students should be able to:

- Describe Ansible inventory concepts.

Ansible inventories

A host inventory defines which hosts Ansible manages. Hosts may belong to groups which are typically used to identify the hosts' role in the datacenter. A host can be a member of more than one group.

There are two ways in which host inventories can be defined. A *static* host inventory may be defined by a text file, or a *dynamic* host inventory may be generated from outside providers.

Static host inventory

An Ansible static host inventory is defined in an INI-like text file, in which each section defines one group of hosts (a *host group*). Each section starts with a host group name enclosed in square brackets ([]). Then *host entries* for each managed host in the group are listed, each on a single line. They consist of the host names or IP addresses of managed hosts which are members of the group. In the following example, the host inventory defines two host groups, **webservers** and **db-servers**.

```
[webservers]
localhost             ansible_connection=local
web1.example.com
web2.example.com:1234 ansible_connection=ssh ansible_user=ftaylor
192.168.3.7

[db-servers]
web1.example.com
db1.example.com
```

Host entries can also define how Ansible communicates with the managed host, including transport and user account information. In the previous example, SSH on **web2.example.com** is configured to listen on a non-standard port, port 1234. The account Ansible should use to log into that host is **ftaylor**.

The default location for the host inventory file is **/etc/ansible/hosts**. The **ansible*** commands will use a different host inventory file when they are used with the **--inventory *PATHNAME*** option, **-i *PATHNAME*** for short.

Defining host inventory groups of groups

Ansible host inventories can include groups of host groups. This is accomplished with the **:children** suffix. The following example creates a new group, called **nwcapitols**, that includes all of the hosts from the **olympia** and **salem** groups.

```
[olympia]
washington1.example.com
washington2.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[salem]
oregon01.example.com
oregon02.example.com

[nwcapitols:children]
olympia
salem
```

Simplifying host inventories with ranges

Ansible host inventories can be simplified by specifying ranges in the host names or IP addresses. Numeric ranges can be specified, but alphabetic ranges are also supported. Ranges have the following syntax:

```
[START:END]
```

Ranges match all the values between *START* and *END*, inclusive. Consider the following examples:

- **`192.168.[4:7].[0:255]`** - all IP addresses in the 192.168.4.0/22 network (192.168.4.0 through 192.168.7.255).
- **`server[01:20].example.com`** - all hosts named **`server01.example.com`** through **`server20.example.com`**.

If leading zeros are included in numeric ranges, they will be used in the pattern. The second example above will not match **`server1.example.com`**. To illustrate this, the following example simplifies the **`olympia`** and **`salem`** group definitions from the earlier example, by using ranges.

```
[olympia]
washington[1:2].example.com

[salem]
oregon[01:02].example.com
```

When in doubt, test the machine's presence in the inventory with the **`ansible`** command:

```
[user@demo ~]$ ansible washington1.example.com --list-hosts
  hosts (1):
    washington1.example.com
[user@demo ~]$ ansible washington01.example.com --list-hosts
 [WARNING]: provided hosts list is empty, only localhost is available

  hosts (0):
```

Defining variables in the host inventory

Values for variables used by playbooks can be specified in host inventory files. These variables would apply to specific hosts or host groups only. Normally it is better to define these *inventory variables* in special directories and not directly in the actual inventory file. This topic will be discussed in more depth elsewhere in the course.

Dynamic host inventory

Ansible host inventory information can also be dynamically generated. Sources for dynamic inventory information include public/private cloud providers, Cobbler system information, an LDAP database, or a configuration management database (CMDB). Ansible includes scripts that

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

handle dynamic host, group, and variable information from the most common providers, such as Amazon EC2, Cobbler, Rackspace Cloud, and OpenStack. For cloud providers, authentication and access information must be defined in files that the scripts can access. This topic will be discussed in more detail later in this course.



References

Inventory — Ansible Documentation
http://docs.ansible.com/ansible/intro_inventory.html

Dynamic Inventory — Ansible Documentation
http://docs.ansible.com/ansible/intro_dynamic_inventory.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Ansible Inventory

Choose the correct answer to the following questions:

1. Which of the following items is ***not*** found in an Ansible inventory file?

   a. Host groups
   b. IP address ranges
   c. Module names
   d. Variable definitions
   e. User authentication information

2.
```
[linux-dev]
cchang.example.com
rlocke.example.com

[windows-dev]
wdinyes.example.com

[development:children]
linux-dev
windows-dev
```

   Given the Ansible inventory in the above exhibit, which host group, or groups, includes the **`rlocke.example.com`** host?

   a. **`linux-dev`**
   b. **`windows-dev`**
   c. **`development`**
   d. Both **`linux-dev`** and **`development`**

3. Which of the following expressions can be used in an Ansible inventory file to match hosts in the 10.1.0.0/16 address range?

   a. **`10.1.0.0/16`**
   b. **`10.1.[0:255].[0:255]`**
   c. **`10.1.[0-255].[0-255]`**
   d. **`10.1.*`**

4. Which of the following can be a source for Ansible dynamic host inventory information?

   a. Cobbler system information
   b. Configuration management database
   c. LDAP database
   d. Scripts that glean information from cloud providers
   e. All of the above

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. Which of the following items is ***not*** found in an Ansible inventory file?

   a. Host groups
   b. IP address ranges
   c. Module names
   d. Variable definitions
   e. User authentication information

2.
```
[linux-dev]
cchang.example.com
rlocke.example.com

[windows-dev]
wdinyes.example.com

[development:children]
linux-dev
windows-dev
```

   Given the Ansible inventory in the above exhibit, which host group, or groups, includes the **`rlocke.example.com`** host?

   a. `linux-dev`
   b. `windows-dev`
   c. `development`
   d. Both `linux-dev` and `development`

3. Which of the following expressions can be used in an Ansible inventory file to match hosts in the 10.1.0.0/16 address range?

   a. `10.1.0.0/16`
   b. `10.1.[0:255].[0:255]`
   c. `10.1.[0-255].[0-255]`
   d. `10.1.*`

4. Which of the following can be a source for Ansible dynamic host inventory information?

   a. Cobbler system information
   b. Configuration management database
   c. LDAP database
   d. Scripts that glean information from cloud providers
   e. All of the above

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Introducing Ansible

Choose the correct answer to the following questions:

1. What is the term this course uses for a host from which system administrators run Ansible?

   a. Ansible master
   b. Client
   c. Control node
   d. Managed host

2. The ***python-simplejson*** package must be installed on which kind of host?

   a. Ansible control node
   b. Red Hat Enterprise Linux 5 managed host
   c. Red Hat Enterprise Linux 6 managed host
   d. Red Hat Enterprise Linux 7 managed host
   e. Windows managed host

3. What is the default location of the static, system-wide Ansible inventory file?

   a. **`/etc/ansible/host.groups`**
   b. **`/etc/ansible/host.inventory`**
   c. **`/etc/ansible/hosts`**
   d. **`/etc/ansible/inventory`**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. What is the term this course uses for a host from which system administrators run Ansible?

   a. Ansible master
   b. Client
   c. Control node
   d. Managed host

2. The ***python-simplejson*** package must be installed on which kind of host?

   a. Ansible control node
   b. Red Hat Enterprise Linux 5 managed host
   c. Red Hat Enterprise Linux 6 managed host
   d. Red Hat Enterprise Linux 7 managed host
   e. Windows managed host

3. What is the default location of the static, system-wide Ansible inventory file?

   a. `/etc/ansible/host.groups`
   b. `/etc/ansible/host.inventory`
   c. `/etc/ansible/hosts`
   d. `/etc/ansible/inventory`

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible is an agentless configuration management tool, based on Python.
- Ansible and its configuration files only need to be available on a system which is used to run Ansible commands, which this course calls a *control node*.
- Ansible host inventory files are INI-like files that identify the hosts and host groups that Ansible manages.
- Ansible copies *modules* from the control node to the *managed hosts*, where it executes them in the order they are specified in the Ansible *playbook*.
- Native SSH is the preferred connection plugin used by Ansible, but the Paramiko plugin allows Red Hat Enterprise Linux 6 control nodes to efficiently communicate with managed hosts.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 2

DEPLOYING ANSIBLE

Overview	
Goal	Install Ansible and run ad hoc commands.
Objectives	• Install Ansible. • Manage Ansible configuration files. • Run Ansible ad hoc commands. • Manage dynamic inventory.
Sections	• Installing Ansible (and Guided Exercise) • Managing Ansible Configuration Files (and Guided Exercise) • Running Ad Hoc Commands (and Guided Exercise) • Managing Dynamic Inventory (and Guided Exercise)
Lab	• Deploying Ansible

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Installing Ansible

Objectives

After completing this section, students should be able to:

- Install Ansible on the control node.
- Discuss connection methods to connect to managed hosts.

Ansible prerequisites

Control node

Unlike other configuration management utilities, such as Puppet and Chef, Ansible uses an agentless architecture. Ansible itself only needs to be installed on the host or hosts from which it will be run. Hosts which will be managed by Ansible do not need to have Ansible installed. This installation involves relatively few steps and has minimal requirements.

Ansible installation on the control node requires only that Python 2, version 2.6 or later be installed. Ansible does not yet use Python 3. To see whether the appropriate version of Python is installed on a Red Hat Enterprise Linux system, use the **yum** command.

```
[root@controlnode ~]# yum list installed python
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
python.x86_64     2.7.5-34.el7     installed
```

At the time of writing, Ansible is not included directly in Red Hat Enterprise Linux. Official instructions on how to obtain, install, and get updates for Ansible for RHEL, as well as for other operating systems and Linux distributions, are available on the Ansible website at *https://www.ansible.com/get-started*.

Managed hosts

Managed hosts do not need to have any special Ansible agent installed. They do need to have Python 2, version 2.4 or later installed. If the version of Python installed on the managed host is earlier than Python 2.5, then it must also have the *python-simplejson* package installed.



Note

At times, it is desirable to use Ansible to manage systems that can not have Python installed. Systems in this category, such as network routers, can be managed using Ansible's **raw** module. Arguments passed to this module are run directly through the configured remote shell instead of going through the module subsystem. However, in most other cases the **raw** module should be avoided.

The Ansible control node communicates with managed hosts over the network. Multiple options are available, but an SSH connection is used by default. Ansible normally connects to the managed host by using the same user name as the one running Ansible on the control node.

To ensure security, SSH sessions require authentication at the initiation of each connection. Relying on password authentication for each connection to each managed hosts quickly becomes

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

unwieldy when the number of managed hosts increases. Therefore, in enterprise environments, key-based authentication is a much more desirable option.

SSH key-based authentication
Users can authenticate **ssh** logins without a password by using *public key authentication*. **ssh** allows users to authenticate using a private-public key scheme. This means that two keys are generated, a private key and a public key.

The private key file is used as the authentication credential and, like a password, must be kept secret and secure. The public key is copied to systems the user wants to log in to and is used to verify the private key. The public key does not need to be secret.

An SSH server that has the public key can issue a challenge that can only be answered by a system holding the private key. As a result, possession of the private key can be used to complete the authentication. This allows system access in a way that doesn't require typing a password every time, but is still secure.

Key generation is performed using the **ssh-keygen** command. This generates the private key, **~/.ssh/id_rsa**, and the public key, **~/.ssh/id_rsa.pub**.

When the SSH keys have been generated, they are stored by default in the **.ssh/** directory of the user's home directory. The file permissions on the private key, **~/.ssh/id_rsa**, should only allow read and write access to the user that owns the file (octal 0600). The file permissions on the public key, **~/.ssh/id_rsa.pub**, should allow all users on the system to read the file but only allow the user that owns the file to have write access (octal 0644).

Before key-based authentication can be used, the public key needs to be copied to the destination system. This can be done with **ssh-copy-id**.

```
[student@controlnode ~]$ ssh-copy-id student@managedhost
```

When the public key has been copied to the destination host, future SSH connections to the host can be authenticated using key-based authentication.

Referencing inventory hosts

When Ansible has been installed, the **ansible** command becomes available for execution on the command line. The various options for the **ansible** command can be displayed with the **--help** option, or its abbreviated **-h** counterpart.

```
[student@controlnode ~]$ ansible -h
Usage: ansible <host-pattern> [options]

Options:
  -a MODULE_ARGS, --args=MODULE_ARGS

...output omitted...

  -v, --verbose         verbose mode (-vvv for more, -vvvv to enable
                        connection debugging)
  --version             show program's version number and exit
```

The **--version** option is helpful for identifying the version of Ansible installed.

```
[student@controlnode ~]$ ansible --version
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ansible 2.0.1.0
...output omitted...
```

To use the **ansible** command for host management, it must be supplied an inventory file which defines the list of hosts to be managed by the control node. One way to do this is to specify the path to the inventory file with the **-i** option to the **ansible** command. To reference inventory hosts, supply a host pattern to the **ansible** command using the following command line syntax.

```
[student@controlnode ~]$ ansible <host-pattern> -i /path/to/inventory/file [options]
```

The **--list-hosts** option can be useful for clarifying which managed hosts are referenced by the host pattern in an **ansible** command. It indicates the number of managed hosts that are referenced as well as enumerates the names of the managed hosts.

Host name

The most basic host pattern is the name for a single managed host listed in the inventory file. This specifies that the host will be the only one in the inventory file that will be acted upon by the **ansible** command.

The following example shows how a single host name is used to reference a host contained in an inventory file and how the **--list-hosts** option can be used to identify the referenced host.

```
[student@controlnode ~]$ cat myinventory
web.example.com
10.1.1.254

[lab]
labhost1.example.com
labhost2.example.com

[test]
test1.example.com
test2.example.com

[datacenter1]
labhost1.example.com
test1.example.com

[datacenter2]
labhost2.example.com
test2.example.com

[datacenter:children]
datacenter1
datacenter2

[new]
192.168.2.1
192.168.2.2

[student@controlnode ~]$ ansible web.example.com -i myinventory --list-hosts
  hosts (1):
    web.example.com
```

IP address

Inventories can identify managed hosts by IP address as well as by host name. The following example shows how a host pattern can be used to reference an IP address contained in an inventory file.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@controlnode ~]$ ansible 192.168.2.1 -i myinventory --list-hosts
  hosts (1):
    192.168.2.1
```

Groups

Host groups defined in an inventory file can also be used as host patterns when executing **ansible** on the command line. When a group name is used as a host pattern, it specifies that **ansible** will act on the hosts that are members of the named group. The following example shows how a host group name is used to reference a group of hosts defined in the inventory file.

```
[student@controlnode ~]$ ansible lab -i myinventory --list-hosts
  hosts (2):
    labhost1.example.com
    labhost2.example.com
```

Lists

Multiple entries in an inventory file can be referenced using a comma-separated list as the host pattern. This comma-separated list can contain any inventory entry, such as host name, IP, group, and so on. The following example shows how a comma-separated list of host name, IP, and group are used to reference hosts which are not defined as a group in the inventory file.

```
[student@controlnode ~]$ ansible labhost1.example.com,test,192.168.2.2 -i myinventory --
list-hosts
  hosts (4):
    labhost1.example.com
    test1.example.com
    test2.example.com
    192.168.2.2
```

All inventory items

At times, it may be desirable to reference all hosts in an inventory file. You can use the Ansible "**all**" host pattern keyword to reference all managed hosts in an inventory, as follows:

```
[student@controlnode ~]$ ansible all -i myinventory --list-hosts
  hosts (6):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
    test2.example.com
    192.168.2.1
    192.168.2.2
```

Wildcards

Another method of accomplishing the same thing as the **all** host pattern is to use the '*****' wildcard character, which matches any string. The following example shows how the '*****' host pattern can be used to reference all hosts defined in an inventory.

```
[student@controlnode ~]$ ansible '*' -i myinventory --list-hosts
  hosts (6):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
    test2.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    web.example.com
    10.1.1.254
    192.168.2.1
    192.168.2.2
```

The asterisk character can be used by itself to reference all managed hosts, but it can also be used in conjunction with partial host name, IP address, or group substrings to match a subset of hosts in an inventory. The following examples demonstrate how the asterisk character can be used in a host pattern to match a subset of hosts defined in an inventory.

The following wildcard host pattern is used to match hosts in the **example.com** domain.

```
[student@controlnode ~]$ ansible '*.example.com' -i myinventory --list-hosts
  hosts (4):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
    test2.example.com
    web.example.com
```

The following example uses a wildcard host pattern to match hosts with an IP address on the **192.168.2.0** network.

```
[student@controlnode ~]$ ansible '192.168.2.*' -i myinventory --list-hosts
  hosts (2):
    192.168.2.1
    192.168.2.2
```

In the following example, a wildcard host pattern references hosts which are members of groups whose names begin with '**datacenter**'.

```
[student@controlnode ~]$ ansible 'datacenter*' -i myinventory --list-hosts
  hosts (4):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
    test2.example.com
```

Advanced host patterns

In addition to wildcards, Ansible allows for the creation of complex host patterns using different exclusion and inclusion logic. Inclusion is accomplished by using the '**:**' character to separate groups in a host pattern to indicate an OR logic. The following example shows the use of a host pattern that references hosts which are members of either the **lab** or **datacenter1** group.

```
[student@controlnode ~]$ ansible lab:datacenter1 -i myinventory --list-hosts
  hosts (3):
    labhost1.example.com
    labhost2.example.com
    test1.example.com
```

In contrast, when used in conjunction with the '**&**' character to separate groups in a host pattern, the '**:&**' characters denote the intersections of two groups in the inventory. The following example shows the use of a host pattern referencing hosts that are members of both the **lab** and **datacenter1** groups.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@controlnode ~]$ ansible 'lab:&datacenter1' -i myinventory --list-hosts

  hosts (1):
    labhost1.example.com
```

Exclusion is accomplished using the '**:**' character in conjunction with the '**!**' character in a host pattern to indicate hosts to exclude. The following example shows the use of a host pattern that references all hosts defined in the **datacenter** group of groups with the exception of **test2.example.com**.

```
[student@controlnode ~]$ ansible 'datacenter:!test2.example.com' -i myinventory --list-
hosts
  hosts (3):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
```

The following example shows the use of a host pattern that references all hosts defined in the inventory with the exception of the managed hosts in the **datacenter1** group.

```
[student@controlnode ~]$ ansible 'all:!datacenter1' -i myinventory --list-hosts
  hosts (3):
    labhost2.example.com
    test2.example.com
    web.example.com
    10.1.1.254
    192.168.2.1
    192.168.2.2
```



Note

More complex host patterns, such as those which use regular expressions, can be created in Ansible. These advanced host patterns are beyond the scope of this course. For more information on their usage, please consult the references listed at the end of this section.



Important

Some wildcard characters available for use in host patterns also have meaning for the shell. Thereby, it is not only good practice but a requirement to have them escaped using, for example, single quotes. These characters include the asterisk (*****), the ampersand (**&**), and the exclamation (**!**).

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

ansible(1) man page

Installation — Ansible Documentation
http://docs.ansible.com/ansible/intro_installation.html

Patterns — Ansible Documentation
http://docs.ansible.com/ansible/intro_patterns.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Installing Ansible

In this exercise, you will install Ansible on a control node and configure it for connections to a managed host.

Outcomes
You should be able to connect to a managed host from a control node using Ansible.

Before you begin
Log in as the **student** user on **workstation** and run **lab install setup**. This setup script ensures that the managed host, **servera**, is reachable on the network.

```
[student@workstation ~]$ lab install setup
```

Steps

1. Verify that the prerequisite version of Python is installed on **workstation**.

```
[student@workstation ~]$ yum list installed python
```

2. Install Ansible on **workstation** so that it can serve the role of the control node.

```
[student@workstation ~]$ sudo yum install -y ansible
```

3. Verify that you can establish an SSH connection from **workstation** to **servera** to ensure that Ansible will be able to connect from the control node to the managed host using SSH. Exit the SSH session after the verification is complete.

```
[student@workstation ~]$ ssh servera.lab.example.com
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to the
 list of known hosts.
[student@servera ~]$ exit
```

4. On the control node, create an inventory file, **/home/student/dep-install/inventory**, which contains a group called **dev** made up of a single managed host, **servera.lab.example.com**.

   4.1. Create and change directory to the **/home/student/dep-install** directory.

```
[student@workstation ~]$ mkdir /home/student/dep-install
[student@workstation ~]$ cd /home/student/dep-install
```

   4.2. Create the inventory file and add the following entries to the file to create the **dev** host group and its single member (**servera.lab.example.com**).

```
[dev]
servera.lab.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5. Execute the **ansible** command with the **-i** option to use the newly created inventory file and use the **--list-hosts** option to verify that the **dev** host group resolves to its single member, **servera**.

```
[student@workstation dep-install]$ ansible dev -i inventory --list-hosts
  hosts (1):
    servera.lab.example.com
```

6. Run **lab install grade** on **workstation** to grade your work.

```
[student@workstation ~]$ lab install grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Ansible Configuration Files

Objectives

After completing this section, students should be able to:

- Discuss Ansible configuration file locations.
- Discuss Ansible configuration file syntax.
- Set some Ansible configuration.

Configuring Ansible

The behavior of an Ansible installation can be customized by modifying settings housed in Ansible's configuration file. Ansible will select its configuration file from one of several possible locations on the control node.

Using `/etc/ansible/ansible.cfg`

When installed, the *ansible* package provides a base configuration file located at **`/etc/ansible/ansible.cfg`**. This file will be used if no other configuration file is found.

Using `~/.ansible.cfg`

Ansible will look for a **`~/.ansible.cfg`** in the user's home directory. This configuration will be used instead of the **`/etc/ansible/ansible.cfg`** if it exists and if there is no **`ansible.cfg`** in the current working directory.

Using `./ansible.cfg`

If an **`ansible.cfg`** file exists in the directory in which the **`ansible`** command is executed, it is used instead of the global file or the user's personal file. This allows administrators to create a directory structure where different environments or projects are housed in separate directories with each directory containing a configuration file tailored with a unique set of settings.



Important

The recommended practice is to create an **`ansible.cfg`** file in a directory from which you run Ansible commands. This directory would also contain any files used by your Ansible project, such as a playbook. This is the most common location used for the Ansible configuration file. It is unusual to use a **`~/.ansible.cfg`** or **`/etc/ansible/ansible.cfg`** file in practice.

Using `$ANSIBLE_CONFIG`

Users can make use of different configuration files by placing them in different directories and then executing Ansible commands from the appropriate directory, but this method can be restrictive and hard to manage as the number of configuration files grows. A more flexible option is to define the location of the configuration file with the **`$ANSIBLE_CONFIG`** environment variable. When this variable is defined, Ansible uses the configuration file the variable specifies instead of any of the previously mentioned configuration files.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuration file precedence

The search order for a configuration file is the reverse of the list above. The global **/etc/ansible/ansible.cfg** will only be used if no other configuration file is specified and the file specified with the **$ANSIBLE_CONFIG** environment variable will override all other configuration files. The first file located in the search order is the one from which Ansible will use configuration settings. Ansible will only use settings from this configuration file. Even if other files with lower precedence exist, their settings will be ignored and not combined with those in the selected configuration file.

Therefore, if users choose to create their own configuration file in favor of the global **/etc/ansible/ansible.cfg** configuration file, they need to duplicate all desired settings from there to their user level configuration file. Settings not defined in the user level configuration file remain unset even if they are set in the global configuration file.

Due to the multitude of locations where Ansible configuration files can be placed, it may be confusing which configuration file is being used by Ansible, especially when multiple files exist on the control node. To clearly identify the configuration file in use, execute the **ansible** command with the **--version** option. In addition to displaying the version of Ansible installed, it also displays the currently active configuration file.

```
[student@controlnode ~]$ ansible --version
ansible 2.0.1.0
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

Another way to display the active Ansible configuration file is to use the **-v** option when executing Ansible commands on the command line.

```
[student@controlnode ~]$ ansible servers --list-hosts -v
Using /etc/ansible/ansible.cfg as config file
...output omitted...
```

Ansible configuration file

The Ansible configuration file consists of several sections with each section containing settings defined as key/value pairs. Section titles are enclosed in square brackets. Settings are grouped under the following six sections in the default Ansible configuration file.

```
[student@controlnode ~]$ grep "^\[" /etc/ansible/ansible.cfg
[defaults]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[accelerate]
[selinux]
```

Most of the settings in the configuration file are grouped under the **[defaults]** section. The **[privilege_escalation]** section contains settings for defining how operations which require escalated privileges will be executed on managed hosts. The **[paramiko_connection]**, **[ssh_connection]**, and **[accelerate]** sections contain settings for optimizing connections to managed hosts. The **[selinux]** section contains settings for defining how SELinux interactions will be configured. While not included in the default global Ansible configuration file

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

provided by the *ansible* package, a **[galaxy]** section is also available for defining parameters related to Ansible Galaxy, which is discussed in a later chapter.

Settings are customized by changing their values in the currently active configuration file and take effect as soon as the file is saved. Some settings are predefined within Ansible with default values and these values are valid even if their respective settings are commented out in the configuration file. To identify the default value of these predefined settings, consult the comments in the global **/etc/ansible/ansible.cfg** configuration file supplied by the *ansible* package. The **ansible** man page also provides information on the default values of these predefined settings.

The following table highlights some commonly modified Ansible configuration settings.

Ansible settings

Setting	Decription
inventory	Location of the Ansible inventory file.
remote user	The remote user account used to establish connections to managed hosts.
become	Enables or disables privilege escalation for operations on managed hosts.
become_method	Defines the privilege escalation method on managed hosts.
become_user	The user account to escalate privileges to on managed hosts.
become_ask_pass	Defines whether privilege escalation on managed hosts should prompt for a password.

Demonstration: Customizing Ansible configuration

1. On **workstation**, log in as the **student** user. Change directory to **/home/student** and execute the **ansible** command with the **--version** option to determine the active Ansible configuration file.

```
[student@workstation ~]$ ansible --version
ansible 2.0.1.0
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

2. Open the **/etc/ansible/ansible.cfg** file and examine the different sections and their settings. Under the **[defaults]** section, locate and examine the **inventory** setting. Also examine the settings under the **[privilege_escalation]** section.

```
[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts

...output omitted...

[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
...output omitted...
```

3. Create a user level Ansible configuration file at **/home/student/.ansible.cfg** and then reverify the active Ansible configuration file.

```
[student@workstation ~]$ touch /home/student/.ansible.cfg
[student@workstation ~]$ ansible --version
ansible 2.0.1.0
  config file = /home/student/.ansible.cfg
...output omitted...
```

4. Create another user level Ansible configuration file at **/home/student/dep-config/ansible.cfg** and then reverify the active Ansible configuration file.

```
[student@workstation ~]$ mkdir /home/student/dep-config
[student@workstation ~]$ touch /home/student/dep-config/ansible.cfg
[student@workstation ~]$ cd /home/student/dep-config
[student@workstation dep-config]$ ansible --version
ansible 2.0.1.0
  config file = /home/student/dep-config/ansible.cfg
...output omitted...
```

5. Create another user level Ansible configuration file at **/home/student/dep-config/myansible.cfg**. Set the **$ANSIBLE_CONFIG** environment variable to the full path name of the new configuration file and then reverify the active Ansible configuration file.

```
[student@workstation dep-config]$ touch /home/student/dep-config/myansible.cfg
[student@workstation dep-config]$ export ANSIBLE_CONFIG=/home/student/dep-config/
myansible.cfg
[student@workstation dep-config]$ ansible --version
ansible 2.0.1.0
  config file = /home/student/dep-config/myansible.cfg
...output omitted...
```

6. Modify the active Ansible configuration file to set the default inventory location to **/home/student/dep-config/inventory**. The **/home/student/dep-config/myansible.cfg** file should have the following contents.

```
[defaults]
inventory=/home/student/dep-config/inventory
```

7. Create a new inventory file, **/home/student/dep-config/inventory**, with the following contents.

```
[classroom]
servera.lab.example.com
serverb.lab.example.com
tower.lab.example.com
```

8. Verify that the new configuration setting is in effect by executing the **ansible** command with the **--list-hosts** option in conjunction with the **classroom** group as a host pattern.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dep-config]$ ansible classroom --list-hosts
  hosts (3):
    servera.lab.example.com
    serverb.lab.example.com
    tower.lab.example.com
```



References

ansible(1) man page

> Configuration file — Ansible Documentation
> http://docs.ansible.com/ansible/intro_configuration.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Ansible Configuration Files

In this exercise, you will customize your Ansible environment.

Outcomes
You should be able to configure your Ansible environment with persistent custom settings.

Before you begin
Log in as the **student** user on **workstation** and run **lab manage setup**. This setup script ensures that the managed host, **servera**, is reachable on the network.

```
[student@workstation ~]$ lab manage setup
```

Steps

1. Create the directory, **/home/student/dep-manage**, to contain the files for this exercise. Change to this newly created directory.

```
[student@workstation ~]$ mkdir /home/student/dep-manage
[student@workstation ~]$ cd /home/student/dep-manage
```

2. Create a directory level Ansible configuration file, **ansible.cfg**, in the newly created directory. The configuration file should configure Ansible so that it uses an inventory located at **/home/student/dep-manage/inventory**. Place the following configuration entries in the **/home/student/dep-manage/ansible.cfg** file.

```
[defaults]
inventory=/home/student/dep-manage/inventory
```

3. In the directory level inventory file, **/home/student/dep-manage/inventory**, create a **myself** host group consisting of the **localhost** host, an **intranetweb** host group consisting of the **servera.lab.example.com** host, and an **everyone** host group comprised of the **myself** and **intranetweb** groups.

   3.1. In **/home/student/dep-manage/inventory**, create the **myself** host group by adding the following entries.

```
[myself]
localhost
```

   3.2. In **/home/student/dep-manage/inventory**, create the **intranetweb** host group by adding the following entries.

```
[intranetweb]
servera.lab.example.com
```

   3.3. In **/home/student/dep-manage/inventory**, create the **everyone** host group by adding the following entries.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[everyone:children]
myself
intranetweb
```

4. Verify the resolution of the **myself** host group with the **ansible** command. Use the **--list-hosts** option to list the managed hosts referenced by the host group.

```
[student@workstation dep-manage]$ ansible myself --list-hosts
  hosts (1):
    localhost
```

5. Activate and configure privilege escalation by adding the appropriate settings under the **[privilege_escalation]** section of the **/home/student/dep-manage/ansible.cfg** file.

   5.1. Create the **privilege_escalation** section in the **/home/student/dep-manage/ansible.cfg** configuration file by adding the following entry.

```
[privilege_escalation]
```

   5.2. Enable privilege escalation by adding the **become** setting and configuring it as **True**.

```
become=True
```

   5.3. Set the privilege escalation to use **sudo** by adding the **become_method** setting and configuring it as **sudo**.

```
become_method=sudo
```

   5.4. Set the privilege escalation user by adding the **become_user** setting and configuring it as **root**.

```
become_user=root
```

   5.5. Enable password prompting during privilege escalation by adding the **become_ask_pass** setting and configuring it as **True**.

```
become_ask_pass=True
```

6. Verify the new changes to **/home/student/dep-manage/inventory** and **/home/student/dep-manage/ansible.cfg**. Use **ansible** with the **--list-hosts** option to resolve the **intranetweb** host group to its managed host members. When prompted for the sudo password, enter **student**.

```
[student@workstation dep-manage]$ ansible intranetweb --list-hosts -v
Using /home/student/dep-manage/ansible.cfg as config file
SUDO password: student
  hosts (1):
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    servera.lab.example.com
```

Notice how privilege escalation is occurring using **sudo** and prompting for a password. Also notice how it reports that **/home/student/dep-manage/ansible.cfg** is the configuration file in use.

7. Verify the resolution of the **everyone** host group with the **ansible** command. Execute the command in verbose mode to verify which configuration file is being used. When prompted for the sudo password, enter **student**.

```
[student@workstation dep-manage]$ ansible everyone --list-hosts -v
Using /home/student/dep-manage/ansible.cfg as config file
SUDO password: student
  hosts (2):
    localhost
    servera.lab.example.com
```

8. Run **lab manage grade** on **workstation** to grade your work.

```
[student@workstation ~]$ lab manage grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Running Ad Hoc Commands

Objectives

After completing this section, students should be able to:

- Run ad hoc commands locally.
- Run ad hoc commands remotely.
- Discuss example uses for ad hoc commands.

Performing ad hoc commands with Ansible

Ansible allows administrators to execute on-demand tasks on managed hosts. These *ad hoc commands* are the most basic operations that can be performed with Ansible.

Ad hoc commands are performed by running **ansible** on the control node, specifying as part of the command which operation should be performed on the managed hosts. Each ad hoc command is capable of performing a single operation. If multiple operations are required on the managed hosts, then an administrator needs to execute a series of ad hoc commands on the control node.

The ad hoc command is an easy way for administrators to get started with using Ansible so they can get a better idea of its use and application in a real world setting. It also serves as an introduction to more advanced Ansible features such as modules, tasks, plays, and playbooks.

Even after learning advanced Ansible features, administrators will undoubtedly still return to ad hoc commands because they offer many practical uses. For configuration purposes, ad hoc commands can be used to quickly make changes to a large number of managed hosts, such as making changes to configuration files and performing software management tasks. Alternatively, ad hoc commands can be used to perform non-invasive commands such as querying a large group of managed hosts for diagnostic information.

Using ad hoc commands

To execute an ad hoc command, administrators need to execute the **ansible** command using the following syntax.

```
ansible host-pattern -m module [-a 'module arguments'] [-i inventory]
```

As previously demonstrated, the host pattern is used to define the list of managed hosts for Ansible to perform the ad hoc command on. The list of managed hosts is determined by applying the host pattern against the default inventory file which is located at **/etc/ansible/hosts**. The **-i** option is used to specify the location of an alternative inventory file.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Note

Managed hosts are typically separate from the control node, but it is possible for the control node to include itself as a managed host.

This may be desirable because it allows the control node to be managed with Ansible without creating a separate control node only for that purpose. To define the control node as a managed host, add the name of the control node, its IP address, the **localhost** name, or the **127.0.0.1** IP address to the inventory.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Using modules in ad hoc commands

The module specified by the **-m** option defines which Ansible module to use in order to perform the remote operation. Modules will be discussed in a later section. For now, think of a module as a tool which is designed to accomplish a specific task.

Arguments are passed to the specified modules using the **-a** option. Some modules are capable of accepting no arguments; other modules can accept multiple arguments. When no argument is needed, omit the **-a** option from the ad hoc command. If multiple arguments need to be specified, supply them as a quoted space-separated list.

```
ansible host-pattern -m module -a 'argument1 argument2' [-i inventory]
```

The list of arguments available for a module can be found in the module's documentation.

Administrators have the option of defining a default module using the **module_name** setting under the **[defaults]** section of the **/etc/ansible/ansible.cfg** Ansible configuration file.

```
# default module name for /usr/bin/ansible
#module_name = command
```

As previously mentioned, some Ansible configuration settings are predefined internally and have values set even if the settings appear commented out in the Ansible configuration file. The **module_name** is one such parameter. It is predefined with the Ansible **command** module as the default module.

The command module

When the **-m** option is omitted in the execution of an ad hoc command, Ansible will consult the Ansible configuration file and use the module defined there. If no module is defined, Ansible uses the internally predefined **command** module. Therefore, the following ad hoc commands are technical equivalents.

```
ansible host-pattern -m command -a 'module arguments'
```

```
ansible host-pattern -a 'module arguments'
```

The default **command** module allows administrators to execute commands on managed hosts. The command to be executed is specified as an argument to the module using the **-a** option. For example, the following command will execute the **hostname** command on the managed hosts referenced by the **mymanagedhosts** host pattern.

```
[student@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname
host1.lab.example.com | SUCCESS | rc=0 >>
host1.lab.example.com
host2.lab.example.com | SUCCESS | rc=0 >>
host2.lab.example.com
```

The previous ad hoc command example returned two lines of output for each managed host. The first line is a status report which shows the name of the managed host that the ad hoc operation was performed on, as well as the outcome of the operation. The second line is the output of the command executed remotely using the Ansible **command** module.

For better readability and parsing of ad hoc command output, administrators may find it useful to have a single line of output for each operation performed on a managed host. A **-o** option is available to display the output of Ansible ad hoc commands in a single line format.

```
[student@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname -o
host1.lab.example.com | SUCCESS | rc=0 >> (stdout) host1.lab.example.com
host2.lab.example.com | SUCCESS | rc=0 >> (stdout) host2.lab.example.com
```



Note

The **command** module allows administrators to quickly execute remote commands on managed hosts. These commands are not processed by the shell on the managed hosts. As such, they cannot access shell environment variables or perform shell operations such as redirection and piping.

For situations where commands require shell processing, administrators can use the **shell** module. Like the **command** module, simply pass the commands to be executed as arguments to the module in the ad hoc command. Ansible will then execute the command remotely on the managed hosts. Unlike the **command** module, the commands will be processed through a shell on the managed hosts. Therefore, shell environment variables will be accessible and shell operations such as redirection and piping will also be available for use.

The following example illustrates the difference between the **command** and **shell** modules. If an attempt is made to execute the bash builtin, **set**, with these two modules, it will only succeed with the **shell** module.

```
[student@demo ~]$ ansible localhost -m command -a set
localhost | FAILED | rc=2 >>
[Errno 2] No such file or directory
[student@demo ~]$ ansible localhost -m shell -a set
localhost | SUCCESS | rc=0 >>
BASH=/bin/sh
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interact
ive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
...output omitted...
```

Ad hoc command configuration

When an ad hoc command is executed, several things occur behind the scenes. First, the Ansible configuration file is consulted for various parameters. The previously mentioned **module_name**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

parameter is one such example. In addition, other parameters are consulted to determine how the connection to the managed hosts will be performed.

Connection settings

When the connection-related parameters have been read, Ansible proceeds with making connections to the managed host. By default, connections to managed hosts are initiated using the SSH protocol. The SSH protocol requires the connection be established using an account on the managed host. This account is referred to by Ansible as the *remote user* and is defined using the **remote_user** setting under the **[defaults]** section of the Ansible configuration file.

```
# default user to use for playbooks if user is not specified
# (/usr/bin/ansible will use current user as default)
#remote_user = root
```

The **remote_user** parameter is commented out by default in **/etc/ansible/ansible.cfg**. With this parameter undefined, ad hoc commands default to connecting to managed hosts using the same remote user account as that of the user account on the control node which executes the ad hoc command.

When an SSH connection has been made to a managed host, Ansible proceeds with using the specified module to perform the ad hoc operation. Once the ad hoc command is completed on a managed host, Ansible displays on the control node any standard output which was produced by the remotely executed operation.

As with all operations, the remote operation will be executed with the permissions of the user which initiated it. Because the operation is initiated with the remote user, it will be restricted by the limits of that user's permissions.

Privilege escalation

After successfully connecting to a managed host as a remote user, Ansible can switch to another user on the host before executing an operation. This is done using Ansible's privilege escalation feature. For example, using the **sudo** command, an Ansible ad hoc command can be executed on a managed host with **root** privileges even if the SSH connection to the managed host was authenticated by a non-privileged remote user.

Configuration settings to enable privilege escalation are located under the **[privilege_escalation]** section of the **ansible.cfg** configuration file.

```
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

Privilege escalation is not enabled by default. To enable privilege escalation, the **become** parameter must be uncommented and defined as **True**.

```
become=True
```

When privilege escalation is enabled, the **become_method**, **become_user**, and **become_ask_pass** parameters come into play. This is true even if these parameters are commented out in **/etc/ansible/ansible.cfg** since they are predefined internally within Ansible. Their predefined values are as follows:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
become_method=sudo
become_user=root
become_ask_pass=False
```

Settings for managed host connections and privilege escalation can be configured in the Ansible configuration file, and they can also be defined using options in ad hoc commands. When defined using options in ad hoc commands, the settings take precedence over those configured in Ansible's configuration file. The following table shows the analogous command line options for each configuration file setting.

Ansible command line options

Setting	Command line option
inventory	**-i**
remote_user	**-u**
become	**--become**, **-b**
become_method	**--become-method**
become_user	**--become-user**
become_ask_pass	**--ask-become-pass**, **-K**

Before configuring these settings using command line options, their currently defined values can be determined by consulting the output of **ansible --help**.

```
[student@controlnode ~]$ ansible --help
...output omitted...
  -b, --become          run operations with become (nopasswd implied)
  --become-method=BECOME_METHOD
                        privilege escalation method to use (default=sudo),
                        valid choices: [ sudo | su | pbrun | pfexec | runas |
                        doas ]
  --become-user=BECOME_USER
...output omitted...
  -u REMOTE_USER, --user=REMOTE_USER
                        connect as this user (default=None)
```



References

ansible(1) man page

Patterns — Ansible Documentation
http://docs.ansible.com/ansible/intro_patterns.html

Introduction to Ad-Hoc Commands — Ansible Documentation
http://docs.ansible.com/ansible/intro_adhoc.html

command - Executes a command on a remote node — Ansible Documentation
http://docs.ansible.com/ansible/command_module.html

shell - Execute commands in nodes — Ansible Documentation
http://docs.ansible.com/ansible/shell_module.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Running Ad Hoc Commands

In this exercise, you will execute ad hoc commands on multiple managed hosts.

Outcomes
You should be able to execute commands on managed hosts on an ad hoc basis using privilege escalation.

We will be executing ad hoc commands on **workstation** and **servera** using the **devops** user account. This account has the same sudo configuration on both **workstation** and **servera**.

Before you begin
Log in as the **student** user on **workstation** and run **lab adhoc setup**. This setup script ensures that the managed host, **servera**, is reachable on the network. It also creates and populates the **/home/student/dep-adhoc** working directory with materials used in this exercise.

```
[student@workstation ~]$ lab adhoc setup
```

Steps

1. Determine the sudo configuration for the **devops** account on both **workstation** and **servera**.

   1.1. Determine the sudo configuration for the **devops** account that was configured when **workstation** was built. Enter **student** if prompted for the password for the **student** account.

   ```
   [student@workstation ~]$ sudo cat /etc/sudoers.d/devops
   [sudo] password for student: student
   devops ALL=(ALL) NOPASSWD: ALL
   ```

   Note that the user has full sudo privileges but does not require password authentication.

   1.2. Determine the sudo configuration for the **devops** account that was configured when **servera** was built.

   ```
   [student@workstation ~]$ ssh devops@servera.lab.example.com
   [devops@servera ~]$ sudo cat /etc/sudoers.d/devops
   devops ALL=(ALL) NOPASSWD: ALL
   [devops@servera ~]$ exit
   ```

   Note that the user has full sudo privileges but does not require password authentication.

2. Change directory to **/home/student/dep-adhoc** and examine the contents of the **ansible.cfg** and **inventory** files.

   ```
   [student@workstation ~]$ cd /home/student/dep-adhoc
   [student@workstation dep-adhoc]$ cat ansible.cfg
   [defaults]
   inventory=inventory
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dep-adhoc]$ cat inventory
[myself]
localhost

[intranetweb]
servera.lab.example.com

[everyone:children]
myself
intranetweb
```

The configuration file in the directory configures the **inventory** file as the Ansible inventory.

3. Using the **command** module, execute an ad hoc command on **workstation** to identify the user account used by Ansible to perform operations on managed hosts. Use the **localhost** host pattern to connect to **workstation** for the ad hoc command execution. Because we are connecting locally, **workstation** is both the control node and managed host.

```
[student@workstation dep-adhoc]$ ansible localhost -m command -a 'id'
localhost | SUCCESS | rc=0 >>
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Note that the ad hoc command was performed on the managed host as the **student** user.

4. Execute the previous ad hoc command on **workstation** but connect and perform the operation with the **devops** user account by using the **-u** option.

```
[student@workstation dep-adhoc]$ ansible localhost -m command -a 'id' -u devops
localhost | SUCCESS | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Note that the ad hoc command was performed on the managed host as the **devops** user.

5. Using the **command** module, execute an ad hoc command on **workstation** to display the contents of the **/etc/motd** file. Execute the command using the **devops** account.

```
[student@workstation dep-adhoc]$ ansible localhost -m command -a 'cat /etc/motd' -u
 devops
localhost | SUCCESS | rc=0 >>
```

Note that the **/etc/motd** file is currently empty.

6. Using the **copy** module, execute an ad hoc command on **workstation** to change the contents of the **/etc/motd** file so that it displays the message '**Managed by Ansible**'. Execute the command using the **devops** account.

```
[student@workstation dep-adhoc]$ ansible localhost -m copy -a 'content="Managed by
 Ansible\n" dest=/etc/motd' -u devops
localhost | FAILED! => {
    "changed": false,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "failed": true,
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    "msg": "Destination /etc not writable"
}
```

The ad hoc command failed due to insufficient permission.

7. Using the **copy** module, execute the previous command again on **workstation** to change the contents of the **/etc/motd** file so that it displays the message '**Managed by Ansible**'. Still use the **devops** user to make the connection to the managed host, but perform the operation as the **root** user using the **--become** and **--become-user** so the required permissions are satisfied.

```
[student@workstation dep-adhoc]$ ansible localhost -m copy -a 'content="Managed by
 Ansible\n" dest=/etc/motd' -u devops --become --become-user root
localhost | SUCCESS => {
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1463518320.68-167292050637471/
source",
    "state": "file",
    "uid": 0
}
```

Note that the command succeeded this time because the ad hoc command was executed with privilege escalation.

8. Using the **copy** module, execute the previous ad hoc command on **servera** to change the contents of the **/etc/motd** file so that it displays the message '**Managed by Ansible**'. Use the **devops** user to make the connection to the managed host, but perform the operation as the **root** user using the **--become** and **--become-user** so the required permissions are satisfied.

```
[student@workstation dep-adhoc]$ ansible servera.lab.example.com -m copy -a
 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become --become-user
 root
servera.lab.example.com | SUCCESS => {
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1464120223.42-93111008249456/
source",
    "state": "file",
    "uid": 0
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

9. Using the **command** module, execute an ad hoc command to verify that the contents of the **/etc/motd** file has been successfully modified on both **workstation** and **servera**. Use the **everyone** host group and the **devops** user to make the connection and perform the operation on the managed hosts.

```
[student@workstation dep-adhoc]$ ansible everyone -m command -a 'cat /etc/motd' -u
 devops
servera.lab.example.com | SUCCESS | rc=0 >>
Managed by Ansible

localhost | SUCCESS | rc=0 >>
Managed by Ansible
```

10. Run **lab adhoc grade** on **workstation** to grade your work.

```
[student@workstation ~]$ lab adhoc grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Dynamic Inventory

Objectives

After completing this section, students should be able to:

- Use an Ansible dynamic inventory to programmatically build an inventory from external data sources.

Using dynamic inventories

By default, Ansible provides a text-based inventory format to define the hosts to be managed. When operating large infrastructures, system information is often available through an external directory service maintained by a monitoring system, or installation servers (such as Zabbix or Cobbler). Ansible supports dynamically building an inventory from these external data sources through the use of scripts which retrieve information from them.

In a similar way, cloud computing and virtualized infrastructures also have information about the instances and virtual machines they are managing, which may be created and deleted in a short time period. Ansible dynamic inventories can also be used to build an inventory of hosts in real time from common cloud and virtualization solutions such as Red Hat OpenStack Platform and Amazon Web Services EC2.

Static inventories in Ansible can be specified either directly in the **`/etc/ansible/hosts`** file, or through the **`-i`** parameter. This is also true for the scripts providing information about dynamic inventories. If the inventory file is executable, it is treated as a dynamic inventory program and Ansible attempts to run it to generate the inventory. If the file is not executable, it is treated as a static inventory.



Note

The inventory location can be configured in the **`ansible.cfg`** configuration file with the *inventory* parameter. By default it is configured to be **`/etc/ansible/hosts`**.

Supported Platforms

A number of existing scripts are available from Ansible's GitHub site at **`https://github.com/ansible/ansible/tree/devel/contrib/inventory`**. These scripts support the dynamic generation of an inventory based on host information available from a large number of platforms, including:

- Private cloud platforms, such as Red Hat OpenStack Platform.
- Public cloud platforms, such as Rackspace Cloud, AWS, or Google Compute Engine.
- Virtualization platforms, such as Ovirt (upstream of Red Hat Enterprise Virtualization).
- Platform-as-a-Service solutions, such as OpenShift.
- Life cycle management tools, such as Spacewalk (upstream of Red Hat Satellite).
- Hosting providers, such as Digital Ocean or Linode.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The script provided for each platform has a different set of requirements (for example, cloud credentials for Red Hat OpenStack Platform), so refer to this section's references to get more information on the requirements for those.

Writing dynamic inventory programs

If a dynamic inventory script does not exist for the directory system or infrastructure in use, it is possible to write a custom dynamic inventory program. It can be written in any programming language, and must return in JSON format when passed appropriate options.

In order for Ansible to use a script to retrieve hosts information from an external inventory system, this script has to support the **`--list`** parameter, returning host group and hosts information similar to the following JSON hash/dictionary. In this example, **`webservers`** is shown as a host group, with **`web1.lab.example.com`** and **`web2.lab.example.com`** as hosts in that group. The **`databases`** group includes the **`db1`** and **`db2`** hosts.

```
[student@workstation ~]$ ./inventoryscript --list
{
  "webservers"  : [ "web1.lab.example.com", "web2.lab.example.com" ],
  "databases"  : [ "db1.lab.example.com", "db2.lab.example.com" ]
}
```

At a minimum, each group should provide a list of host names or IP addresses for its associated hosts. The script also needs to support the **`--host`** ***`hostname`***, returning an empty JSON hash/dictionary, or a JSON hash/dictionary with the variables associated to that host, such as the following ones:

```
[student@workstation ~]$ ./inventoryscript --host demoserver
{
    "ntpserver" : "ntp.lab.example.com",
    "dnsserver" : "dns.lab.example.com"
}
```



Note

A script creating a dynamic inventory has to be executable in order for Ansible to use it.

Working with multiple inventories

Ansible supports the use of multiple inventories in the same run. If either the value passed to the **`-i`** parameter or the value of the *inventory* parameter in the **`/etc/ansible/ansible.cfg`** configuration, is a directory, the files included in the directory will be used to retrieve information from multiple inventories, either static or dynamic. The executable files within that directory are used to retrieve dynamic inventories, and the other files are used as static inventories.

When multiple inventory files exist, they are examined in alphabetical order. Therefore, it is important that a file's name follows that of another file in alphabetical order if its contents are dependent on contents of the other file. The following example shows how the **`inventorya`** file has a datacenter group which includes the **`webservers`** group defined in the **`inventoryb`** file. This will cause an error since the **`inventorya`** file will be read prior to the **`inventoryb`** file and the **`webserver`** group will not yet be defined when the **`inventorya`** file is being processed.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation test]$ ls -la inventory/
total 8
-rw-rw-r--. 1 student student 33 May 18 15:57 inventorya
-rw-rw-r--. 1 student student 61 May 18 15:58 inventoryb
[student@workstation test]$ cat inventory/inventorya
[datacenter:children]
webservers
[student@workstation test]$ cat inventory/inventoryb
[webservers]
servera.lab.example.com
serverb.lab.example.com
[student@workstation test]$ ansible datacenter --list-hosts
ERROR! inventory/inventorya:2: Section [datacenter:children] includes undefined group:
 webservers
```

Ansible can be configured to ignore files in an inventory directory if they end with certain suffixes. More information is available in the documentation listed in the following References section.



References

Dynamic Inventory — Ansible Documentation
http://docs.ansible.com/ansible/intro_dynamic_inventory.html

Developing Dynamic Inventory Sources — Ansible Documentation
http://docs.ansible.com/ansible/developing_inventory.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Dynamic Inventory

In this exercise, you will fix a custom script that retrieves a dynamic inventory.

Outcomes
You should be able to:
- Manage a dynamic inventory.

Before you begin
Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab deploy-dynamic setup** script. It checks if Ansible is installed on **workstation** and also creates a working directory for this exercise.

```
[student@workstation ~]$ lab deploy-dynamic setup
```

Steps
1. On workstation, change to the working directory for the exercise, **/home/student/dep-dynamic**.

   ```
   [student@workstation ~]$ cd /home/student/dep-dynamic
   ```

2. Create an **ansible.cfg** Ansible configuration file in the working directory and populate it with the following entries so that the **inventory** directory is configured as the default inventory.

   ```
   [defaults]
   inventory = inventory
   ```

3. Create the **/home/student/dep-dynamic/inventory** directory.

   ```
   [student@workstation dep-dynamic]$ mkdir inventory
   ```

4. Download the **inventorya.py**, **inventoryw.py**, and **hosts** files from **http://materials.example.com/dynamic/** to the **/home/student/dep-dynamic/inventory** directory. Both of the files ending in **.py** are scripts that retrieve dynamic inventories. The dynamic inventory used by the **inventorya.py** script has the *webservers* group, which includes the **servera.lab.example.com** host, and the dynamic inventory used by the **inventoryw.py** script includes the **workstation.lab.example.com** host. The **hosts** file defines the *servers* group, which is a parent group of the *webservers* group.

   ```
   [student@workstation dep-dynamic]$ wget http://materials.example.com/dynamic/
   inventorya.py -O inventory/inventorya.py
   [student@workstation dep-dynamic]$ wget http://materials.example.com/dynamic/
   inventoryw.py -O inventory/inventoryw.py
   [student@workstation dep-dynamic]$ wget http://materials.example.com/dynamic/hosts -
   O inventory/hosts
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5. Use the **ansible** command to list the managed hosts associated with the *webservers* group using the **inventorya.py** script. It raises an error about the permissions on the dynamic inventory script.

```
[student@workstation dep-dynamic]$ ansible -i inventory/inventorya.py webservers --
list-hosts
ERROR! The file inventory/inventorya.py looks like it should be an
 executable inventory script, but is not marked executable. Perhaps you
 want to correct this with `chmod +x inventory/inventory.py`?
```

6. Check the current permissions for the **inventorya.py** script, and change them to 755.

```
[student@workstation dep-dynamic]$ ls -la inventory/inventorya.py
-rw-r--r--. 1 student student 0 Apr 29 14:20 inventory/inventorya.py
[student@workstation dep-dynamic]$ chmod 755 inventory/inventorya.py
```

7. Change the permissions for the **inventoryw.py** script to 755.

```
[student@workstation dep-dynamic]$ chmod 755 inventory/inventoryw.py
```

8. Check the current output for the **inventorya.py** script using the **--list** parameter. The hosts associated with the *webservers* group are displayed.

```
[student@workstation dep-dynamic]$ inventory/inventorya.py --list
{"webservers": {"hosts": ["servera.lab.example.com"], "vars": {} } }
```

9. Check the current output for the **inventoryw.py** script using the **--list** parameter. The **workstation.lab.example.com** host is displayed.

```
[student@workstation dep-dynamic]$ inventory/inventoryw.py --list
{"all": {"hosts": ["workstation.lab.example.com"], "vars": {} } }
```

10. Check the *servers* group definition in the **/home/student/dep-dynamic/inventory/hosts** file. The *webservers* group defined in the dynamic inventory is configured as a child of the *servers* group.

```
[student@workstation dep-dynamic]$ cat inventory/hosts
[servers:children]
webservers
```

11. Run the following command to verify the list of hosts in the **webservers** group. It raises an error about the **webservers** group being undefined.

```
[student@workstation dep-dynamic]$ ansible webservers --list-hosts
ERROR! inventory/hosts:46: Section [servers:children] includes undefined group:
 webservers
```

12. Because the static and dynamic inventories are executed in alphabetical order, the **inventorya.py** script, which defines the **webservers** group, must be executed *before*

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

the **hosts** file, which includes the **webservers** group as a dependency. Rename the **inventorya.py** script to **ainventory.py** to solve the previous error.

```
[student@workstation dep-dynamic]$ mv inventory/inventorya.py inventory/
ainventory.py
```

13. Rerun the following command to verify the list of hosts in the **webservers** group. It should work without any errors.

```
[student@workstation dep-dynamic]$ ansible webservers --list
  hosts (1):
    servera.lab.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Deploying Ansible

In this lab, you will configure an Ansible control node for connections to inventory hosts and use ad hoc commands to perform actions on managed hosts.

Outcomes
You should be able to configure an Ansible environment and conduct operations on managed hosts using ad hoc commands.

You have been asked to configure **workstation** as an Ansible control node so that it can be used to manage a new managed host, **serverb**. Begin by verifying that the prerequisite software is installed on both the control node and the new managed hosts and that the two hosts can communicate successfully over the SSH protocol.

Then create a working directory, **/home/student/dep-lab**. In this directory, create an **ansible.cfg** configuration file and also an **inventory** directory.

The new managed host will be used to host Internet facing websites. Create the static inventory file, **/home/student/dep-lab/inventory/inventory**, by downloading it from **http://materials.example.com/dynamic/inventory**. Also download the dynamic inventory script file from **http://materials.example.com/dynamic/binventory.py** to the **/home/student/dep-lab/inventory** directory. The dynamic inventory script will define a new host group, **internetweb**. Modify the static inventory file, **/home/student/dep-lab/inventory/inventory**, to add the new host group to the **everyone** group.

Manage the Ansible environment on the control node from the **/home/student/dep-lab** directory using the configuration file contained in the directory. In the **[defaults]** section, configure the control node so that it will connect to managed host as the **devops** user. In the **[privilege_escalation]** section, set privilege escalation to be disabled by default, but when enabled it should escalate privileges using **sudo** with the **root** user account and with no password authentication.

When configuration is complete, use Ansible's ad hoc command feature and the **copy** module to set the contents of the **/etc/motd** file on **serverb** to '**This server is managed by Ansible.**'. Then use an ad hoc command to verify that the contents of the **/etc/motd** file on **serverb**.

Before you begin
Log in as the **student** user on **workstation** and run **lab deploy setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network.

```
[student@workstation ~]$ lab deploy setup
```

Steps:
1. Verify the Ansible installation on the control node.

2. Verify that **sshd** is running on the managed host and accepts connections with key authentication from the control node. Exit the SSH session when finished.

3. Configure the control node so that it will connect to managed hosts as the **devops** user. Also, set privilege escalation to be disabled by default because it is good practice to

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

only use the least privilege required for operations. If privilege escalation is performed, the configuration should execute it using **sudo** with the **root** user account and with no password authentication.

4. Create the **/home/student/dep-lab/inventory** directory and download both the static inventory file and the dynamic inventory script located at **http://materials.example.com/dynamic/inventory** and **http://materials.example.com/dynamic/binventory.py**, respectively to this directory. The **binventory.py** script returns the details for the *internetweb* group. Configure the *internetweb* group as a child of the existing *everyone* group that is defined in the static inventory file.

5. Execute an ad hoc command using privilege escalation to verify that **devops** is now the remote user and that privilege escalation is now disabled by default.

6. Execute an ad hoc command using the **copy** module and privilege escalation to modify the contents of the **/etc/motd** file on **serverb**. The contents of the file should display the message '**This server is managed by Ansible.**'. Then use an ad hoc command to verify that the contents of the **/etc/motd** file on **serverb**.

7. Run **lab deploy grade** on **workstation** to grade your work.

```
[student@workstation dep-lab]$ lab deploy grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will configure an Ansible control node for connections to inventory hosts and use ad hoc commands to perform actions on managed hosts.

Outcomes
You should be able to configure an Ansible environment and conduct operations on managed hosts using ad hoc commands.

You have been asked to configure **workstation** as an Ansible control node so that it can be used to manage a new managed host, **serverb**. Begin by verifying that the prerequisite software is installed on both the control node and the new managed hosts and that the two hosts can communicate successfully over the SSH protocol.

Then create a working directory, **/home/student/dep-lab**. In this directory, create an **ansible.cfg** configuration file and also an **inventory** directory.

The new managed host will be used to host Internet facing websites. Create the static inventory file, **/home/student/dep-lab/inventory/inventory**, by downloading it from **http://materials.example.com/dynamic/inventory**. Also download the dynamic inventory script file from **http://materials.example.com/dynamic/binventory.py** to the **/home/student/dep-lab/inventory** directory. The dynamic inventory script will define a new host group, **internetweb**. Modify the static inventory file, **/home/student/dep-lab/inventory/inventory**, to add the new host group to the **everyone** group.

Manage the Ansible environment on the control node from the **/home/student/dep-lab** directory using the configuration file contained in the directory. In the **[defaults]** section, configure the control node so that it will connect to managed host as the **devops** user. In the **[privilege_escalation]** section, set privilege escalation to be disabled by default, but when enabled it should escalate privileges using **sudo** with the **root** user account and with no password authentication.

When configuration is complete, use Ansible's ad hoc command feature and the **copy** module to set the contents of the **/etc/motd** file on **serverb** to '**This server is managed by Ansible.**'. Then use an ad hoc command to verify that the contents of the **/etc/motd** file on **serverb**.

Before you begin
Log in as the **student** user on **workstation** and run **lab deploy setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network.

```
[student@workstation ~]$ lab deploy setup
```

Steps:
1. Verify the Ansible installation on the control node.

   1.1. Verify that the *ansible* package is installed.

   ```
   [student@workstation ~]$ yum list installed ansible
   Installed Packages
   ansible.noarch     2.0.1.0-2.el7     @ansible
   ```

   1.2. Determine the configuration file that is in use by the Ansible installation by executing the **ansible** command using the **--version** option.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ ansible --version
ansible 2.0.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

2. Verify that **sshd** is running on the managed host and accepts connections with key authentication from the control node. Exit the SSH session when finished.

   2.1. Verify that the SSH daemon is running on the managed host and accepting connections with key authentication.

```
[student@workstation ~]$ ssh serverb.lab.example.com 'hostname'
Warning: Permanently added 'serverb.lab.example.com,172.25.250.11' (ECDSA) to
 the list of known hosts.
serverb.lab.example.com
```

   2.2. Verify that the control node can connect to the managed host as the **devops** user with key authentication.

```
[student@workstation ~]$ ssh devops@serverb.lab.example.com
Last login: Thu Mar 31 16:07:06 2016 from workstation.lab.example.com
[devops@serverb ~]$
```

   2.3. Verify that the **devops** user has full sudo privileges and does not require password authentication. Exit the SSH session when the verification has been completed.

```
[devops@serverb ~]$ sudo -l
Matching Defaults entries for devops on this host:
    requiretty, !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\:/
usr/local/sbin\:/usr/local/bin

User devops may run the following commands on this host:
    (ALL) NOPASSWD: ALL
[devops@serverb ~]$ exit
```

3. Configure the control node so that it will connect to managed hosts as the **devops** user. Also, set privilege escalation to be disabled by default because it is good practice to only use the least privilege required for operations. If privilege escalation is performed, the configuration should execute it using **sudo** with the **root** user account and with no password authentication.

   3.1. Create and change to the working directory for the exercise.

```
[student@workstation ~]$ mkdir /home/student/dep-lab
[student@workstation ~]$ cd /home/student/dep-lab
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3.2. Create the **/home/student/dep-lab/ansible.cfg** file and add the following entries to create the **[defaults]** section and use it to set the **remote_user** parameter to **devops** so that connections to managed hosts will use the **devops** account on the managed hosts. The entries added also configures Ansible to use the directory **/home/student/dep-lab/inventory** as the default inventory directory.

```
[defaults]
remote_user=devops
inventory=inventory
```

3.3. In the **/home/student/dep-lab/ansible.cfg** file, create the **[privilege_escalation]** section and add the following entries to disable privilege escalation and set the privilege escalation method to use the **root** account with **sudo** and without password authentication.

```
[privilege_escalation]
become=False
become_method=sudo
become_user=root
become_ask_pass=False
```

4. Create the **/home/student/dep-lab/inventory** directory and download both the static inventory file and the dynamic inventory script located at **http://materials.example.com/dynamic/inventory** and **http://materials.example.com/dynamic/binventory.py**, respectively to this directory. The **binventory.py** script returns the details for the *internetweb* group. Configure the *internetweb* group as a child of the existing *everyone* group that is defined in the static inventory file.

4.1. Create the **/home/student/dep-lab/inventory** directory.

```
[student@workstation dep-lab]$ mkdir inventory
```

4.2. Download the **http://materials.example.com/dynamic/inventory** file to the **/home/student/dep-lab/inventory** directory.

```
[student@workstation dep-lab]$ wget http://materials.example.com/dynamic/
inventory -O inventory/inventory
```

4.3. Download the **http://materials.example.com/dynamic/binventory.py** script to the **/home/student/dep-lab/inventory** directory, and change its permission to 755.

```
[student@workstation dep-lab]$ wget http://materials.example.com/dynamic/
binventory.py -O inventory/binventory.py
[student@workstation dep-lab]$ chmod 755 inventory/binventory.py
```

4.4. Configure the *internetweb* group as a child of the existing *everyone* group by adding the following entry to the **inventory** file.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
...
[everyone:children]
myself
intranetweb
internetweb
... Output omitted ...
```

5. Execute an ad hoc command using privilege escalation to verify that **devops** is now the remote user and that privilege escalation is now disabled by default.

```
[student@workstation dep-lab]$ ansible serverb.lab.example.com -m command -a 'id'
serverb.lab.example.com | SUCCESS | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops) context=unconfined_u:
 unconfined_r:unconfined_t:s0-s0:c0.c1023
```

6. Execute an ad hoc command using the **copy** module and privilege escalation to modify the contents of the **/etc/motd** file on **serverb**. The contents of the file should display the message '**This server is managed by Ansible.**'. Then use an ad hoc command to verify that the contents of the **/etc/motd** file on **serverb**.

```
[student@workstation dep-lab]$ ansible serverb.lab.example.com -m copy -a
 'content="This server is managed by Ansible.\n" dest=/etc/motd' --become
serverb.lab.example.com | SUCCESS => {
    "changed": true,
    "checksum": "1e2105d6df22c8d696bc82d3c613cbcb88f36fa5",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "3e6ebda65f0497d3af806622baf7b625",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 34,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1464121888.97-234193031694771/
source",
    "state": "file",
    "uid": 0
}
[student@workstation dep-lab]$ ansible serverb.lab.example.com -m command -a 'cat /
etc/motd'
serverb.lab.example.com | SUCCESS | rc=0 >>
This server is managed by Ansible.
```

7. Run **lab deploy grade** on **workstation** to grade your work.

```
[student@workstation dep-lab]$ lab deploy grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Any system on which Ansible is installed and which has access to the right configuration files and playbooks to manage remote systems (*managed hosts*) is a *control node*.
- The managed hosts are defined in the *inventory*. Host patterns are used to reference managed hosts defined in an inventory.
- Inventories can be a static file or dynamically generated by a program from an external source such as a directory service or cloud management system.
- The location of the inventory is controlled by the Ansible configuration file in use, but most frequently is kept with the playbook files.
- Ansible looks for its configuration file in a number of places in order of precedence. The first configuration file found is used; all others are ignored.
- The **`ansible`** command is used to perform one-time *ad hoc commands* on managed hosts.
- Ad hoc commands determine the operation to perform through the use of *modules* and their arguments.
- Ad hoc commands requiring additional permissions can make use of Ansible's *privilege escalation* features.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 3

IMPLEMENTING PLAYBOOKS

Overview	
Goal	Write Ansible plays and execute a playbook.
Objectives	• Write YAML files. • Implement Ansible playbooks. • Write and execute a playbook.
Sections	• Writing YAML files (and Guided Exercise) • Implementing Modules (and Guided Exercise) • Implementing Ansible Playbooks (and Guided Exercise)
Lab	• Implementing Playbooks

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Writing YAML Files

Objectives

After completing this section, students should be able to:

- Write YAML files.

Introduction to YAML

Ansible playbooks are written using the *YAML Ain't Markup Language (YAML)* language. Therefore, it is necessary to understand the basics of YAML syntax to compose an Ansible playbook.

YAML was designed primarily for the representation of data structures such as *lists* and *associative arrays* in an easy to write, human-readable format. This design objective is accomplished primarily by abandoning traditional enclosure syntax, such as brackets, braces, or opening and closing tags, commonly used by other languages to denote the structure of a data hierarchy. Instead, in YAML, data hierarchy structures are maintained using outline indentation.

YAML files optionally begin with a three dash start of document marker and are optionally terminated with a three dot end of file marker. Between the beginning and ending document markers, data structures are represented using an outline format which uses space characters for indentation. There is no strict requirement regarding the number of space characters used for indentation other than data elements must be further indented than their parents to indicate nested relationships. Data elements at the same level in the data hierarchy must have the same indentation. Blank lines can be optionally added for readability.



Important

Indentation can only be performed using the space character. Indentation is very critical to the proper interpretation of YAML. Since tabs are treated differently by various editors and tools, YAML forbids the use of tabs for indentation.

Users of the **vim** text editor can modify its action in response to **Tab** key entries. For example, with the addition of the following line to the user's **$HOME/.vimrc**, when **vim** detects that you are editing a YAML file, it will perform a two space indentation when the **Tab** key is pressed, will autoindent subsequent lines, and will expand tabs into spaces.

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

The following is a sample YAML file showing a simple data hierarchy.

```
---
title:            My book

author:
    first_name:   John
    last_name:    Doe

publish_date:     2016-01-01
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
chapters:
    - number:     1
      title:      Chapter 1 Title
      pages:      10

    - number:     2
      title:      Chapter 2 Title
      pages:      7
...
```

Using YAML in Ansible playbooks

Ansible playbooks are written in a list format. The items in the list are key/value pairs. Composing a playbook requires only a basic knowledge of YAML syntax.

Start and end of file markers

YAML files are initiated with a beginning of document marker made up of three dashes. Playbooks can be initiated with this line. However, it is optional and does not affect the execution of a playbook.

YAML files are terminated with an end of document marker made up of three periods. This is also optional in a playbook.

```
---
...output omitted...
...
```

Strings

Strings in YAML do not require enclosure in quotations even if there are spaces contained in the string. If desired, strings can be enclosed in either double-quotes or single-quotes.

```
this is a string
```

```
'this is another string'
```

```
"this is yet another a string"
```

There are two ways to write multi-line strings. One way uses the vertical bar (**|**) character to denote that newline characters within the string are to be preserved.

```
include_newlines: |
          Example Company
          123 Main Street
          Atlanta, GA 30303
```

The other way to write multi-line strings uses the greater-than (**>**) character to indicate that newline characters are to be converted to spaces and that leading white spaces in the lines are to be removed. This method is often used to break long strings at space characters so that they can span multiple lines for better readability.

```
fold_newlines: >
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
          This is
          a very long,
          long, long, long
          sentence.
```

Dictionaries

Key/value data pairs used in YAML are also referred to as *dictionaries*, *hashes*, or *associative arrays*. In key/value pairs, keys are separated from values using a delimiter string composed of a colon and a space.

```
key: value
```

Dictionaries are commonly expressed in indented block format.

```
---
  name: Automation using Ansible
  code: DO407
```

Dictionaries can also be expressed in inline block format where multiple key/value pairs are enclosed between curly braces and separated using a delimiter string composed of a comma and space.

```
---
  {name: Automation using Ansible, code: DO407}
```

Lists

In YAML, lists are like arrays in other programming languages. To represent a list of items, a single dash followed by a space is used to prefix each list item.

```
---
 - red
 - green
 - blue
```

Lists can also be expressed in an inline format where multiple list items are enclosed between brackets and separated using a delimiter string composed of a comma and space.

```
---
fruits:
  [red, green, blue]
```

Comments

Comments can also be used to aid readability. In YAML, comments are initiated by a hash symbol (**#**) and can exist at the end of any line, blank or non-blank. If used on a non-blank line, precede the hash symbol with a space.

```
# This is a YAML comment
```

```
some data # This is also a YAML comment
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Verifying YAML syntax

YAML syntax errors in a playbook will cause its execution to fail. Upon failure, the execution output may or may not assist in pinpointing the exact source of the syntax error.

Therefore, it is advisable that the YAML syntax in a playbook be verified prior to its execution. There are several ways to do this.

Python

One way to verify YAML syntax in a playbook is to read in the playbook YAML file using Python programming language. Administrators who are familiar with Python can use a command similar to the following example which requires the installation of the *PyYAML* package.

```
[student@demo ~] python -c 'import yaml, sys; print yaml.load(sys.stdin)' < myyaml.yml
```

If no syntax error exists, Python prints the contents of the YAML file to stdout in JSON format. The following example demonstrates the use of this method on a YAML file with valid syntax.

```
[student@demo ~] cat myyaml.yml
---
- first item
- second item
- third item
...
[student@demo ~] python -c 'import yaml, sys; print yaml.load(sys.stdin)' < myyaml.yml
['first item', 'second item', 'third item']
```

The following example shows what happens when a file contains incorrect YAML syntax.

```
[student@demo ~] cat myyaml.yml
---
- first item
- second item
-third item
...
[student@demo ~] python -c 'import yaml, sys; print yaml.load(sys.stdin)' < myyaml.yml
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib64/python2.7/site-packages/yaml/__init__.py", line 71, in load
    return loader.get_single_data()
...output omitted...
yaml.scanner.ScannerError: while scanning a simple key
  in "<stdin>", line 4, column 1
could not found expected ':'
  in "<stdin>", line 5, column 1
```

YAML Lint

For administrators who are not familiar with Python, there are many online YAML syntax verification tools available. One example is the *YAML Lint* [http://yamllint.com/] website. Copy and paste the YAML contents in a playbook into the form on the home page and then submit the form. The web page reports the results of the syntax verification, as well as display a formatted version of the originally submitted content.

The YAML Lint website reports the error message "**Valid YAML!**" when the following correct YAML contents are submitted.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
- first item
- second item
- third item
...
```

However, it reports the error message "**(<unknown>): could not find expected ':' while scanning a simple key at line 4 column 1** " when the following incorrect YAML contents are submitted.

```
---
- first item
- second item
-third item
...
```

Ansible command line tools

Ansible offers a native feature for administrators to validate the YAML syntax in a playbook. The **ansible-playbook** command, used to execute playbooks, includes a **--syntax-check** option that checks for syntax errors.

While the Python and YAML Lint methods previously mentioned are capable of validating YAML syntax, syntax verification using **ansible-playbook** is the preferred method of verifying YAML syntax within an Ansible playbook. This method conducts a more rigorous review and ensures that required data elements specific to Ansible playbooks are not missing.

The following example demonstrates how **ansible-playbook** determines that a valid YAML file is not valid as a playbook.

```
[student@demo ~] cat myyaml.yml
---
- first item
- second item
- third item
...

[student@workstation ~]$ ansible-playbook myyaml.yml
ERROR! playbook entries must be either a valid play or an include statement

The error appears to have been in '/home/student/myyaml.yml': line 2, column 3, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

---
- first item
  ^ here
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

YAML Ain't Markup Language (YAML) Version 1.2
http://www.yaml.org/spec/1.2/spec.html

YAML
https://en.wikipedia.org/wiki/YAML

YAML Syntax — Ansible Documentation
http://docs.ansible.com/ansible/YAMLSyntax.html

YAML Reference Card
http://yaml.org/refcard.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Writing YAML Files

In this exercise, you will write a YAML file and verify its syntax.

Outcomes
You should be able to create and verify YAML files containing lists and dictionaries.

Before you begin
Log in to **workstation** as the **student** user.

Steps
1. Create and change to the working directory, **/home/student/imp-yaml**, for this exercise.

```
[student@workstation ~] mkdir /home/student/imp-yaml
```

```
[student@workstation ~] cd /home/student/imp-yaml
```

2. Create and open a new YAML file, **/home/student/imp-yaml/dictionaries.yaml**. Add contents to the file in YAML format to represent a list of two dictionaries, **en2es** and **es2en**. The first dictionary, **en2es**, contains the following key/value pairs.

Key	Value
number	numero
word	palabra

The second dictionary, **es2en**, contains the following key/value pairs.

Key	Value
perro	dog
gato	cat

2.1. Add the following entries to create the list's first dictionary, **en2es**, along with its key/value pairs. Separate the keys and values with a colon and a space. Indent the key/value pair entries with spaces to indicate that they are contained in the **en2es** dictionary.

```
- en2es:
   number: numero
   word: palabra
```

2.2. Add the following entries to create the list's second dictionary, **es2en**, along with its key/value pairs. Separate the keys and values with a colon and a space. Indent the key/value pair entries with spaces to indicate that they are contained in the **en2es** dictionary.

```
- es2en:
   perro: dog
   gato: cat
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3. Create and open a new YAML file, **/home/student/imp-yaml/numbers.yaml**. Add contents to the file in YAML format to represent a list of two dictionaries, **numbers** and **numeros**. The first list, **numbers**, contains the following items:

   - one
   - two
   - three

   The second list, **numeros**, contains the following items:

   - uno
   - dos
   - tres

   3.1. Add the following entries to create the list's first list, **numbers**, along with its items. Indent list items with spaces and then prefix them with the hyphen character to indicate that they are contained in the **numbers** list.

```
- numbers:
  - one
  - two
  - three
```

   3.2. Add the following entries to create the list's second list, **numeros**, along with its items. Indent list items with spaces and then prefix them with the hyphen character to indicate that they are contained in the **numeros** list.

```
- numeros:
  - uno
  - dos
  - tres
```

4. Validate the YAML in the **/home/student/imp-yaml/dictionaries.yaml** by reading the file using Python's **system** and **yaml** modules.

   4.1. Execute the following Python command to read in the **dictionaries.yaml** file from stdin. If there are no errors, the file contents will be written to stdout in a singe line serialized format.

```
[student@workstation imp-yaml] python -c 'import yaml, sys; print
 yaml.load(sys.stdin)' < dictionaries.yaml
[{'en2es': {'word': 'palabra', 'number': 'numero'}}, {'es2en': {'gato': 'cat',
 'perro': 'dog'}}]
```

   4.2. Verify that the exit status of the command is **0** which signifies that the file is syntactically correct.

```
[student@workstation imp-yaml] echo $?
0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4.3. Verify that your single line serialized output matches that shown previously. A match indicates that the file's contents created the desired data structures. A mismatch indicates that the file's contents did not create the desired data structures.

5. Validate the YAML in the **/home/student/imp-yaml/numbers.yaml** by reading the file using Python's **system** and **yaml** modules.

5.1. Execute the following Python command to read in the **numbers.yaml** file from stdin. If there are no errors, the file contents will be written to stdout in a singe line serialized format.

```
[student@workstation imp-yaml] python -c 'import yaml, sys; print
 yaml.load(sys.stdin)' < numbers.yaml
[{'numbers': ['one', 'two', 'three']}, {'numeros': ['uno', 'dos', 'tres']}]
```

5.2. Verify that the exit status of the command is **0** which signifies that the file is syntactically correct.

```
[student@workstation imp-yaml] echo $?
0
```

5.3. Verify that your single line serialized output matches that shown previously. A match indicates that the file's contents created the desired data structures. A mismatch indicates that the file's contents did not create the desired data structures.

6. Run **lab yaml grade** on **workstation** to grade your work.

```
[student@workstation imp-yaml]$ lab yaml grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Implementing Modules

Objectives

After completing this section, students should be able to:

- Explain what Ansible modules are, where they are installed, and how to use an existing module.
- Show examples of common modules.

Introduction to modules

Modules are programs that Ansible uses to perform operations on managed hosts. They are ready-to-use tools designed to perform specific operations. Modules can be executed from the **`ansible`** command line or used in playbooks to execute tasks. When run, modules are copied to the managed host and executed there.

Ansible comes packaged with over 400 modules available for use. These prepackaged modules can be used to perform a wide range of tasks, such as cloud, user, package, and service management.

Core, extras, and custom modules

There are three types of Ansible modules:

1. *Core modules* are included with Ansible and are written and maintained by the Ansible development team. Core modules are the most important modules and are used for common administration tasks.
2. *Extras modules* are currently included with Ansible but may be promoted to core or shipped separately in the future. They are generally not maintained by the Ansible team but by the community. Typically, these modules implement features for managing newer technologies such as OpenStack.
3. *Custom modules* are modules developed by end users and not shipped by Ansible. If a module does not already exist for a task, an administrator can write a new module to implement it.

Core and Extras modules are always available. Ansible looks for custom modules on the control node in directories defined by the **`$ANSIBLE_LIBRARY`** environment variable or if that is not set, by the **`library`** parameter in the current Ansible configuration file. Ansible will also look for modules in the **`./library`** directory relative to the location of the playbook being used.

```
library = /usr/share/my_modules/
```

Module categories

For better organization and management, Ansible modules are grouped into the following functional categories.

- Cloud
- Clustering

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Commands
- Database
- Files
- Inventory
- Messaging
- Monitoring
- Network
- Notification
- Packaging
- Source Control
- System
- Utilities
- Web Infrastructure
- Windows

On the Ansible documentation website, module documentation is indexed by the preceding list of categories to assist administrators in their search for a module that suits their specific task.

On Red Hat Enterprise Linux 7 systems, modules are installed in the **`/usr/lib/python2.7/site-packages/ansible/modules`** directory. Core and extras modules are stored under separate directories. Modules within the two directories are organized into category subdirectories.

Module documentation

The large number of modules packaged with Ansible provides administrators with many tools for common administrative tasks. To familiarize themselves with the modules available, administrators can consult the Ansible documentation website, **`docs.ansible.com`**. The module index on the website allows administrators to search for available modules for functions. For example, modules for user and service management can be found under the **`Systems Modules`** and modules for database administration can be found under **`Database Modules`**.

For each module, the Ansible documentation website provides a summary of its functions and instructions on how each specific function can be invoked with options to the module. The documentation also provides useful examples showing the use of each module and its options.

Module documentation is also available locally on the Ansible control node and is accessible using the **`ansible-doc`** command. To see a list of the modules available on a control node, run the **`ansible-doc -l`** command. This displays a list of module names and a synopsis of their function.

```
[student@workstation modules]$ ansible-doc -l
a10_server                      Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
a10_service_group               Manage A10 Networks devices' service groups
a10_virtual_server              Manage A10 Networks devices' virtual servers
acl                             Sets and retrieves file ACL information.
add_host                        add a host (and alternatively a group) to the ansible-
playbook in-memory inventory
airbrake_deployment             Notify airbrake about app deployments
alternatives                    Manages alternative programs for common commands
apache2_module                  enables/disables a module of the Apache2 webserver
apk                             Manages apk packages
apt                             Manages apt-packages
...output omitted...
```

Detailed documentation on a specific module can be displayed by passing the module name to **ansible-doc**. Like the Ansible documentation website, the command provides a synopsis of the module's function, details of its various options, and examples. The following example shows the documentation displayed for the **yum** module.

```
[student@workstation modules]$ ansible-doc yum
> YUM

  Installs, upgrade, removes, and lists packages and groups with the
  `yum' package manager.

Options (= is mandatory):

- conf_file
        The remote yum configuration file to use for the transaction.
        [Default: None]

- disable_gpg_check
        Whether to disable the GPG checking of signatures of packages
        being installed. Has an effect only if state is `present' or
        `latest'. (Choices: yes, no) [Default: no]

...output omitted...

EXAMPLES:
- name: install the latest version of Apache
  yum: name=httpd state=latest

- name: remove the Apache package
  yum: name=httpd state=absent

...output omitted...
```

The **ansible-doc** command also offers a **-s** (or **--snippet**) option, which produces example output that can serve as a model for how to use a particular module in a playbook. This output can serve as a starter template which can be included in a playbook to implement the module for task execution. Comments are included in the output to remind administrators of the use of each option. The following example shows the snippet output for the **yum** module.

```
[student@workstation modules]$ ansible-doc -s yum
- name: Manages packages with the `yum' package manager
  action: yum
      conf_file             # The remote yum configuration ...
      disable_gpg_check     # Whether to disable the GPG ...
      disablerepo           # `Repoid' of repositories ...
      enablerepo            # `Repoid' of repositories ...
      exclude               # Package name(s) to exclude ...
      list                  # Various (non-idempotent) ...
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      name=                 # Package name, or package ...
      state                 # Whether to install ....
      update_cache          # Force updating the cache ...
```

Invoking modules

There are multiple ways to work with Ansible modules, depending on the context and the administrator's needs.

- Modules can be called as part of an ad hoc command, using the **ansible** command. The **-m** option allows administrators to specify the name of the module to use. The following command uses the **ping** module to test connectivity to all managed hosts. This module connects and authenticates to the managed host and verifies that the Python requirements for Ansible are met.

```
[student@controlnode ~]$ ansible -m ping all
```

- Modules can also be called in playbooks, as part of a **task**. The following snippet shows how the **yum** module can be invoked with the name of a package and its desired state as arguments.

```
tasks:
- name: Installs a package
  yum:
    name: postfix
    state: latest
```

Demonstration: Discovering Ansible modules

This demonstration will use Ansible ad hoc commands to show some common operations using the **yum**, **service**, and **uri** core modules. The demonstration will be conducted out of the **/home/student/imp-moddemo** directory on **workstation** using the supplied configuration file and inventory. The **workstation** host will serve as both the control node and the managed host for this demonstration. It is assumed that SSH public key authentication is configured between the control node and the managed host, and that both the managed host and **materials.example.com** are running Apache HTTP web servers. The control node is already configured to connect to the managed host as the **devops** user, which has full sudo privileges that does not require password authentication. Privilege escalation has been enabled in the control node's configuration and will be performed using sudo as the root user and without password prompting. The *tree* package is not installed on **workstation** at the start of the demonstration.

The **yum** core module will be used to install the *tree* package on the managed host. Then the **service** core module will be used to restart the Apache HTTP web server on workstation. Finally, the **uri** core module will be used to see what web pages on **materials.example.com** are visible to **workstation**.

1. Change directory to the working directory for the demonstration.

```
[student@workstation ~]$ cd /home/student/imp-moddemo
```

2. Determine the status of the *tree* package and the availability of the package for installation on **workstation**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation imp-moddemo]$ yum list installed tree
Loaded plugins: langpacks, search-disabled-repos
Error: No matching Packages to list
```

3. Install the *tree* package by executing an ad hoc command which uses the **yum** module. Be sure to activate privilege escalation because the command has to be executed with **root** privileges.

```
[student@workstation imp-moddemo]$ ansible localhost -m yum -a "name=tree
 state=present"
localhost | SUCCESS => {
"changed": true,
"msg": "",
"rc": 0,
...output omitted...
```

4. Reverify the status of the *tree* package on **workstation**.

```
[student@workstation imp-moddemo]$ yum list installed tree
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
tree.x86_64    1.6.0-10.el7    rhel_dvd
```

The package is now installed.

5. Determine the status of the **httpd** service on **workstation**.

```
[student@workstation imp-moddemo]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
 disabled)
  Drop-In: /etc/systemd/system/httpd.service.d
       └─training.conf
   Active: active (running) since Mon 2016-04-18 14:23:47 EDT; 38min ago
 Docs: man:httpd(8)
       man:apachectl(8)
  Process: 328 ExecStop=/bin/kill -WINCH ${MAINPID} (code=exited, status=0/SUCCESS)
  Process: 12363 ExecReload=/usr/sbin/httpd $OPTIONS -k graceful (code=exited,
 status=0/SUCCESS)
 Main PID: 333 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:   0 B/sec"
   CGroup: /system.slice/httpd.service
       ├─333 /usr/sbin/httpd -DFOREGROUND
       ├─334 /usr/sbin/httpd -DFOREGROUND
       ├─335 /usr/sbin/httpd -DFOREGROUND
       ├─336 /usr/sbin/httpd -DFOREGROUND
       ├─337 /usr/sbin/httpd -DFOREGROUND
       └─338 /usr/sbin/httpd -DFOREGROUND

Apr 18 14:23:47 workstation.lab.example.com systemd[1]: Starting The Apache HTTP
 Server...
Apr 18 14:23:47 workstation.lab.example.com systemd[1]: Started The Apache HTTP
 Server.
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6. Restart the **httpd** service by executing an ad hoc command which uses the **service** module. Be sure to activate privilege escalation because the command has to be executed with **root** privileges.

```
[student@workstation imp-moddemo]$ ansible localhost -m service -a "name=httpd
 state=restarted"
localhost | SUCCESS => {
"changed": true,
"name": "httpd",
"state": "started"
}
```

7. Reverify the status of the **httpd** service on **workstation**.

```
[student@workstation imp-moddemo]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
 disabled)
  Drop-In: /etc/systemd/system/httpd.service.d
       └─training.conf
   Active: active (running) since Mon 2016-04-18 15:03:43 EDT; 3s ago
 Docs: man:httpd(8)
       man:apachectl(8)
  Process: 1559 ExecStop=/bin/kill -WINCH ${MAINPID} (code=exited, status=0/SUCCESS)
  Process: 12363 ExecReload=/usr/sbin/httpd $OPTIONS -k graceful (code=exited,
 status=0/SUCCESS)
 Main PID: 1590 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
       ├─1590 /usr/sbin/httpd -DFOREGROUND
       ├─1591 /usr/sbin/httpd -DFOREGROUND
       ├─1592 /usr/sbin/httpd -DFOREGROUND
       ├─1593 /usr/sbin/httpd -DFOREGROUND
       ├─1594 /usr/sbin/httpd -DFOREGROUND
       └─1595 /usr/sbin/httpd -DFOREGROUND

Apr 18 15:03:43 workstation.lab.example.com systemd[1]: Starting The Apache HTTP
 Server...
Apr 18 15:03:43 workstation.lab.example.com systemd[1]: Started The Apache HTTP
 Server.
```

The system messages indicate that the service was successfully restarted.

8. Make an HTTP request to the **http://materials.example.com** URL from **workstation** and evaluate the HTTP status return code.

```
[student@workstation imp-moddemo]$ ansible localhost -m uri -a "url=http://
materials.example.com/"
localhost | SUCCESS => {
"changed": false,
"content_length": "2457",
"content_location": "http://materials.example.com/",
"content_type": "text/html;charset=ISO-8859-1",
"date": "Mon, 18 Apr 2016 19:10:28 GMT",
"redirected": false,
"server": "Apache/2.4.6 (Red Hat Enterprise Linux)",
"status": 200
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The module executed successfully and reports that an HTTP status return code of **200** was received.

9. Make an HTTP request to the **http://materials.example.com/nothing** URL from **workstation** and evaluate the HTTP status return code.

```
[student@workstation imp-moddemo]$ ansible localhost -m uri -a "url=http://
materials.example.com/nothing"
localhost | FAILED! => {
"changed": false,
"content": "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0//EN\">\n<html><head>
\n<title>404 Not Found<title>\n<head><body>\n<h1>Not Found<h1>\n<p>The requested
 URL /nothing was not found on this server.<p>\n<body><html>\n",
"content_length": "205",
"content_type": "text/html; charset=iso-8859-1",
"date": "Mon, 18 Apr 2016 19:14:08 GMT",
"failed": true,
"msg": "Status code was not [200]",
"redirected": false,
"server": "Apache/2.4.6 (Red Hat Enterprise Linux)",
"status": 404
}
```

The module execution reports a failure and shows that an HTTP status return code of **404** was received.



References

ansible-doc(1) man page

> About modules — Ansible Documentation
> http://docs.ansible.com/ansible/modules.html

> Ansible modules - these modules ship with Ansible — Ansible Documentation
> https://github.com/ansible/ansible-modules-core

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Modules

In this exercise, you will perform administrative tasks on remote systems using Ansible modules.

Outcomes
With Ansible ad hoc commands, use modules to perform remote operations on managed hosts.

A developer has requested that you install the Apache software on **servera**. He also requested that the **httpd** service be enabled and started once the software is installed.

Manage **servera** from the **/home/student/imp-module** directory using the **ansible.cfg** configuration file and **inventory** inventory file provided in the directory. Ansible is already configured to use the **devops** user to connect to managed hosts. The **devops** user has **sudo** access to the **root** account on the managed hosts. Privilege escalation support has not yet been enabled in the Ansible configuration file. This is needed to allow the control node to run modules needing **root** access on the managed hosts.

The **servera** managed host is already defined in the **inventory** inventory file. Perform the remote operations on **servera** using the appropriate modules in Ansible ad hoc commands.

Before you begin
Log in as the **student** user on **workstation** and run **lab module setup**. This setup script ensures that the managed host, **servera**, is reachable on the network and that the Ansible inventory and Ansible configuration files are both correctly configured.

```
[student@workstation ~]$ lab module setup
```

Steps
1. Change directory to the working directory, **/home/student/imp-module**.

   ```
   [student@workstation ~]$ cd /home/student/imp-module
   ```

2. Verify that **servera** is defined in the inventory using the **ansible** command and its **--list-hosts** option to list managed hosts.

   ```
   [student@workstation imp-module]$ ansible servera.lab.example.com --list-hosts
     hosts (1):
       servera.lab.example.com
   ```

3. Look for the **-u** option in the output generated by the **ansible --help** command to verify the remote user configuration is set to the **devops** remote user.

   ```
   [student@workstation imp-module]$ ansible --help | grep -A1 -w -- -u
     -u REMOTE_USER, --user=REMOTE_USER
                           connect as this user (default=devops)
   ```

4. Look in **/home/student/imp-module/ansible.cfg** to determine the **inventory** and **remote_user** settings and to determine if privilege escalation is enabled and how privilege escalation settings are configured.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4.1. Open the **/home/student/imp-module/ansible.cfg** file in read-only mode using the **view** command.

```
[student@workstation imp-module]$ view /home/student/imp-module/ansible.cfg
```

4.2. Review the settings configured for the inventory and the remote user.

```
[defaults]
inventory=inventory
remote_user=devops
...output omitted...
```

4.3. Go to the **[privilege_escalation]** section of the file and review the settings configured in that section.

```
...output omitted...
[privilege_escalation]
become=False
become_method=sudo
become_user=root
become_ask_pass=False
```

Privilege escalation is disabled. If enabled, the privilege escalation would occur using **sudo** with the **root** account and with no password prompting.

5. Using the **ping** module, determine if the **servera** managed host is reachable on the network by executing the following Ansible ad hoc command.

```
[student@workstation imp-module]$ ansible servera.lab.example.com -m ping
servera.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

The module's **SUCCESS** output indicates that the managed host is reachable.

6. Using the **yum** module, install the latest available version of the *httpd* package by executing the following ad hoc command. Pass the package name and the desired state as arguments to the module in the form of a key/value pair list.

```
[student@workstation imp-module]$ ansible servera.lab.example.com -m yum -a
 "name=httpd state=latest"
servera.lab.example.com | FAILED! => {
    "changed": true,
    "failed": true,
    "msg": "You need to be root to perform this command.\n",
    "rc": 1,
    "results": [
        "Loaded plugins: langpacks, search-disabled-repos\n"
    ]
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The module's output indicates that the ad hoc command failed because it was not executed as the **root** user.

7. Reexecute the previous ad hoc command but include the **-b** option to enable privilege escalation.

```
[student@workstation imp-module]$ ansible servera.lab.example.com -m yum -a
 "name=httpd state=latest" -b
servera.lab.example.com | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
...output omitted...
    ]
}
```

The module's output reports that the ad hoc command succeeded.

8. Using the **service** module, enable and start the *httpd* service by executing the following ad hoc command. Pass the service name and the desired state and enabled status as arguments to the module in the form of a key/value pair list. Because the command requires **root** privileges, use the **-b** option to enable privilege escalation.

```
[student@workstation imp-module]$ ansible servera.lab.example.com -m service -a
 "name=httpd enabled=yes state=started" -b
servera.lab.example.com | SUCCESS => {
    "changed": true,
    "enabled": true,
    "name": "httpd",
    "state": "started"
}
```

The module's output reports that the ad hoc command succeeded.

9. Run **lab module grade** on **workstation** to grade your work.

```
[student@workstation imp-module]$ lab module grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Implementing Ansible Playbooks

Objectives

After completing this section, students should be able to:

- Write and execute a playbook.

Playbooks and ad hoc commands

Ansible modules can be used to perform operations on managed hosts using simple ad hoc commands. While useful for simple operations, ad hoc commands are not suited for the challenges of complex configuration management or orchestration scenarios.

Ad hoc commands can only invoke one module and one set of arguments at a time. When multiple operations are required, they must be executed using multiple ad hoc commands.

Playbooks are files which describe the desired configurations or procedural steps to implement on managed hosts. Playbooks offer a powerful and flexible solution for configuration management and deployment. Playbooks can change lengthy, complex administrative tasks into easily repeatable routines with predictable and successful outcomes.

Playbook basics

Playbooks are text files written in YAML format. Compared to the languages used in other configuration management tools such as Puppet, the syntax used in playbooks is much easier to write and comprehend.

Playbooks are named after the sports analogy that a playbook contains a collection of plays, with each play being a standardized plan that can be executed. Ansible playbooks also contain *plays*, where each play serves to define a set of operations to perform on a specific set of managed hosts. These operations are called *tasks*, and the managed hosts are referred to as *hosts*. Tasks are performed by invoking Ansible modules and passing them the necessary arguments to accomplish the desired operation.



Important

The ordering of the contents within a playbook is important, because Ansible executes plays and tasks in the order they are presented.

Ansible playbooks should be written so that tasks do not make unnecessary changes if the managed hosts are already in the correct state. In other words, if a playbook is run once to put the hosts in the correct state, the playbook should be written so it is safe to run it a second time and it should make no further changes to the configured systems. A playbook with this property is *idempotent*. Most Ansible modules are idempotent, so it is relatively easy to ensure this is true.

Each playbook can contain one or more plays. Because a play applies a set of tasks to a set of hosts, multiple plays are required when a situation calls for a different set of tasks to be performed on one set of hosts and a different set of tasks to be performed on another set of hosts.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Note

Although multiple plays can be defined in a playbook, only a single playbook can be defined within a given YAML file. If multiple playbooks are desired, they must each be created as separate YAML files.

Writing a playbook

Playbook files may begin with the YAML document marker (**---**). This marker is optional and has no impact on the execution of the playbook.

Conversely, the YAML document terminator (**...**) may be used to terminate a playbook file. This entry is also optional, and if it is not present the end of the file will automatically be the end of the playbook.

Within a playbook file, plays are expressed in a list context where each play is defined using a YAML dictionary data structure. Therefore, the beginning of each play begins with a single dash followed by a space.

```
-
```

Within a play, administrators can define various attributes. An attribute definition consists of the attribute name followed by a colon and a space (:) and then the attribute value.

```
attribute: value
```

Following YAML syntax, each attribute in a play needs to be indented with spaces to be at the same indentation level to indicate their place in the data structure hierarchy.

The following example shows how two attributes are indented to indicate their relationship to the parent play. The dash in the first entry marks the beginning of a play using the list item syntax. The second entry is space-indented to the same indentation level as the first entry to convey that the two attributes refer to the same parent play.

```
---
- attribute1: value1
  attribute2: value2
```

Name attribute

The **name** attribute can be used to give a descriptive label to a play. This is an optional attribute, but it is a recommended practice to name all tasks with a label. This is especially useful when a playbook contains multiple plays.

```
name: my first play
```

Hosts attribute

The set of managed hosts to perform tasks on is defined using the **hosts** attribute. Because this component is integral to the concept of a play, the host attribute must be defined in every

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

play. The host attribute is defined using host patterns to reference hosts from the inventory as previously discussed.

```
hosts: managedhost.example.com
```

User attributes
Tasks in playbooks are normally executed through a network connection to the managed hosts. As with ad hoc commands, the user account used for these task execution is dependent on various parameters in Ansible's configuration file, **/etc/ansible/ansible.cfg**. The user that runs the tasks can be defined by the **remote_user** parameter. However, if privilege escalation is enabled, other parameters such **become_user** can also have an impact.

If the remote user defined in the Ansible configuration for task execution is not suitable, it can be overwritten using the **remote_user** attribute within a play.

```
remote_user: remoteuser
```

Privilege escalation attributes
Additional attributes are also available to define privilege escalation parameters from within a playbook. The **become** Boolean parameter can be used to enable or disable privilege escalation regardless of how it is defined in the Ansible configuration file.

```
become: True/False
```

If privilege escalation is enabled, the **become_method** attribute can be used to define the privilege escalation method to use during a specific play. The example below dictates that **sudo** be used for privilege escalation.

```
become_method: sudo
```

Additionally, with privilege escalation enabled, the **become_user** attribute can define the user account to be used for privilege escalation within the context of a specific play.

```
become_user: privileged_user
```

Tasks attribute
The main component of a play, the operations to be executed on the managed hosts, are defined using the **tasks** attribute. This attribute is defined as a list of dictionaries. Each task in the tasks list is composed of a set of key/value pairs.

In the following example, the **tasks** list consists of a single task item. The first entry of the task item defines a name for the task. The second entry invokes the **service** module and supplies its arguments as the values.

```
tasks:
- name: first task
  service: name=httpd enabled=true
```

In the preceding example, the dash in the first entry marks the beginning of the list of attributes which pertain to the task. The second entry is space-indented to the same indentation level as the first entry to convey that the two attributes refer to the same parent task.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

If multiple tasks are desired, the same syntax is repeated for each task.

```
tasks:
- name: first task
  service: name=httpd enabled=true
- name: second task
  service: name=sshd enabled=true
...output omitted...
- name: last task
  service: name=sshd enabled=true
```



Important

Use **ansible-doc** to find and learn how to use modules for your tasks.

When possible, try to avoid the **command**, **shell**, and **raw** modules in playbooks, as simple as they may seem to use. Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.

For example, this task using the **shell** module is not idempotent. Every time the play is run, it will rewrite **/etc/resolv.conf** even if it already consists of the line "nameserver 192.0.2.1".

```
- name: Non-idempotent approach with shell module
  shell: echo "nameserver 192.0.2.1" > /etc/resolv.conf
```

A number of things could be done to use the **shell** module in an idempotent way, and sometimes making those changes and using **shell** is the best approach. But a quicker solution may be to use **ansible-doc** to discover the **copy** module and use that to get the desired effect.

The following example will not rewrite the file **/etc/resolv.conf** if it already consists of the right content:

```
- name: Idempotent approach with copy module
  copy:
    dest: /etc/resolv.conf
    content: "nameserver 192.0.2.1\n"
```

The **copy** module is special-purpose and can easily test to see if the state has already been met, and if it has will make no changes. The **shell** module allows a lot of flexibility, but also requires more attention to ensure that it runs in an idempotent way.

Idempotent playbooks can be run repeatedly to ensure systems are in a particular state without disrupting those systems if they already are.

Comments

As is good practice with all programming languages and configuration files, comments can be added to Ansible playbooks. Comments in playbooks are preceded by the pound sign, **#**. Comments should be used throughout a playbook to enhance readability and documentation.

```
# This is a comment
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

A simple playbook

The example below shows a simple playbook constructed with the components and syntax previously discussed.

```
---
# This is a simple playbook with a single play
-  name: a simple play
   hosts: managedhost.example.com
   user: remoteuser
   become: yes
   become_method: sudo
   become_user: root
   tasks:
   - name: first task
     service: name=httpd enabled=true
   - name: second task
     service: name=sshd enabled=true
...
```

Playbook formatting

There are several additional options for formatting playbooks. These options can be used to improve readability and organization of playbook contents.

The tasks dictionary in a playbook contains a key/value pair entry where the key is the name of the module being invoked and the value is the list of arguments which need to be passed to the module. Depending on the scenario, the list of arguments may be extremely long. There are two formatting options which can be applied under these circumstances.

Multi-line formatting

Long module arguments can be broken up using a multi-line format. When using this format, the line needs to be broken on a space and the continuation lines need to be properly indented with spaces. Continuation lines must be indented with enough spaces to exceed the indentation level of the first line. The following lines are examples of equivalent tasks, with the first example formatted using a single line format and the second formatted with a multi-line format.

```
tasks:
- name: first task
  service: name=httpd enabled=true state=started
```

```
tasks:
- name: first task
  service: name=httpd
           enabled=true
           state=started
```

Dictionary formatting

Another way of dealing with long module arguments is to format them using YAML's dictionary structure. When using this format, the option/value pairs passed as arguments are converted to dictionary entries. These entries need to be space indented past the indentation level of the module entry to indicate their child relation to it. The following are examples of equivalent tasks, with the first example formatted using a single line format and the second with its arguments structured as a dictionary.

```
tasks:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
- name: first task
  service: name=httpd enabled=true state=started
```

```
tasks:
- name: first task
  service:
    name: httpd
    enabled: true
    state: started
```

Blocks

Complex playbooks may contain a long list of tasks. Some tasks in the list may be related in their function.

With Ansible version 2.0, *blocks* offer another alternative to task organization. Blocks can be used to group related tasks together. This not only improves readability but also allows task parameters to be performed on a block level when writing more complex playbooks.

The following examples show how a list of tasks can be organized into distinct groups with the use of blocks.

```
tasks:
- name: first task
  yum:
    name: httpd
    state: latest
- name: second task
  yum:
    name: openssh-server
    state: latest
- name: third task
  service:
    name:httpd
    enabled:true
    state: started
- name: fourth task
  service:
    name: sshd
    enabled:true
    state:started
```

```
tasks:
- block:
  - name: first package task
    yum:
      name: httpd
      state: latest
  - name: second package task
    yum:
      name: openssh-server
      state: latest
- block:
  - name: first service task
    service:
      name: httpd
      enabled: true
      state: started
  - name: second service task
    service:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      name: sshd
      enabled: true
      state: started
```

Multiple plays

As previously mentioned, a playbook can contain one or more plays. Because plays map managed hosts to tasks, scenarios that require different tasks to be performed on different hosts, such as orchestration, necessitate the use of different plays. Rather than having plays in separate playbook files, multiple plays can be placed in the same playbook file. Plays are expressed in a list context, so the start of each play is indicated by a preceding dash and space.

The following example shows the format for creating a simple playbook with multiple plays.

```
---
# This is a simple playbook with two plays

-  name: first play
   hosts: web.example.com
   tasks:
   - name: first task
     service:
       name: httpd
       enabled: true

-  name: second play
   hosts: database.example.com
   tasks:
   - name: first task
     service:
       name: mariadb
       enabled: true
...
```

Executing playbooks

Playbooks are executed using the **`ansible-playbook`** command. The command is executed on the control node and the name of the playbook to be executed is passed as an argument.

When the playbook is executed, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.

The following example shows the execution of a simple playbook.

```
[student@controlnode imp-playdemo]$ cat webserver.yml
---
- name: play to setup web server
  hosts: servera.lab.example.com
  tasks:
  - name: latest httpd version installed
    yum:
      name: httpd
      state: latest
...

[student@controlnode imp-playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ************************************************

TASK [setup] *******************************************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ***************************************
ok: [servera.lab.example.com]

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

The preceding output demonstrates why it is useful to use the name attributes for plays and tasks in a playbook event even though its use is optional. The value set for the name attribute is displayed in the output of the playbook execution. For playbooks with multiple plays and tasks, using the name attributes greatly facilitates monitoring the progress of a playbook's execution.

Syntax verification

Prior to executing a playbook, it is good practice to perform a verification to ensure that the syntax of its contents is correct. The **ansible-playbook** command offers a **--syntax-check** option which can be used to verify the syntax of a playbook file. The following example shows the successful syntax verification of a playbook.

```
[student@controlnode ~]$ ansible-playbook --syntax-check webserver.yml

playbook: webserver.yml
```

When syntax verification fails, a syntax error is reported. The output also includes the approximate location of the syntax issue in the playbook. The following example shows the failed syntax verification of a playbook where the space separator is missing after the **name** attribute for the play.

```
[student@controlnode ~]$ ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.


The error appears to have been in '/home/student/webserver.yml': line 3, column 8, but
 may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

- name:play to setup web server
  hosts: servera.lab.example.com
       ^ here
```

Executing a dry run

Another helpful option is the **-C** option. This causes Ansible to report what changes would have occurred if the playbook were executed, but does not make any actual changes to managed hosts.

The following example shows the dry run of a playbook containing a single task for ensuring that the latest version of *httpd* package is installed on a managed host. Note that the dry run reports that the task would effect a change on the managed host.

```
[student@controlnode ~]$ ansible-playbook -C webserver.yml

PLAY [play to setup web server] *********************************************

TASK [setup] ****************************************************************
ok: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [latest httpd version installed] ***************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

Step-by-step execution
When developing new playbooks, it may be helpful to execute the playbook interactively. The **ansible-playbook** command offers the **--step** option for this purpose.

When executed with this option, Ansible steps through each task in the playbook. Prior to executing each task, it prompts the user for input. User can choose '**y**' to execute the task, '**n**' to skip the task, or '**c**' to exit step-by-step execution and execute the remaining tasks non-interactively.

The following example shows the step-by-step run of a playbook containing a task for ensuring that the latest version of the *httpd* package is installed on a managed host.

```
[student@controlnode ~]$ ansible-playbook --step webserver.yml

PLAY [play to setup web server] *************************************************
Perform task: TASK: setup (y/n/c): y

Perform task: TASK: setup (y/n/c): *****************************************

TASK [setup] ********************************************************************
ok: [servera.lab.example.com]
Perform task: TASK: latest httpd version installed (y/n/c): y

Perform task: TASK: latest httpd version installed (y/n/c): ********************

TASK [latest httpd version installed] ***************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

Demonstration: Writing and executing playbooks

This demonstration illustrates the creation and use of Ansible playbooks. The demonstration will be conducted out of the **/home/student/imp-playdemo** directory on **workstation** using the supplied configuration file, inventory, and playbooks. It is assumed that **workstation** is the control node, and that **servera.lab.example.com** is a managed host. The control node also manages itself as a managed host. The managed hosts have a **devops** user that is able to get **root** access through sudo without a password. SSH authentication keys are set up to give student login access to the devops user.

The supplied configuration file specifies the remote user and the location of the inventory. It also configures Ansible logging to be enabled on the control node and managed hosts.

1. Log in as the **student** user on **workstation**. Change directory to the working directory and review the contents of the directory.

```
[student@workstation ~]$ cd /home/student/imp-playdemo
[student@workstation imp-playdemo]$ ls -la
total 12
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
drwxrwxr-x. 3 student student   50 May 18 21:03 .
drwx------. 7 student student 4096 May 18 21:03 ..
-rw-rw-r--. 1 student student  138 May 18 21:03 ansible.cfg
-rw-rw-r--. 1 student student   34 May 18 21:03 inventory
drwxrwxr-x. 2 student student    6 May 18 21:03 log
[student@workstation imp-playdemo]$ cat ansible.cfg
[defaults]
remote_user=devops
inventory=inventory
log_path=/home/student/imp-playdemo/log/ansible.log
no_log=False
no_target_syslog=False
[student@workstation imp-playdemo]$ cat inventory
localhost
servera.lab.example.com
```

2. Review the **/home/student/imp-playdemo/ftpclient.yml** playbook. It ensures that the latest version of the *lftp* package is installed on the local managed host. This operation will require escalated privileges, so the playbook enables privilege escalation using **sudo** and the **root** user. The connection to the managed host is made with the **devops** user.

```
---
- name: ftp client installed
  hosts: localhost
  remote_user: devops
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: latest lftp version installed
    yum:
      name: lftp
      state: latest
...
```

3. Review the**/home/student/imp-playdemo/ftpserver.yml** playbook. It ensures that the latest version of the *vsftpd* and *firewalld* packages are installed on **servera**. This operation will require escalated privileges, so the playbook enables privilege escalation using **sudo** and the **root** user. The connection to the managed host is made with the **devops** user.

   The playbook ensures that **firewalld** allows incoming connections for the FTP service. It also ensures that the **vsftpd** and **firewalld** services are enabled and running.

   Related tasks are grouped together using blocks.

```
---
- name: ftp server installed
  hosts: servera.lab.example.com
  remote_user: devops
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - block:
    - name: latest vsftpd version installed
      yum:
        name: vsftpd
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        state: latest
    - name: latest firewalld version installed
      yum:
        name: firewalld
        state: latest
  - block:
    - name: firewalld permits ftp service
      firewalld:
        service: ftp
        permanent: true
        state: enabled
        immediate: yes
  - block:
    - name: vsftpd enabled and running
      service:
        name: vsftpd
        enabled: true
        state: started
    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started
...
```

4. Execute the **/home/student/imp-playdemo/ftpclient.yml** playbook.

```
[student@workstation imp-playdemo]$ ansible-playbook ftpclient.yml

PLAY [ftp client installed] ****************************************************

TASK [setup] *******************************************************************
ok: [localhost]

TASK [latest lftp version installed] *******************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```

5. Execute the **/home/student/imp-playdemo/ftpserver.yml** playbook.

```
[student@workstation imp-playdemo]$ ansible-playbook ftpserver.yml

PLAY [ftp server installed] ****************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [latest vsftpd version installed] *****************************************
changed: [servera.lab.example.com]

TASK [latest firewalld version installed] **************************************
ok: [servera.lab.example.com]

TASK [firewalld permits ftp service] *******************************************
changed: [servera.lab.example.com]

TASK [vsftpd enabled and running] **********************************************
changed: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [firewalld enabled and running] ***************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=6    changed=3    unreachable=0    failed=0
```

6. Review the log messages generated on the control node.

```
[student@workstation imp-playdemo]$ cat /home/student/imp-playdemo/log/ansible.log
2016-04-22 18:40:29,327 p=4940 u=student |  PLAY [ftp client installed]
 ****************************************************
2016-04-22 18:40:29,348 p=4940 u=student |  TASK [setup]
 *******************************************************************
2016-04-22 18:40:29,841 p=4940 u=student |  ok: [localhost]
2016-04-22 18:40:29,843 p=4940 u=student |  TASK [latest lftp version installed]
 ****************************************
...output omitted
2016-04-22 18:41:03,140 p=5131 u=student |  TASK [firewalld enabled and running]
 ****************************************
2016-04-22 18:41:04,190 p=5131 u=student |  ok: [servera.lab.example.com]
2016-04-22 18:41:04,191 p=5131 u=student |  PLAY RECAP
 *********************************************************************
2016-04-22 18:41:04,191 p=5131 u=student |  servera.lab.example.com    : ok=6
 changed=3    unreachable=0    failed=0
```

7. Login to **servera** as the **root** user and review the messages generated in **/var/log/messages** by the playbook execution.

```
[root@servera ~]# grep 'python: ansible' /var/log/messages
May 18 22:47:58 servera python: ansible-setup Invoked with filter=* fact_path=/etc/
ansible/facts.d
May 18 22:47:58 servera python: ansible-yum Invoked with name=['vsftpd'] list=None
 install_repoquery=True conf_file=None disable_gpg_check=False state=latest
 disablerepo=None update_cache=False enablerepo=None exclude=None
May 18 22:48:01 servera python: ansible-yum Invoked with name=['firewalld']
 list=None install_repoquery=True conf_file=None disable_gpg_check=False
 state=latest disablerepo=None update_cache=False enablerepo=None exclude=None
May 18 22:48:02 servera python: ansible-firewalld Invoked with service=ftp zone=None
 immediate=True source=None state=enabled permanent=True timeout=0 port=None
 rich_rule=None
May 18 22:48:02 servera python: ansible-service Invoked with name=vsftpd
 pattern=None enabled=True state=started sleep=None arguments= runlevel=default
May 18 22:48:02 servera python: ansible-service Invoked with name=firewalld
 pattern=None enabled=True state=started sleep=None arguments= runlevel=default
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

ansible-playbook(1) man page

Intro to Playbooks — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_intro.html

Playbooks — Ansible Documentation
http://docs.ansible.com/ansible/playbooks.html

Check Mode ("Dry Run") — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_checkmode.html

Start and Step — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_startnstep.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Ansible Playbooks

In this exercise, you will write and use an Ansible playbook to perform administration tasks on a managed host.

Outcomes
You should be able to construct and execute a playbook to manage configuration and perform administration on a managed host.

A developer has asked you to configure Ansible to automate the setup of web servers for your company's intranet web site. The developer is using the host **servera** to develop the web site and test your Ansible playbook.

A working directory, **/home/student/imp-playbook**, has been created on **workstation** for the purpose of managing the managed node, **servera**. The directory has already been populated with an **ansible.cfg** configuration file and an **inventory** inventory file. The managed host, **servera**, is already defined in this inventory file.

The developer needs the managed host to have the latest versions of the *httpd* and *firewalld* packages installed. Also, the **httpd** and **firewalld** services need to be enabled and running. Lastly, **firewalld** should allow remote systems access to the **HTTP** service.

Construct a playbook on the control node, **workstation**, called **/home/student/imp-playbook/intranet.yml**. Create a play in this playbook that configures the managed host as requested by the developer. Also include a task to create the **/var/www/html/index.html** file to test the installation. Populate this file with the message '**Welcome to the example.com intranet!**'.

The playbook should also contain another play which performs a test from the control node to ensure that the web server is accessible across the network. This play should be comprised of a task which makes an HTTP request to **http://servera.lab.example.com/index.html** and verifies that the HTTP status return code is 200.

After the playbook is written, verify its syntax and then execute the playbook to implement the configuration. Verify your work by executing **lab playbook grade**.

Before you begin
Log in as the **student** user on **workstation** and run **lab playbook setup**. This setup script ensures that the managed host, **servera**, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory are installed on the control node.

```
[student@workstation ~]$ lab playbook setup
```

Steps
1. Change directory to the working directory, **/home/student/imp-playbook**.

   ```
   [student@workstation ~] cd /home/student/imp-playbook
   ```

2. Create a new playbook, **/home/student/imp-playbook/intranet.yml**, and define a play for the tasks to be performed on the managed host.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

2.1. Create and open a new playbook, **/home/student/imp-playbook/intranet.yml**, and add a line consisting of three dashes to the beginning of the file to indicate the start of the YAML file.

```
---
```

2.2. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to denote the start of a play with a name of **intranet services**.

```
- name: intranet services
```

2.3. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to indicate that the play applies to the **servera** managed host. Be sure to indent the line with two spaces to indicate that it is contained by the play.

```
  hosts: servera.lab.example.com
```

2.4. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to enable privilege escalation. Be sure to indent the line with two spaces to indicate that it is contained by the play.

```
  become: yes
```

3. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to define the beginning of the **tasks** list. Be sure to indent the line with two spaces to indicate that it is contained by the play.

```
  tasks:
```

4. Add the necessary lines to the **/home/student/imp-playbook/intranet.yml** file to define the group of package management tasks.

4.1. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to create a new block for the tasks of ensuring that the latest versions of the necessary packages are installed. Be sure to indent the line with two spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

```
  - block:
```

4.2. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task for ensuring that the latest version of the *httpd* package is installed. Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with six spaces and calls the **yum** module. The remaining entries are indented with eight

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

spaces and pass the necessary arguments to ensure that the latest version of the *httpd* package is installed.

```
    - name: latest httpd version installed
      yum:
        name: httpd
        state: latest
```

4.3. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task for ensuring that the latest version of the *firewalld* package is installed. Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with six spaces and calls the **yum** module. The remaining entries are indented with eight spaces and pass the necessary arguments to ensure that the latest version of the *firewalld* package is installed.

```
    - name: latest firewalld version installed
      yum:
        name: firewalld
        state: latest
```

5. Add the necessary lines to the **/home/student/imp-playbook/intranet.yml** file to define the firewall configuration task.

5.1. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create a new block to configure the **firewalld**. Be sure to indent the line with two spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

```
  - block:
```

5.2. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task to ensure **firewalld** opens HTTP service to remote systems. Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with six spaces and calls the **firewalld** module. The remaining entries are indented with eight spaces and pass the necessary arguments to ensure that access to the HTTP service is permanently allowed.

```
    - name: firewalld permits http service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes
```

6. Add the necessary lines to the **/home/student/imp-playbook/intranet.yml** file to define the service management tasks.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6.1. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create a new block for service management tasks. Be sure to indent the line with two spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

```
  - block:
```

6.2. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task to ensure the **httpd** service is enabled and running. Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with six spaces and calls the **service** module. The remaining entries are indented with eight spaces and pass the necessary arguments to ensure that the httpd service is enabled and running.

```
    - name: httpd enabled and running
      service:
        name: httpd
        enabled: true
        state: started
```

6.3. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task for ensuring that the **firewalld** service is enabled and running. Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with eight spaces and calls the **service** module. The remaining entries are indented with ten spaces and pass the necessary arguments to ensure that the firewalld service is enabled and started.

```
    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started
```

7. Add the necessary lines to the **/home/student/imp-playbook/intranet.yml** file to define the task for generating web content for testing.

7.1. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create a new block for web content management tasks. Be sure to indent the line with two spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

```
  - block:
```

7.2. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task for populating web content into **/var/www/html/index.html**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Be sure to indent the line with four spaces, a dash, and a space. This indicates that the block is contained by the play and that it is an item in the **tasks** list.

The first entry provides a descriptive name for the task. The second entry is indented with six spaces and calls the **copy** module. The remaining entries are indented with eight spaces and pass the necessary arguments to populate the web content.

```
    - name: test html page
      copy:
        content: "Welcome to the example.com intranet!\n"
        dest: /var/www/html/index.html
```

8. In **/home/student/imp-playbook/intranet.yml**, define another play for the tasks to be performed on the control node.

   8.1. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to denote the start of a second play called '**test**'.

```
- name: test
```

   8.2. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to indicate that the play applies to the **localhost** managed host. Be sure to indent the line with two spaces to indicate that it is contained by the play.

```
  hosts: localhost
```

9. Add the following line to the **/home/student/imp-playbook/intranet.yml** file to define the beginning of the **tasks** list. Be sure to indent the line with two spaces to indicate that it is contained by the play.

```
  tasks:
```

10. Add the following lines to the **/home/student/imp-playbook/intranet.yml** file to create the task for verifying web services from the control node. Be sure to indent the first line with two spaces, a dash, and a space. This indicates that the task is contained by the play and that it is an item in the **tasks** list.

    The first entry provides a descriptive name for the task. The second entry is indented with four spaces and calls the **uri** module. The remaining entries are indented with six spaces and pass the necessary arguments to execute a query for web content from the control node to the managed host and verify the status code received.

```
  - name: connect to intranet
    uri:
      url: http://servera.lab.example.com
      status_code: 200
```

11. Look at the final **/home/student/imp-playbook/intranet.yml** playbook and verify that it has the following structured content. Save the file.

```
---
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
- name: intranet services
  hosts: servera.lab.example.com
  become: yes
  tasks:
  - block:
    - name: latest httpd version installed
      yum:
        name: httpd
        state: latest
    - name: latest firewalld version installed
      yum:
        name: firewalld
        state: latest
  - block:
    - name: firewalld permits http service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes
  - block:
    - name: httpd enabled and running
      service:
        name: httpd
        enabled: true
        state: started
    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started
  - block:
    - name: test html page
      copy:
        content: "Welcome to the example.com intranet!\n"
        dest: /var/www/html/index.html
- name: test
  hosts: localhost
  tasks:
  - name: connect to intranet
    uri:
      url: http://servera.lab.example.com
      status_code: 200
```

12. Verify the syntax of the **intranet.yml** playbook by executing the **ansible-playbook** command with the **--syntax-check** option.

```
[student@workstation imp-playbook]$ ansible-playbook --syntax-check intranet.yml

playbook: intranet.yml
```

13. Execute the playbook. Read through the output generated to ensure that all tasks completed successfully.

```
[student@workstation imp-playbook]$ ansible-playbook intranet.yml

PLAY [intranet services] *******************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [latest httpd version installed] ***************************************
ok: [servera.lab.example.com]

TASK [latest firewalld version installed] ***********************************
ok: [servera.lab.example.com]

TASK [firewalld permits http service] ***************************************
ok: [servera.lab.example.com]

TASK [httpd enabled and running] ********************************************
ok: [servera.lab.example.com]

TASK [firewalld enabled and running] ****************************************
changed: [servera.lab.example.com]

TASK [test html page ] ******************************************************
changed: [servera.lab.example.com]

PLAY [test] *****************************************************************

TASK [setup] ****************************************************************
ok: [localhost]

TASK [connect to intranet] **************************************************
ok: [localhost]

PLAY RECAP ******************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com    : ok=7    changed=2    unreachable=0    failed=0
```

14. Run **lab playbook grade** on **workstation** to grade your work.

```
[student@workstation imp-playbook]$ lab playbook grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Playbooks

In this lab, you will configure and perform administrative tasks on managed hosts using a playbook.

Outcomes
You should be able to construct and execute a playbook to install, configure, and verify the status of web and database services on a managed host.

A developer responsible for the company's Internet website has asked you to write an Ansible playbook to automate the setup of his server environment on **serverb.lab.example.com**.

A working directory, **/home/student/imp-lab**, has been created on **workstation** for the purpose of managing the managed node, **serverb**. The directory has already been populated with an **ansible.cfg** configuration file and an **inventory** inventory file. The managed host, **serverb**, is already defined in this inventory file.

The application being developed on **serverb** will require the installation of the latest versions of the *firewalld*, *httpd*, *php*, *php-mysql*, and *mariadb-server* packages. The **firewalld**, **httpd**, and **mariadb** services also need to be enabled and running. The **firewalld** service must also provide access for remote systems to the web site provided by the **httpd** service.

Construct a playbook on the control node, **workstation**, called **/home/student/imp-lab/internet.yml**. Create a play in this playbook that configures the managed host as requested by the developer. Place the package installation tasks within a block. Place the firewall configuration task within a separate block. Place the service management tasks within another separate block. In a final block, include a task that uses the **get_url** module to fetch and populate the content for the **/var/www/html/index.php** file on the managed host to test the installation. This content is available for download at **http://materials.example.com/grading/var/www/html/index.php**.

The playbook should also contain another play which performs a simple test from the control node to ensure that the web server is accessible across the network as expected. This play should be comprised of a task which uses the **uri** module to make an HTTP request to **http://serverb.lab.example.com/index.php** and verifies that the HTTP status return code is 200.

After the playbook is written, verify its syntax and then execute the playbook to implement the configuration. Verify your work by executing **lab playbookinternet grade**

Before you begin
Log in as the **student** user on **workstation** and run **lab playbookinternet setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network. It will also ensure that the correct Ansible configuration file and inventory are installed on the control node.

```
[student@workstation ~]$ lab playbookinternet setup
```

Steps
1. Change directory to the working directory, **/home/student/imp-lab**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

2. Create a new playbook, **/home/student/imp-lab/internet.yml**, and add the necessary entries to start a first play named "**internet services**" and specify its intended managed host, **serverb.lab.example.com**. Also add an entry to enable privilege escalation.

3. Add the necessary entries to the **/home/student/imp-lab/internet.yml** file to define the tasks in the first play for configuring the managed host. Define blocks of tasks for package management, firewall configuration, service management, and content creation.

4. In **/home/student/imp-lab/internet.yml**, define another play for the task to be performed on the control node to test access to the web server that should be running on the **serverb** managed host. This play does not require privilege escalation.

5. Verify the syntax of the **internet.yml** playbook using the **ansible-playbook** command.

6. Execute the playbook. Read through the output generated to ensure that all tasks completed successfully.

7. Run **lab playbookinternet grade** on **workstation** to grade your work.

```
[student@workstation imp-lab]$ lab playbookinternet grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will configure and perform administrative tasks on managed hosts using a playbook.

Outcomes

You should be able to construct and execute a playbook to install, configure, and verify the status of web and database services on a managed host.

A developer responsible for the company's Internet website has asked you to write an Ansible playbook to automate the setup of his server environment on **serverb.lab.example.com**.

A working directory, **/home/student/imp-lab**, has been created on **workstation** for the purpose of managing the managed node, **serverb**. The directory has already been populated with an **ansible.cfg** configuration file and an **inventory** inventory file. The managed host, **serverb**, is already defined in this inventory file.

The application being developed on **serverb** will require the installation of the latest versions of the *firewalld*, *httpd*, *php*, *php-mysql*, and *mariadb-server* packages. The **firewalld**, **httpd**, and **mariadb** services also need to be enabled and running. The **firewalld** service must also provide access for remote systems to the web site provided by the **httpd** service.

Construct a playbook on the control node, **workstation**, called **/home/student/imp-lab/internet.yml**. Create a play in this playbook that configures the managed host as requested by the developer. Place the package installation tasks within a block. Place the firewall configuration task within a separate block. Place the service management tasks within another separate block. In a final block, include a task that uses the **get_url** module to fetch and populate the content for the **/var/www/html/index.php** file on the managed host to test the installation. This content is available for download at **http://materials.example.com/grading/var/www/html/index.php**.

The playbook should also contain another play which performs a simple test from the control node to ensure that the web server is accessible across the network as expected. This play should be comprised of a task which uses the **uri** module to make an HTTP request to **http://serverb.lab.example.com/index.php** and verifies that the HTTP status return code is 200.

After the playbook is written, verify its syntax and then execute the playbook to implement the configuration. Verify your work by executing **lab playbookinternet grade**

Before you begin

Log in as the **student** user on **workstation** and run **lab playbookinternet setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network. It will also ensure that the correct Ansible configuration file and inventory are installed on the control node.

```
[student@workstation ~]$ lab playbookinternet setup
```

Steps

1. Change directory to the working directory, **/home/student/imp-lab**.

```
[student@workstation ~] cd /home/student/imp-lab
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

2. Create a new playbook, **/home/student/imp-lab/internet.yml**, and add the necessary entries to start a first play named "**internet services**" and specify its intended managed host, **serverb.lab.example.com**. Also add an entry to enable privilege escalation.

   2.1. Add the following entry to the beginning of **/home/student/imp-lab/internet.yml** to begin the YAML format.

   ```
   ---
   ```

   2.2. Add the following entry to the **/home/student/imp-lab/internet.yml** file to denote the start of a play with a name of **internet services**.

   ```
   - name: internet services
   ```

   2.3. Add the following entry to the **/home/student/imp-lab/internet.yml** file to indicate that the play applies to the **serverb** managed host. Be sure to indent the entry with two spaces to indicate that it is contained by the play.

   ```
     hosts: serverb.lab.example.com
   ```

   2.4. Add the following entry to the **/home/student/imp-lab/internet.yml** file to enable privilege escalation. Be sure to indent the entry with two spaces to indicate that it is contained by the play.

   ```
     become: yes
   ```

3. Add the necessary entries to the **/home/student/imp-lab/internet.yml** file to define the tasks in the first play for configuring the managed host. Define blocks of tasks for package management, firewall configuration, service management, and content creation.

   3.1. Add the following entry to the **/home/student/imp-lab/internet.yml** file to define the beginning of the **tasks** list. Be sure to indent the entry with two spaces to indicate that it is contained by the play.

   ```
     tasks:
   ```

   3.2. Add the following entry to the **/home/student/imp-lab/internet.yml** file to create a new block for the tasks of ensuring that the latest versions of the necessary packages are installed.

   ```
   - block:
     - name: latest httpd version installed
       yum:
         name: httpd
         state: latest
     - name: latest firewalld version installed
       yum:
         name: firewalld
         state: latest
     - name: latest mariadb-server version installed
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    yum:
      name: mariadb-server
      state: latest
  - name: latest php version installed
    yum:
      name: php
      state: latest
  - name: latest php-mysql version installed
    yum:
      name: php-mysql
      state: latest
```

3.3. Add the necessary entries to the **/home/student/imp-lab/internet.yml** file to define a separate block for the firewall configuration task.

```
- block:
  - name: firewalld permits http service
    firewalld:
      service: http
      permanent: true
      state: enabled
      immediate: yes
```

3.4. Add the necessary entries to the **/home/student/imp-lab/internet.yml** file to define a separate block for the service management tasks.

```
- block:
  - name: httpd enabled and running
    service:
      name: httpd
      enabled: true
      state: started
  - name: mariadb enabled and running
    service:
      name: mariadb
      enabled: true
      state: started
  - name: firewalld enabled and running
    service:
      name: firewalld
      enabled: true
      state: started
```

3.5. Add the necessary entries to the **/home/student/imp-lab/internet.yml** file to define the final task block for generating web content for testing.

```
- block:
  - name: get test php page
    get_url:
      url: "http://materials.example.com/grading/var/www/html/index.php"
      dest: /var/www/html/index.php
      mode: 0644
```

4. In **/home/student/imp-lab/internet.yml**, define another play for the task to be performed on the control node to test access to the web server that should be running on the **serverb** managed host. This play does not require privilege escalation.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4.1. Add the following entry to the **/home/student/imp-lab/internet.yml** file to denote the start of a second play with a name of **test**.

```
- name: test
```

4.2. Add the following entry to the **/home/student/imp-lab/internet.yml** file to indicate that the play applies to the **localhost** managed host. Be sure to indent the entry with two spaces to indicate that it is contained by the play.

```
  hosts: localhost
```

4.3. Add an entry to the **/home/student/imp-lab/internet.yml** file to define the beginning of the **tasks** list. Be sure to indent the entry with two spaces to indicate that it is contained by the play.

```
  tasks:
```

4.4. Add entries to the **/home/student/imp-lab/internet.yml** file to create the task for verifying the managed host's web services from the control node.

```
  - name: connect to internet web server
    uri:
      url: http://serverb.lab.example.com
      status_code: 200
```

5. Verify the syntax of the **internet.yml** playbook using the **ansible-playbook** command.

```
[student@workstation imp-lab]$ cat internet.yml
---
- name: internet services
  hosts: serverb.lab.example.com
  become: yes
  tasks:
  - block:
    - name: latest httpd version installed
      yum:
        name: httpd
        state: latest
    - name: latest firewalld version installed
      yum:
        name: firewalld
        state: latest
    - name: latest mariadb-server version installed
      yum:
        name: mariadb-server
        state: latest
    - name: latest php version installed
      yum:
        name: php
        state: latest
    - name: latest php-mysql version installed
      yum:
        name: php-mysql
        state: latest
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  - block:
    - name: firewalld permits http service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes
  - block:
    - name: httpd enabled and running
      service:
        name: httpd
        enabled: true
        state: started
    - name: mariadb enabled and running
      service:
        name: mariadb
        enabled: true
        state: started
    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started
  - block:
    - name: get test php page
      get_url:
        url: "http://materials.example.com/grading/var/www/html/index.php"
        dest: /var/www/html/index.php
        mode: 0644
- name: test
  hosts: localhost
  tasks:
  - name: connect to internet web server
    uri:
      url: http://serverb.lab.example.com
      status_code: 200
[student@workstation imp-lab]$ ansible-playbook --syntax-check internet.yml

playbook: internet.yml
```

6. Execute the playbook. Read through the output generated to ensure that all tasks completed successfully.

```
[student@workstation imp-lab]$ ansible-playbook internet.yml
PLAY [internet services] *******************************************************

TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]

TASK [latest httpd version installed] ******************************************
changed: [serverb.lab.example.com]

TASK [latest firewalld version installed] **************************************
ok: [serverb.lab.example.com]

TASK [latest mariadb-server version installed] *********************************
changed: [serverb.lab.example.com]

TASK [latest php version installed] ********************************************
changed: [serverb.lab.example.com]

TASK [latest php-mysql version installed] **************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
changed: [serverb.lab.example.com]

TASK [firewalld permits http service] ***************************************
changed: [serverb.lab.example.com]

TASK [httpd enabled and running] ********************************************
changed: [serverb.lab.example.com]

TASK [mariadb enabled and running] ******************************************
changed: [serverb.lab.example.com]

TASK [firewalld enabled and running] ****************************************
changed: [serverb.lab.example.com]

TASK [get test php page] ****************************************************
changed: [serverb.lab.example.com]

PLAY [test] *****************************************************************

TASK [setup] ****************************************************************
ok: [localhost]

TASK [connect to internet web server] ***************************************
ok: [localhost]

PLAY RECAP ******************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
serverb.lab.example.com    : ok=11   changed=9    unreachable=0    failed=0
```

7. Run **lab playbookinternet grade** on **workstation** to grade your work.

```
[student@workstation imp-lab]$ lab playbookinternet grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible *playbooks* are used to describe standard tasks needed to configure managed hosts
- Ansible playbooks are written in YAML format
- YAML files are structured using space indentation to represent data hierarchy
- Tasks are implemented using standardized code packaged as Ansible *modules*
- The **`ansible-doc`** command can list installed modules, and provide documentation and example code snippets of how to use them in playbooks
- The **`ansible-playbook`** command is used to verify playbook syntax and execute playbooks

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 4

MANAGING VARIABLES AND INCLUSIONS

Overview	
Goal	To describe variable scope and precedence, manage variables and facts in a play, and manage inclusions.
Objectives	• Manage variables in Ansible projects • Manage facts in playbooks • Include variables and tasks from external files into a playbook
Sections	• Managing Variables (and Guided Exercise) • Managing Facts (and Guided Exercise) • Managing Inclusions (and Guided Exercise)
Lab	• Lab: Managing Variables and Inclusions

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Variables

Objectives

After completing this section, students should be able to:

- Manage variables in Ansible projects

Introduction to Ansible variables

Ansible supports variables that can be used to store values that can be reused throughout files in an entire Ansible project. This can help simplify creation and maintenance of a project and reduce the incidence of errors.

Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project. Some examples of values that variables might contain include

- Users to create
- Packages to install
- Services to restart
- Files to remove
- Archives to retrieve from the Internet

Naming variables

Variables have names which consist of a string that must start with a letter and can only contain letters, numbers, and underscores.

Consider the following table that shows the difference between invalid and valid variable names:i

Ansible variable names	
Invalid variable names	**Valid variable names**
`web server`	`web_server`
`remote.file`	`remote_file`
`1st file`	`file_1` or `file1`
`remoteserver$1`	`remote_server_1` or `remote_server1`

Defining variables

Variables can be defined in a bewildering variety of places in an Ansible project. However, this can be simplified to three basic scope levels:

- *Global scope*: Variables set from the command line or Ansible configuration
- *Play scope*: Variables set in the play and related structures
- *Host scope*: Variables set on host groups and individual hosts by the inventory, fact gathering, or registered tasks

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

If the same variable name is defined at more than one level, the higher wins. So variables defined by the inventory are overridden by variables defined by the playbook, which are overridden by variables defined on the command line.

A detailed discussion of variable precedence is available in the Ansible documentation, a link to which is provided in the References at the end of this section.

Variables in playbooks

Defining variables in playbooks

When writing playbooks, administrators can use their own variables and call them in a task. For example, a variable **web_package** can be defined with a value of **httpd** and called by the **yum** module in order to install the *httpd* package.

Playbook variables can be defined in multiple ways. One of the simplest is to place it in a **vars** block at the beginning of a playbook:

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

It is also possible to define playbook variables in external files. In this case, instead of using **vars**, the **vars_files** directive may be used, followed by a list of external variable files relative to the playbook that should be read:

```
- hosts: all
  vars_files:
    - vars/users.yml
```

The playbook variables are then defined in that file or those files in YAML format:

```
user: joe
home: /home/joe
```

Using variables in playbooks

Once variables have been declared, administrators can use the variables in tasks. Variables are referenced by placing the variable name in double curly braces. Ansible substitutes the variable with its value when the task is executed.

```
vars:
  user: joe

tasks:
  # This line will read: Creates the user joe
  - name: Creates the user {{ user }}
    user:
      # This line will create the user named Joe
      name: "{{ user }}"
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Important

When a variable is used as the first element to start a value, quotes are mandatory. This prevents Ansible from considering the variable as starting a YAML dictionary. The following message appears if quotes are missing:

```
yum:
     name: {{ service}}
            ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```

Host variables and group variables

Inventory variables that apply directly to hosts fall into two broad categories: *host variables* that apply to a specific host, and *group variables* that apply to all hosts in a host group or in a group of host groups. Host variables take precedence over group variables, but variables defined by a playbook take precedence over both.

One way to define host variables and group variables is to do it directly in the inventory file. This is an older approach and not preferred, but may be encountered by users:

- This is a host variable, **ansible_user**, being defined for the host **demo.example.com**.

```
[servers]
demo.example.com  ansible_user=joe
```

- In this example, a group variable **user** is being defined for the group **servers**.

```
[servers]
demo1.example.com
demo2.example.com

[servers:vars]
user=joe
```

- Finally, in this example a group variable **user** is being defined for the group **servers**, which happens to consist of two host groups each with two servers.

```
[servers1]
demo1.example.com
demo2.example.com

[servers2]
demo3.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
demo4.example.com

[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

Among the disadvantages of this approach, it makes the inventory file more difficult to work with, mixes information about hosts and variables in the same file, and uses an obsolete syntax.

Using group_vars and host_vars directories

The preferred approach is to create two directories in the same working directory as the inventory file or directory, **group_vars** and **host_vars**. These directories contain files defining group variables and host variables, respectively.



Important

The recommended practice is to define inventory variables using **host_vars** and **group_vars** directories, and not to define them directly in the inventory file or files.

To define group variables for the group **servers**, a YAML file named **group_vars/servers** would be created, and then the contents of that file would set variables to values using the same syntax as a playbook:

```
user: joe
```

Likewise, to define host variables for a particular host, a file with a name matching the host is created in **host_vars** to contain the host variables.

The following examples illustrate this approach in more detail. Consider the following scenario where there are two data centers to manage that has the following inventory file in **~/project/inventory**:

```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

- If a general value needs to be defined for all servers in both datacenters, a group variable can be set for **datacenters**:

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- If the value to define varies for each datacenter, a group variable can be set for each datacenter:

```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

- If the value to be defined varies for each host in every datacenter, using host variables is recommended:

```
[admin@station project]$ cat ~/project/host_vars/demo1.example.com
package: httpd
[admin@station project]$ cat ~/project/host_vars/demo2.example.com
package: apache
[admin@station project]$ cat ~/project/host_vars/demo3.example.com
package: mariadb-server
[admin@station project]$ cat ~/project/host_vars/demo4.example.com
package: mysql-server
```

The directory structure for **project**, if it contained all of the example files above, might look like this:

```
project
├── ansible.cfg
├── group_vars
│   ├── datacenters
│   ├── datacenters1
│   └── datacenters2
├── host_vars
│   ├── demo1.example.com
│   ├── demo2.example.com
│   ├── demo3.example.com
│   └── demo4.example.com
├── inventory
└── playbook.yml
```

Overriding variables from the command line

Inventory variables are overridden by variables set in a playbook, but both kinds of variables may be overridden through arguments passed to the **ansible** or **ansible-playbook** commands on the command line. This can be useful in a case where the defined value for a variable needs to be overridden for a single host for a one-off run of a playbook. For example:

```
[user@demo ~]$ ansible-playbook demo2.example.com main.yml -e "package=apache"
```

Variables and arrays

Instead of assigning a piece of configuration data that relates to the same element (a list of packages, a list of services, a list of users, etc.) to multiple variables, administrators can use *arrays*. One interesting consequence of this is that an array can be browsed.

For instance, consider the following snippet:

```
user1_first_name: Bob
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user3_home_dir: /users/acook
```

This could be rewritten as an array called **users**:

```
users:
  bjones:
    first_name: Bob
    last_name: Jones
    home_dir: /users/bjones
  acook:
    first_name: Anne
    last_name: Cook
    home_dir: /users/acook
```

Users can then be accessed using the following variables:

```
# Returns 'Bob'
users.bjones.first_name

# Returns '/users/acook'
users.acook.home_dir
```

Because the variable is defined as a Python *dictionary*, an alternative syntax is available.

```
# Returns 'Bob'
users['bjones']['first_name']

# Returns '/users/acook'
users['acook']['home_dir']
```



Important

The dot notation can cause problems if the key names are the same as names of Python methods or attributes, such as **discard**, **copy**, **add**, and so on. Using the brackets notation can help avoid errors.

Both syntaxes are valid, but to make troubleshooting easier, it is recommended that one syntax used consistently in all files throughout any given Ansible project.

Registered variables

Administrators can capture the output of a command by using the **register** statement. The output is saved into a variable that could be used later for either debugging purposes or in order to achieve something else, such as a particular configuration based on a command's output.

The following playbook demonstrates how to capture the output of a command for debugging purposes:

```
---
- name: Installs a package and prints the result
  hosts: all
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  tasks:
    - name: Install the package
      yum:
        name: httpd
        state: installed
      register: install_result

    - debug: var=install_result
```

When the playbook is run, the **debug** module is used to dump the value of the **install_result** registered variable to the terminal.

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY [Installs a package and prints the result] ***************************

TASK [setup] *******************************************************************
ok: [demo.example.com]

TASK [Install the package] *****************************************************
ok: [demo.example.com]

TASK [debug] *******************************************************************
ok: [demo.example.com] => {
    "install_result": {
        "changed": false,
        "msg": "",
        "rc": 0,
        "results": [
            "httpd-2.4.6-40.el7.x86_64 providing httpd is already installed"
        ]
    }
}

PLAY RECAP *********************************************************************
demo.example.com    : ok=3    changed=0    unreachable=0    failed=0
```

Demonstration: Managing variables

1. From **workstation**, as the **student** user, change into the **~/demo_variables-playbook** directory.

   ```
   [student@workstation ~]$ cd demo_variables-playbook
   [student@workstation demo_variables-playbook]$
   ```

2. Look at the **inventory** file. Notice that there are two host groups, **webservers** and **dbservers**, which are children of the larger host group **servers**. The **servers** host group has variables set the old way, directly in the inventory file, including one that sets **package** to **httpd**.

   ```
   [webservers]
   servera.lab.example.com

   [dbservers]
   servera.lab.example.com

   [servers:children]
   webservers
   dbservers
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[servers:vars]
ansible_user=devops
ansible_become=yes
package=httpd
```

3. Create a new playbook, **playbook.yml**, for all hosts. Using the **yum** module, install the package specified by the **package** variable.

```
---
- hosts: all
  tasks:
    - name: Installs the "{{ package }}" package
      yum:
        name: "{{ package }}"
        state: latest
```

4. Run the playbook using the **ansible-playbook** command. Watch the output as Ansible installs the *httpd* package.

```
[student@workstation demo_variables-playbook]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Installs the "httpd" package] ********************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

5. Run an ad hoc command to ensure the *httpd* package has been successfully installed.

```
[student@workstation demo_variables-playbook]$ ansible servers \
> -a 'yum list installed httpd'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
httpd.x86_64                      2.4.6-40.el7                        @rhel_dvd
```

6. Create the **group_vars** directory and a new file **group_vars/dbservers** to set the variable **package** to **mariadb-server** for the **dbservers** host group in the recommended way.

```
[student@workstation demo_variables-playbook]$ cat group_vars/dbservers
package: mariadb-server
```

7. Run the playbook again using the **ansible-playbook** command. Watch Ansible install the *mariadb-server* package.

   The **package** variable for the more specific host group **dbservers** took precedence over the one for its parent host group **servers**.

```
[student@workstation demo_variables-playbook]$ ansible-playbook playbook.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Installs the "mariadb-server" package] ***********************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

The output indicates the variable defined for the hosts group has been overridden.

8. Run an ad hoc command to confirm the **mariadb-server** package has been successfully installed.

```
[student@workstation demo_variables-playbook]$ ansible dbservers \
> -a 'yum list installed mariadb-server'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
mariadb-server.x86_64                1:5.5.44-2.el7                 @rhel_dvd
```

9. For **servera.lab.example.com**, set the variable **package** to **screen**.
Do this by creating a new directory, **host_vars**, and a file **host_vars/servera.lab.example.com** that sets the variable in the recommended way.

```
[student@workstation demo_variables-playbook]$ cat host_vars/servera.lab.example.com
package: screen
```

10. Run the playbook again. Watch the output as Ansible installs the *screen* package.

The host-specific value of the **package** variable overrides any value set by the host's host groups.

```
[student@workstation demo_variables-playbook]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Installs the "screen" package] *******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

11. Run an ad hoc command to confirm the **screen** package has been successfully installed.

```
[student@workstation demo_variables-playbook]$ ansible servera.lab.example.com \
> -a 'yum list installed screen'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
screen.x86_64            4.1.0-0.21.20120314git3c2946.el7             rhel_dvd
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

12. Run the **ansible-playbook** command again, this time using the **-e** option to override the **package** variable.

```
[student@workstation demo_variables-playbook]$ ansible-playbook playbook.yml \
> -e 'package=mutt'
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Installs the "mutt" package] *********************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

13. Run an ad hoc command to confirm the *mutt* package is installed.

```
[student@workstation demo_variables-playbook]$ ansible all \
> -a 'yum list installed mutt'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
mutt.x86_64     5:1.5.21-26.el7         @rhel_dvd
```



References

Inventory — Ansible Documentation
http://docs.ansible.com/ansible/intro_inventory.html

Variables — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_variables.htm

Variable Precedence: Where Should I Put A Variable?
http://docs.ansible.com/ansible/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

YAML Syntax — Ansible Documentation
http://docs.ansible.com/ansible/YAMLSyntax.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Variables

In this exercise, you will define and use variables in a playbook.

Outcomes
You should be able to:

- Define variables in a playbook.
- Create various tasks that include defined variables.

Before you begin
From **workstation**, run the lab setup script to confirm the environment is ready for the exercise to begin. The script creates the working directory, called **dev-vars-playbook**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab manage-variables-playbooks setup
```

Steps
1. From **workstation**, as the **student** user, change into the **~/dev-vars-playbook** directory.

   ```
   [student@workstation ~]$ cd dev-vars-playbook
   [student@workstation dev-vars-playbook]$
   ```

2. Over the next several steps, you will create a playbook that installs the Apache web server and opens the ports for the service to be reachable. The playbook queries the web server to ensure it is up and running.

   First, create the playbook **playbook.yml** and define the following variables in the **vars** section: **web_pkg**, which defines the name of the package to install for the web server; **firewall_pkg**, which defines the name of the firewall package; **web_service** for the name of the web service to manage; and **firewall_service** for the name of the firewall service to manage. Add the **python_pkg** variable to install a required package for the **uri** module; and **rule**, which defines the service to open.

   ```
   ---
   - name: Install Apache and start the service
     hosts: webserver
     vars:
       web_pkg: httpd
       firewall_pkg: firewalld
       web_service: httpd
       firewall_service: firewalld
       python_pkg: python-httplib2
       rule: http
   ```

3. Create the **tasks** block and create the first task which uses the **yum** module to install the required packages.

   ```
     tasks:
       - name: Install the required packages
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    yum:
      name:
        - "{{ web_pkg  }}"
        - "{{ firewall_pkg }}"
        - "{{ python_pkg }}"
      state: latest
```

4. Create two tasks to start and enable the **`httpd`** and **`firewalld`** services.

```
  - name: Start and enable the {{ firewall_service }} service
    service:
      name: "{{ firewall_service }}"
      enabled: true
      state: started

  - name: Start and enable the {{ web_service }} service
    service:
      name: "{{ web_service }}"
      enabled: true
      state: started
```

5. Add a task that will create some content in **`/var/www/html/index.html`**.

```
  - name: Create web content to be served
    copy:
      content: "Example web content"
      dest: /var/www/html/index.html
```

6. Add a task that will use the **`firewalld`** module to add a rule for the web service.

```
  - name: Open the port for {{ rule }}
    firewalld:
      service: "{{ rule }}"
      permanent: true
      immediate: true
      state: enabled
```

7. Create a new play that will query the web service to ensure everything has been correctly configured. It should run on **`localhost`**. The **`uri`** module can be used to check a URL. For this task, check for a status code of 200 to confirm the server is running and properly configured.

```
- name: Verify the Apache service
  hosts: localhost
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://servera.lab.example.com
        status_code: 200
```

8. When completed with all tasks, the playbook should appear as follows. Review the playbook and ensure all the tasks are defined.

```
---
- name: Install Apache and start the service
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  hosts: webserver
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python-httplib2
    rule: http

  tasks:
    - name: Install the required packages
      yum:
        name:
          - "{{ web_pkg  }}"
          - "{{ firewall_pkg }}"
          - "{{ python_pkg }}"
        state: latest

    - name: Start and enable the {{ firewall_service }} service
      service:
        name: "{{ firewall_service }}"
        enabled: true
        state: started

    - name: Start and enable the {{ web_service }} service
      service:
        name: "{{ web_service }}"
        enabled: true
        state: started

    - name: Create web content to be served
      copy:
        content: "Example web content"
        dest: /var/www/html/index.html

    - name: Open the port for {{ rule }}
      firewalld:
        service: "{{ rule }}"
        permanent: true
        immediate: true
        state: enabled

- name: Verify the Apache service
  hosts: localhost
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://servera.lab.example.com
        status_code: 200
```

9. The playbook can be run using the **`ansible-playbook`** command. Watch the output as Ansible starts by installing the packages, starting and enabling the services, and ensuring the web server is reachable.

```
[student@workstation dev-vars-playbook]$ ansible-playbook playbook.yml

PLAY [Install Apache and start the service] ***********************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Install the required packages] ******************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
changed: [servera.lab.example.com]

TASK [Start and enable the firewalld service] *********************************
changed: [servera.lab.example.com]

TASK [Start and enable the httpd service] *************************************
changed: [servera.lab.example.com]

TASK [Create web content to be served] ****************************************
changed: [servera.lab.example.com]

TASK [Open the port for http] *************************************************
changed: [servera.lab.example.com]

PLAY [Verify the Apache service] **********************************************

TASK [setup] ******************************************************************
ok: [localhost]

TASK [Ensure the webserver is reachable] **************************************
ok: [localhost]

PLAY RECAP ********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com    : ok=6    changed=5    unreachable=0    failed=0
```

Evaluation

From *workstation*, run the **lab manage-variables-playbooks grade** command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab manage-variables-playbooks grade
```

Cleanup

Run the **lab manage-variables-playbooks cleanup** command to undo the changes made to **servera**.

```
[student@workstation ~]$ lab manage-variables-playbooks cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Facts

Objectives

After completing this section, students should be able to:

- Manage facts in playbooks

Ansible facts

Ansible *facts* are variables that are automatically discovered by Ansible from a managed host. Facts are pulled by the **setup** module and contain useful information stored into variables that administrators can reuse. Ansible facts can be part of playbooks, in conditionals, loops, or any other dynamic statement that depends on a value for a managed host; for example:

- A server can be restarted depending on the current kernel version.
- The MySQL configuration file can be customized depending on the available memory.
- Users can be created depending on the host name.

There are virtually no limits to how Ansible facts can be used, since they can be part of blocks, loops, conditionals, and so on. Ansible facts are a convenient way to retrieve the state of a managed node and decide which action to take based on its state. Facts provide information about

- The host name
- The kernel version
- The network interfaces
- The IP addresses
- The version of the operating system
- Various environment variables
- The number of CPUs
- The available or free memory
- The available disk space

The following snippet shows some of the facts Ansible gathered from a managed node:

```
[user@demo ~]$ ansible demo1.example.com -m setup
ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        "ansible_bios_version": "0.5.1",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
            "LANG": "en_US.UTF-8",
            "console": "ttyS0,115200n8",
            "crashkernel": "auto",
            "net.ifnames": "0",
            "no_timer_check": true,
            "ro": true,
            "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
        }
... Output omitted ...
```

The output is returned in JSON, and each value is stored in a Python dictionary. Administrators can then browse the dictionaries to retrieve a particular value. The following table shows some of the facts gathered from a managed node that may be useful in a playbook:

Ansible facts

Fact	*Variable*
Hostname	**{{ ansible_hostname }}**
Main IPv4 address (based on routing)	**{{ ansible_default_ipv4.address }}**
Main disk first partition size (based on disk name, such as **vda**, **vdb**, and so on.)	**{{ ansible_devices.vda.partitions.vda1.size }}**
DNS servers	**{{ ansible_dns.nameservers }}**
kernel version	**{{ ansible_kernel }}**

When a fact is used in a playbook, Ansible dynamically substitutes the variable name with the corresponding value:

```
---
- hosts: all
  tasks:
  - name: Prints various Ansible facts
    debug:
      msg: >
        The default IPv4 address of {{ ansible_fqdn }}
        is {{ ansible_default_ipv4.address }}
```

The following output shows how Ansible was able to query the managed node and dynamically use the system information to update the variable. Moreover, facts can be also used to create dynamic groups of hosts that match particular criteria.

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [demo1.example.com]

TASK [Prints various Ansible facts] ********************************************
ok: [demo1.example.com] => {
    "msg": "The default IPv4 address of demo1.example.com is
            172.25.250.10"
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
demo1.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

Facts filters

Ansible facts contain extensive information about the system. Administrators can use Ansible *filters* in order to limit the results returned when gathering facts from a managed node. Filters can be used to:

- Only retrieve information about network cards.
- Only retrieve information about disks.
- Only retrieve information about users.

To use filters, the expression needs to be passed as an option, using **`-a 'filter=EXPRESSION'`**. For example, to only return information about **`eth0`**, a filter can be applied on the **`ansible_eth0`** element:

```
[user@demo ~]$ ansible demo1.example.com -m setup -a 'filter=ansible_eth0'
demo1.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "172.25.250.10",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
            "ipv6": [
                {
                    "address": "fe80::5054:ff:fe00:fa0a",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "52:54:00:00:fa:0a",
            "module": "virtio_net",
            "mtu": 1500,
            "pciid": "virtio0",
            "promisc": false,
            "type": "ether"
        }
    },
    "changed": false
}
```

Notice that Ansible only returns the facts in the **`ansible_eth0`** array.

Administrators can manually disable facts for managed hosts if a large number of servers are being managed. To disable facts, set **`gather_facts`** to **`no`** in the playbook:

```
---
- hosts: large_farm
  gather_facts: no
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Facts are always gathered by Ansible unless overridden in this way.

Custom facts

Administrators can create their own facts and push them to a managed node. Upon creation, the custom facts will be integrated and read by the **setup** module. Custom facts can be used for:

- Defining a particular value for a system based on a custom script.
- Defining a value based on a program execution.

Custom facts are found by Ansible if the facts file is saved in the **/etc/ansible/facts.d** directory. Files must have **.fact** as an extension. A facts file is a plain-text file in INI or JSON format. An INI facts file contains a top level defined by a section, followed by the key-value pairs for the facts to define:

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

If supplied in JSON, the following syntax needs to be used:

```
{
  "packages": {
    "web_package": "httpd",
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```

For both formats, the result returned by Ansible is the same and will be put in the **ansible_local** level. Using filters, it is then possible to ensure custom facts have been successfully installed and retrieved:

```
[user@demo ~]$ ansible demo1.example.com -m setup -a 'filter=ansible_local'
demo1.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "users": {
                    "user1": "joe",
                    "user2": "jane"
                }
            }
        }
    },
    "changed": false
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Custom facts can be used the same way as default facts in playbooks:

```
---
- hosts: all
  tasks:
  - name: Prints various Ansible facts
    debug:
      msg: The package to install on {{ ansible_fqdn }} is
 {{ ansible_local.custom.packages.web_package }}
```

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [demo1.example.com]

TASK [Prints various Ansible facts] ********************************************
ok: [demo1.example.com] => {
    "msg": "The package to install on demo1.example.com is httpd"
}

PLAY RECAP *********************************************************************
demo1.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

Demonstration: Managing facts

1. From **workstation**, as the **student** user, change into the **~/demo-vars-facts** directory.

   ```
   [student@workstation ~]$ cd demo-vars-facts
   [student@workstation demo-vars-facts]$
   ```

2. Create the **custom.fact** INI file. Create the **package** section and define two entries for the packages to install.

   ```
   [packages]
   web_package: httpd
   db_package: mariadb-server
   ```

3. Create the **setup_facts.yml** playbook to install the facts file on the managed host.

   ```
   ---
   - name: Installs remote facts
     hosts: lamp
     vars:
       remote_dir: /etc/ansible/facts.d
       facts_file: custom.fact
     tasks:
       - name: Creates the remote directory
         file:
           state: directory
           recurse: yes
           path: "{{ remote_dir }}"
       - name: Installs the new facts
         copy:
           src: "{{ facts_file }}"
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        dest: "{{ remote_dir }}"
```

4. Using the **ansible** command, use the **setup** module to discover some of the available facts on **servera.lab.example.com**.

```
[student@workstation demo-vars-facts]$ ansible lamp -m setup
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
        "ansible_bios_version": "0.5.1",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
            "LANG": "en_US.UTF-8",
            "console": "ttyS0,115200n8",
            "crashkernel": "auto",
            "net.ifnames": "0",
            "no_timer_check": true,
            "ro": true,
            "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
        },
...
```

5. Using the **filter** module, filter the facts returned by the module to see what the DNS resolver settings for the managed host currently are.

```
[student@workstation demo-vars-facts]$ ansible lamp -m setup -a 'filter=ansible_dns'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_dns": {
            "nameservers": [
                "172.25.250.254"
            ],
            "search": [
                "lab.example.com",
                "example.com"
            ]
        }
    },
    "changed": false
}
```

6. Run the **setup_facts.yml** playbook to install the custom facts on **servera.lab.example.com**.

```
[student@workstation demo-vars-facts]$ ansible-playbook setup_facts.yml

PLAY [Installs remote facts] ***************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Creates the remote directory] *********************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ok: [servera.lab.example.com]

TASK [Installs the new facts] *************************************************
changed: [servera.lab.example.com]

PLAY RECAP ********************************************************************
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

7. Use the **setup** module again to ensure the custom facts were set up on **servera**. Filter on **ansible_local** to display only the custom facts.

```
[student@workstation demo-vars-facts]$ ansible lamp -m setup \
> -a 'filter=ansible_local'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                }
            }
        }
    },
    "changed": false
}
```

Notice that the two new facts were successfully installed. They can now be used in playbooks.

8. Create the main **playbook.yml** playbook for all servers in the **lamp** group. Create a new task to install the packages defined by the Ansible custom facts.

```
---
- hosts: lamp
  tasks:
    - name: Installs packages on {{ ansible_fqdn }}
      yum:
        name:
          -  "{{ ansible_local.custom.packages.db_package }}"
          -  "{{ ansible_local.custom.packages.web_package }}"
        state: latest
      register: result
    - name: Displays the result of the command
      debug:
        var: result
```

9. Run the playbook and watch the output that prints information about the packages that have been installed.

```
[student@workstation demo-vars-facts]$ ansible-playbook playbook.yml
PLAY [Installs packages] ******************************************************

TASK [setup] ******************************************************************
ok: [servera.lab.example.com]

TASK [Installs packages on servera.lab.example.com] ***************************
changed: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [Displays the result of the command] ********************************
ok: [servera.lab.example.com] => {
    "result": {
        "changed": true,
        "msg": "",
        "rc": 0,
        "results": [
...
```

10. Use an ad hoc command to confirm both *httpd* and *mariadb-server* were successfully installed:

```
[student@workstation demo-vars-facts]$ ansible lamp -a \
> 'yum list installed httpd mariadb-server'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
httpd.x86_64                            2.4.6-40.el7                    @rhel_dvd
mariadb-server.x86_64                   1:5.5.44-2.el7                  @rhel_dvd
```



References

setup - Gathers facts about remote hosts — Ansible Documentation
http://docs.ansible.com/ansible/setup_module.html

Local Facts (Facts.d) — Variables — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_variables.html#local-facts-facts-d

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Facts

In this exercise, you will gather Ansible facts from a managed host and use them in playbooks.

Outcomes
You should be able to:

- Gather facts from a host.
- Create various tasks that use the gathered facts.

Before you begin
From **workstation**, run the lab setup script to confirm the environment is ready for the exercise to begin. The script creates the working directory, called **dev-vars-facts**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab manage-variables-facts setup
```

Steps
1. From **workstation**, as the **student** user, change into the **~/dev-vars-facts** directory.

```
[student@workstation ~]$ cd dev-vars-facts
[student@workstation dev-vars-facts]$
```

2. The Ansible **setup** module retrieves facts from a system. Run an ad hoc command to retrieve the facts for all servers in the **webserver** group. The output will display all the facts gathered for **servera.lab.example.com** in JSON format. Review some of the variables displayed.

```
[student@workstation dev-vars-facts]$ ansible webserver -m setup
... Output omitted ...
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
... Output omitted ...
```

3. Filter the facts matching the **ansible_user** expression. Append a wildcard to match all facts starting with **ansible_user**.

```
[student@workstation dev-vars-facts]$ ansible webserver -m setup -a
 'filter=ansible_user*'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_user_dir": "/root",
        "ansible_user_gecos": "root",
        "ansible_user_gid": 0,
        "ansible_user_id": "root",
        "ansible_user_shell": "/bin/bash",
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        "ansible_user_uid": 0,
        "ansible_userspace_architecture": "x86_64",
        "ansible_userspace_bits": "64"
    },
    "changed": false
}
```

4. Create a fact file named **custom.fact**. The fact file defines the package to install and the service to start on **servera**. The file should read as follows:

```
[general]
package = httpd
service = httpd
state = started
```

5. Create the **setup_facts.yml** playbook to make the **/etc/ansible/facts.d** remote directory and to save the **custom.fact** file to that directory.

```
---
- name: Install remote facts
  hosts: webserver
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: Create the remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: Install the new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
```

6. Run an ad hoc command with the **setup** module. Since user-defined facts are put into the **ansible_local** section, use a filter to display only this section. There should not be any custom facts at this point.

```
[student@workstation dev-vars-facts]$ ansible webserver -m setup -a
 'filter=ansible_local'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {},
    "changed": false
}
```

7. Run the **setup_facts.yml** playbook.

```
[student@workstation dev-vars-facts]$ ansible-playbook setup_facts.yml

PLAY [Install remote facts] ****************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Create the remote directory] **********************************************
changed: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [Install the new facts] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

8. To ensure the new facts have been properly installed, run an ad hoc command with the **setup** module again. Display only the **ansible_local** section. The custom facts should appear.

```
[student@workstation dev-vars-facts]$ ansible webserver -m setup -a
 'filter=ansible_local'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "custom": {
                "general": {
                    "package": "httpd",
                    "service": "httpd",
                    "state": "started"
                }
            }
        }
    },
    "changed": false
}
```

9. It is now possible to create the main playbook that uses both default and user facts to configure **servera**. Over the next several steps, you will add to the playbook file. Start the **playbook.yml** playbook file with the following:

```
---
- name: Install Apache and starts the service
  hosts: webserver
```

10. Continue editing the **playbook.yml** file by creating the first task that installs the *httpd* package. Use the user fact for the name of the package.

```
  tasks:
    - name: Install the required package
      yum:
        name: "{{ ansible_local.custom.general.package }}"
        state: latest
```

11. Create another task that uses the custom fact to start the **httpd** service.

```
    - name: Start the service
      service:
        name: "{{ ansible_local.custom.general.service }}"
        state: "{{ ansible_local.custom.general.state }}"
```

12. When completed with all the tasks, the full playbook should look like the following. Review the playbook and ensure all the tasks are defined.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
- name: Install Apache and starts the service
  hosts: webserver

  tasks:
    - name: Install the required package
      yum:
        name: "{{ ansible_local.custom.general.package }}"
        state: latest

    - name: Start the service
      service:
        name: "{{ ansible_local.custom.general.service }}"
        state: "{{ ansible_local.custom.general.state }}"
```

13. Before running the playbook, use an ad hoc command to verify the **httpd** service is not currently running on **servera**.

```
[student@workstation dev-vars-facts]$ ansible servera.lab.example.com -m command -a
 'systemctl status httpd'
servera.lab.example.com | FAILED | rc=3 >>
● httpd.service
   Loaded: not-found (Reason: No such file or directory)
   Active: inactive (dead)
... Output omitted ...
```

14. Run the playbook using the **ansible-playbook** command. Watch the output as Ansible starts by installing the package, then enabling the service.

```
[student@workstation dev-vars-facts]$ ansible-playbook playbook.yml

PLAY [Install Apache and start the service] ************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Install the required package] ********************************************
changed: [servera.lab.example.com]

TASK [Start the service] *******************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

15. Use an ad hoc command to execute **systemctl** to check if the **httpd** service is now running on **servera**.

```
[student@workstation dev-vars-facts]$ ansible servera.lab.example.com -m command -a
 'systemctl status httpd'
servera.lab.example.com | SUCCESS | rc=0 >>
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset:
 disabled)
   Active: active (running) since Mon 2016-05-16 17:17:20 PDT; 12s ago
     Docs: man:httpd(8)
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
           man:apachectl(8)
 Main PID: 32658 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:   0 B/sec"
   CGroup: /system.slice/httpd.service
... Output omitted ...
```

Evaluation

From *workstation*, run the **lab manage-variables-facts** command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab manage-variables-facts grade
```

Cleanup

Run the **lab manage-variables-facts** command to clean up the lab.

```
[student@workstation ~]$ lab manage-variables-facts cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Inclusions

After completing this section, students should be able to:

- Include variables and tasks from external files into a playbook

Inclusions

When working with complex or long playbooks, administrators can use separate files to divide tasks and lists of variables into smaller pieces for easier management. There are multiple ways to include task files and variables in a playbook.

- Tasks can be included in a playbook from an external file by using the **`include`** directive.

```
tasks:
  - name: Include tasks to install the database server
    include: tasks/db_server.yml
```

- The **`include_vars`** module can include variables defined in either JSON or YAML files, overriding host variables and playbook variables already defined.

```
 tasks:
  - name: Include the variables from a YAML or JSON file
    include_vars: vars/variables.yml
```

Using multiple, external files for tasks and variables is a convenient way to build the main playbook in a modular way, and facilitates reuse of Ansible elements across multiple playbooks.

Including tasks

Consider the following examples where it might be useful to manage a set of tasks as a file separate from the playbook:

- If fresh servers require complete configuration, administrators could create various sets of tasks for creating users, installing packages, configuring services, configuring privileges, setting up access to a shared file system, hardening the servers, installing security updates, and installing a monitoring agent. Each of these sets of tasks could be managed through a separate self-contained task file.
- If servers are managed collectively by the developers, the system administrators and the database administrators, every organization can write its own set of tasks that will be reviewed and integrated by the systems manager.
- If a server requires a particular configuration, it can be integrated as a set of tasks executed based on a conditional (that is, including the tasks only if specific criteria are met).
- If a group of servers need to run a particular task or set of tasks, the tasks might only be run on a server if it is part of a specific host group.

The **`include`** directive allows administrators to have a *task file* inserted at a particular point in a playbook. A task file is simply a file that contains a flat list of tasks:

```
- name: Installs the {{ package }} package
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  yum:
    name: "{{ package }}"
    state: latest

- name: Starts the {{ service }} service
  service:
    name: "{{ service }}"
    state:  "{{ state }}"
```

The **include** directive is used in the playbook to specify what task file to include at what point in the playbook. A **vars** directive can be used to set variables used by the task file, and will override playbook variables, inventory variables, registered variables, and facts defined in the task file.

Assume the task file in the previous example was saved as the file **environment.yml**. An **include** directive with variables could be used in a playbook, like this:

```
---
- name: Install, start, and enable services
  hosts: all
  tasks:
  - name: Includes the tasks file and defines the variables
    include: environment.yml
    vars:
      package: mariadb-server
      service: mariadb
      state: started
    register: output

  - name: Debugs the included tasks
    debug:
      var: output
```

The following output shows how the tasks have been included and executed. In this case, the *mariadb-server* package has been installed because the variable **package** with a value of **mariadb-server** has been set by the **include** directive.

```
[user@demo ~]$ ansible-playbook.yml
PLAY [Install, start, and enable services] ********************************

TASK [setup] *******************************************************************
ok: [demo1.example.com]

TASK [Includes the tasks file and defines the variables] ******************
included: /home/student/demo-vars-inclusions/environment.yml for
          demo1.example.com

TASK [Installs the mariadb-server package] ********************************
ok: [demo1.example.com]

TASK [Starts the mariadb service] *****************************************
changed: [demo1.example.com]

TASK [Debugs the included tasks] ******************************************
ok: [demo1.example.com] => {
    "output": {
        "changed": false,
        "include": "environment.yml",
        "include_variables": {}
    }
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
}

PLAY RECAP *********************************************************************
demo1.example.com    : ok=5    changed=1    unreachable=0    failed=0
```



Important

Tasks included in a playbook are evaluated based on their order in the playbook.

For example, a playbook may have a task to start a service provided by a software package, and a second task included from an external file that installs that software package. The external task must be included in the playbook before the task to start the package's service is declared in the playbook.

Administrators can create a dedicated directory for task files, and save all task files in that directory. Then the playbook simply includes task files from that directory in the playbook. This allows construction of a complex playbook while making it easier to manage its structure and components.



Note

For complex projects, *roles* provide a powerful way to organize included task files and playbooks. This topic will be discussed later in the course.

Including variables

Just as tasks can be included in playbooks, variables can be externally defined and included in playbooks. There are many ways in which this can be done. Some of the ways to set variables include:

- Inventory variables defined in the inventory file or in external files in **`host_vars`** and **`group_vars`** directories
- Facts and registered variables
- Playbook variables defined in the playbook file with **`vars`** or in an external file through **`vars_files`**

The **`include_vars`** module is one more way to set variables in a playbook from an external file. The interesting thing about this method is that it is done through a module, and it overrides any values set through the above methods. This can be useful when combining task files that set values for their variables which you want to override, or in conjunction with conditional execution to set certain values only when specific conditions are met.

Included variable files may be defined using either JSON or YAML, YAML being the preferred syntax. The **`include_vars`** module takes the path of the variable file to import as an argument.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Important

If a task will requires the variable settings from a variable file, the administrator must include the variable file in the playbook before the task is defined.

Consider the following snippets that demonstrate how to define variables and include them in a playbook with **`include_vars`**. The following YAML file, **`variables.yml`**, contains two variables that define the packages to install:

```
---
packages:
  web_package: httpd
  db_package: mariadb-server
```

To import these two variables in a playbook, the **`include_vars`** module can be used:

```
---
- name: Install web application packages
  hosts: all
  tasks:
  - name: Includes the tasks file and defines the variables
    include_vars: variables.yml

  - name: Debugs the variables imported
    debug:
      msg: >
        "{{ packages['web_package'] }} and {{ packages.db_package }}
        have been imported"
```

The following output shows that the variables have been successfully imported.

```
[user@prompt ~]$ ansible-playbook playbook.yml
PLAY [Install web application packages] ***********************************

TASK [setup] *******************************************************************
ok: [demo.example.com]

TASK [Includes the tasks file and defines the variables] ******************
ok: [demo.example.com]

TASK [Debugs the variables imported] ***************************************
ok: [demo.example.com] => {
    "msg": "httpd and mariadb-server have been imported"
}

PLAY RECAP *********************************************************************
demo.example.com    : ok=3    changed=0    unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Important

When deciding where to define variables, try to keep things simple.

In most cases, it is not necessary to use **include_vars**, **host_vars** and **group_vars** files, playbook **vars_files** directives, inlined variables in playbook and inventory files, and command-line overrides all at the same time. In fact, that much complexity could make it hard to easily maintain your Ansible project.

1. From **workstation**, as the **student** user, change into the **~/demo-vars-inclusions** directory.

```
[student@workstation ~]$ cd demo-vars-inclusions
[student@workstation demo-vars-inclusions]$
```

2. Define the **paths.yml** variables file and create a dictionary that sets some system paths. Specify the **fileserver** base path as **/home/student/srv/filer/**, with the **ansible_fqdn** variable as the subdirectory. Specify the **dbpath** base path as **/home/student/srv/database/**, with the **ansible_fqdn** variable as the subdirectory.

```
---
paths:
  fileserver: /home/student/srv/filer/{{ ansible_fqdn }}
  dbpath: /home/student/srv/database/{{ ansible_fqdn }}
```

3. Create the **fileservers.yml** playbook and include the **paths.yml** variables file. The **fileserver** will be created using the variable defined previously in the **paths.yml** variables file.

```
---
- name: Configure fileservers
  hosts: fileservers
  tasks:
    - name: Imports the variables file
      include_vars: paths.yml

    - name: Creates the remote directory
      file:
        path: "{{ paths.fileserver }}"
        state: directory
        mode: 0755
      register: result

    - name: Debugs the results
      debug:
        var: result
```

4. Run the **fileservers.yml** playbook and examine the output.

```
[student@workstation demo-vars-inclusions]$ ansible-playbook fileservers.yml
PLAY [Configure fileservers] ***************************************************

TASK [setup] *******************************************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ok: [servera.lab.example.com]

TASK [Imports the variables file] ***************************************
ok: [servera.lab.example.com]

TASK [Creates the remote directory] *************************************
changed: [servera.lab.example.com]

TASK [Debugs the results] ***********************************************
ok: [servera.lab.example.com] => {
    "result": {
        "changed": true,
        "gid": 0,
        "group": "root",
        "mode": "0755",
        "owner": "root",
        "path": "/home/student/srv/filer/servera.lab.example.com",
        "secontext": "unconfined_u:object_r:user_home_t:s0",
        "size": 6,
        "state": "directory",
        "uid": 0
    }
}

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=1    unreachable=0    failed=0
```

The output shows the directory structure that has been created by Ansible, which matches the path that has been set by the **paths.fileserver** variable.

5. Create the **dbservers.yml** playbook. Include the variable file and use the **paths.dbpath** variable to create the directory structure.

```
---
- name: Configure DB Servers
  hosts: dbservers
  tasks:
    - name: Imports the variables file
      include_vars: paths.yml

    - name: Creates the remote directory
      file:
        path: "{{ paths.dbpath }}"
        state: directory
        mode: 0755
      register: result

    - name: Debugs the results
      debug:
        var: result
```

6. Run the **dbservers.yml** playbook and examine the output.

```
[student@workstation demo-vars-inclusions]$ ansible-playbook dbservers.yml
PLAY [Configure DB Servers] ***************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Imports the variables file] ***************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ok: [servera.lab.example.com]

TASK [Creates the remote directory] ************************************
changed: [servera.lab.example.com]

TASK [Debugs the results] **********************************************
ok: [servera.lab.example.com] => {
    "result": {
        "changed": true,
        "gid": 0,
        "group": "root",
        "mode": "0755",
        "owner": "root",
        "path": "/home/student/srv/database/servera.lab.example.com",
        "secontext": "unconfined_u:object_r:user_home_t:s0",
        "size": 6,
        "state": "directory",
        "uid": 0
    }
}

PLAY RECAP *************************************************************
servera.lab.example.com    : ok=4    changed=1    unreachable=0    failed=0
```

7. Create another variable file, **`package.yml`**, and define the name of the package to install.

```
---
packages:
  web_pkg: httpd
```

8. Create a task file, called **`install_package.yml`**, and create a basic task that installs a package.

```
---
- name: Installs {{ packages.web_pkg }}
  yum:
    name: "{{ packages.web_pkg }}"
    state: latest
```

9. Create the **`playbook.yml`** playbook. Define it for the hosts in the **`fileservers`** group. Include both variable files as well as the task file.

```
---
- name: Install fileserver packages
  hosts: fileservers
  tasks:
    - name: Includes the variable
      include_vars: package.yml

    - name: Installs the package
      include: install_package.yml
```

10. Run the playbook and watch the output.

```
[student@workstation demo-vars-inclusions]$ ansible-playbook playbook.yml
PLAY [Install fileserver packages] *************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Includes the variable] ***************************************************
ok: [servera.lab.example.com]

TASK [Installs the package] ****************************************************
included: /home/student/demo-vars-inclusions/install_package.yml
          for servera.lab.example.com

TASK [Installs httpd] **********************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=1    unreachable=0    failed=0
```

Note the *httpd* package has been installed from a task, whereas the name of the package to install is defined in the variables file.

11. Update the **playbook.yml** playbook to override the name of the package to install. Append a **vars** block to the **include** statement and define a dictionary to override the name of the package to install.

```
---
- name: Install fileserver packages
  hosts: fileservers
  tasks:
    - name: Includes the variable
      include_vars: package.yml

    - name: Installs the package
      include: install_package.yml
      vars:
        packages:
          web_pkg: tomcat
```

12. Run the playbook and watch the output as Ansible installs the *tomcat* package.

```
[student@workstation demo-vars-inclusions]$ ansible-playbook playbook.yml
PLAY [Install fileserver packages] *********************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Includes the variable] ***************************************************
ok: [servera.lab.example.com]

TASK [Installs the package] ****************************************************
included: /home/student/demo-vars-inclusions/install_package.yml
          for servera.lab.example.com

TASK [Installs tomcat] *********************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=1    unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Playbook Roles and Include Statements — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_roles.html

include_vars - Load variables from files, dynamically within a task — Ansible Documentation
http://docs.ansible.com/ansible/include_vars_module.html

file - Sets attributes of files — Ansible Documentation
http://docs.ansible.com/ansible/file_module.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Inclusions

In this exercise, you will manage inclusions in Ansible playbooks.

Outcomes
You should be able to:

- Define variables and tasks in separate files.
- Include variable files and task files in playbooks.

Before you begin
From **workstation**, run the lab setup script to confirm the environment is ready for the lab to begin. The script creates the working directory **dev-vars-inclusions**.

```
[student@workstation ~]$ lab manage-variables-inclusions setup
```

Steps

1. From **workstation**, as the **student** user, change into the directory **~/dev-vars-inclusions**.

```
[student@workstation ~]$ cd dev-vars-inclusions
[student@workstation dev-vars-inclusions]$
```

2. One task file, one variable file, and one playbook will be created for this exercise. The variable file defines, in YAML format, a variable used by the playbook. The task file defines the required tasks and includes variables that will be passed later on as arguments.

   2.1. Create a directory called **tasks** and change into that directory.

```
[student@workstation dev-vars-inclusions]$ mkdir tasks && cd tasks
[student@workstation tasks]$
```

   2.2. In the **tasks** directory, create the **environment.yml** task file. Define the two tasks that install and start the web server; use the **package** variable for the package name, **service** for the service name, and **svc_state** for the service state.

```
---
  - name: Install the {{ package }} package
    yum:
      name: "{{ package }}"
      state: latest
  - name: Start the {{ service }} service
    service:
      name: "{{ service }}"
      state: "{{ svc_state }}"
```

   2.3. Change back into the main project directory. Create a directory named **vars** and change into that directory.

```
[student@workstation tasks]$ cd ..
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dev-vars-inclusions]$ mkdir vars
[student@workstation dev-vars-inclusions]$ cd vars
[student@workstation vars]$
```

2.4. In the **vars** directory, create the **variables.yml** variables file. The file defines the **firewall_pkg** variable in YAML format. The file should read as follows:

```
---
firewall_pkg: firewalld
```

2.5. Change back to the top-level project directory for the playbook.

```
[student@workstation vars]$ cd ..
[student@workstation dev-vars-inclusions]$
```

3. Create and edit the main playbook, named **playbook.yml**. The playbook imports the tasks as well as the variables; and it installs the **firewalld** service and configures it.

3.1. Start by adding the **webserver** host group. Define a **rule** variable with a value of **http**.

```
---
- hosts: webserver
  vars:
    rule: http
```

3.2. Continue editing the **playbook.yml** file. Define the first task, which uses the **include_vars** module to import extra variables in the playbook. The variables are used by other tasks in the playbook. Include the **variables.yml** variable file created previously.

```
  tasks:
    - name: Include the variables from the YAML file
      include_vars: vars/variables.yml
```

3.3. Define the second task which uses the **include** module to include the base **environment.yml** playbook. Because the three defined variables are used in the base playbook, but are not defined, include a **vars** block. Set three variables in the **vars** section: **package** set as **httpd**, **service** set as **httpd**, and **svc_state** set as **started**.

```
    - name: Include the environment file and set the variables
      include: tasks/environment.yml
      vars:
        package: httpd
        service: httpd
        svc_state: started
```

3.4. Create three more tasks: one that installs the *firewalld* package, one that starts the **firewalld** service, and one that adds a rule for the HTTP service. Use the variables that were defined previously.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Create a task that installs the *firewalld* package using the **firewall_pkg** variable.

```
    - name: Install the firewall
      yum:
        name: "{{ firewall_pkg }}"
        state: latest
```

Create a task that starts the **firewalld** service.

```
    - name: Start the firewall
      service:
        name: firewalld
        state: started
        enabled: true
```

Create a task that adds a firewall rule for the HTTP service using the **rule** variable.

```
    - name: Open the port for {{ rule }}
      firewalld:
        service: "{{ rule }}"
        immediate: true
        permanent: true
        state: enabled
```

3.5. Finally, add a task that creates the **index.html** file for the web server using the **copy** module. Create the file with the Ansible **ansible_fqdn** fact, which returns the fully qualified domain name. Also include a time stamp in the file using an Ansible fact. The task should read as follows:

```
    - name: Create index.html
      copy:
        content: "{{ ansible_fqdn }} has been customized using Ansible on the
 {{ ansible_date_time.date }}\n"
        dest: /var/www/html/index.html
```

3.6. When you have completed the main **playbook.yml** playbook, it should appear as follows:

```
---
- hosts: webserver
  vars:
    rule: http
  tasks:
    - name: Include the variables from the YAML file
      include_vars: vars/variables.yml

    - name: Include the environment file and set the variables
      include: tasks/environment.yml
      vars:
        package: httpd
        service: httpd
        svc_state: started

    - name: Install the firewall
      yum:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        name: "{{ firewall_pkg }}"
        state: latest

    - name: Start the firewall
      service:
        name: firewalld
        state: started
        enabled: true

    - name: Open the port for {{ rule }}
      firewalld:
        service: "{{ rule }}"
        immediate: true
        permanent: true
        state: enabled

    - name: Create index.html
      copy:
        content: "{{ ansible_fqdn }} has been customized using Ansible on the
  {{ ansible_date_time.date }}\n"
        dest: /var/www/html/index.html
```

4. Run the playbook using the **`ansible-playbook`** command. Watch the output as Ansible starts by including the **`environment.yml`** playbook and running its tasks, then keeps executing the tasks defined in the main playbook.

```
[student@workstation dev-vars-inclusions]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Include the variables from the YAML file] ********************************
ok: [servera.lab.example.com]

TASK [Include the environment file and set the variables] **********************
included: /home/student/dev-vars-inclusions/tasks/environment.yml for
 servera.lab.example.com

TASK [Install and start the web server] ***********************************
changed: [servera.lab.example.com]

TASK [Start the service] *******************************************************
changed: [servera.lab.example.com]

TASK [Install the firewall] ****************************************************
changed: [servera.lab.example.com]

TASK [Start the firewall] ******************************************************
changed: [servera.lab.example.com]

TASK [Open the port for http] **************************************************
changed: [servera.lab.example.com]

TASK [Create index.html] *******************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=9     changed=4    unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5. Use **curl** to ensure the web server is reachable from **workstation**. Because the **index.html** has been created, the output should appear as follows:

```
[student@workstation dev-vars-inclusions]$ curl http://servera.lab.example.com
servera.lab.example.com has been customized using Ansible on the 2016-03-31
```

Evaluation

Run the **lab manage-variables-inclusions** command from **workstation** to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab manage-variables-inclusions grade
```

Cleanup

Run the **lab manage-variables-inclusions cleanup** command to clean up after the lab.

```
[student@workstation ~]$ lab manage-variables-inclusions cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Managing Variables and Inclusions

In this lab, you will deploy a database as well as a web server. For this lab, various Ansible modules, variables, and tasks will be defined.

Outcomes

You should be able to:

- Define variables in a playbook.
- Create custom facts for a managed host and use these facts in the main playbook.
- Create a separate set of tasks to include in the main playbook.
- Define a variable in a variable file and include the variable file in the main playbook.

Before you begin

From **workstation.lab.example.com**, open a new terminal and run the setup script to prepare the environment. The script creates the project directory, called **lab-managing-vars**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab manage-variables setup
```

Steps

1. Create all of the files related to this lab in, or below, the **lab-managing-vars** project directory.

2. **Defining custom facts**

   Create a facts file in INI format called **custom.fact**. Create a section called **packages** and define two facts: one called **db_package**, with a value of **mariadb-server**, and one called **web_package**, with a value of **httpd**. Create a section called **services** with two facts: one called **db_service**, with a value of **mariadb**, and one called **web_service**, with a value of **httpd**.

   Define a playbook, called **setup_facts.yml**, that will install the facts on **serverb**.

3. **Installing facts**

   Run the playbook to install the custom facts and verify the facts are available as Ansible facts.

4. **Defining variables**

   Create a directory for variables, called **vars**. Define a YAML variable file, called **main.yml**, in that directory that defines a new variable, called **web_root**, with a value of **/var/www/html**.

5. **Defining tasks**

   From the project directory, **lab-managing-vars**, create a directory for tasks, called **tasks**. Define a task file in the subdirectory, called **main.yml**, that instructs Ansible to

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

install both the web server package and the database package using facts Ansible gathered from **serverb.lab.example.com**. When they are installed, start the two services.

6. **Defining the main playbook**

   Create the main playbook **playbook.yml** in the top-level directory for this lab. The playbook should be in the following order: target the **lamp** hosts groups and define a new variable, **firewall** with a value of **firewalld**.

   Create the following tasks:
   - A task that includes the variable file **main.yml**.
   - A task that includes the tasks defined in the tasks file.
   - A task for installing the latest version of the *firewall* package.
   - A task for for starting the **firewall** service.
   - A task for opening TCP port 80 permanently.
   - A task that uses the **copy** module to create the **index.html** page in the directory the variable defines.

   The **index.html** should appear as follows:

   ```
   serverb.lab.example.com (172.25.250.11) has been customized by Ansible
   ```

   Both the host name and the IP address should use Ansible facts.

7. **Running the playbook**

   Run the playbook **playbook.yml** created in the previous step.

8. **Testing the deployment**

   From **workstation**, use **curl** to ensure the web server is reachable. Also use an ad hoc command to connect to **serverb** as the **devops** user and ensure the **mariadb** service is running.

Evaluation
Run the **lab manage-variables grade** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab manage-variables grade
```

Cleanup
Run the **lab manage-variables cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab manage-variables cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will deploy a database as well as a web server. For this lab, various Ansible modules, variables, and tasks will be defined.

Outcomes
You should be able to:

- Define variables in a playbook.
- Create custom facts for a managed host and use these facts in the main playbook.
- Create a separate set of tasks to include in the main playbook.
- Define a variable in a variable file and include the variable file in the main playbook.

Before you begin
From **workstation.lab.example.com**, open a new terminal and run the setup script to prepare the environment. The script creates the project directory, called **lab-managing-vars**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab manage-variables setup
```

Steps

1. Create all of the files related to this lab in, or below, the **lab-managing-vars** project directory.

   ```
   [student@workstation ~]$ cd lab-managing-vars
   [student@workstation lab-managing-vars]$
   ```

2. **Defining custom facts**

   Create a facts file in INI format called **custom.fact**. Create a section called **packages** and define two facts: one called **db_package**, with a value of **mariadb-server**, and one called **web_package**, with a value of **httpd**. Create a section called **services** with two facts: one called **db_service**, with a value of **mariadb**, and one called **web_service**, with a value of **httpd**.

   Define a playbook, called **setup_facts.yml**, that will install the facts on **serverb**.

   2.1. Create the fact file **custom.fact**. The file should appear as follows:

   ```
   [packages]
   db_package = mariadb-server
   web_package = httpd

   [services]
   db_service = mariadb
   web_service = httpd
   ```

   2.2. From the project directory, create the **setup_facts.yml** playbook to install the facts on the managed host, **serverb.lab.example.com**. Use the **file** and **copy** modules to install the custom facts. The playbook should appear as follows:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
- name: Install remote facts
  hosts: lamp
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: Create the remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: Install the new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
```

3. **Installing facts**

Run the playbook to install the custom facts and verify the facts are available as Ansible facts.

3.1. Run the playbook.

```
[student@workstation lab-managing-vars]$ ansible-playbook setup_facts.yml
PLAY [Install remote facts] ****************************************************

TASK [setup] *******************************************************************
changed: [serverb.lab.example.com]

TASK [Create the remote directory] **********************************************
ok: [serverb.lab.example.com]

TASK [Install the new facts] ***************************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com   : ok=3    changed=2    unreachable=0    failed=0
```

3.2. Ensure the newly created facts can be retrieved.

```
[student@workstation lab-managing-vars]$ ansible lamp -m setup -a
 'filter=ansible_local*'
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "services": {
                    "db_service": "mariadb",
                    "web_service": "httpd"
                }
            }
        }
    },
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    "changed": false
}
```

4. **Defining variables**

   Create a directory for variables, called **vars**. Define a YAML variable file, called **main.yml**, in that directory that defines a new variable, called **web_root**, with a value of **/var/www/html**.

   4.1. Create the variables directory, **vars**, inside the project directory.

   ```
   [student@workstation lab-managing-vars]$ mkdir vars
   ```

   4.2. Create the variables file **vars/main.yml**. The file should contain the following content:

   ```
   ---
   web_root: /var/www/html
   ```

5. **Defining tasks**

   From the project directory, **lab-managing-vars**, create a directory for tasks, called **tasks**. Define a task file in the subdirectory, called **main.yml**, that instructs Ansible to install both the web server package and the database package using facts Ansible gathered from **serverb.lab.example.com**. When they are installed, start the two services.

   5.1. Create the **tasks** directory inside the project directory.

   ```
   [student@workstation lab-managing-vars]$ mkdir tasks
   ```

   5.2. Create the tasks file, **tasks/main.yml**. The tasks should install both the database and the web server, and start the services **httpd** and **mariadb**. Use the custom Ansible facts for the name of the services to manage. The file should appear as follows:

   ```
   ---
     - name: Install and start the database and web servers
       yum:
         name:
           - "{{ ansible_local.custom.packages.db_package }}"
           - "{{ ansible_local.custom.packages.web_package }}"
         state: latest

     - name: Start the database service
       service:
         name: "{{ ansible_local.custom.services.db_service }}"
         state: started
         enabled: true

     - name: Start the web service
       service:
         name: "{{ ansible_local.custom.services.web_service }}"
         state: started
         enabled: true
   ```

6. **Defining the main playbook**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Create the main playbook **playbook.yml** in the top-level directory for this lab. The playbook should be in the following order: target the **lamp** hosts groups and define a new variable, **firewall** with a value of **firewalld**.

Create the following tasks:

- A task that includes the variable file **main.yml**.
- A task that includes the tasks defined in the tasks file.
- A task for installing the latest version of the *firewall* package.
- A task for for starting the **firewall** service.
- A task for opening TCP port 80 permanently.
- A task that uses the **copy** module to create the **index.html** page in the directory the variable defines.

The **index.html** should appear as follows:

```
serverb.lab.example.com (172.25.250.11) has been customized by Ansible
```

Both the host name and the IP address should use Ansible facts.

6.1. The following steps edit the single **playbook.yml** file.

Create the main playbook, **playbook.yml**, for the hosts in the **lamp** hosts group, and define the **firewall** variable.

```
---
- hosts: lamp
  vars:
    firewall: firewalld
```

6.2. Add the **tasks** block and define the first task that includes the variables file located under **vars/main.yml**.

```
  tasks:
    - name: Include the variable file
      include_vars: vars/main.yml
```

6.3. Create the second task, which imports the tasks file located under **tasks/main.yml**.

```
    - name: Include the tasks
      include: tasks/main.yml
```

6.4. Create the tasks that install the firewall, start the service, open port 80, and reload the service.

```
    - name: Install the firewall
      yum:
        name: "{{ firewall }}"
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        state: latest

    - name: Start the firewall
      service:
        name: "{{ firewall }}"
        state: started
        enabled: true

    - name: Open the port for the web server
      firewalld:
        service: http
        state: enabled
        immediate: true
        permanent: true
```

6.5. Finally, create the task that uses the **copy** module to create a custom main page, **index.html**. Use the variable **web_root**, defined in the variables file, for the home directory of the web server.

```
    - name: Create index.html
      copy:
        content: "{{ ansible_fqdn }}({{ ansible_default_ipv4.address }}) has
 been customized by Ansible\n"
        dest: "{{ web_root }}/index.html"
```

6.6. When complete, the tree should appear as follows:

```
[student@workstation lab-managing-vars]$ tree
.
├── ansible.cfg
├── custom.fact
├── inventory
├── playbook.yml
├── setup_facts.yml
├── tasks
│   └── main.yml
└── vars
    └── main.yml

2 directories, 7 files
```

6.7. The main playbook should appear as follows:

```
---
- hosts: lamp
  vars:
    firewall: firewalld

  tasks:
    - name: Include the variable file
      include_vars: vars/main.yml

    - name: Include the tasks
      include: tasks/main.yml

    - name: Install the firewall
      yum:
        name: "{{ firewall }}"
        state: latest
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    - name: Start the firewall
      service:
        name: "{{ firewall }}"
        state: started
        enabled: true

    - name: Open the port for the web server
      firewalld:
        service: http
        state: enabled
        immediate: true
        permanent: true

    - name: Create index.html
      copy:
        content: "{{ ansible_fqdn }}({{ ansible_default_ipv4.address }}) has
 been customized by Ansible\n"
        dest: "{{ web_root }}/index.html"
```

7. **Running the playbook**

   Run the playbook **`playbook.yml`** created in the previous step.

   7.1. Using the **`ansible-playbook`** command, run the playbook.

```
[student@workstation lab-managing-vars]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

... Output omitted ...

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=10    changed=5    unreachable=0    failed=0
```

8. **Testing the deployment**

   From **`workstation`**, use **`curl`** to ensure the web server is reachable. Also use an ad hoc command to connect to **`serverb`** as the **`devops`** user and ensure the **`mariadb`** service is running.

   8.1. From **`workstation`**, use **`curl`** to ensure the web server has been successfully started and it is reachable. If the following message appears, it indicates that the web server has been installed, the firewall has been updated with a new rule and the included variable has been successfully used.

```
[student@workstation lab-managing-vars]$ curl http://serverb
serverb.lab.example.com(172.25.250.11) has been customized by Ansible
```

   8.2. Use an ad hoc Ansible command to ensure the **`mariadb`** service is running on **`serverb.lab.example.com`**.

```
[student@workstation lab-managing-vars]$ ansible lamp -a 'systemctl status
 mariadb'
serverb.lab.example.com | SUCCESS | rc=0 >>
● mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor
 preset: disabled)
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
   Active: active (running) since Fri 2016-04-01 10:50:40 PDT; 7min ago
... Output omitted ...
```

Evaluation

Run the **lab manage-variables grade** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab manage-variables grade
```

Cleanup

Run the **lab manage-variables cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab manage-variables cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible *variables* allow administrators to reuse values across files in an entire Ansible project
- Variables have names which consist of a string that must start with a letter and can only contain letters, numbers, and underscores
- Variables can be defined for hosts and host groups in the inventory, for playbooks, by facts and external files, and from the command line
- It is better to store inventory variables in files in the **`host_vars`** and **`group_vars`** directory relative to the inventory than in the inventory file itself
- Ansible *facts* are variables that are automatically discovered by Ansible from a managed host
- In a playbook, when a variable is used at the start of a value, quotes are mandatory
- The **`register`** keyword can be used to capture the output of a command in a variable.
- Both **`include`** and **`include_vars`** modules can be used to include tasks or variable files in YAML or JSON format in playbooks.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 5

IMPLEMENTING TASK CONTROL

Overview	
Goal	Manage task control, handlers, and tags in Ansible playbooks
Objectives	• Construct conditionals and loops in a playbook • Implement handlers in a playbook • Implement tags in a playbook • Resolve errors in a playbook
Sections	• Constructing Flow Control (and Guided Exercise) • Implementing Handlers (and Guided Exercise) • Implementing Tags (and Guided Exercise) • Handling Errors (and Guided Exercise)
Lab	• Implementing Task Control

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Constructing Flow Control

Objectives

After completing this section, students should be able to:

- Implement loops in playbooks
- Construct conditionals in playbooks
- Combine loops and conditionals

Ansible loops

Ansible supports many loop formats in order to iterate over a set of values defined in an array. Using loops saves administrators from writing repetitive tasks that use the same module; for example, if there are multiple users to create or multiple packages to install, an array can be defined and a task created that iterates over the array instead of using the **yum** module twice. To pass a loop as an argument, the **item** keyword needs to be used for Ansible to parse the array. The following loops are supported by Ansible:

- **Simple loops**: A simple loop is a list of items that Ansible reads and iterate over. They are defined by providing a list of items to the **with_items** keyword. Consider the following snippet that uses the **yum** module twice in order to install two packages:

```
- yum:
    name: postfix
    state: latest

- yum:
    name: dovecot
    state: latest
```

These two similar tasks using the **yum** module can be rewritten with a simple loop so that only one task is needed to install both packages:

```
- yum:
    name: "{{ item }}"
    state: latest
  with_items:
    - postfix
    - dovecot
```

The previous code can be replaced by having the packages inside an array called with the **with_items** keyword; the following playbook shows how to pass the array **mail_services** as an argument; the module will loop over the array to retrieve the name of the packages to install:

```
vars:
  mail_services:
    - postfix
    - dovecot

tasks:
  - yum:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      name: "{{ item }}"
      state: latest
    with_items: "{{ mail_services }}"
```

- **List of hashes**: When passing arrays as arguments, the array can be a list of hashes. The following snippet shows how a multidimensional array (an array with key-pair values) is passed to the **`user`** module in order to customize both the name and the group:

```
- user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  with_items:
    - { name: 'jane', groups: 'wheel' }
    - { name: 'joe', groups: 'root' }
```

- **Nested loops**: Administrators can also use nested loops, which are loops inside of loops called with the **`with_nested`** keyword. When nested loops are used, Ansible iterates over the first array as long as there are values in it. For example, when multiple MySQL privileges are required for multiple users, administrators can create a multidimensional array and invoke it with the **`with_nested`** keyword. The following snippet shows how to use the **`mysql_user`** module in a nested loop to grant two users a set of three privileges:

```
- mysql_user:
    name: "{{ item[0] }}"
    priv: "{{ item[1] }}.*:ALL"
    append_privs: yes
    password: redhat

  with_nested:
    - [ 'joe', 'jane' ]
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

  In the previous example, the name of the users to create can also be an array defined in a variable. The following snippet shows how users are defined inside an array and passed as the first item of the array:

```
vars:
  users:
    - joe
    - jane

tasks:
  - mysql_user:
      name: "{{ item[0] }}"
      priv: "{{ item[1] }}.*:ALL"
      append_privs: yes
      password: redhat

    with_nested:
      - "{{ users }}"
      - [ 'clientdb', 'employeedb', 'providerdb' ]
```

The following table shows some additional types of loops supported by Ansible.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Ansible Loops

Loop Keyword	Description
with_file	Takes a list of control node file names. **item** is set to the content of each file in sequence.
with_fileglob	Takes a file name globbing pattern. **item** is set to each file in a directory on the control node that matches that pattern, in sequence, non-recursively.
with_sequence	Generates a sequence of items in increasing numerical order. Can take **start** and **end** arguments which have a decimal, octal, or hexadecimal integer value.
with_random_choices	Takes a list. **item** is set to one of the list items at random.

To capture the output of a module that uses an array, the **register** keyword can be used with an array. Ansible will save the output in the variable. This can be useful if administrators want to review the result of the execution of a module. The following snippet shows how to register the content of the array after the iteration:

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
```

The **echo** will then contain the following information:

```
{
    "changed": true,
    "msg": "All items completed",
    "results": [
        {
            "changed": true,
            "cmd": "echo \"one\" ",
            "delta": "0:00:00.003110",
            "end": "2013-12-19 12:00:05.187153",
            "invocation": {
                "module_args": "echo \"one\"",
                "module_name": "shell"
            },
            "item": "one",
            "rc": 0,
            "start": "2013-12-19 12:00:05.184043",
            "stderr": "",
            "stdout": "one"
        },
        {
            "changed": true,
            "cmd": "echo \"two\" ",
            "delta": "0:00:00.002920",
            "end": "2013-12-19 12:00:05.245502",
            "invocation": {
                "module_args": "echo \"two\"",
                "module_name": "shell"
            },
            "item": "two",
            "rc": 0,
            "start": "2013-12-19 12:00:05.242582",
            "stderr": "",
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
            "stdout": "two"
        }
    ]
}
```

Conditionals

Ansible can use *conditionals* to execute tasks or plays when certain conditions are met. For example, a conditional can be used to determine the available memory on a managed host before Ansible installs or configures a service.

Conditionals allow administrators to differentiate between managed hosts and assign them functional roles based on the conditions that they meet. Playbook variables, registered variables, and Ansible facts can all be tested with conditionals. Operators such as string comparison, mathematical operators, and Booleans are available.

The following examples illustrate some ways in which conditionals can be used by Ansible.

- A hard limit can be defined in a variable (for example, **min_memory**) and compare it against the available memory on a managed host.
- The output of a command can be captured and evaluated by Ansible to determine whether or not a task completed before taking further action. For example, if a program fails, then a batch will need to be skipped.
- Ansible facts can be used to determine the managed host network configuration and decide which template file to send (for example, network bonding or trunking).
- The number of CPUs can be evaluated to determine how to properly tune a web server.
- Registered variables can be compared with defined variables to check a service change. For example, this can be used to verify the MD5 checksum of a file.

The following table shows some of the operators that administrators can use when working with conditionals:

Ansible conditionals operators

Operator	Example
Equal	**"{{ max_memory }} == 512"**
Less than	**"{{ min_memory }} < 128"**
Greater than	**"{{ min_memory }} > 256"**
Less than or equal to	**"{{ min_memory }} <= 256"**
Greater than or equal to	**"{{ min_memory }} >= 512"**
Not equal to	**"{{ min_memory }} != 512"**
Variable exists	**"{{ min_memory }} is defined"**
Variable does not exist	**"{{ min_memory }} is not defined"**
Variable is set to **1**, **True**, or **yes**	**"{{ available_memory }}"**
Variable is set to **0**, **False**, or **no**	**"not {{ available_memory }}"**
Value is present in a variable or an array	**"{{ users }} in users["db_admins"]"**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Important

The **==** operator used to test equality in conditions must not be confused with the **=** operator used for assigning a value to a variable.

For example,

- **`var="this is my variable"`** assigns the value **`this is my variable`** to the variable **`var`**
- **`{{ var }} == this is my variable`** tests the value of the variable **`var`** to see if it has been assigned the string **`this is my variable`**

Ansible **when** statement

To implement a conditional on an element, the **when** statement must be used, followed by the condition to test. When the statement is present, Ansible will evaluate it prior to executing the task. The following snippet shows a basic implementation of a **when** statement. Before creating the user **`db_admin`**, Ansible must ensure the managed host is part of the **`databases`** group:

```
- name: Create the database admin
  user:
    name: db_admin
  when: inventory_hostname in groups["databases"]
```



Important

Notice the placement of the **when** statement. Because the **when** statement is not a module variable, it must be placed "outside" the module by being indented at the top level of the task.

The **when** statement does not have to be at the top of the task. This breaks from the "top-down" ordering that is normal for Ansible.

Multiple conditions

One **when** statement can be used to evaluate multiple values. To do so, conditionals can be combined with the **and** and **or** keywords or grouped with parentheses.

The following snippets show some examples of how to express multiple conditions.

- With the **and** operation, both conditions have to be true for the entire conditional statement to be met. For example, the following condition will be met if the installed kernel is the specified version and if the **`inventory_hostname`** is in the **`staging`** group:

```
ansible_kernel == 3.10.0-327.el7.x86_64 and inventory_hostname in groups['staging']
```

- If a conditional statement should be met when either condition is true, then the **or** statement should be used. For example, the following condition is met if the machine is running either Red Hat Enterprise Linux or Fedora:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

- More complex conditional statements can be clearly expressed through grouping conditions with parentheses to ensure that they are correctly interpreted. For example, the following conditional statement is met if the machine is running either Red Hat Enterprise Linux 7 or Fedora 23:

```
(ansible_distribution == "RedHat" and ansible_distribution_major_version == 7) or
  (ansible_distribution == "Fedora" and ansible_distribution_major_version == 23)
```

Loops and conditionals can be combined. In the following example, the *mariadb-server* package will be installed by the **yum** module if there is a file system mounted on **/** with more than 300 MB free. The **ansible_mounts** fact is a list of dictionaries, each one representing facts about one mounted file system. The loop iterates over each dictionary in the list, and the conditional statement is not met unless a dictionary is found representing a mounted file system where both conditions are true.

```
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: latest
  with_items: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 300000000
```



Important

When combining **when** with **with_items**, be aware that the **when** statement is processed for each item.

Here is another example combining conditionals and registered variables. The following annotated playbook will restart the **httpd** service only if the **postfix** service is running.

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix (1)
      ignore_errors: yes (2)
      register: result (3)

    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0 (4)
```

1. Is Postfix running or not?
2. If it is not running and the command "fails", do not stop processing
3. Saves information on the module's result in a variable named **result**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4 Evaluates the output of the Postfix task. If the exit code of the **`systemctl`** command is 0, then Postfix is active and this task will restart the **`httpd`** service.

Using Booleans

Booleans are variables that take one of two possible values, which can be expressed as:

- **`True`** or **`Yes`** or **`1`**
- **`False`** or **`No`** or **`0`**

Booleans can be used as a simple switch to enable or disable tasks.

- To enable the task:

```
---
- hosts: all
  vars:
    my_service: True

  tasks:
    - name: Installs a package
      yum:
        name: httpd
      when: my_service
```

- To disable the task:

```
---
- hosts: all
  vars:
    my_service: False

  tasks:
    - name: Installs a package
      yum:
        name: httpd
      when: my_service
```



References

Loops — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_loops.html

Conditionals — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_conditionals.html

What Makes A Valid Variable Name — Variables — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_variables.html#what-makes-a-valid-variable-name

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Constructing Flow Control

In this exercise, you will construct conditionals and loops in Ansible playbooks

Outcomes
You should be able to:
- Implement Ansible conditionals using the **when** statement.
- Use Ansible **with_items** loops in conjunction with conditionals.

Before you begin
From **workstation**, run the lab setup script to confirm the environment is ready for the lab to begin. The script creates the working directory, called **dev-flowcontrol**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab task-control-flowcontrol setup
```

Steps
1. As the **student** user on **workstation**, change to the directory **/home/student/dev-flowcontrol**.

   ```
   [student@workstation ~]$ cd dev-flowcontrol
   [student@workstation dev-flowcontrol]$
   ```

2. Create a task file named **configure_database.yml**. This will define the tasks to install the extra packages, update **/etc/my.cnf** from a copy stored on a web site, and start **mariadb** on the managed hosts. The include file can and will use the variables you defined in the **playbook.yml** file and inventory. The **get_url** module will need to set **force=yes** so that the **my.cnf** file is updated even if it already exists on the managed host, and will need to set correct permissions as well as SELinux contexts on the **/etc/my.cnf** file. When you are finished, save the file and exit the editor. The file should read as follows:

   ```
   - yum:
       name: "{{ extra_packages }}"

   - get_url:
       url: "http://materials.example.com/task_control/my.cnf"
       dest: "{{ configure_database_path }}"
       owner: mysql
       group: mysql
       mode: 0644
       seuser: system_u
       setype: mysqld_etc_t
       force: yes

   - service:
       name: "{{ db_service }}"
       state: started
       enabled: true
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3. In the same directory, create the **playbook.yml** playbook. Define a list variable, **db_users**, that consists of a list of two users, **db_admin** and **db_user**. Add a **configure_database_path** variable set to the file **/etc/my.cnf**.

   Create a task that uses a loop to create the users only if the managed host belongs to the databases host group. The file should read as follows:

```
---
- hosts: all
  vars:
    db_package: mariadb-server
    db_service: mariadb
    db_users:
      - db_admin
      - db_user
    configure_database_path: /etc/my.cnf

  tasks:
  - name: Create the MariaDB users
    user:
      name: "{{ item }}"
    with_items: "{{ db_users }}"
    when: inventory_hostname in groups['databases']
```

4. In the playbook, add a task that uses the **db_package** variable to install the database software, only if the variable has been defined. The task should read as follows:

```
  - name: Install the database server
    yum:
      name: "{{ db_package }}"
    when: db_package is defined
```

5. In the playbook, create a task to do basic configuration of the database. The task will run only when **configure_database_path** is defined. This task should include the **configure_database.yml** task file and define a local array **extra_packages** which will be used to specify additional packages needed for this configuration. Set that list variable to include a list of three packages: *mariadb-bench*, *mariadb-libs*, and *mariadb-test*. When done, save the playbook and exit the editor.

```
  - name: Configure the database software
    include: configure_database.yml
    vars:
      extra_packages:
        - mariadb-bench
        - mariadb-libs
        - mariadb-test
    when: configure_database_path is defined
```

6. Check the final **playbook.yml** file before running it. It should now read in its entirety:

```
---
- hosts: all
  vars:
    db_package: mariadb-server
    db_service: mariadb
    db_users:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      - db_admin
      - db_user
    configure_database_path: /etc/my.cnf

  tasks:
  - name: Create the MariaDB users
    user:
      name: "{{ item }}"
    with_items: "{{ db_users }}"
    when: inventory_hostname in groups['databases']

  - name: Install the database server
    yum:
      name: "{{ db_package }}"
    when: db_package is defined

  - name: Configure the database software
    include: configure_database.yml
    vars:
      extra_packages:
        - mariadb-bench
        - mariadb-libs
        - mariadb-test
    when: configure_database_path is defined
```

7. Run the playbook to install and configure the database on the managed hosts.

```
[student@workstation dev-flowcontrol]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

... Output omitted ...

TASK [Includes the configuration] **********************************************
included: /home/student/dev-flowcontrol/configure_database.yml
          for servera.lab.example.com
```

The output confirms the task file has been successfully included and executed.

8. Manually verify that the necessary packages have been installed on **servera**, that the **/etc/my.cnf** file is in place with the correct permissions, and that the two users have been created.

8.1. Use an ad hoc command from **workstation** to **servera** to confirm the packages have been installed.

```
[student@workstation dev-flowcontrol]$ ansible all -a 'yum list installed
 mariadb-bench mariadb-libs mariadb-test'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
mariadb-bench.x86_64                1:5.5.44-2.el7                  @rhel_dvd
mariadb-libs.x86_64                 1:5.5.44-2.el7                  installed
mariadb-test.x86_64                 1:5.5.44-2.el7                  @rhel_dvd
```

8.2. Confirm the **my.cnf** file has been successfully copied under **/etc/**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dev-flowcontrol]$ ansible all -a 'grep Ansible /etc/my.cnf'
servera.lab.example.com | SUCCESS | rc=0 >>
# Ansible file
```

8.3. Confirm the two users have been created.

```
[student@workstation dev-flowcontrol]$ ansible all -a 'id db_user'
servera.lab.example.com | SUCCESS | rc=0 >>
uid=1003(db_user) gid=1003(db_user) groups=1003(db_user)
[student@workstation dev-flowcontrol]$ ansible all -a 'id db_admin'
servera.lab.example.com | SUCCESS | rc=0 >>
uid=1002(db_admin) gid=1002(db_admin) groups=1002(db_admin)
```

Evaluation

Run the **`lab task-control-flowcontrol grade`** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab task-control-flowcontrol grade
```

Cleanup

Run the **`lab task-control-flowcontrol cleanup`** command to cleanup after the lab.

```
[student@workstation ~]$ lab task-control-flowcontrol cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Implementing Handlers

Objectives

After completing this section, students should be able to:

- Implement handlers in playbooks

Ansible handlers

Ansible modules are designed to be *idempotent*. This means that in a properly written playbook, the playbook and its tasks can be run multiple times without changing the managed host, unless they need to make a change in order to get the managed host to the desired state.

However, sometimes when a task does make a change to the system, a further task may need to be run. For example, a change to a service's configuration file may then require that the service be reloaded so that the changed configuration takes effect.

Handlers are tasks that respond to a notification triggered by other tasks. Each handler has a globally-unique name, and is triggered at the end of a block of tasks in a playbook. If no task notifies the handler by name, it will not run. If one or more tasks notify the handler, it will run exactly once after all other tasks in the play have completed. Because handlers are tasks, administrators can use the same modules in handlers that they would for any other task. Normally, handlers are used to reboot hosts and restart services.

Handlers can be seen as *inactive* tasks that only get triggered when explicitly invoked using a **`notify`** statement. The following snippet shows how the Apache server is only restarted by the **`restart_apache`** task when a configuration file is updated and notifies it:

```
tasks:
  - name: copy demo.example.conf configuration template (1)
    copy:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf
    notify: (2)
      - restart_apache (3)

handlers: (4)
  - name: restart_apache (5)
    service: (6)
      name: httpd
      state: restarted
```

1. The task that notifies the handler.
2. The **`notify`** statement indicates the task needs to trigger a handler.
3. The name of the handler to run.
4. The statement starts the handlers section.
5. The name of the handler invoked by tasks.
6. The module to use for the handler.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

In the previous example, the **restart_apache** handler will trigger when notified by the **copy** task that a change happened. A task may call more than one handler in their **notify** section. Ansible treats the **notify** statement as an array and iterates over the handler names:

```
tasks:
  - name: copy demo.example.conf configuration template
    copy:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf
    notify:
      - restart_mysql
      - restart_apache

handlers:
  - name: restart_mysql
    service:
      name: mariadb
      state: restarted

  - name: restart_apache
    service:
      name: httpd
      state: restarted
```

Using handlers

As discussed in the Ansible documentation, there are some important things to remember about using handlers:

- Handlers are always run in the order in which the **handlers** section is written in the play, not in the order in which they are listed by the **notify** statement on a particular task.
- Handlers run after all other tasks in the play complete. A handler called by a task in the **tasks:** part of the playbook will not run until *all* of the tasks under **tasks:** have been processed.
- Handler names live in a global namespace. If two handlers are incorrectly given the same name, only one will run.
- Handlers defined inside an **include** can not be notified.
- Even if more than one task notifies a handler, the handler will only run once. If no tasks notify it, a handler will not run.
- If a task that includes a **notify** does not execute (for example, a package is already installed), the handler will not be notified. The handler will be skipped unless another task notifies it. Ansible notifies handlers only if the task acquires the **CHANGED** status.



Important

Handlers are meant to perform an action upon the execution of a task; they should not be used as a replacement for tasks.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Intro to Playbooks — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_intro.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Handlers

In this exercise, you will implement handlers in playbooks.

Outcomes
You should be able to:
- Define handlers in playbooks and notify them for configuration change.

Before you begin
Run **`lab task-control-handlers setup`** from **`workstation`** to configure the environment for the exercise. The script creates the **`dev-handlers`** project directory as well as the Ansible configuration file and the host inventory file.

```
[student@workstation ~]$ lab task-control-handlers setup
```

Steps
1. From **`workstation.lab.example.com`**, open a new terminal and change to the **`dev-handlers`** project directory.

   ```
   [student@workstation ~]$ cd dev-handlers
   [student@workstation dev-handlers]$
   ```

2. In that directory, use a text editor to create the **`configure_db.yml`** playbook file. This file will install a database server and create some users; when the database server is installed, the playbook restarts the service.

   2.1. Start the playbook with the initialization of some variables: **`db_packages`**, which defines the name of the packages to install for the database service; **`db_service`**, which defines the name of the database service; **`src_file`** for the URL of the configuration file to install; and **`dst_file`** for the location of the installed configuration file on the managed hosts. The playbook should read as follows:

   ```
   ---
   - hosts: databases
     vars:
       db_packages:
         - mariadb-server
         - MySQL-python
       db_service: mariadb
       src_file: "http://materials.example.com/task_control/my.cnf.template"
       dst_file: /etc/my.cnf
   ```

   2.2. In the **`configure_db.yml`** file, define a task that uses the **`yum`** module to install the required database packages as defined by the **`db_packages`** variable. Notify the handler **`start_service`** in order to start the service. The task should read as follows:

   ```
     tasks:
       - name: Install {{ db_packages }} package
         yum:
           name: "{{ item }}"
           state: latest
         with_items: "{{ db_packages }}"
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      notify:
        - start_service
```

2.3. Add a task to download **my.cnf.template** to **/etc/my.cnf** on the managed host, using the **get_url** module. Add a condition that notifies the handler **restart_service** as well as **set_password**, to restart the database service and set the administrative password. The task should read:

```
    - name: Download and install {{ dst_file }}
      get_url:
        url: "{{ src_file }}"
        dest: "{{ dst_file }}"
        owner: mysql
        group: mysql
        force: yes
      notify:
        - restart_service
        - set_password
```

2.4. Define the three handlers the tasks needs. The **start_service** handle will start the **mariadb** service; the **restart_service** handler will restart the **mariadb** service; and the **set_password** handler will set the administrative password for the database service.

Define the **start_service** handler. It should read as follows:

```
  handlers:
    - name: start_service
      service:
        name: "{{ db_service }}"
        state: started
```

2.5. Define the second handler, **restart_service**. It should read as follows:

```
    - name: restart_service
      service:
        name: "{{ db_service }}"
        state: restarted
```

2.6. Finally, define the handler that sets the administrative password. The handler will use the **mysql_user** module to perform the command. The handler should read as follows:

```
    - name: set_password
      mysql_user:
        name: root
        password: redhat
```

2.7. When completed, the playbook should look like the following:

```
---
- hosts: databases
  vars:
    db_packages:
      - mariadb-server
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      - MySQL-python
    db_service: mariadb
    src_file: "http://materials.example.com/task_control/my.cnf.template"
    dst_file: /etc/my.cnf

  tasks:
    - name: Install {{ db_packages }} package
      yum:
        name: "{{ item }}"
        state: latest
      with_items: "{{ db_packages }}"
      notify:
        - start_service
    - name: Download and install {{ dst_file }}
      get_url:
        url: "{{ src_file }}"
        dest: "{{ dst_file }}"
        owner: mysql
        group: mysql
        force: yes
      notify:
        - restart_service
        - set_password

  handlers:
    - name: start_service
      service:
        name: "{{ db_service }}"
        state: started

    - name: restart_service
      service:
        name: "{{ db_service }}"
        state: restarted

    - name: set_password
      mysql_user:
        name: root
        password: redhat
```

3. Run the **configure_db.yml** playbook. Notice the output shows the handlers are being executed.

```
[student@workstation dev-handlers]# ansible-playbook configure_db.yml

PLAY ***************************************************************************

... Output omitted ...

RUNNING HANDLER [start_service] ************************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [restart_service] **********************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [set_password] *************************************************
changed: [servera.lab.example.com]
```

4. Run the playbook again. This time the handlers are skipped.

```
[student@workstation dev-handlers]# ansible-playbook configure_db.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PLAY ***************************************************************

... Output omitted ...

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
```

5. Because handlers are executed once, they can help prevent failures. Update the playbook to add a task after installing **/etc/my.cnf** that sets the MySQL admin password like the **set_password** handler. This will show you why using a handler in this situation is better than a simple task. The task should read as follows:

```
    - name: Set the MySQL password
      mysql_user:
        name: root
        password: redhat
```

6. Run the playbook again. The task should fail since the MySQL password has already been set.

```
[student@workstation dev-handlers]# ansible-playbook configure_db.yml

TASK [Set the MySQL password] **************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "unable to connect to database, check login_user and login_password are
 correct or /root/.my.cnf has the credentials. Exception message: (1045, \"Access
 denied for user 'root'@'localhost' (using password: NO)\")"}

NO MORE HOSTS LEFT *************************************************************
        to retry, use: --limit @configure_db.retry

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=0    unreachable=0    failed=1
```

Cleanup

Run the **lab task-control-handlers cleanup** command to clean up after completing the lab.

```
[student@workstation ~]$ lab task-control-handlers cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Implementing Tags

Objectives

After completing this section, students should be able to:

- Set tags to control which parts of a playbook should be run.

Tagging Ansible resources

For long playbooks, it is useful to be able to run subsets of the tasks in the playbook. To do this, *tags* can be set on specific resources as a text label. Tagging a resource only requires that the **`tags`** keyword be used, followed by a list of tags to apply. When plays are tagged, the **`--tags`** option can be used with **`ansible-playbook`** to filter the playbook to only execute specific tagged plays. Tags are available for the following resources:

- In playbooks, each task can be tagged, using the **`tags`** keyword:

```
tasks:
  - name: Install {{ item }} package
    yum: name={{ item }} state=installed
    with_items:
      - postfix
      - mariadb-server
    tags:
      - packages
```

- When a task file is included in a playbook, the task can be tagged, allowing administrators to set a global tag for the **`include`** statement:

```
- include: common.yml
  tags: [webproxy, webserver]
```



Note

When tagging a role or an **`include`** statement, all tasks they define are also tagged.

```
roles:
  - { role: databases, tags: ['production', 'staging'] }
```



Important

When roles or **`include`** statements are tagged, the tag is not a way to exclude some of the tagged tasks the included files contain. Tags in this context are a way to apply a global tag to all tasks.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing tagged resources

When a tag has been applied to a resource, the **ansible-playbook** command can be used with the **--tags** or **--skip-tags** argument to either execute the tagged resources, or prevent the tagged resources from being included in the play. The following playbook contains two tasks; the first one is tagged with the **production** tag, the other one does not have any tag associated with it:

```
---
- hosts: all
  tasks:
    - name: My tagged task
      yum:
        name: httpd
        state: latest
      tags: production

    - name: Installs postfix
      yum:
        name: postfix
        state: latest
```

To only run the first task, the **--tags** argument can be used:

```
[user@demo ~]$ ansible-playbook main.yml --tags 'production'
TASK [My tagged task] ***********************************************
ok: [demo.example.com]
... Output omitted ...
```

Because the **--tags** option was specified, the playbook only ran one task, the task tagged with the **production** tag. To skip the first task and only run the second task, the **--skip-tags** option can be used:

```
[user@demo ~]$ ansible-playbook main.yml --skip-tags 'production'
... Output omitted ...

TASK [Installs postfix] *********************************************
ok: [demo.example.com]
```

Special tags

Ansible has a special tag that can be assigned in a playbook: **always**. This tag causes the task to always be executed even if the **--skip-tags** option is used, unless explicitly skipped with **--skip-tags always**.

There are three special tags that can be used from the command-line with the **--tags** option:

1. The **tagged** keyword is used to run any tagged resource.
2. The **untagged** keyword does the opposite of the **tagged** keyword by excluding all tagged resources from the play.
3. The **all** keyword allows administrators to include all tasks in the play. This is the default behavior of the command line.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Demonstration: Implementing Tags

1. From **workstation.lab.example.com**, open a new terminal and change to the **demo-tags** project directory. Create the **prepare_db.yml** task file and define a task that installs the required services for the database. Tag the task as **dev**, and have it notify the **start_db** handler. The file should read as follows:

```
---
- name: Install database packages
  yum:
    name: "{{ item }}"
    state: latest
  with_items: "{{ db_packages }}"
  tags:
    - dev
  notify:
    - start_db
```

2. Define a second task to retrieve the database configuration file if the **dev** tag is active in the execution environment, and that restarts that database service. The task will use a conditional to ensure the configuration file path is defined. The task should read as follows:

```
- name: Get small config file
  get_url:
    url: "{{ db_config_src_path_small }}"
    dest: "{{ db_config_path }}"
  when: db_config_src_path_small is defined
  notify:
    - restart_db
  tags:
    - dev
```

3. Define a third task to retrieve a different database configuration file if the **prod** tag is active in the execution environment, and that restarts the database service. Like the previous task, this task will use a conditional to ensure that the configuration file path is defined. The task should read as follows:

```
- name: Get large config file
  get_url:
    url: "{{ db_config_src_path_large }}"
    dest: "{{ db_config_path }}"
  when: db_config_src_path_large is defined
  notify:
    - restart_db
  tags:
    - prod
```

4. Define a task that sets the Message of The Day for the managed host. Tag the task with the **dev** task. The task should read as follows:

```
- name: Update motd for development
  copy:
    content: "This is a development server"
    dest: /etc/motd
  tags:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    - dev
```

5. Repeat the previous step but change both the tag as well as the command. The task should read as follows:

```
- name: Update motd for production
  copy:
    content: "This is a production server"
    dest: /etc/motd
  tags:
    - prod
```

6. When completed, the task file should read as follows:

```
---
- name: Install database packages
  yum:
    name: "{{ item }}"
    state: latest
  with_items: "{{ db_packages }}"
  tags:
    - dev
  notify:
    - start_db

- name: Get small config file
  get_url:
    url: "{{ db_config_src_path_small }}"
    dest: "{{ db_config_path }}"
  when: db_config_src_path_small is defined
  notify:
    - restart_db
  tags:
    - dev

- name: Get large config file
  get_url:
    url: "{{ db_config_src_path_large }}"
    dest: "{{ db_config_path }}"
  when: db_config_src_path_large is defined
  notify:
    - restart_db
  tags:
    - prod

- name: Update motd for development
  copy:
    content: "This is a development server"
    dest: /etc/motd
  tags:
    - dev

- name: Update motd for production
  copy:
    content: "This is a production server"
    dest: /etc/motd
  tags:
    - prod
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

7. In the project directory, create the main playbook, **playbook.yml** for servers in the **databases** group. The playbook will define the variables required by the task file upon import. The playbook should read as follows:

```
---
- hosts: all
  vars:
    db_packages:
      - mariadb-server
      - MySQL-python
    db_config_url: http://materials.example.com/task_control
    db_config_src_path_small: "{{ db_config_url }}/my.cnf.small"
    db_config_src_path_large: "{{ db_config_url }}/my.cnf.large"
    db_config_path: /etc/my.cnf
    db_service: mariadb
```

8. Define the first task that includes the task file; the task file will use a conditional to ensure the managed host is in the group **databases**. The task should read as follows:

```
  tasks:
  - include:
      prepare_db.yml
    when: inventory_hostname in groups['databases']
```

9. Define the two handlers the task file requires, **start_db** and **restart_db**. The **handlers** block should read as follows:

```
  handlers:
  - name: start_db
    service:
      name: "{{ db_service }}"
      state: started

  - name: restart_db
    service:
      name: "{{ db_service }}"
      state: restarted
```

10. When completed, the playbook should read as follows:

```
---
- hosts: all
  vars:
    db_packages:
      - mariadb-server
      - MySQL-python
    db_config_url: http://materials.example.com/task_control
    db_config_src_path_small: "{{ db_config_url }}/my.cnf.small"
    db_config_src_path_large: "{{ db_config_url }}/my.cnf.large"
    db_config_path: /etc/my.cnf
    db_service: mariadb

  tasks:
  - include:
      prepare_db.yml
    when: inventory_hostname in groups['databases']
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  handlers:
  - name: start_db
    service:
      name: "{{ db_service }}"
      state: started

  - name: restart_db
    service:
      name: "{{ db_service }}"
      state: restarted
```

11. Run the playbook, applying the **dev** tag using the **--tags** option.

```
[student@workstation demo-tags]$ ansible-playbook playbook.yml --tags 'dev'
... Output omitted ...
TASK [Get small config file] ***************************************************
changed: [servera.lab.example.com]

TASK [Update motd for development] *********************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

Notice the configuration file that has been retrieved (**my.cnf.small**).

12. Run an ad hoc command to display the **/etc/motd** file on **servera**.

```
[student@workstation dev-tags]$ ansible databases -a 'cat /etc/motd'
servera.lab.example.com | SUCCESS | rc=0 >>
This is a development server
```

13. Run the playbook again, this time skipping the tasks tagged as **dev**:

```
[student@workstation demo-tags]$ ansible-playbook playbook.yml --skip-tags 'dev'
... Output omitted ...
TASK [Get large config file] ***************************************************
ok: [servera.lab.example.com]

TASK [Update motd for production] **********************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

14. Run an ad hoc command to display the **/etc/motd** file on **servera**.

```
[student@workstation dev-tags]$ ansible databases -a 'cat /etc/motd'
servera.lab.example.com | SUCCESS | rc=0 >>
This is a production server
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Tags — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_tags.html

Task And Handler Organization For A Role — Best Practices — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_best_practices.html#task-and-handler-organization-for-a-role

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Tags

In this exercise, you will implement tags in a playbook and run the playbook.

Outcomes
You should be able to:
- Tag Ansible tasks.
- Filter tasks based on tags when running playbooks.

Before you begin
Run the **`lab task-control-tags setup`** command from **`workstation`** to prepare the environment for this exercise. The script creates the working directory, called **`dev-tags`**, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab task-control-tags setup
```

Steps
1. Change to the **`dev-tags`** project directory that the lab script created.

   ```
   [student@workstation ~]$ cd dev-tags
   [student@workstation dev-tags]$
   ```

2. The following steps will edit the same **`configure_mail.yml`** task file.

   In the project directory, create the **`configure_mail.yml`** task file. The task file contains instructions to install the required packages for the mail server, as well as instructions to retrieve the configuration files for the mail server.

   2.1. Create the first task that uses the **`yum`** module to install the *postfix* package. Notify the **`start_postfix`** handler, and tag the task as **`server`** using the **`tags`** keyword. The task should read as follows:

   ```
   ---
   - name: Install postfix
     yum:
       name: postfix
       state: latest
     tags:
       - server
     notify:
       - start_postfix
   ```

   2.2. Add a task that installs the *dovecot* package using the **`yum`** module. It should notify the **`start_dovecot`** handler. Tag the task as **`client`**. The task should read as follows:

   ```
   - name: Install dovecot
     yum:
       name: dovecot
       state: latest
     tags:
       - client
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  notify:
    - start_dovecot
```

2.3. Define a task that uses the **get_url** module to retrieve the Postfix configuration file. Notify the **restart_postfix** handler and tag the task as **server**. The task should read as follows:

```
- name: Download main.cf configuration
  get_url:
    url: http://materials.example.com/task_control/main.cf
    dest: /etc/postfix/main.cf
  tags:
    - server
  notify:
    - restart_postfix
```

2.4. When completed, the task file should read as follows:

```
---
- name: Install postfix
  yum:
    name: postfix
    state: latest
  tags:
    - server
  notify:
    - start_postfix

- name: Install dovecot
  yum:
    name: dovecot
    state: latest
  tags:
    - client
  notify:
    - start_dovecot

- name: Download main.cf configuration
  get_url:
    url: http://materials.example.com/task_control/main.cf
    dest: /etc/postfix/main.cf
  tags:
    - server
  notify:
    - restart_postfix
```

3. The following steps will edit the same **playbook.yml** playbook file.

3.1. Create a playbook file named **playbook.yml**. Define the playbook for all hosts. The playbook should read as follows:

```
---
- hosts: all
```

3.2. Define the task that includes the task file **configure_mail.yml** using the **include** module. Add a conditional to only run the task for the hosts in the **mailservers** group. The task should read as follows:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  tasks:
   - name: Include configure_mail.yml
     include:
       configure_mail.yml
     when: inventory_hostname in groups['mailservers']
```

3.3. Define the three handlers the task file requires: **`start_postfix`**, **`start_dovecot`**, and **`restart_postfix`**.

Define **`start_postfix`** handler to start the mail server. The handler should read as follows:

```
  handlers:
    - name: start_postfix
      service:
        name: postfix
        state: started
```

3.4. Define the **`start_dovecot`** handler to start the mail client. The handler should read as follows:

```
    - name: start_dovecot
      service:
        name: dovecot
        state: started
```

3.5. Define the **`restart_postfix`** handler that restarts the mail server. The handler should read as follows:

```
    - name: restart_postfix
      service:
        name: postfix
        state: restarted
```

3.6. When completed, the playbook should read as follows:

```
---
- hosts: all

  tasks:
   - name: Include configure_mail.yml
     include:
       configure_mail.yml
     when: inventory_hostname in groups['mailservers']

  handlers:
    - name: start_postfix
      service:
        name: postfix
        state: started

    - name: start_dovecot
      service:
        name: dovecot
        state: started
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  - name: restart_postfix
    service:
      name: postfix
      state: restarted
```

4. Run the playbook, only applying the **server** tagged tasks using the **--tags** option. Notice how only the **start_postfix** handler gets triggered.

```
[student@workstation dev-tags]$ ansible-playbook playbook.yml --tags 'server'
... Output omitted ...
RUNNING HANDLER [start_postfix] ************************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

5. Run an ad hoc command to ensure the *postfix* package has been successfully installed.

```
[student@workstation dev-tags]$ ansible mailservers -a 'yum list installed postfix'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
postfix.x86_64                    2:2.10.1-6.el7                    @rhel_dvd
```

6. Run the playbook again, but this time skip the tasks tagged with the **server** tag. The play will install the *dovecot* package, because the task is tagged with the **client** tag, and it will trigger the **start_dovecot** handler.

```
[student@workstation dev-tags]$ ansible-playbook playbook.yml --skip-tags 'server'
... Output omitted ...
TASK [Install dovecot] *********************************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [start_dovecot] ************************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

7. Run an ad hoc command to ensure the *dovecot* package has been successfully installed.

```
[student@workstation dev-tags]$ ansible mailservers -a 'yum list installed dovecot'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
dovecot.x86_64                    1:2.2.10-5.el7                    @rhel_dvd
```

Evaluation

Run the **lab task-control-tags grade** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab task-control-tags grade
```

Cleanup

Run the **lab task-control-tags cleanup** command to cleanup after the lab.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab task-control-tags cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Handling Errors

Objectives

After completing this section, students should be able to:

- Resolve errors in a playbook.

Errors in plays

Ansible evaluates the return codes of modules and commands to determine whether a task succeeded or failed. Normally, when a task fails Ansible immediately aborts the plays, skipping all subsequent tasks.

However, sometimes an administrator may want to have playbook execution continue even if a task fails. For example, it may be expected that a particular command will fail. There are a number of Ansible features that may be used to better manage task errors:

- **Ignore the failed task**: By default, if a task fails, the play is aborted; however, this behavior can be overridden by skipping failed tasks. To do so, the **`ignore_errors`** keyword needs to be used in a task. The following snippet shows how to use **`ignore_errors`** to continue playbook execution even if the task fails. For example, if the *notapkg* package does not exist the yum module will fail, but having **`ignore_errors`** set to **`yes`** will allow execution to continue.

```
- yum:
    name: notapkg
    state: latest
  ignore_errors: yes
```

- **Force execution of handlers**: By default, if a task that notifies a handler fails, the handler will be skipped as well. Administrators can override this behavior by using the **`force_handlers`** keyword in task. This forces the handler to be called even if the task fails. The following snippet shows hows to use the **`force_handlers`** keyword in a task to forcefully execute the handler even if the task fails:

```
---
- hosts: all
  force_handlers: yes
  tasks:
    - yum:
        name: notapkg
        state: latest
      notify: restart_database

  handlers:
    - name: restart_database
      service:
        name: mariadb
        state: restarted
```

- **Override the `failed` state**: A task itself can succeed, yet administrators may want to mark the task as failed based on specific criteria. To do so, the **`failed_when`** keyword can be used with a task. This is often used with modules that execute remote commands and capture the output in a variable. For example, administrators can run a script that outputs an error

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

message and use that message to define the failed state for the task. The following snippet shows how the **failed_when** keyword can be used in a task:

```
tasks:
  - shell:
      cmd: /usr/local/bin/create_users.sh
    register: command_result
    failed_when: "'Password missing' in command_result.stdout"
```

- **Override the changed state**: When a task updates a managed host, it acquires the **changed** state; however, if a task does not perform any change on the managed node, handlers are skipped. The **changed_when** keyword can be used to override the default behavior of what triggers the **changed** state. For example, if administrators want to restart a service every time a playbook runs, the **changed_when** keyword can be added to the task. The following snippet shows how a handler can be triggered every time by forcing the **changed** state:

```
tasks:
  - shell:
      cmd: /usr/local/bin/upgrade-database
    register: command_result
    changed_when: "'Success' in command_result.stdout"
    notify:
      - restart_database

handlers:
  - name: restart_database
     service:
       name: mariadb
       state: restarted
```

Ansible blocks and error handling

In playbooks, *blocks* are clauses that enclose tasks. Blocks allow for logical grouping of tasks, and can be used to control how tasks are executed. For example, administrators can define a main set of tasks and a set of extra tasks that will only be executed if the first set fails. To do so, blocks in playbooks can be used with three keywords:

- **block**: Defines the main tasks to run.
- **rescue**: Defines the tasks that will be run if the tasks defined in the **block** clause fails.
- **always**: Defines the tasks that will always run independently of the success or failure of tasks defined in the **block** and **rescue** clauses.

The following example shows how to implement a block in a playbook. Even if tasks defined in the **block** clause fail, tasks defined in the **rescue** and **always** clause will be executed:

```
tasks:
  - block:
      - shell:
          cmd: /usr/local/lib/upgrade-database

    rescue:
      - shell:
          cmd: /usr/local/lib/create-users

    always:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    - service:
        name: mariadb
        state: restarted
```

Demonstration: Handling errors

1. The setup script created a project directory, called **demo-failures**. It contains an initial playbook called **playbook.yml**. The playbook targets all hosts in the **mailservers** group. It initializes four variables: **maildir_path** with a value of **/home/john/Maildir**, **maildir** with a value of **/home/student/Maildir**, **mail_package** with a value of **postfix**, and **mail_service** with a value of **postfix**. The playbook begins with the following:

```
---
- hosts: mailservers
  vars:
    maildir_path: /home/john/Maildir
    maildir: /home/student/Maildir
    mail_package: postfix
    mail_service: postfix
```

2. It also has a **tasks** section that defines a task that uses the **copy** module to install a directory to the managed host.

```
  tasks:
    - name: Create {{ maildir_path }}
      copy:
        src: "{{ maildir }}"
        dest: "{{ maildir_path }}"
        mode: 0755
```

3. It has a second task that installs the package that the **mail_package** variable defines.

```
    - name: Install {{ mail_package }} package
      yum:
        name: "{{ mail_package }}"
        state: latest
```

4. Run the playbook. Watch as the first task fails, resulting in the failure of the play.

```
[student@workstation demo-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Create /home/john/Maildir] ***********************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "could not find src=/home/student/Maildir"}

NO MORE HOSTS LEFT *************************************************************
        to retry, use: --limit @playbook.retry
... Output omitted ...
```

5. Update the first task by adding the **ignore_errors** keyword with a value of **yes**. The updated task should read as follows:

```
    - name: Create {{ maildir_path }}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      copy:
        src: "{{ maildir }}"
        dest: "{{ maildir_path }}"
        mode: 0755
      ignore_errors: yes
```

6. Run the playbook and watch as the second task runs, despite the fact that the first task failed.

```
[student@workstation demo-failures]$ ansible-playbook.playbook.yml
... Output omitted ...
TASK [Create /home/john/Maildir] ***********************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "could not find src=/home/student/Maildir"}
...ignoring

TASK [Install postfix package] *************************************************
changed: [servera.lab.example.com]
.... Output omitted ...
```

7. Create a block to nest the two existing tasks. Define a new task that starts the mail server.

   7.1. Create a block in the **tasks** section and nest the tasks into it. Remove the line **ignore_errors: yes** for the first task. The **block** section should read as follows:

```
    - block:
      - name: Create {{ maildir_path }}
        copy:
          src: "{{ maildir }}"
          dest: "{{ maildir_path }}"
          mode: 0755
```

   7.2. Move the task that installs the *postfix* package into the **rescue** clause. Update the task to also install the *dovecot* package. The clause should read as follows:

```
      rescue:
      - name: Install mail packages
        yum:
          name: "{{ item }}"
          state: latest
        with_items:
          - "{{ mail_package }}"
          - dovecot
```

   7.3. Add an **always** clause. It should create a task to start both the **postfix** and **dovecot** services. Register the output of the command in the **command_result** variable. The clause should read as follows:

```
      always:
      - name: Start mail services
        service:
          name: "{{ item }}"
          state: started
        with_items:
          - "{{ mail_service }}"
          - dovecot
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        register: command_result
```

8. Add a **debug** task outside the block that outputs the **command_result** variable. The task should read as follows:

```
    - debug:
        var: command_result
```

9. The playbook should now read as follows:

```
---
- hosts: mailservers
  vars:
    maildir_path: /home/john/Maildir
    maildir: /home/student/Maildir
    mail_package: postfix
    mail_service: postfix

  tasks:
    - block:
      - name: Create {{ maildir_path }}
        copy:
          src: "{{ maildir }}"
          dest: "{{ maildir_path }}"
          mode: 0755

      rescue:
      - name: Install mail packages
        yum:
          name: "{{ item }}"
          state: latest
        with_items:
          - "{{ mail_package }}"
          - dovecot

      always:
      - name: Start mail services
        service:
          name: "{{ item }}"
          state: started
        with_items:
          - "{{ mail_service }}"
          - dovecot
        register: command_result

    - debug:
        var: command_result
```

10. Run the playbook, and watch as the **rescue** and **always** clauses run despite the fact that the first task failed.

```
[student@workstation demo-failures]$ ansible-playbook.playbook.yml
... Output omitted ...
TASK [Create /home/john/Maildir] ***********************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false,
"failed": true, "msg": "could not find src=/home/student/Maildir"}

TASK [Install mail packages] ***************************************************
changed: [servera.lab.example.com] => (item=[u'postfix', u'dovecot'])
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [Start mail services] *****************************************************
changed: [servera.lab.example.com] => (item=postfix)
changed: [servera.lab.example.com] => (item=dovecot)

TASK [debug] *******************************************************************
ok: [servera.lab.example.com] => {
    "command_result": {
        "changed": true,
        "msg": "All items completed",
        "results": [
            {
... Output omitted ...
```

11. To restart the two services every time, the **changed_when** keyword can be used. To do so, capture the output of the first task into a variable; the variable will be used to force the state of the task that installs the packages. A **handlers** section will be added to the playbook.

    11.1. Update the task in the **block** section by saving the output of the command into the **command_output** variable. The task should read as follows:

```
      - name: Create {{ maildir_path }}
        copy:
          src: "{{ maildir }}"
          dest: "{{ maildir_path }}"
          mode: 0755
        register: command_output
```

    11.2. Update the task in the **rescue** section by adding a **changed_when** condition as well as a **notify** section for the **restart_dovecot** handler. The condition will read the **command_output** variable to forcefully mark the task as changed. The task should read as follows:

```
      - name: Install mail packages
        yum:
          name: "{{ item }}"
          state: latest
        with_items:
          - "{{ mail_package }}"
          - dovecot
        changed_when: "'not find' in command_output.msg"
        notify: restart_dovecot
```

    11.3. Update the task in the **always** section to restart the service defined by the **mail_service** variable. The **always** section should read as follows:

```
      - name: Start mail services
        service:
          name: "{{ mail_service }}"
          state: restarted
        register: command_result
```

    11.4. Append a **handlers** section to restart the **dovecot** service. The **handlers** section should read as follows:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
handlers:
  - name: restart_dovecot
    service:
      name: dovecot
      state: restarted
```

11.5. When updated, the **tasks** and **handlers** sections should read as follows:

```
tasks:
  - block:
    - name: Create {{ maildir_path }}
      copy:
        src: "{{ maildir }}"
        dest: "{{ maildir_path }}"
        mode: 0755
      register: command_output

    rescue:
    - name: Install mail packages
      yum:
        name: "{{ item }}"
        state: latest
      with_items:
        - "{{ mail_package }}"
        - dovecot
      changed_when: "'not find' in command_output.msg"
      notify: restart_dovecot

    always:
    - name: Start mail services
      service:
        name: "{{ mail_service }}"
        state: restarted
      register: command_result

  - debug:
      var: command_result

handlers:
  - name: restart_dovecot
    service:
      name: dovecot
      state: restarted
```

12. Run the playbook. Watch as the handler is called, despite the fact that the *dovecot* package is already installed:

```
[student@workstation demo-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Install mail packages] ***************************************************
changed: [servera.lab.example.com] => (item=[u'postfix', u'dovecot'])
... Output omitted ...
RUNNING HANDLER [restart_dovecot] **********************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Error Handling in Playbooks — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_error_handling.html

Error Handling — Blocks — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_blocks.html#error-handling

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Handling Errors

In this exercise, you will handle errors in Ansible playbooks using various features

Outcomes
You should be able to:

- Ignore failed commands during the execution of playbooks.
- Force execution of handlers.
- Override what constitutes a failure in tasks.
- Override the **changed** state for tasks.
- Implement blocks/rescue/always in playbooks.

Before you begin
From **workstation**, run the lab setup script to confirm the environment is ready for the lab to begin. The script creates the working directory **dev-failures**.

```
[student@workstation ~]$ lab task-control-failures setup
```

Steps

1. From **workstation.lab.example.com**, change to the **dev-failures** project directory.

   ```
   [student@workstation ~]$ cd dev-failures
   [student@workstation dev-failures]$
   ```

2. The lab script created an Ansible configuration file as well as an inventory file that contains the server **servera.lab.example.com** in the group **databases**. Review the file before proceeding.

3. Create the **playbook.yml** playbook. Initialize three variables: **web_package** with a value of **http**, **db_package** with a value of **mariadb-server** and **db_service** with a value of **mariadb**. The variables will be used to install the required packages and start the server.

   The **http** value is an intentional error in the package name. The file should read as follows:

   ```
   ---
   - hosts: databases
     vars:
       web_package: http
       db_package: mariadb-server
       db_service: mariadb
   ```

4. Define two tasks that use the **yum** module and the two variables, **web_package** and **db_package**. The task will install the required packages. The tasks should read as follows:

   ```
     tasks:
       - name: Install {{ web_package }} package
         yum:
           name: "{{ web_package }}"
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        state: latest

    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: latest
```

5. Run the playbook and watch the output of the play.

```
[student@workstation dev-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "No Package matching 'http' found available, installed or updated",
 "rc": 0, "results": []}
... Output omitted ...
```

The task failed because there is no existing package called **`http`**. Because the first task failed, the second task was skipped.

6. Update the first task to ignore any errors by adding the **`ignore_errors`** keyword. The tasks should read as follows:

```
  tasks:
    - name: Install {{ web_package }} package
      yum:
        name: "{{ web_package }}"
        state: latest
      ignore_errors: yes

    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: latest
```

7. Run the playbook another time and watch the output of the play.

```
[student@workstation dev-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "No Package matching 'http' found available, installed or updated",
 "rc": 0, "results": []}
...ignoring

TASK [Install mariadb-server package] ******************************************
ok: [servera.lab.example.com]
... Output omitted ...
```

Despite the fact that the first task failed, Ansible executed the second one.

8. Insert a new task at the beginning of the playbook. The task will execute a remote command and capture the output. The output of the command is used by the task that installs the *mariadb-server* package to override what Ansible considers as a failure.

    8.1. In the playbook, insert a task at the beginning of the playbook that executes a remote command and saves the output in the **`command_result`** variable. The play should also

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

continue if the task fails; to do so, add the **ignore_errors** keyword. The task should read as follows:

```
    - name: Check {{ web_package }} installation status
      command: yum list installed "{{ web_package }}"
      register: command_result
      ignore_errors: yes
```

8.2. Run the playbook to ensure the first two tasks are skipped.

```
[student@workstation dev-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Check http installation status] ***************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": true, "cmd":
 ["yum", "list", "installed", "http"], "delta": "0:00:00.269811", "end":
 "2016-05-18 07:10:18.446872", "failed": true, "rc": 1, "start": "2016-05-18
 07:10:18.177061", "stderr": "Error: No matching Packages to list", "stdout":
 "Loaded plugins: langpacks, search-disabled-repos", "stdout_lines": ["Loaded
 plugins: langpacks, search-disabled-repos"], "warnings": ["Consider using yum
 module rather than running yum"]}
...ignoring

TASK [Install http package] *************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "No Package matching 'http' found available, installed or updated",
 "rc": 0, "results": []}
...ignoring
... Output omitted ...
```

8.3. Update the task that installs the *mariadb-server* package. Add a condition that indicates that the task should be considered as failed if the keyword **Error** is present in the variable **command_result**. The task should read as follows:

```
    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: latest
      when: "'Error' in command_result.stdout"
```

8.4. Before running the playbook, run an ad hoc command to remove the **mariadb-server** package from the **databases** managed hosts.

```
[student@workstation dev-failures]$ ansible databases -a 'yum -y remove mariadb-
server'
servera.lab.example.com | SUCCESS | rc=0 >>
...  Output omitted ...
Removed:
  mariadb-server.x86_64 1:5.5.44-2.el7
... Output omitted ...
```

8.5. Run the playbook and watch how the latest task is skipped as well.

```
[student@workstation dev-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Install mariadb-server package] ***************************************
skipping: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
... Output omitted ...
```

9. Update the task that installs the *mariadb-server* package by overriding what triggers the **changed** state. To do so, use the return code saved in the **command_result.rc** variable.

   9.1. Update the last task by commenting out the **when** condition and adding a **changed_when** condition in order to override the **changed** state for the task. The condition will use the return code the registered variable contains. The task should read as follows:

```
    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: latest
      # when: "'Error' in command_result.stdout"
      changed_when: "command_result.rc == 1"
```

   9.2. Execute the playbook twice; the first execution will reinstall the *mariadb-server* package. The following output shows the result of the second execution; as you can see, the task is marked as **changed** despite the fact that the *mariadb-server* package has already been installed:

```
... Output omitted ...
TASK [Install mariadb-server package] ******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=1    unreachable=0    failed=0
```

   9.3. Update the playbook by nesting the first two tasks in a **block** cause. Remove the lines that use the **ignore_errors** conditional. The block should read as follows:

```
    - block:
      - name: Check {{ web_package }} installation status
        command: yum list installed "{{ web_package }}"
        register: command_result

      - name: Install {{ web_package }} package
        yum:
          name: "{{ web_package }}"
          state: latest
```

   9.4. Nest the task that installs the *mariadb-server* package in a **rescue** clause and remove the conditional that overrides the changed result. The task will be executed even if the previous tasks fail. The block should read as follows:

```
      rescue:
        - name: Install {{ db_package }} package
          yum:
            name: "{{ db_package }}"
            state: latest
```

   9.5. Finally, add an **always** clause that will start the database server upon installation using the **service** module. The clause should read as follows:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    always:
      - name: Start {{ db_service }} service
        service:
          name: "{{ db_service }}"
          state: started
```

9.6. Once updated, the task section should read as follows:

```
  tasks:
    - block:
      - name: Check {{ web_package }} installation status
        command: yum list installed "{{ web_package }}"
        register: command_result

      - name: Install {{ web_package }} package
        yum:
          name: "{{ web_package }}"
          state: latest

      rescue:
        - name: Install {{ db_package }} package
          yum:
            name: "{{ db_package }}"
            state: latest

      always:
        - name: Start {{ db_service }} service
          service:
            name: "{{ db_service }}"
            state: started
```

9.7. Before running the playbook, remove the *mariadb-server* package from the **databases** managed hosts.

```
[student@workstation dev-failures]$ ansible databases -a 'yum -y remove mariadb-
server'
```

9.8. Run the playbook, and watch as despite failure for the first two tasks, Ansible installs the *mariadb-server* package and starts the **mariadb** service.

```
[student@workstation dev-failures]$ ansible-playbook playbook.yml
... Output omitted ...
TASK [Install mariadb-server package] ******************************************
changed: [servera.lab.example.com]

TASK [Start mariadb service ] **************************************************
changed: [servera.lab.example.com]
... Output omitted ...
```

Evaluation

From *workstation*, run the **lab task-control-failures** command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab task-control-failures grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Cleanup

Run the **lab task-control-failures cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab task-control-failures cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Task Control

In this lab, you will install the Apache web server and secure it using **mod_ssl**. You will use various Ansible conditionals to deploy the environment.

Outcomes

You should be able to:

- Define conditionals in Ansible playbooks.
- Set up loops that iterate over elements.
- Define handlers in playbooks.
- Define tags and use them in conditionals.
- Handle errors in playbooks.

Before you begin

Log in as the **student** user on **workstation** and run **lab task-control setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory are installed on the control node.

```
[student@workstation ~]$ lab task-control setup
```

Steps

1. From **workstation.lab.example.com**, change to the **lab-task-control** project directory.

2. **Defining tasks for the web server**

   In the top-level directory for this lab, create the **install_packages.yml** task file. Define a task that installs the latest version of the *httpd* and *mod_ssl* packages. For the two packages to install, use the variables called **web_package** and **ssl_package**. The variables will be defined later on in the main playbook. Use a loop for installing the packages; moreover, the packages should only be installed if the server belongs to the **webservers** group, and only if the available memory on the system is greater than the amount of memory the **memory** variable defines. The variable will be set upon import of the task; use an Ansible fact to determine the available memory on the managed host.

   Add a task that starts the service defined by the **web_service** variable.

3. **Defining tasks for the web server's configuration**

   Create the **configure_web.yml** task file. Add a task that checks whether or not the *httpd* package is installed and register the output in a variable. Update the condition to consider the task as failed based on the return code of the command (the return code is **1** when a package is not installed).

   Create a block that executes only if the *httpd* package is installed (use the return code that has been captured in the first task). The block should start with a task that retrieves the file that the **https_uri** variable defines (the variable will be set in the main playbook) and copy it to **serverb.lab.example.com** in the **/etc/httpd/conf.d/** directory.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Define a task that creates the **ssl** directory under **/etc/httpd/conf.d/** on the managed host with a mode of **0755**. The directory will store the SSL certificates.

Define a task that creates the **logs** directory under **/var/www/html/** on the managed host with a mode of **0755**. The directory will store the SSL logs.

Define a task that uses the **stat** module to ensure the **/etc/httpd/conf.d/ssl.conf** file exists, and capture the output in a variable. Define a task that renames the **/etc/httpd/conf.d/ssl.conf** file as **/etc/httpd/conf.d/ssl.conf.bak**, only if the file exists (use the captured output from the previous task).

Define another task that retrieves and extracts the SSL certificates file that the **ssl_uri** variable defines (the variable will be set in the main playbook). Extract the file under the **/etc/httpd/conf.d/ssl/** directory.

Configure this task to notify the **restart_services** handler.

Define the task that creates the **index.html** file under the **/var/www/html/** directory. The file should use Ansible facts and should read as follows:

```
serverb.lab.example.com (172.25.250.11) has been customized by Ansible
```

4. **Defining tasks for the firewall**

   Create the **configure_firewall.yml** task file. Start with a task that installs the package that the **fw_package** variable defines (the variable will be set in the main playbook). Create a task that starts the service specified by the **fw_service** variable.

   Create a task that adds firewall rules for the **http** and **https** services using a loop. The rules should be applied immediately and persistently. Tag all tasks with the **production** tag.

5. **Defining the main playbook**

   In the top-level project directory for this lab, create the main playbook, **playbook.yml**. The playbook should only apply to hosts in the **webservers** group. Define a block that imports the following three task files: **install_packages.yml**, **configure_web.yml**, and **configure_firewall.yml**.

   For the task that imports the **install_packages.yml** playbook, define the following variables: **memory** with a value of **256**, **web_package** with a value of **httpd** and **ssl_package** with a value of **mod_ssl**.

   For the task that imports the **configure_web.yml** file, define the following variables: **https_uri** with a value of **http://materials.example.com/task_control/https.conf**, and **ssl_uri** with a value of **http://materials.example.com/task_control/ssl.tar.gz**.

   For the task that imports the **configure_firewall.yml** playbook, add a condition to only import the tasks tagged with the **production** tag. Define the **fw_package** and **fw_service** variables with a value of **firewalld**.

   In the **rescue** clause for the block, define a task to install the **httpd** service. Notify the **restart_services** handler to start the service upon its installation.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Add an **always** statement that uses the **shell** module to query the status of the **httpd** service using **systemctl**.

Define the **restart_services** handle to restart both the **httpd** and **firewalld** services using a loop.

6. **Executing the playbook.yml playbook**

Run the **playbook.yml** playbook to set up the environment. Ensure the web server has been correctly configured by querying the home page of the web server (**https://serverb.example.com**) using **curl** with the **-k** option to allow insecure connections. The output should read as follows:

```
server.lab.example.com (172.25.250.11) has been customized by Ansible
```

Evaluation

Run the **lab task-control grade** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab task-control grade
```

Cleanup

Run the **lab task-control cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab task-control cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will install the Apache web server and secure it using **mod_ssl**. You will use various Ansible conditionals to deploy the environment.

Outcomes

You should be able to:

- Define conditionals in Ansible playbooks.
- Set up loops that iterate over elements.
- Define handlers in playbooks.
- Define tags and use them in conditionals.
- Handle errors in playbooks.

Before you begin

Log in as the **student** user on **workstation** and run **lab task-control setup**. This setup script ensures that the managed host, **serverb**, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory are installed on the control node.

```
[student@workstation ~]$ lab task-control setup
```

Steps

1. From **workstation.lab.example.com**, change to the **lab-task-control** project directory.

   ```
   [student@workstation ~]$ cd lab-task-control
   [student@workstation lab-task-control]$
   ```

2. **Defining tasks for the web server**

   In the top-level directory for this lab, create the **install_packages.yml** task file. Define a task that installs the latest version of the *httpd* and *mod_ssl* packages. For the two packages to install, use the variables called **web_package** and **ssl_package**. The variables will be defined later on in the main playbook. Use a loop for installing the packages; moreover, the packages should only be installed if the server belongs to the **webservers** group, and only if the available memory on the system is greater than the amount of memory the **memory** variable defines. The variable will be set upon import of the task; use an Ansible fact to determine the available memory on the managed host.

   Add a task that starts the service defined by the **web_service** variable.

   2.1. The following steps will edit the single **install_packages.yml** task file.

   In the top-level directory for this lab, create the **install_packages.yml** task file. Start by defining the task that uses the **yum** module in order to install the required packages. For the packages, use a loop and two variables: **web_package** and **ssl_package**. The two variables will be set by the main playbook.

   ```
   ---
   - name: Installs the required packages
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  yum:
    name: "{{ item }}"
  with_items:
    - "{{ web_package }}"
    - "{{ ssl_package }}"
```

2.2. Continue editing **`install_packages.yml`**. Add a **`when`** clause in order to install the packages only if:

1. The managed host is in the **`webservers`** group.
2. The amount of memory on the managed host is greater than the amount the **`memory`** variable defines. For the amount of memory the system has, the **`ansible_memory_mb.real.total`** can be used.

The **`when`** clause should read as follows:

```
  when:
    - inventory_hostname in groups["webservers"]
    - "{{ ansible_memory_mb.real.total }} > {{ memory }}"
```

2.3. Finally, add the task that starts the service defined by the **`web_service`** variable.

```
- name: Starts the service
  service:
    name: "{{ web_service }}"
    state: started
```

2.4. When completed, the file should read as follows:

```
---
- name: Installs the required packages
  yum:
    name: "{{ item }}"
  with_items:
    - "{{ web_package }}"
    - "{{ ssl_package }}"
  when:
    - inventory_hostname in groups["webservers"]
    - "{{ ansible_memory_mb.real.total }} > {{ memory }}"

- name: Starts the service
  service:
    name: "{{ web_service }}"
    state: started
```

3. **Defining tasks for the web server's configuration**

Create the **`configure_web.yml`** task file. Add a task that checks whether or not the *httpd* package is installed and register the output in a variable. Update the condition to consider the task as failed based on the return code of the command (the return code is **1** when a package is not installed).

Create a block that executes only if the *httpd* package is installed (use the return code that has been captured in the first task). The block should start with a task that retrieves the file

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

that the **https_uri** variable defines (the variable will be set in the main playbook) and copy it to **serverb.lab.example.com** in the **/etc/httpd/conf.d/** directory.

Define a task that creates the **ssl** directory under **/etc/httpd/conf.d/** on the managed host with a mode of **0755**. The directory will store the SSL certificates.

Define a task that creates the **logs** directory under **/var/www/html/** on the managed host with a mode of **0755**. The directory will store the SSL logs.

Define a task that uses the **stat** module to ensure the **/etc/httpd/conf.d/ssl.conf** file exists, and capture the output in a variable. Define a task that renames the **/etc/httpd/conf.d/ssl.conf** file as **/etc/httpd/conf.d/ssl.conf.bak**, only if the file exists (use the captured output from the previous task).

Define another task that retrieves and extracts the SSL certificates file that the **ssl_uri** variable defines (the variable will be set in the main playbook). Extract the file under the **/etc/httpd/conf.d/ssl/** directory.

Configure this task to notify the **restart_services** handler.

Define the task that creates the **index.html** file under the **/var/www/html/** directory. The file should use Ansible facts and should read as follows:

```
serverb.lab.example.com (172.25.250.11) has been customized by Ansible
```

3.1. The following steps will edit the single **configure_web.yml** file.

In the top-level directory of this lab, create the **configure_web.yml** tasks file. Start with a task that uses the **shell** module to determine whether or not the **httpd** package is installed. The **failed_when** variable will be used to override how Ansible should consider the task as failed by using the return code.

```
---
- shell:
    rpm -q httpd
  register: rpm_check
  failed_when: rpm_check.rc == 1
```

3.2. Continue editing the **configure_web.yml** file. Create a block that contains the tasks for configuring the files. Start the block with a task that uses the **get_url** module to retrieve the Apache SSL configuration file. Use the **https_uri** variable for the **url** and **/etc/httpd/conf.d/** for the remote path on the managed host.

```
- block:
  - get_url:
      url: "{{ https_uri }}"
      dest: /etc/httpd/conf.d/
```

3.3. Create the **/etc/httpd/conf.d/ssl** remote directory with a mode of **0755**.

```
  - file:
      path: /etc/httpd/conf.d/ssl
      state: directory
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    mode: 0755
```

3.4. Create the **/var/www/html/logs** remote directory with a mode of **0755**.

```
- file:
    path: /var/www/html/logs
    state: directory
    mode: 0755
```

3.5. Ensure the **/etc/httpd/conf.d/ssl.conf** file exists. Capture the output in the **ssl_file** variable using the **register** statement.

```
- stat:
    path: /etc/httpd/conf.d/ssl.conf
  register: ssl_file
```

3.6. Create the task that renames the **/etc/httpd/conf.d/ssl.conf** file as **/etc/httpd/conf.d/ssl.conf.bak**. The task will evaluate the content of the **ssl_file** variable before attempting to rename the file.

```
- shell:
    mv /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.conf.bak
  when: ssl_file.stat.exists
```

3.7. Create the task that uses the **unarchive** module to retrieve the remote SSL configuration files. Use the **ssl_uri** variable for the source and **/etc/httpd/conf.d/ssl/** as the destination. Instruct the task to notify the **restart_services** handler when the file has been copied.

```
- unarchive:
    src: "{{ ssl_uri }}"
    dest: /etc/httpd/conf.d/ssl/
    copy: no
  notify:
    - restart_services
```

3.8. Add the last task that creates the **index.html** file under **/var/www/html/** on the managed host. The page should read as follows:

```
severb.lab.example.com (172.25.250.11) has been customized by Ansible
```

Use the two following Ansible facts to create the page: **ansible_fqdn**, and **ansible_default_ipv4.address**.

```
- copy:
    content: "{{ ansible_fqdn }} ({{ ansible_default_ipv4.address }}) has been
customized by Ansible\n"
    dest: /var/www/html/index.html
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3.9. Finally, make sure the block only runs if the *httpd* package is installed. To do so, add a **when** clause that parses the return code contained in the **rpm_check** registered variable.

```
  when:
    rpm_check.rc == 0
```

3.10. When completed, the file should read as follows:

```
---
- shell:
    rpm -q httpd
  register: rpm_check
  failed_when: rpm_check.rc == 1

- block:
  - get_url:
      url: "{{ https_uri }}"
      dest: /etc/httpd/conf.d/

  - file:
      path: /etc/httpd/conf.d/ssl
      state: directory
      mode: 0755

  - file:
      path: /var/www/html/logs
      state: directory
      mode: 0755

  - stat:
      path: /etc/httpd/conf.d/ssl.conf
    register: ssl_file

  - shell:
      mv /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.conf.bak
    when: ssl_file.stat.exists

  - unarchive:
      src: "{{ ssl_uri }}"
      dest: /etc/httpd/conf.d/ssl/
      copy: no
    notify:
      - restart_services

  - copy:
      content: "{{ ansible_fqdn }} ({{ ansible_default_ipv4.address }}) has been
 customized by Ansible\n"
      dest: /var/www/html/index.html

  when:
    rpm_check.rc == 0
```

4. **Defining tasks for the firewall**

Create the **configure_firewall.yml** task file. Start with a task that installs the package that the **fw_package** variable defines (the variable will be set in the main playbook). Create a task that starts the service specified by the **fw_service** variable.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Create a task that adds firewall rules for the **http** and **https** services using a loop. The rules should be applied immediately and persistently. Tag all tasks with the **production** tag.

4.1. The following steps will edit the single **configure_firewall.yml** file.

Define the task that uses the **yum** module to install latest version of the firewall service. Tag the task with the **production** tag. The task should read as follows:

```
---
- yum:
    name: "{{ fw_package }}"
    state: latest
  tags: production
```

4.2. Continue editing the **configure_firewall.yml** file. Add the task that starts the firewall service using the **fw_service** variable and tag it as **production**. The task should read as follows:

```
- service:
    name: "{{ fw_service }}"
    state: started
  tags: production
```

4.3. Write the task that uses the **firewalld** module to add the **http** and **https** service rules to the firewall. The rules should be applied immediately as well as persistently. Use a loop for the two rules. Tag the task as **production**.

```
- firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  with_items:
    - http
    - https
  tags: production
```

4.4. When completed, the file should read as follows:

```
---
- yum:
    name: "{{ fw_package }}"
    state: latest
  tags: production

- service:
    name: "{{ fw_service }}"
    state: started
  tags: production

- firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  with_items:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    - http
    - https
  tags: production
```

5. **Defining the main playbook**

   In the top-level project directory for this lab, create the main playbook, **playbook.yml**. The playbook should only apply to hosts in the **webservers** group. Define a block that imports the following three task files: **install_packages.yml**, **configure_web.yml**, and **configure_firewall.yml**.

   For the task that imports the **install_packages.yml** playbook, define the following variables: **memory** with a value of **256**, **web_package** with a value of **httpd** and **ssl_package** with a value of **mod_ssl**.

   For the task that imports the **configure_web.yml** file, define the following variables: **https_uri** with a value of **http://materials.example.com/task_control/https.conf**, and **ssl_uri** with a value of **http://materials.example.com/task_control/ssl.tar.gz**.

   For the task that imports the **configure_firewall.yml** playbook, add a condition to only import the tasks tagged with the **production** tag. Define the **fw_package** and **fw_service** variables with a value of **firewalld**.

   In the **rescue** clause for the block, define a task to install the **httpd** service. Notify the **restart_services** handler to start the service upon its installation.

   Add an **always** statement that uses the **shell** module to query the status of the **httpd** service using **systemctl**.

   Define the **restart_services** handle to restart both the **httpd** and **firewalld** services using a loop.

   5.1. The following steps will edit the single **playbook.yml** file.

   Create the **playbook.yml** playbook and start by targeting the hosts in the **webservers** host group.

```
---
- hosts: webservers
```

   5.2. Continue editing the **playbook.yml** file. Create a block for importing the three task files, using the **include** statement. For the first **include**, use **install_packages.yml** as the name of the file to import. Define the four variables required by the file:

   1. **memory**, with a value of **256**
   2. **web_package**, with a value of **httpd**
   3. **ssl_package**, with a value of **mod_ssl**
   4. **web_service**, with a value of **httpd**

   Add the following to the **playbook.yml** file:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  tasks:
    - block:
      - include: install_packages.yml
        vars:
          memory: 256
          web_package: httpd
          ssl_package: mod_ssl
          web_service: httpd
```

5.3. For the second **include**, use **configure_web.yml** as the name of the file to import. Define the two variables required by the file:

- **https_uri**, with a value of **http://materials.example.com/task_control/https.conf**
- **ssl_uri**, with a value of **http://materials.example.com/task_control/ssl.tar.gz**

Add the following to the **playbook.yml** file:

```
      - include: configure_web.yml
        vars:
          https_uri: http://materials.example.com/task_control/https.conf
          ssl_uri: http://materials.example.com/task_control/ssl.tar.gz
```

5.4. For the third **include**, use **configure_firewall.yml** as the name of the file to import. Define the variables required by the file, **fw_package** and **fw_service**, both with a value of **firewalld**. Import only the tasks that are tagged with **production**.

```
      - include: configure_firewall.yml
        vars:
          fw_package: firewalld
          fw_service: firewalld
        tags: production
```

5.5. Create the **rescue** clause for the block that installs the latest version of the *httpd* package and notifies the **restart_services** handler upon the package installation. Add a **debug** statement that reads:

```
Failed to import and run all the tasks; installing the web server manually
```

```
      rescue:
      - yum:
          name: httpd
          state: latest
        notify:
          - restart_services

      - debug:
          msg: "Failed to import and run all the tasks; installing the web
 server manually"
```

5.6. In the **always** clause, use the **shell** module to query the status of the **httpd** service, using **systemctl**. Add the following to the **playbook.yml** file:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      always:
      - shell:
          cmd: "systemctl status httpd"
```

5.7. Define the **restart_services** handler that uses a loop to restart both the **firewalld** and **httpd** services. Add the following to the **playbook.yml** file:

```
  handlers:
    - name: restart_services
      service:
        name: "{{ item }}"
        state: restarted
      with_items:
        - httpd
        - firewalld
```

5.8. When completed, the playbook should read as follows:

```
---
- hosts: webservers
  tasks:
    - block:
      - include: install_packages.yml
        vars:
          memory: 256
          web_package: httpd
          ssl_package: mod_ssl
          web_service: httpd
      - include: configure_web.yml
        vars:
          https_uri: http://materials.example.com/task_control/https.conf
          ssl_uri: http://materials.example.com/task_control/ssl.tar.gz
      - include: configure_firewall.yml
        vars:
          fw_package: firewalld
          fw_service: firewalld
        tags: production

      rescue:
      - yum:
          name: httpd
          state: latest
        notify:
          - restart_services

      - debug:
          msg: "Failed to import and run all the tasks; installing the web
 server manually"

      always:
      - shell:
          cmd: "systemctl status httpd"

  handlers:
    - name: restart_services
      service:
        name: "{{ item }}"
        state: restarted
      with_items:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        - httpd
        - firewalld
```

6. **Executing the `playbook.yml` playbook**

Run the **`playbook.yml`** playbook to set up the environment. Ensure the web server has been correctly configured by querying the home page of the web server (**`https://serverb.example.com`**) using **`curl`** with the **`-k`** option to allow insecure connections. The output should read as follows:

```
server.lab.example.com (172.25.250.11) has been customized by Ansible
```

6.1. Using the **`ansible-playbook`** command, run the **`playbook.yml`** playbook. The playbook should:

- Import and run the tasks that install the web server packages only if there is enough memory on the managed host.
- Import and run the tasks that configure SSL for the web server.
- Import and run the tasks that create the firewall rule for the web server to be reachable.

```
[student@workstation lab-task-control]$ ansible-playbook playbook.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]
...
RUNNING HANDLER [restart_services] *********************************************
changed: [serverb.lab.example.com] => (item=httpd)
changed: [serverb.lab.example.com] => (item=firewalld)

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=19   changed=13   unreachable=0    failed=0
```

6.2. Using **`curl`** to confirm the web page is available. The **`-k`** option allows you to bypass any SSL strict checking.

```
[student@workstation lab-task-control]$ curl -k https://serverb.lab.example.com
serverb.lab.example.com (172.25.250.11) has been customized by Ansible
```

Evaluation

Run the **`lab task-control grade`** command from *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab task-control grade
```

Cleanup

Run the **`lab task-control cleanup`** command to cleanup after the lab.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab task-control cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- *Loops* can be used to iterate over a set of values. They can be a list of items in sequence or random order, files, an autogenerated sequence of numbers, or other things
- *Conditionals* can be used to execute tasks or plays only when certain conditions have been met
- Conditions can be tested with various operators, including string comparisons, mathematical operators, and Boolean values
- *Handlers* are special tasks that execute at the end of the play if notified by other tasks
- *Tags* are used to mark certain tasks to be skipped or executed based on what tags a task has
- Tasks can be configured to handle error conditions by ignoring task failure, forcing handlers to be called even if the task failed, mark a task as failed when it succeeded, or override the behavior that causes a task to be marked as changed
- *Blocks* can be used to group tasks as a unit and execute other tasks depending on whether or not all the tasks in the block succeed

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 6

IMPLEMENTING JINJA2 TEMPLATES

Overview	
Goal	Implement a Jinja2 template
Objectives	• Describe Jinja2 templates • Implement Jinja2 templates
Sections	• Describing Jinja2 Templates (and Quiz) • Implementing Jinja2 Templates (and Guided Exercise)
Lab	• Implementing Jinja2 Templates

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Describing Jinja2 Templates

Objectives

After completing this section, students should be able to:

- Describe Jinja2 templates
- Describe the differences between YAML files and Jinja2 templates
- Describe variables, control structures, and comment use in Jinja2 templates

Introduction to Jinja2

Ansible uses the Jinja2 templating system to modify files before they are distributed to managed hosts. Generally speaking, it is preferable to avoid modifying configuration files through logic in templates. However, templates can be useful when systems need to have slightly modified versions of the same file. Ansible also uses Jinja2 to reference variables in playbooks.

Ansible allows Jinja2 loops and conditionals to be used in templates, but they are not allowed in playbooks. Ansible playbooks are completely machine-parseable YAML. This is an important feature, because it means it is possible to have code generate pieces of files, or to have other third-party tools read Ansible files. Not everyone will need this feature, but it can unlock interesting possibilities.

Delimiters

Variables or logic expressions are placed between tags, or delimiters. For example, Jinja2 templates use **{% *EXPR* %}** for expressions or logic (for example, loops), while **{{ *EXPR* }}** are used for outputting the results of an expression or a variable to the end user. The latter tag, when rendered, is replaced with a value or values, and are seen by the end user. Use **{# *COMMENT* #}** syntax to enclose comments.

In the following example the first line includes a comment that will not be included in the final file. The variable references in the second line are replaced with the values of the system facts being referenced.

```
{# /etc/hosts line #}
{{ ansible_default_ipv4.address }}    {{ ansible_hostname }}
```

Control Structures

Often the result of a play depends on the value of a variable, fact (something learned about the remote system), or previous task result. In some cases, the values of variables depend on other variables. Further, additional groups can be created to managed hosts based on whether the hosts match other criteria. There are many options to control execution flow in Ansible.

Loops

Jinja2 uses the **for** statement to provide looping functionality. In the following example, the **user** variable is replaced with all the values included in **users**.

```
{% for user in users %}
      {{ user }}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
{% endfor %}
```

The following **for** statement runs through all the values in the **users** variable, replacing **myuser** with each value, except when the value is **Snoopy**.

```
{# for statement #}
{% for myuser in users if not myuser == "Snoopy"%}
{{loop.index}} - {{ myuser }}
{% endfor %}
```

The **loop.index** variable expands to the index number that the loop is currently on. It has a value of 1 the first time the loop executes, and it increments by 1 through each iteration.

Conditionals

Jinja2 uses the **if** statement to provide conditional control. In the following example, the result is displayed if the **finished** variable is **True**.

```
{% if finished %}
    {{ result }}
{% endif %}
```

Variable Filters

Jinja2 provides filters which change the output format for template expressions (for example, to JSON). There are filters available for languages such as YAML or JSON. The **to_json** filter formats the expression output using JSON, and the **to_yaml** filter uses YAML as the formatting syntax.

```
{{ output | to_json }}
{{ output | to_yaml }}
```

Additional filters are available, such as the **to_nice_json** and **to_nice_yaml** filters, which format the expression output in either JSON or YAML human readable format.

```
{{ output | to_nice_json }}
{{ output | to_nice_yaml }}
```

Both the **from_json** and **from_yaml** filters expect a string in either JSON or YAML format and parse it.

```
{{ output | from_json }}
{{ output | from_yaml }}
```

The expressions used with **when** clauses in Ansible playbooks are Jinja2 expressions. Built-in Ansible filters that are used to test return values include **failed**, **changed**, **succeeded**, and **skipped**. The following task shows how filters can be used inside of conditional expressions.

```
tasks:
... Output omitted ...
  - debug: msg="the execution was aborted"
    when: returnvalue | failed
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Issues to be Aware of with YAML vs. Jinja2 in Ansible

The use of some Jinja2 expressions inside of a YAML playbook may change the meaning for those expressions, so they require some adjustments in the syntax used.

1. YAML syntax requires quotes when a value starts with a variable reference (**{{ }}**). The quotes prevent the parser from treating the expression as the start of a YAML dictionary. For example, the following playbook snippet will fail:

```
- hosts: app_servers
  vars:
    app_path: {{ base_path }}/bin
```

Instead, use the following syntax:

```
- hosts: app_servers
  vars:
    app_path: "{{ base_path }}/bin"
```

2. When there is a need to include nested **{{...}}** elements, the braces around the inner ones must be removed. Consider the following playbook snippet:

```
    - name: display the host value
      debug:
        msg: hostname = {{ params[ {{ host_ip }} ] }}, IPaddr = {{ host_ip }}
```

Ansible raises the following error when it tries to run it:

```
... Output omitted ...
TASK [display the host value] ************************************************
fatal: [localhost]: FAILED! => {"failed": true, "msg": "template error
 while templating string: expected token ':', got '}'. String: hostname =
 {{ params[ {{ host_ip }} ] }}, IPaddr = {{ host_ip }}"}
... Output omitted ...
```

Use the following syntax instead. It will run without error.

```
    - name: display the host value
      debug:
        msg: hostname = {{ params[ host_ip ] }}, IPaddr = {{ host_ip }}
```

Now the playbook will run without error.



References

Template Designer Documentation — Jinja2 Documentation
http://jinja.pocoo.org/docs/dev/templates/

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Describing Jinja2 Templates

Choose the correct answers to the following questions:

1. What is the main purpose of Jinja2 templates usage within Ansible?

   a. Modify files before they are distributed to managed hosts
   b. Provide an enhanced alternative to YAML for playbooks
   c. Add conditional and loop use inside of playbooks
   d. Support object use in templates

2. Which three features are included in the Jinja2 templates? (Choose three.)

   a. Variable filters
   b. Objects
   c. Loops
   d. Conditionals
   e. Iterators

3. Which two delimiters are allowed in Jinja2 templates? (Choose two.)

   a. {{ ... }} for variables
   b. {$ ... $} for expressions
   c. {% ... %} for expressions
   d. {@ ... @} for variables
   e. {(...)} for functions

4. Which three of the following filters are supported in Jinja2 templates? (Choose three.)

   a. to_nice_json
   b. from_nice_json
   c. to_nice_yaml
   d. from_nice_yaml
   e. to_yaml

5. Which two lines show valid Jinja2 usage in YAML? (Choose two.)

   a. remote_host: "{{ host_name }}"
   b. remote_host: {{ "host_name" }}
   c. remote_host: params[{{ host_name }}]
   d. remote_host: "{{ params[host_name] }}"

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answers to the following questions:

1. What is the main purpose of Jinja2 templates usage within Ansible?

   a. Modify files before they are distributed to managed hosts
   b. Provide an enhanced alternative to YAML for playbooks
   c. Add conditional and loop use inside of playbooks
   d. Support object use in templates

2. Which three features are included in the Jinja2 templates? (Choose three.)

   a. Variable filters
   b. Objects
   c. Loops
   d. Conditionals
   e. Iterators

3. Which two delimiters are allowed in Jinja2 templates? (Choose two.)

   a. {{ ... }} for variables
   b. {$ ... $} for expressions
   c. {% ... %} for expressions
   d. {@ ... @} for variables
   e. {(...)} for functions

4. Which three of the following filters are supported in Jinja2 templates? (Choose three.)

   a. to_nice_json
   b. from_nice_json
   c. to_nice_yaml
   d. from_nice_yaml
   e. to_yaml

5. Which two lines show valid Jinja2 usage in YAML? (Choose two.)

   a. remote_host: "{{ host_name }}"
   b. remote_host: {{ "host_name" }}
   c. remote_host: params[{{ host_name }}]
   d. remote_host: "{{ params[host_name] }}"

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Implementing Jinja2 Templates

Objectives

After completing this section, students should be able to:

- Build a template file
- Use the template file in a playbook

Building a Jinja2 template

A Jinja2 template is composed of multiple elements: data, variables and expressions. Those variables and expressions are replaced with their values when the Jinja2 template is rendered. The variables used in the template can be specified in the **vars** section of the playbook. It is possible to use the managed hosts' facts as variables on a template.



Note

Remember that the facts associated with a managed host can be obtained using the **ansible *system_hostname* -i *inventory_file* -m setup** command.

The following example shows how to create a template with variables using two of the facts retrieved by Ansible from managed hosts: **ansible_hostname** and **ansible_date_time.date**. When the associated playbook is executed, those two facts will be replaced by their values in the managed host being configured.



Note

A file containing a Jinja2 template does not need any specific file extension (for example, **.j2**).

```
Welcome to {{ ansible_hostname }}.
Today's date is: {{ ansible_date_time.date }}.
```

The following example uses the **for** statement, and assumes a **myhosts** variable has been defined in the inventory file being used. This variable would contain a list of hosts to be managed. With the following **for** statement, all hosts in the **myhosts** group from the inventory would be listed.

```
{% for myhost in groups['myhosts'] %}
{{ myhost }}
{% endfor %}
```

It is recommended practice to include at the beginning of a Jinja2 template the **ansible_managed** variable, in order to reflect that the file is managed by Ansible. This will be reflected into the file created in the managed host, with a string including the date, user ID and managed host hostname. This variable by default has the following value.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on
 {host}
```

To include the **ansible_managed** variable inside a Jinja2 template, use the following syntax:

```
{{ ansible_managed }}
```

Using Jinja2 Templates in Playbooks

Jinja2 templates are a powerful tool to customize configuration files to be deployed on the managed hosts. When the Jinja2 template for a configuration file has been created, it can be deployed to the managed hosts using the **template** module, which supports the transfer of a local file in the control node to the managed hosts.

To use the **template** module, use the following syntax. The value associated with the **src** key specifies the source Jinja2 template, and the value associated to the **dest** key specifies the file to be created on the destination hosts.

```
tasks:

    - name: template render
      template:
        src: /tmp/j2-template.j2
        dest: /tmp/dest-config-file.txt
```



Note

The template module allows you to configure, in the destination file, settings such as the owner, group or mode, or validate it against a command (for example, **visudo -c**)

As seen in previous chapters, thanks to Jinja2 syntax, it is also possible to use variables, with or without filters, inside the YAML definition of a playbook, including facts.



References

template - Templates a file out to a remote server — Ansible Documentation
http://docs.ansible.com/ansible/template_module.html

Variables — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_variables.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Implementing Jinja2 Templates

In this exercise, you will create a simple template file that delivers a custom motd file.

Outcomes
You should be able to:
- Build a template file.
- Use the template file in a playbook.

Before you begin
Log in to **workstation** as **student** using **student** as a password.

On **workstation**, run the **lab jinja2-implement setup** script. It checks if Ansible is installed on **workstation**, and creates the **/home/student/jinja2** directory, downloading the **ansible.cfg** file on it. It also downloads the **motd.yml**, **motd.j2**, and **inventory** files into the **/home/student/jinja2/files** directory.

```
[student@workstation ~]$ lab jinja2-implement setup
```



Note

All the files used along this exercise are available in **workstation** in the **/home/student/jinja2/files** directory.

Steps
1. On workstation, go to the **/home/student/jinja2** directory.

   ```
   [student@workstation ~]$ cd ~/jinja2/
   ```

2. Create the **inventory** file in the current directory. This file configures two groups: *webservers* and *workstations*. Include the system *servera.lab.example.com* in the *webservers* group, and the system *workstation.lab.example.com* in the *workstations* group.

   ```
   [webservers]
   servera.lab.example.com

   [workstations]
   workstation.lab.example.com
   ```

3. Create a template for the Message of the Day. Include it in the **motd.j2** file in the current directory. Include the following variables in the template:

   - **ansible_hostname** to retrieve the managed host hostname.
   - **ansible_date_time.date** for the managed host date.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- **system_owner** for the email of the owner of the system. This variable requires to be defined, with an appropriate value, in the **vars** section of the playbook template.

```
This is the system {{ ansible_hostname }}.
Today's date is: {{ ansible_date_time.date }}.
Only use this system with permission.
You can ask {{ system_owner }} for access.
```

4. Create a playbook in a new file in the current directory, named **motd.yml**. Define the **system_owner** variable in the **vars** section, and include a task for the *template* module, which maps the **motd.j2** Jinja2 template to the remote file **/etc/motd** in the managed hosts. Set the owner and group to *root*, and the mode to *0644*.

```
---
- hosts: all
  user: devops
  become: true
  vars:
    system_owner: clyde@example.com
  tasks:
    - template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```

5. Run the playbook included in the **motd.yml** file.

```
[student@workstation jinja2]$ ansible-playbook motd.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]
ok: [workstation.lab.example.com]

TASK [template] ****************************************************************
changed: [servera.lab.example.com]
changed: [workstation.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com      : ok=2    changed=1    unreachable=0    failed=0
workstation.lab.example.com  : ok=2    changed=1    unreachable=0    failed=0
```

6. Log in to **servera.lab.example.com** using the *devops* user, to verify the motd is displayed when logging in. Log out when you have finished.

```
[student@workstation jinja2]$ ssh devops@servera.lab.example.com
This is the system servera.
Today's date is: 2016-04-11.
Only use this system with permission.
You can ask clyde@example.com for access.
[devops@servera ~]# exit
Connection to servera.lab.example.com closed.
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Evaluation

From **workstation**, run the **lab jinja2-implement grade** script to confirm success on this exercise.

```
[student@workstation ~]$ lab jinja2-implement grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Jinja2 Templates

In this lab, you will create a file using a Jinja2 template on a playbook.

Outcomes
You should be able to:
- Build a template file.
- Use the template file in a playbook.

Before you begin
Log in to **workstation** as **student** using **student** as a password.

On **workstation**, run the **lab jinja2-lab setup** script. It checks if Ansible is installed on **workstation**, and creates the **/home/student/jinja2-lab** directory, downloading the **ansible.cfg** file on it. It also downloads the **motd.yml**, **motd.j2**, and **inventory** files into the **/home/student/jinja2-lab/files** directory.

```
[student@workstation ~]$ lab jinja2-lab setup
```



Note

All the files used in this exercise are available in **workstation** in the **/home/student/jinja2-lab/files** directory.

Steps
1. Create an inventory file, named **inventory**, in the **/home/student/jinja2-lab** directory. This inventory file defines the group **servers** which has the **serverb.lab.example.com** managed host associated to it.
2. Identify the facts in **serverb.lab.example.com** which show what is the status of the system memory.
3. Create a template for the Message of the Day, named **motd.j2**, in the current directory. Use the facts previously identified.
4. Create a new playbook file in the current directory, named **motd.yml**. Using the *template* module, configure the **motd.j2** Jinja2 template file previously created, to be mapped to the file **/etc/motd** in the managed hosts. This file has the *root* user as owner and group, and its permissions are *0644*. Configure the playbook so it uses the *devops* user, and setup the **become** parameter to be *true*.
5. Run the playbook included in the **motd.yml** file.
6. Check that the playbook included in the *motd.yml* file has been executed correctly.

Evaluation
From **workstation**, run the **lab jinja2-lab grade** script to confirm success on this exercise.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab jinja2-lab grade
```

Cleanup

From **workstation**, run the **lab jinja2-lab grade** script to confirm success on this exercise.

```
[student@workstation ~]$ lab jinja2-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will create a file using a Jinja2 template on a playbook.

Outcomes
You should be able to:
- Build a template file.
- Use the template file in a playbook.

Before you begin
Log in to **workstation** as **student** using **student** as a password.

On **workstation**, run the **lab jinja2-lab setup** script. It checks if Ansible is installed on **workstation**, and creates the **/home/student/jinja2-lab** directory, downloading the **ansible.cfg** file on it. It also downloads the **motd.yml**, **motd.j2**, and **inventory** files into the **/home/student/jinja2-lab/files** directory.

```
[student@workstation ~]$ lab jinja2-lab setup
```



Note

All the files used in this exercise are available in **workstation** in the **/home/student/jinja2-lab/files** directory.

Steps
1. Create an inventory file, named **inventory**, in the **/home/student/jinja2-lab** directory. This inventory file defines the group **servers** which has the **serverb.lab.example.com** managed host associated to it.

   1.1. On workstation, go to the **/home/student/jinja2-lab** directory.

   ```
   [student@workstation ~]$ cd ~/jinja2-lab/
   ```

   1.2. Create the **inventory** file in the current directory. This file configures one group: *servers*. Include the system *serverb.lab.example.com* in the *servers* group.

   ```
   [servers]
   serverb.lab.example.com
   ```

2. Identify the facts in **serverb.lab.example.com** which show what is the status of the system memory.

   2.1. Use the *setup* module to get a list of all the facts for the **serverb.lab.example.com** managed host. Both the **ansible_memfree_mb** and **ansible_memtotal_mb** facts provide information about the free memory and the total memory of the managed host.

   ```
   [student@workstation jinja2-lab]$ ansible serverb.lab.example.com -m setup
   serverb.lab.example.com | SUCCESS => {
       "ansible_facts": {
   ... Output omitted ...
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
 "ansible_memfree_mb": 741,
... Output omitted ...
 "ansible_memtotal_mb": 992,
... Output omitted ...
    },
    "changed": false
}
```

3. Create a template for the Message of the Day, named **motd.j2**, in the current directory. Use the facts previously identified.

   3.1. Create a new file, named **motd.j2**, in the current directory. Use both the **ansible_memfree_mb** and **ansible_memtotal_mb** facts variables to create a Message of the Day.

```
[student@workstation jinja2-lab]$ cat motd.j2
This system's total memory is: {{ ansible_memtotal_mb }} MBs.
The current free memory is: {{ ansible_memfree_mb }} MBs.
```

4. Create a new playbook file in the current directory, named **motd.yml**. Using the *template* module, configure the **motd.j2** Jinja2 template file previously created, to be mapped to the file **/etc/motd** in the managed hosts. This file has the *root* user as owner and group, and its permissions are *0644*. Configure the playbook so it uses the *devops* user, and setup the **become** parameter to be *true*.

   4.1. Create a new playbook file in the current directory, named **motd.yml**. Use the *template* module, configuring the **motd.j2** Jinja2 template file previously created in the **src** parameter, and **/etc/motd** in the **dest** parameter. Configure the **owner** and **group** parameters to be *root*, and the **mode** parameter to be *0644*. Use the *devops* user for the **user** parameter, and configure the **become** parameter to be *true*.

```
[student@workstation jinja2-lab]$ cat motd.yml
---
- hosts: all
  user: devops
  become: true
  tasks:
    - template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```

5. Run the playbook included in the **motd.yml** file.

   5.1. Run the playbook included in the **motd.yml** file.

```
[student@workstation jinja2-lab]$ ansible-playbook motd.yml
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]

TASK [template] ****************************************************************
changed: [serverb.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
serverb.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0
```

6. Check that the playbook included in the *motd.yml* file has been executed correctly.

   6.1. Log in to **serverb.lab.example.com** using the *devops* user, to verify the motd is displayed when logging in. Log out when you have finished.

```
[student@workstation jinja2-lab]$ ssh devops@serverb.lab.example.com
This system's total memory is: 992 MBs.
The current free memory is: 741 MBs.
[devops@serverb ~]$ logout
```

Evaluation

From **workstation**, run the **lab jinja2-lab grade** script to confirm success on this exercise.

```
[student@workstation ~]$ lab jinja2-lab grade
```

Cleanup

From **workstation**, run the **lab jinja2-lab grade** script to confirm success on this exercise.

```
[student@workstation ~]$ lab jinja2-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible uses the Jinja2 templating system to render files before they are distributed to managed hosts.
- YAML allows the use of Jinja2 based variables, with or without filters, inside the definition of a playbook.
- A Jinja2 template is usually composed of two elements: variables and expressions. Those variables and expressions are replaced with their values when the Jinja2 template is rendered.
- Filters in Jinja2 are a way of transforming template expressions from one kind of data into another.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 7

IMPLEMENTING ROLES

Overview	
Goal	Create and manage roles
Objectives	• Describe the structure and behavior of a role • Create a role • Deploy roles with Ansible Galaxy
Sections	• Describing Role Structure (and Quiz) • Creating Roles (and Guided Exercise) • Deploying Roles with Ansible Galaxy (and Guided Exercise)
Lab	• Implementing Roles

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Describing Role Structure

Objectives

After completing this section, students should be able to:

- Describe the structure and behavior of a role.
- Define role dependencies.

Structuring Ansible playbooks with roles

Data centers have a variety of different types of hosts. Some serve as web servers, others as database servers, and others can have software development tools installed and configured on them. An Ansible playbook, with tasks and handlers to handle all of these cases, would become large and complex over time. Ansible *roles* allow administrators to organize their playbooks into separate, smaller playbooks and files.

Roles provide Ansible with a way to load tasks, handlers, and variables from external files. Static files and templates can also be associated and referenced by a role. The files that define a role have specific names and are organized in a rigid directory structure, which will be discussed later. Roles can be written so they are general purpose and can be reused.

Use of Ansible roles has the following benefits:

- Roles group content, allowing easy sharing of code with others
- Roles can be written that define the essential elements of a system type: web server, database server, git repository, or other purpose
- Roles make larger projects more manageable
- Roles can be developed in parallel by different administrators

Examining the Ansible role structure

An Ansible role's functionality is defined by its directory structure. The top-level directory defines the name of the role itself. Some of the subdirectories contain YAML files, named **`main.yml`**. The **`files`** and **`templates`** subdirectories can contain objects referenced by the YAML files.

The following **`tree`** command displays the directory structure of the **`user.example`** role.

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Ansible role subdirectories

Subdirectory	Function
defaults	The **main.yml** file in this directory contains the default values of role variables that can be overwritten when the role is used.
files	This directory contains static files that are referenced by role tasks.
handlers	The **main.yml** file in this directory contains the role's handler definitions.
meta	The **main.yml** file in this directory contains information about the role, including author, license, platforms, and optional role dependencies.
tasks	The **main.yml** file in this directory contains the role's task definitions.
templates	This directory contains Jinja2 templates that are referenced by role tasks.
tests	This directory can contain an inventory and **test.yml** playbook that can be used to test the role.
vars	The **main.yml** file in this directory defines the role's variable values.

Defining variables and defaults

Role variables are defined by creating a **vars/main.yml** file with key: value pairs in the role directory hierarchy. They are referenced in the role YAML file like any other variable: **{{ *VAR_NAME* }}**. These variables have a high priority and can not be overridden by inventory variables.

Default variables allow default values to be set for variables of included or dependent roles. They are defined by creating a **defaults/main.yml** file with key: value pairs in the role directory hierarchy. Default variables have the lowest priority of any variables available. They can be easily overridden by any other variable, including inventory variables.

Define a specific variable in either **vars/main.yml** or **defaults/main.yml**, but not in both places. Default variables should be used when it is intended that their values will be overridden.

Using Ansible roles in a playbook

Using roles in a playbook are straightforward. The following example shows how to use Ansible roles.

```
---
- hosts: remote.example.com
  roles:
    - role1
    - role2
```

For each role specified, the role tasks, role handlers, role variables, and role dependencies will be included in the playbook, in that order. Any **copy**, **script**, **template**, or **include** tasks in the role can reference the relevant files, templates, or tasks without absolute or relative path names.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Ansible will look for them in the role's **files**, **templates**, or **tasks** respectively, based on their use.

The following example shows the alternative syntax for using a role in a playbook. **role1** is used in the same way as the previous example. Default variable values are overridden when **role2** is used.

```
---
- hosts: remote.example.com
  roles:
    - role: role1
    - role: role2
      var1: val1
      var2: val2
```

Defining role dependencies

Role dependencies allow a role to include other roles as dependencies in a playbook. For example, a role that defines a documentation server may depend upon another role that installs and configures a web server. Dependencies are defined in the **meta/main.yml** file in the role directory hierarchy.

The following is a sample **meta/main.yml** file.

```
---
dependencies:
  - { role: apache, port: 8080 }
  - { role: postgres, dbname: serverlist, admin_user: felix }
```

By default, roles are only added as a dependency to a playbook once. If another role also lists it as a dependency it will not be run again. This behavior can be overridden by setting the **allow_duplicates** variable to **yes** in the **meta/main.yml** file.

Controlling order of execution

Normally, the tasks of roles execute before the tasks of the playbooks that use them. Ansible provides a way of overriding this default behavior: the **pre_tasks** and **post_tasks** tasks. The **pre_tasks** tasks are performed before any roles are applied. The **post_tasks** tasks are performed after all the roles have completed.

```
---
- hosts: remote.example.com
  pre_tasks:
    - debug:
        msg: 'hello'
  roles:
    - role1
    - role2
  tasks:
    - debug:
        msg: 'still busy'
  post_tasks:
    - debug:
        msg: 'goodbye'
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Playbook Roles and Include Statements — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_roles.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Describing Role Structure

Choose the correct answer to the following questions:

1. Roles are:

    a. Configuration settings that allow specific users to run Ansible playbooks.
    b. Playbooks for a data center.
    c. Collection of YAML task files and supporting items arranged in a specific structure for easy sharing, portability, and reuse.

2. Which of the following can be specified in roles?

    a. Handlers
    b. Tasks
    c. Templates
    d. Variables
    e. All of the above

3. Which file declares role dependencies?

    a. The Ansible playbook that uses the role.
    b. The **`meta/main.yml`** file inside the role hierarchy.
    c. The **`meta/main.yml`** file in the project directory.
    d. Role dependencies cannot be defined in Ansible.

4. Which file in a role's directory hierarchy should contain the initial values of variables that might be used as parameters to the role?

    a. **`defaults/main.yml`**
    b. **`meta/main.yml`**
    c. **`vars/main.yml`**
    d. The host inventory file.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. Roles are:

   a. Configuration settings that allow specific users to run Ansible playbooks.
   b. Playbooks for a data center.
   c. Collection of YAML task files and supporting items arranged in a specific structure for easy sharing, portability, and reuse.

2. Which of the following can be specified in roles?

   a. Handlers
   b. Tasks
   c. Templates
   d. Variables
   e. All of the above

3. Which file declares role dependencies?

   a. The Ansible playbook that uses the role.
   b. The `meta/main.yml` file inside the role hierarchy.
   c. The `meta/main.yml` file in the project directory.
   d. Role dependencies cannot be defined in Ansible.

4. Which file in a role's directory hierarchy should contain the initial values of variables that might be used as parameters to the role?

   a. `defaults/main.yml`
   b. `meta/main.yml`
   c. `vars/main.yml`
   d. The host inventory file.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Creating Roles

Objectives

After completing this section, students should be able to:

- Create an Ansible role.
- Properly reference an Ansible role in a playbook.

Creating roles in Ansible requires no special development tools. Creating and using a role is a three step process:

1. Create the role directory structure.
2. Define the role content.
3. Use the role in a playbook.

Creating the role directory structure

Ansible looks for roles in a subdirectory called **roles** in the project directory. Roles can also be kept in the directories referenced by the **roles_path** variable in Ansible configuration files. This variable contains a colon-separated list of directories to search.

Each role has its own directory with specially named subdirectories. The following directory structure contains the files that define the **motd** role.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    ├── tasks
    │   └── main.yml
    └── templates
        └── motd.j2
```

The **files** subdirectory contains fixed-content files and the **templates** subdirectory contains templates that can be deployed by the role when it is used. The other subdirectories can contain **main.yml** files that define default variable values, handlers, tasks, role metadata, or variables, depending on the subdirectory they are in. If a subdirectory exists but is empty, such as **handlers** in the previous example, it is ignored. If a role does not utilize a feature, the subdirectory can be omitted altogether, for example the **meta** and **vars** subdirectories in the previous example.

Defining the role content

After the directory structure is created, the content of the Ansible role must be defined. A good place to start would be the ***ROLENAME*/tasks/main.yml** file. This file defines which modules to call on the managed hosts that this role is applied.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The following **tasks/main.yml** file manages the **/etc/motd** file on managed hosts. It uses the **template** module to copy the template named **motd.j2** to the managed host. The template is retrieved from the **templates** subdirectory of the role.

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file
  template:
    src: templates/motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

The following command displays the contents of the **templates/motd.j2** template of the **motd** role. It references Ansible facts and a **system-owner** variable.

```
[user@host ~]$ cat roles/motd/templates/motd.j2
This is the system {{ ansible_hostname }}.

Today's date is: {{ ansible_date_time.date }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

The role can define a default value for the **system_owner** variable. The **defaults/main.yml** file in the role's directory structure is where these values are set.

The following **defaults/main.yml** file sets the **system_owner** variable to **user@host.example.com**. This will be the email address that is written in the **/etc/motd** file of managed hosts that this role is applied to.

```
[user@host ~]$ cat roles/motd/defaults/main.yml
---
system_owner: user@host.example.com
```

Using the role in a playbook

To access a role, reference it in the **roles:** section of a playbook. The following playbook refers to the **motd** role. Because no variables are specified, the role will be applied with its default variable values.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true

  roles:
    - motd
```

When the playbook is executed, tasks performed because of a role can be identified by the role name prefix. The following sample output illustrates this with the **motd: deliver motd file** message.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[user@host ~]$ ansible-playbook -i inventory use-motd-role.yml

PLAY [use motd role playbook] **************************************************

TASK [setup] *******************************************************************
ok: [remote.example.com]

TASK [motd : deliver motd file] ************************************************
changed: [remote.example.com]

PLAY RECAP *********************************************************************
remote.example.com         : ok=2    changed=1    unreachable=0    failed=0
```

Changing a role's behavior with variables

Variables can be used with roles, like parameters, to override previously defined, default values. When they are referenced, the variable: value pairs must be specified as well.

The following example shows how to use the **motd** role with a different value for the **system_owner** role variable. The value specified, **someone@host.example.com**, will replace the variable reference when the role is applied to a managed host.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true

  roles:
    - role: motd
      system_owner: someone@host.example.com
```



References

Playbook Roles and Include Statements — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_roles.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Creating Roles

In this exercise, you will create two roles that use variables and parameters: **myvhost** and **myfirewall**.

The **myvhost** role will install and configure the Apache service on a host. A template is provided that will be used for **/etc/httpd/conf.d/vhost.conf**: **vhost.conf.j2**.

The **myfirewall** role installs, enables, and starts the **firewalld** daemon. It opens the firewall service port specified by the **firewall_service** variable.

Outcomes
You should be able to create Ansible roles that use variables, files, templates, tasks, and handlers to deploy a network service and enable a working firewall.

Before you begin
Reset **servera**.

From workstation, run the command **lab creating-roles setup** to prepare the environment for this exercise. This will create the working directory, called **dev-roles**, and populate it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab creating-roles setup
```

Steps

1. Log in to your **workstation** host as **student**. Change to the **dev-roles** working directory.

   ```
   [student@workstation ~]$ cd dev-roles
   [student@workstation dev-roles]$
   ```

2. Create the directory structure for a role called **myvhost**. The role will include fixed files, templates, tasks, and handlers. A dependency will be created later, so it should have a **meta** subdirectory.

   ```
   [student@workstation dev-roles]$ mkdir -p roles/myvhost/{files,handlers}
   [student@workstation dev-roles]$ mkdir roles/myvhost/{meta,tasks,templates}
   ```

3. Create the **main.yml** file in the **tasks** subdirectory of the role. The role should perform four tasks:

   - Install the *httpd* package.
   - Start and enable the **httpd** service.
   - Download the HTML content into the virtual host **DocumentRoot** directory.
   - Install the template configuration file that configures the webserver.

   3.1. Use a text editor to create a file called **roles/myvhost/tasks/main.yml**. Include code to use the **yum** module to install the *httpd* package. The file contents should look like the following:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
# tasks file for myvhost

- name: install httpd
  yum:
    name: httpd
    state: latest
```

3.2. Add additional code to the **tasks/main.yml** file to use the **service** module to start and enable the **httpd** service.

```
- name: start and enable httpd service
  service:
    name: httpd
    state: started
    enabled: true
```

3.3. Add another stanza to copy the HTML content from the role to the virtual host **DocumentRoot** directory. Use the **copy** module and include a trailing slash after the source directory name. This will cause the module to copy the contents of the **html** directory immediately below the destination directory (similar to **rsync** usage). The **ansible_hostname** variable will expand to the short host name of the managed host.

```
- name: deliver html content
  copy:
    src: html/
    dest: "/var/www/vhosts/{{ ansible_hostname }}"
```

3.4. Add another stanza to use the **template** module to create **/etc/httpd/conf.d/vhost.conf** on the managed host. It should call a handler to restart the **httpd** daemon when this file is updated.

```
- name: template vhost file
  template:
    src: vhost.conf.j2
    dest: /etc/httpd/conf.d/vhost.conf
    owner: root
    group: root
    mode: 0644
  notify:
    - restart httpd
```

3.5. Save your changes and exit the **tasks/main.yml** file.

4. Create the handler for restarting the **httpd** service. Use a text editor to create a file called **roles/myvhost/handlers/main.yml**. Include code to use the **service** module. The file contents should look like the following:

```
---
# handlers file for myvhost

- name: restart httpd
  service:
    name: httpd
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    state: restarted
```

5. Create the HTML content that will be served by the webserver.

   5.1. The role task that called the **copy** module referred to an **html** directory as the **src**. Create this directory below the **files** subdirectory of the role.

```
[student@workstation dev-roles]$ mkdir -p roles/myvhost/files/html
```

   5.2. Create an **index.html** file below that directory with the contents: "simple index". Be sure to use this string verbatim because the grading script looks for it.

```
[student@workstation dev-roles]$ echo 'simple index' > roles/myvhost/files/html/
index.html
```

6. Move the **vhost.conf.j2** template from the project directory to the role's **templates** subdirectory.

```
[student@workstation dev-roles]$ mv vhost.conf.j2 roles/myvhost/templates/
```

7. Test the **myvhost** role to make sure it works properly.

   7.1. Write a playbook that uses the role, called **use-vhost-role.yml**. It should have the following content:

```
---
- name: use vhost role playbook
  hosts: webservers

  pre_tasks:
    - debug:
        msg: 'Beginning web server configuration.'

  roles:
    - myvhost

  post_tasks:
    - debug:
        msg: 'Web server has been configured.'
```

   7.2. Run the playbook. Review the output to confirm that Ansible performed the actions on the web server, **servera**.

```
[student@workstation dev-roles]$ ansible-playbook use-vhost-role.yml

PLAY [use vhost role playbook] *************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [debug] *******************************************************************
ok: [servera.lab.example.com] => {
    "msg": "Beginning web server configuration."
}
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [myvhost : install httpd] *************************************************
changed: [servera.lab.example.com]

TASK [myvhost : start and enable httpd service] ********************************
changed: [servera.lab.example.com]

TASK [myvhost : deliver html content] ******************************************
changed: [servera.lab.example.com]

TASK [myvhost : template vhost file] *******************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [myvhost : restart httpd] **************************************
changed: [servera.lab.example.com]

TASK [debug] *******************************************************************
ok: [servera.lab.example.com] => {
    "msg": "Web server has been configured."
}

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
```

7.3. Run ad hoc commands to confirm that the role worked. The *httpd* package should be installed and the **`httpd`** service should be running.

```
[student@workstation dev-roles]$ ansible webservers -a 'yum list installed
 httpd'
servera.lab.example.com | SUCCESS | rc=0 >>
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
httpd.x86_64                        2.4.6-40.el7                        @rhel_dvd
[student@workstation dev-roles]$ ansible webservers -a 'systemctl is-active
 httpd'
servera.lab.example.com | SUCCESS | rc=0 >>
active
[student@workstation dev-roles]$ ansible webservers -a 'systemctl is-enabled
 httpd'
servera.lab.example.com | SUCCESS | rc=0 >>
enabled
```

7.4. The Apache configuration should be installed with template variables expanded.

```
[student@workstation dev-roles]$ ansible webservers -a 'cat /etc/httpd/conf.d/
vhost.conf'
servera.lab.example.com | SUCCESS | rc=0 >>
# Ansible managed: /home/student/dev-roles/roles/myvhost/templates/vhost.conf.j2
 modified on 2016-04-15 10:01:12 by student on workstation.lab.example.com

<VirtualHost *:80>
    ServerAdmin webmaster@servera.lab.example.com
    ServerName servera.lab.example.com
    ErrorLog logs/servera-error.log
    CustomLog logs/servera-common.log common
    DocumentRoot /var/www/vhosts/servera/

    <Directory /var/www/vhosts/servera/>
        Options +Indexes +FollowSymlinks +Includes
        Order allow,deny
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        Allow from all
    </Directory>
</VirtualHost>
```

7.5. The HTML content should be found in a directory called **/var/www/vhosts/servera**. The **index.html** file should contain the string "simple index".

```
[student@workstation dev-roles]$ ansible webservers -a 'cat /var/www/vhosts/
servera/index.html'
servera.lab.example.com | SUCCESS | rc=0 >>
simple index
```

7.6. Use a web browser on **servera** to check if the web content is available locally. It should succeed and display content.

```
[student@workstation dev-roles]$ ansible webservers -a 'curl -s http://
localhost'
servera.lab.example.com | SUCCESS | rc=0 >>
simple index
```

7.7. Use a web browser on **workstation** to check if content is available from **http://servera.lab.example.com**. If a firewall is running on **servera**, you will see the following error message:

```
[student@workstation dev-roles]$ curl -S http://servera.lab.example.com
curl: (7) Failed connect to servera.lab.example.com:80; No route to host
```

This is because the firewall port for HTTP is not open. If the web content successfully displays, it is because a firewall is not running on **servera**.

8. Create the role directory structure for a role called **myfirewall**.

```
[student@workstation dev-roles]$ mkdir -p roles/myfirewall/{defaults,handlers,tasks}
```

9. Create the **main.yml** file in the **tasks** subdirectory of the role. The role should perform three tasks:

- Install the *firewalld* package.
- Start and enable the **firewalld** service.
- Open a firewall service port.

9.1. Use a text editor to create a file called **roles/myfirewall/tasks/main.yml**. Include code to use the **yum** module to install the *firewalld* package. The file contents should look like the following:

```
---
# tasks file for myfirewall

- name: install firewalld
  yum:
    name: firewalld
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    state: latest
```

9.2. Add additional code to the **tasks/main.yml** file to use the **service** module to start and enable the **firewalld** service.

```
- name: start and enable firewalld service
  service:
    name: firewalld
    state: started
    enabled: true
```

9.3. Add another stanza to use the **firewalld** module to immediately, and persistently, open the service port specified by the **firewall_service** variable. It should look like the following:

```
- name: firewall services config
  firewalld:
    state: enabled
    immediate: true
    permanent: true
    service: "{{ firewall_service }}"
```

9.4. Save your changes and exit the **tasks/main.yml** file.

10. Create the handler for restarting the **firewalld** service. Use a text editor to create a file called **roles/myfirewall/handlers/main.yml**. Include code to use the **service** module. The file contents should look like the following:

```
---
# handlers file for myfirewall

- name: restart firewalld
  service:
    name: firewalld
    state: restarted
```

11. Create the file that defines the default value for the **firewall_service** variable. It should have a default value of **ssh** initially. We will override the value to open the port for **http** when we use the role in a later step.

    Use a text editor to create a file called **roles/myfirewall/defaults/main.yml**. It should contain the following content:

```
---
# defaults file for myfirewall

firewall_service: ssh
```

12. Modify the **myvhost** role to include the **myfirewall** role as a dependency, then retest the modified role.

    12.1. Use a text editor to create a file, called **roles/myvhost/meta/main.yml**, that makes **myvhost** depend on the **myfirewall** role. The **firewall_service** variable should

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

be set to **http** so the correct service port is opened. By using the role in this way, the default value of **ssh** for **firewall_service** will be ignored. The explicitly assigned value of **http** will be used instead.

The resulting file should look like the following:

```
---
dependencies:
  - { role: myfirewall, firewall_service: http }
```

12.2. Run the playbook again. Confirm the additional **myfirewall** tasks are successfully executed.

```
[student@workstation dev-roles]$ ansible-playbook use-vhost-role.yml

PLAY [use vhost role playbook] *************************************************

... Output omitted ...

TASK [myfirewall : install firewalld] ******************************************
ok: [servera.lab.example.com]

TASK [myfirewall : start and enable firewalld daemon] **************************
ok: [servera.lab.example.com]

TASK [myfirewall : firewall services config] ***********************************
changed: [servera.lab.example.com]

... Output omitted ...

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=11   changed=6    unreachable=0    failed=0
```

12.3. Confirm the web server content is available to remote clients.

```
[student@workstation dev-roles]$ curl http://servera.lab.example.com
simple index
```

Evaluation

From *workstation*, run the **lab creating-roles grade** command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab creating-roles grade
```

Cleanup

Run the **lab creating-roles cleanup** command to cleanup the managed host.

```
[student@workstation ~]$ lab creating-roles cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Deploying Roles with Ansible Galaxy

Objectives

After completing this section, students should be able to:

- Locate Ansible roles in the Ansible Galaxy website.
- Deploy roles with Ansible Galaxy.

Ansible Galaxy

Ansible Galaxy [https://galaxy.ansible.com] is a public library of Ansible roles written by a variety of Ansible administrators and users. It is an archive that contains thousands of Ansible roles and it has a searchable database that helps Ansible users identify roles that might help them accomplish an administrative task. Ansible Galaxy includes links to documentation and videos for new Ansible users and role developers.

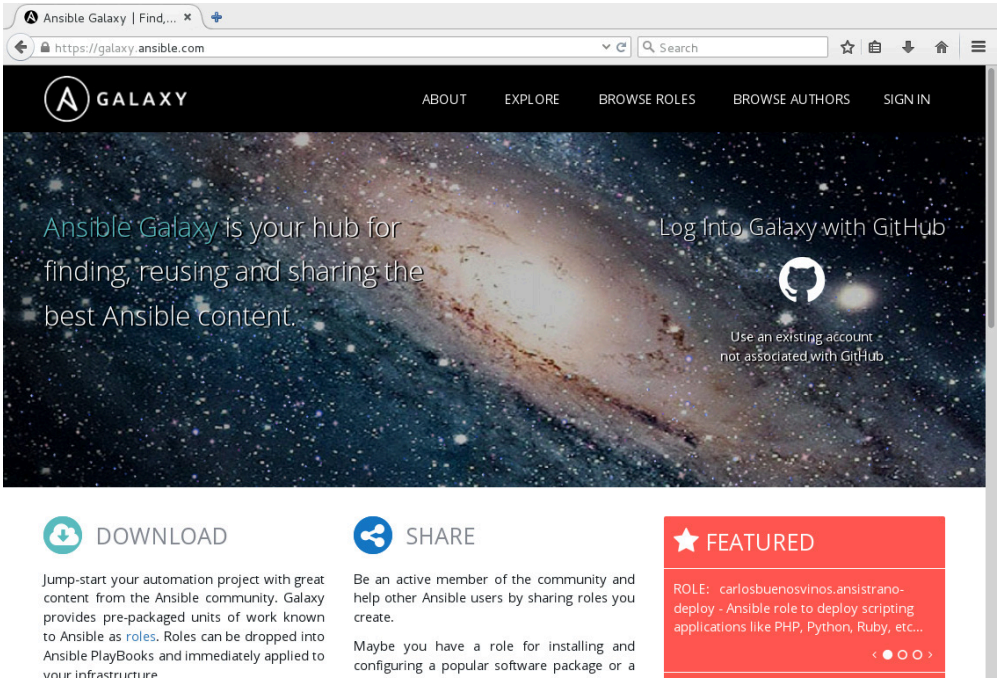


Figure 7.1: Ansible Galaxy home page

Accessing tips for using Ansible Galaxy

The **ABOUT** tab on the Ansible Galaxy website home page leads to a page that describes how to use Ansible Galaxy. There is content that describes how to download and use roles from Ansible Galaxy. Instructions on how to develop roles and upload them to Ansible Galaxy are also on that page.

There are counters associated with each role that is found on the Ansible Galaxy website. Users can vote for the usefulness of a role by clicking the star button. The number of stars a role has is a clue to its popularity among the Ansible community. Users can also watch a role. The number of watchers for a role give an indication of community interest in a role under development. Finally,

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

the number of times a role is downloaded from Ansible Galaxy is maintained with the role. This count is an indication of how many Ansible users actually use the role.

The **EXPLORE** tab on the Ansible Galaxy website home page shows the most active, the most starred, and the most watched roles on Ansible Galaxy. This page also displays the list of the top role authors and contributors on the site.

Browsing Ansible roles and authors
The **BROWSE ROLES** and **BROWSE AUTHORS** tabs on the Ansible Galaxy website home page give users access to information about the roles published on Ansible Galaxy. Users can search for an Ansible role by its name, or by other role attributes. The authors of roles published in Ansible Galaxy can be searched by name.

The following figure shows the search results that displayed after a keyword search was performed. The term "install git" was entered into the **Search roles** box.

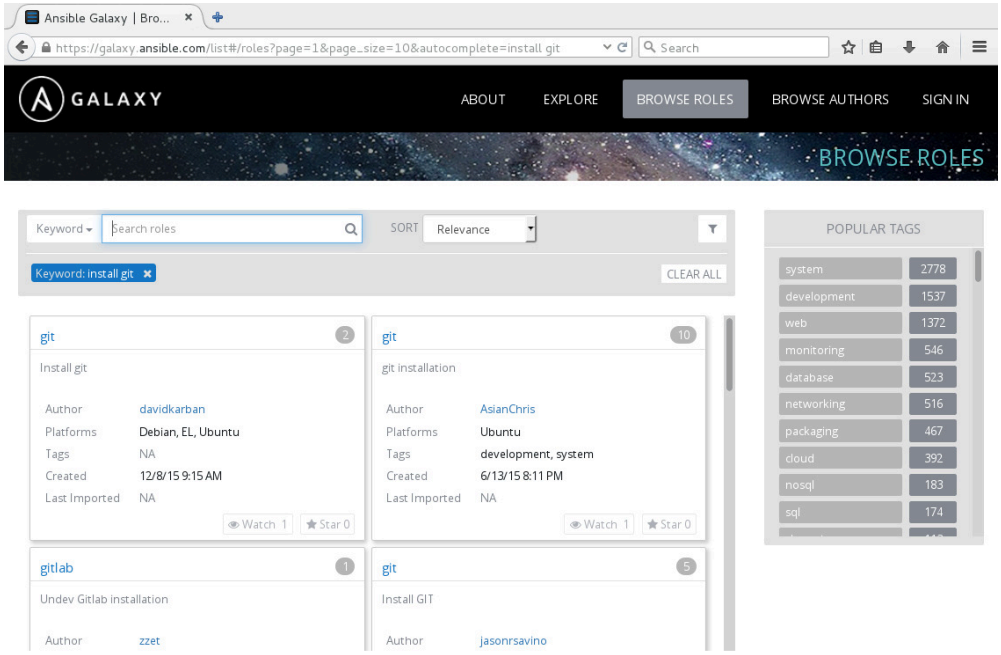


Figure 7.2: Ansible Galaxy search screen

The pulldown menu to the left of the **Search roles** box allow searches to be performed on keywords, author IDs, platform, and tags. Possible platform values include **EL** for Enterprise Linux, and **Fedora**. Tags are often assigned to roles to indicate their use in the data center. Possible tag values include **system**, **development**, **web**, **packaging**, and others.

The ansible-galaxy command-line tool

The **ansible-galaxy** command line tool can be used to search for, display information about, install, list, remove, or initialize roles.

Identifying and installing roles
The **ansible-galaxy search** subcommand searches Ansible Galaxy for the string specified as an argument. The **--author**, **--platforms**, and **--galaxy-tags** options can be used to narrow the search results. The following example displays the names of roles that include "install" and "git" in their description, and are available for the Enterprise Linux (**el**) platform.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[user@host ~]$ ansible-galaxy search 'install git' --platforms el

Found 77 roles matching your search:

 Name                                Description
 ----                                -----------
 zzet.gitlab                         Undev Gitlab installation
 jasonrsavino.git                    Install GIT
 samdoran.gitlab                     Install GitLab CE Omnibus
 kbrebanov.git                       Installs git
 AsianChris.git                      git installation
... Output omitted ...
```

The **ansible-galaxy info** subcommand displays more detailed information about a role. The following command displays information about the **davidkarban.git** role, available from Ansible Galaxy. Because the information requires more than one screen to display, **ansible-galaxy** uses **less** to display the role's information.

```
[user@host ~]$ ansible-galaxy info davidkarban.git
[DEPRECATION WARNING]: The comma separated role spec format, use the
yaml/explicit format instead.. This feature will be removed in a future release.
 Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less

Role: davidkarban.git
        description: Install git
        active: True
        commit:
        commit_message:
        commit_url:
        company: David Karban
        created: 2015-12-08T09:15:48.542Z
        download_count: 1
        forks_count: 0
        github_branch:
        github_repo: ansible-git
        github_user: davidkarban
        id: 6422
        is_valid: True
        issue_tracker_url: https://github.com/davidkarban/ansible-git/issues
        license: license (GPLv2, CC-BY, etc)
        min_ansible_version: 1.2
        modified: 2016-04-20T20:13:35.549Z
        namespace: davidkarban
        open_issues_count: 0
        path: /etc/ansible/roles
:
```

The **ansible-galaxy install** subcommand downloads a role from Ansible Galaxy, then installs it locally on the control node. The default installation location for roles is **/etc/ansible/roles**. This location can be overridden by either the value of the **role_path** configuration variable, or a **-p *DIRECTORY*** option on the command-line.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[user@host ~]$ ansible-galaxy install davidkarban.git -p roles/
- downloading role 'git', owned by davidkarban
- downloading role from https://github.com/davidkarban/ansible-git/archive/master.tar.gz
- extracting davidkarban.git to roles/davidkarban.git
- davidkarban.git was installed successfully
[user@host ~]$ ls roles/
davidkarban.git
```

Multiple roles can be installed with a single **ansible-galaxy install** command when a requirements file is specified with the **-r** option. The requirements file is a YAML file that specifies which roles to download and install. A **name** value can be used to determine what the role will be called locally.

There are many types of role sources that can be specified in a requirements file. The following example shows how to specify a download from Ansible Galaxy and another download from another web server.

```
[user@host ~]$ cat roles2install.yml
# From Galaxy
- src: author.rolename

# From a webserver, where the role is packaged in a gzipped tar archive
- src: https://webserver.example.com/files/sample.tgz
  name: ftpserver-role
[user@host ~]$ ansible-galaxy init -r roles2install.yml
```

Roles downloaded and installed from Ansible Galaxy can be used in playbooks like any other role. They are referenced in the **roles:** section using their full ***AUTHOR.NAME*** role name. The following **use-git-role.yml** playbook references the **davidkarban.git** role.

```
[user@host ~]$ cat use-git-role.yml
---
- name: use davidkarban.git role playbook
  hosts: remote.example.com
  user: devops
  become: true

  roles:
    - davidkarban.git
```

Using the playbook causes *git* to be installed on the **remote.example.com** server. The full role name, including the author, is displayed as its tasks are executed by the playbook.

```
[user@host ~]$ ssh remote.example.com yum list installed git
Loaded plugins: langpacks, search-disabled-repos
Error: No matching Packages to list
[user@host ~]$ ansible-playbook use-git-role.yml

PLAY [use davidkarban.git role playbook] ***********************************

TASK [setup] *******************************************************************
ok: [remote.example.com]

TASK [davidkarban.git : Load the OS specific variables] ************************
ok: [remote.example.com]

TASK [davidkarban.git : Install the packages in Redhat derivates] **************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
... Output omitted ...

PLAY RECAP *********************************************************************
remote.example.com         : ok=3    changed=1    unreachable=0    failed=0

[user@host ~]$ ssh remote.example.com yum list installed git
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
git.x86_64                       1.8.3.1-5.el7                       @rhel_dvd
```

Managing downloaded modules

The **ansible-galaxy** command can manage local roles. The roles are found in the **roles** directory of the current project, or they can be found in one of the directories listed in the **roles_path** variable. The **ansible-galaxy list** subcommand lists the roles that are found locally.

```
[user@host ~]$ ansible-galaxy list
- davidkarban.git, master
- student.bash_env, (unknown version)
```

A role can be removed locally with the **ansible-galaxy remove** subcommand.

```
[user@host ~]$ ansible-galaxy remove student.bash_env
- successfully removed student.bash_env
[user@host ~]$ ansible-galaxy list
- davidkarban.git, master
```

Using ansible-galaxy to create roles

The **ansible-galaxy init** command creates a directory structure for a new role that will be developed. The author and name of the role is specified as an argument to the command and it creates the directory structure in the current directory. **ansible-galaxy** interacts with the Ansible Galaxy website API when it performs most operations. The **--offline** option permits the **init** command to be used when Internet access is unavailable.

```
[user@host roles]$ ansible-galaxy init --offline student.example
- student.example was created successfully
[user@host roles]$ ls student.example/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```



References

Ansible Galaxy — Ansible Documentation
http://docs.ansible.com/ansible/galaxy.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Deploying Roles with Ansible Galaxy

In this exercise, you will use Ansible Galaxy to download and install an Ansible role. You will also use it to initialize the directory for a new role to be developed.

Outcomes
You should be able to use Ansible Galaxy to initialize a new Ansible role and download and install an existing role.

Before you begin
From workstation, run the command **`lab ansible-galaxy setup`** to prepare the environment for this exercise. This will create the working directory, called **`dev-roles`**, and populate it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab ansible-galaxy setup
```

Steps

1. Log in to your **`workstation`** host as **`student`**. Change to the **`dev-roles`** working directory.

   ```
   [student@workstation ~]$ cd dev-roles
   [student@workstation dev-roles]$
   ```

2. Launch your favorite text editor and create a requirements file, called **`install-roles.yml`**. It should contain the following content:

   ```
   ---
   # install-roles.yml

   - src: http://materials.example.com/roles-library/student.bash_env.tgz
     name: student.bash_env
   ```

   The **`src`** value specifies the URL where the existing Ansible role exists. The **`name`** value defines where to save the role locally.

3. Use the **`ansible-galaxy`** command to use the requirements file you just created to download and install the **`student.bash_env`** role.

   3.1. Display the **`roles`** subdirectory before the role has been installed, for comparison.

   ```
   [student@workstation dev-roles]$ ls roles/
   myfirewall  myvhost
   ```

   3.2. Use Ansible Galaxy to download and install the role. The **`-p roles`** option specifies the path to the directory where roles are stored locally. The **`-r install-roles.yml`** option specifies the requirements file listing the roles to download and install.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dev-roles]$ ansible-galaxy install -p roles -r install-
roles.yml
- downloading role from http://materials.example.com/roles-library/
student.bash_env.tgz
- extracting student.bash_env to roles/student.bash_env
- student.bash_env was installed successfully
```

3.3. Display the **roles** subdirectory after the role has been installed. Confirm it has a new subdirectory, called **student.bash_env**, matching the **name** value specified in the YAML file.

```
[student@workstation dev-roles]$ ls roles/
myfirewall  myvhost  student.bash_env
```

4. Create a playbook, called **use-bash_env-role.yml**, that uses the **student.bash_env** role. It should execute on the **webservers** host group as the **devops** user. The contents of the playbook should look like the following:

```
---
- name: use student.bash_env role playbook
  hosts: webservers
  user: devops
  become: true

  roles:
    - student.bash_env
```

5. Run the playbook. The **student.bash_env** role creates standard template configuration files in **/etc/skel** on the managed host. The files it creates include **.bashrc**, **.bash_profile**, and **.vimrc**.

```
[student@workstation dev-roles]$ ansible-playbook -i inventory use-bash_env-role.yml

PLAY [use student.bash_env role playbook] ***********************************

TASK [setup] ****************************************************************
ok: [servera.lab.example.com]

TASK [student.bash_env : put away .bashrc] **********************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bash_profile] ****************************
ok: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ***********************************
changed: [servera.lab.example.com]

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=4    changed=2    unreachable=0    failed=0
```

6. Compare the template files in the **student.bash_env** role with the files on the managed host. They will differ when the templates have variables that will be expanded on the managed host.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6.1. Use the **md5sum** command to generate checksums of the templates.

```
[student@workstation dev-roles]$ md5sum roles/student.bash_env/templates/*
f939eb71a81a9da364410b799e817202  roles/student.bash_env/templates/
_bash_profile.j2
989764ae6830691e7468599db8d194ba  roles/student.bash_env/templates/_bashrc.j2
a7320d70317cd23e0c2083489b532cdf  roles/student.bash_env/templates/_vimrc.j2
```

6.2. Display the MD5 checksums of the files on the managed host, **servera**, for comparison.

```
[student@workstation dev-roles]$ ssh servera md5sum /etc/skel/
{.bash_profile,.bashrc,.vimrc}
f939eb71a81a9da364410b799e817202  /etc/skel/.bash_profile
7f6d35286702531c9bef441516334404  /etc/skel/.bashrc
a7320d70317cd23e0c2083489b532cdf  /etc/skel/.vimrc
```

The sums are the same for **.bash_profile** and **.vimrc**.

6.3. Display the contents of the **_bashrc.j2** template. It uses a variable for some of its content, and that is why it is different from the file on the managed host.

```
[student@workstation dev-roles]$ cat roles/student.bash_env/templates/_bashrc.j2
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging
 feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

PS1="{{ default_prompt }}"
```

6.4. Display the last few lines of the **/etc/skel/.bashrc** file on **servera**. You should see that it has a line that defines a default **PS1** prompt that came from the template.

```
[student@workstation dev-roles]$ ssh servera tail -n5 /etc/skel/.bashrc

# User specific aliases and functions

PS1="[student prompt \W]\$ "
```

7. The **ansible-galaxy** command can be used to initialize the subdirectories for a new role. When creating a new role, instead of creating the directory structure by hand, **ansible-galaxy init --offline** will do the work for you.

7.1. Use the **ansible-galaxy** command with the **init** subcommand to create a new role called **empty.example**. The **-p roles** option specifies the path to the directory where roles are stored locally.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation dev-roles]$ ansible-galaxy init --offline -p roles
 empty.example
- empty.example was created successfully
```

7.2. A new subdirectory is created in the **roles** directory for the new role that is being defined.

```
[student@workstation dev-roles]$ ls roles/
empty.example  myfirewall  myvhost  student.bash_env
```

7.3. List the directory that was created for the new role. It has all of the subdirectories that could be used in a role definition.

```
[student@workstation dev-roles]$ ls roles/empty.example/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
[student@workstation dev-roles]$ ls roles/empty.example/*
roles/empty.example/README.md

roles/empty.example/defaults:
main.yml

roles/empty.example/files:

roles/empty.example/handlers:
main.yml

roles/empty.example/meta:
main.yml

roles/empty.example/tasks:
main.yml

roles/empty.example/templates:

roles/empty.example/tests:
inventory  test.yml

roles/empty.example/vars:
main.yml
```

7.4. Display the contents of the **tasks/main.yml** file of the new role. Notice that it is a simple YAML stub with a comment with the role's name. All of the YAML files created by Ansible Galaxy contain similar content.

```
[student@workstation dev-roles]$ cat roles/empty.example/tasks/main.yml
---
# tasks file for empty.example
```

Evaluation

From *workstation*, run the **lab ansible-galaxy grade** command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-galaxy grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Cleanup

Run the **lab ansible-galaxy cleanup** command to cleanup the managed host.

```
[student@workstation ~]$ lab ansible-galaxy cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Roles

In this lab, you will create two roles that use variables and parameters: **student.myenv** and **myapache**.

The **student.myenv** role customizes a system with required packages and a helpful script for all users. It will customize a user account, specified by the **myenv_user** variable, with a profile picture and an extra command alias in their **~/.bashrc** file.

The **myapache** role will install and configure the Apache service on a host. Two templates are provided which will be used for the **/etc/httpd/conf/httpd.conf** and the **/var/www/html/index.html** files: **apache_httpdconf.j2** and **apache_indexhtml.j2** respectively.

Outcomes
You should be able to create Ansible roles that use variables, files, templates, tasks, and handlers to customize user environments and deploy a network service.

Before you begin
Prepare your systems for this exercise by running the **lab ansible-roles-lab setup** command on your **workstation** system. This will create the working directory, called **lab-roles**, and populate it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab ansible-roles-lab setup
```

Steps

1. Log in to your **workstation** host as **student**. Change to the **lab-roles** working directory.

   ```
   [student@workstation ~]$ cd lab-roles
   [student@workstation lab-roles]$
   ```

2. Create directories to contain the Ansible roles. They should be contained in the **student.myenv** and **myapache** directories, below **~student/lab-roles/roles** on **workstation**.

3. Install the **mkcd.sh.j2** file as a template for the **student.myenv** role.

4. Make **/usr/share/icons/hicolor/48x48/apps/system-logo-icon.png** available as a static profile image for the **student.myenv** role. It will ultimately be called **profile.png** on the managed host in a user's home directory.

5. Define **student.myenv** role tasks to perform the following steps:

   - Install packages defined by the **myenv_packages** variable.
   - Copy the standard profile picture to the user's home directory as **~/profile.png**. Use the **myenv_user** variable for the user name.
   - Add the following line to the user's **~/.bashrc** file: **alias tree='tree -C'**. Use the **myenv_user** variable for the user name. Hint: The **lineinfile** module might be well suited for this task.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Install the **mkcd.sh.j2** template as **/etc/profile.d/mkcd.sh**. It should have user:group ownership of **root**:**root** and have **-rw-r--r--** permissions.

The role should fail with an error message when the **myenv_user** variable is an empty string.

6. Define the **myenv_packages** variable for the **student.myenv** role so it contains the following packages: *git*, *tree*, and *vim-enhanced*.

7. Assign the empty string as the default value for the **myenv_user** variable.

8. Create a playbook, called **myenv.yml**, that runs on all hosts. It should use the **student.myenv** role, but do not set the **myenv_user** variable. Test the **student.myenv** role and confirm that it fails.

9. Update the **myenv.yml** playbook so that it uses the **student.myenv** role, setting the **myenv_user** variable to **student**. Test the **student.myenv** role and confirm that it works properly.

10. Install the Apache-related Jinja2 template files, in the **lab-roles** project directory, to the **myapache** role.

11. Create a handler that will restart the **httpd** service.

12. Define **myapache** role tasks to perform the following steps:

    - Install the *httpd* and *firewalld* packages.
    - Copy the **apache_httpdconf.j2** template to **/etc/httpd/conf/httpd.conf**. The target file should be owned by **root** with **-r--r--r--** permissions. Restart Apache using the handler created previously.
    - Copy the **apache_indexhtml.j2** template to **/var/www/html/index.html**. The target file should be owned by **root** with **-r--r--r--** permissions.
    - Start and enable the **httpd** and **firewalld** services.
    - Open port 80/tcp on the firewall.

    Package installation should always occur when this role is used, but the other tasks should only occur when the **apache_enable** variable is set to **true**. The role should restart the Apache service when the configuration file is updated.

13. Create the default variable values for the **myapache** role. The **apache_enable** variable should have a default value of **false**.

14. Create a playbook, called **apache.yml**, that runs on **serverb**. It should use the **myapache** role, but use the default value of the **apache_enable** variable. Test the **myapache** role and confirm that it installs the packages, but does not deploy the web server.

15. Modify the **apache.yml** playbook so that it uses the **myapache** role, setting the **apache_enable** variable to **true**. Test the **myapache** role and confirm that it works properly.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Evaluation
Grade your work by running the **lab ansible-roles-lab grade** command from your **workstation** machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-roles-lab grade
```

Cleanup
Run the **lab ansible-roles-lab cleanup** command to cleanup the managed host.

```
[student@workstation ~]$ lab ansible-roles-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will create two roles that use variables and parameters: **student.myenv** and **myapache**.

The **student.myenv** role customizes a system with required packages and a helpful script for all users. It will customize a user account, specified by the **myenv_user** variable, with a profile picture and an extra command alias in their **~/.bashrc** file.

The **myapache** role will install and configure the Apache service on a host. Two templates are provided which will be used for the **/etc/httpd/conf/httpd.conf** and the **/var/www/html/index.html** files: **apache_httpdconf.j2** and **apache_indexhtml.j2** respectively.

Outcomes

You should be able to create Ansible roles that use variables, files, templates, tasks, and handlers to customize user environments and deploy a network service.

Before you begin

Prepare your systems for this exercise by running the **lab ansible-roles-lab setup** command on your **workstation** system. This will create the working directory, called **lab-roles**, and populate it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab ansible-roles-lab setup
```

Steps

1. Log in to your **workstation** host as **student**. Change to the **lab-roles** working directory.

   ```
   [student@workstation ~]$ cd lab-roles
   [student@workstation lab-roles]$
   ```

2. Create directories to contain the Ansible roles. They should be contained in the **student.myenv** and **myapache** directories, below **~student/lab-roles/roles** on **workstation**.

   The **ansible-galaxy** command can be used to create the subdirectories for the roles.

   ```
   [student@workstation lab-roles]$ ansible-galaxy init --offline -p roles
    student.myenv
   - student.myenv was created successfully
   [student@workstation lab-roles]$ ansible-galaxy init --offline -p roles myapache
   - myapache was created successfully
   ```

3. Install the **mkcd.sh.j2** file as a template for the **student.myenv** role.

   The lab setup script copied the file to the **lab-roles** working directory. Move it to the **roles/student.myenv/templates/** subdirectory.

   ```
   [student@workstation lab-roles]$ mv mkcd.sh.j2 roles/student.myenv/templates/
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4. Make **/usr/share/icons/hicolor/48x48/apps/system-logo-icon.png** available as a static profile image for the **student.myenv** role. It will ultimately be called **profile.png** on the managed host in a user's home directory.

   Copy the file, renaming it to **roles/student.myenv/files/profile.png**.

```
[student@workstation lab-roles]$ cp /usr/share/icons/hicolor/48x48/apps/system-logo-
icon.png roles/student.myenv/files/profile.png
```

5. Define **student.myenv** role tasks to perform the following steps:

   - Install packages defined by the **myenv_packages** variable.
   - Copy the standard profile picture to the user's home directory as **~/profile.png**. Use the **myenv_user** variable for the user name.
   - Add the following line to the user's **~/.bashrc** file: **alias tree='tree -C'**. Use the **myenv_user** variable for the user name. Hint: The **lineinfile** module might be well suited for this task.
   - Install the **mkcd.sh.j2** template as **/etc/profile.d/mkcd.sh**. It should have user:group ownership of **root**:**root** and have **-rw-r--r--** permissions.

   The role should fail with an error message when the **myenv_user** variable is an empty string.

   Modify **roles/student.myenv/tasks/main.yml** so that it contains the following:

```
---
# tasks file for student.myenv

- name: check myenv_user default
  fail:
    msg: You must specify the variable `myenv_user` to use this role!
  when: "myenv_user == ''"

- name: install my packages
  yum:
    name: "{{ item }}"
    state: installed
  with_items: "{{ myenv_packages }}"

- name: copy placeholder profile pic
  copy:
    src: profile.png
    dest: "~{{ myenv_user }}/profile.png"

- name: set an alias in `.bashrc`
  lineinfile:
    line: "alias tree='tree -C'"
    dest: "~{{ myenv_user }}/.bashrc"

- name: template out mkcd function
  template:
    src: mkcd.sh.j2
    dest: /etc/profile.d/mkcd.sh
    owner: root
    group: root
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    mode: 0644
```

6. Define the **myenv_packages** variable for the **student.myenv** role so it contains the following packages: *git*, *tree*, and *vim-enhanced*.

   Create **roles/student.myenv/vars/main.yml** with the following contents:

```
---
# vars file for student.myenv

myenv_packages:
  - 'git'
  - 'tree'
  - 'vim-enhanced'
```

7. Assign the empty string as the default value for the **myenv_user** variable.

   Create **roles/student.myenv/defaults/main.yml** with the following contents:

```
---
# defaults file for student.myenv

myenv_user: ''
```

8. Create a playbook, called **myenv.yml**, that runs on all hosts. It should use the **student.myenv** role, but do not set the **myenv_user** variable. Test the **student.myenv** role and confirm that it fails.

   8.1. Use a text editor to create the **myenv.yml** playbook. It should look like the following:

```
---
- name: setup my personal environment
  hosts: all
  roles:
    - student.myenv
```

   8.2. Run the **myenv.yml** playbook. Check the **ansible-playbook** output to make sure it fails.

```
[student@workstation lab-roles]$ ansible-playbook myenv.yml

PLAY [setup my personal environment] ******************************************

TASK [setup] ******************************************************************
ok: [tower.lab.example.com]
ok: [serverb.lab.example.com]

TASK [student.myenv : check myenv_user default] *******************************
fatal: [tower.lab.example.com]: FAILED! => {"changed": false, "failed":
 true, "msg": "You must specify the variable `myenv_user` to use this
 role!"}
fatal: [serverb.lab.example.com]: FAILED! => {"changed": false,
 "failed": true, "msg": "You must specify the variable `myenv_user`
 to use this role!"}

NO MORE HOSTS LEFT ************************************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        to retry, use: --limit @myenv.retry

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=1    changed=0    unreachable=0    failed=1
tower.lab.example.com      : ok=1    changed=0    unreachable=0    failed=1
```

9. Update the **myenv.yml** playbook so that it uses the **student.myenv** role, setting the **myenv_user** variable to **student**. Test the **student.myenv** role and confirm that it works properly.

   9.1. Use a text editor to modify the **myenv.yml** playbook. It should look like the following:

```
---
- name: setup my personal environment
  hosts: all
  roles:
    - role: student.myenv
      myenv_user: student
```

   9.2. Run the **myenv.yml** playbook. Check the **ansible-playbook** output to make sure the tasks ran properly.

```
[student@workstation lab-roles]$ ansible-playbook myenv.yml

PLAY [setup my personal environment] *******************************************

TASK [setup] *******************************************************************
ok: [tower.lab.example.com]
ok: [serverb.lab.example.com]

TASK [student.myenv : check myenv_user default] ********************************
skipping: [serverb.lab.example.com]
skipping: [tower.lab.example.com]

TASK [student.myenv : install my packages] *************************************
changed: [serverb.lab.example.com] => (item=[u'vim-enhanced', u'tree', u'git'])
changed: [tower.lab.example.com] => (item=[u'vim-enhanced', u'tree', u'git'])

TASK [student.myenv : copy placeholder profile pic] ****************************
changed: [serverb.lab.example.com]
changed: [tower.lab.example.com]

TASK [student.myenv : set an alias in `.bashrc`] *******************************
changed: [tower.lab.example.com]
changed: [serverb.lab.example.com]

TASK [student.myenv : template out mkcd function] ******************************
changed: [tower.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=5    changed=4    unreachable=0    failed=0
tower.lab.example.com      : ok=5    changed=4    unreachable=0    failed=0
```

10. Install the Apache-related Jinja2 template files, in the **lab-roles** project directory, to the **myapache** role.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

10.1. The lab setup script copied the templates in the **lab-roles** working directory. Move them to the **roles/myapache/templates/** subdirectory.

```
[student@workstation lab-roles]$ mv apache_*.j2 roles/myapache/templates
```

10.2. List the **roles/myapache/templates/** subdirectory to confirm the templates are in place.

```
[student@workstation lab-roles]$ ls roles/myapache/templates
apache_httpdconf.j2  apache_indexhtml.j2
```

11. Create a handler that will restart the **httpd** service.

Modify **roles/myapache/handlers/main.yml** so that it contains the following:

```
---
# handlers file for myapache

- name: restart apache
  service:
    name: httpd
    state: restarted
```

12. Define **myapache** role tasks to perform the following steps:

- Install the *httpd* and *firewalld* packages.
- Copy the **apache_httpdconf.j2** template to **/etc/httpd/conf/httpd.conf**. The target file should be owned by **root** with **-r--r--r--** permissions. Restart Apache using the handler created previously.
- Copy the **apache_indexhtml.j2** template to **/var/www/html/index.html**. The target file should be owned by **root** with **-r--r--r--** permissions.
- Start and enable the **httpd** and **firewalld** services.
- Open port 80/tcp on the firewall.

Package installation should always occur when this role is used, but the other tasks should only occur when the **apache_enable** variable is set to **true**. The role should restart the Apache service when the configuration file is updated.

Modify **roles/myapache/tasks/main.yml** so that it contains the following:

```
---
# tasks file for myapache

- name: install apache package
  yum:
    name: httpd
    state: latest

- name: install firewalld package
  yum:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    name: firewalld
    state: latest

- name: template out apache configuration file
  template:
    src: apache_httpdconf.j2
    dest: /etc/httpd/conf/httpd.conf
    owner: root
    group: root
    mode: 0444
  notify:
    - restart apache
  when: apache_enable

- name: template out apache index.html
  template:
    src: apache_indexhtml.j2
    dest: /var/www/html/index.html
    owner: root
    group: root
    mode: 0444
  when: apache_enable

- name: start and enable apache daemon
  service:
    name: httpd
    state: started
    enabled: true
  when: apache_enable

- name: start and enable firewalld daemon
  service:
    name: firewalld
    state: started
    enabled: true
  when: apache_enable

- name: open http firewall port
  firewalld:
    port: 80/tcp
    immediate: true
    permanent: true
    state: enabled
  when: apache_enable
```

13. Create the default variable values for the **myapache** role. The **apache_enable** variable should have a default value of **false**.

    Modify **roles/myapache/defaults/main.yml** so it contains the following:

```
---
# defaults file for myapache

apache_enable: false
```

14. Create a playbook, called **apache.yml**, that runs on **serverb**. It should use the **myapache** role, but use the default value of the **apache_enable** variable. Test the **myapache** role and confirm that it installs the packages, but does not deploy the web server.

    14.1. Use a text editor to create the **apache.yml** playbook. It should look like the following:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
- name: setup apache on serverb.lab.example.com
  hosts: serverb.lab.example.com
  roles:
    - myapache
```

14.2. Run the **apache.yml** playbook. Check the **ansible-playbook** output to make sure it installs the needed packages, but skips the remaining tasks.

```
[student@workstation lab-roles]$ ansible-playbook apache.yml

PLAY [setup apache on serverb.lab.example.com] *********************************

TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]

TASK [myapache : install apache package] ***************************************
ok: [serverb.lab.example.com]

TASK [myapache : install firewalld package] ************************************
ok: [serverb.lab.example.com]

TASK [myapache : template out apache configuration file] ***********************
skipping: [serverb.lab.example.com]

TASK [myapache : template out apache index.html] *******************************
skipping: [serverb.lab.example.com]

TASK [myapache : start and enable apache daemon] *******************************
skipping: [serverb.lab.example.com]

TASK [myapache : open http firewall port] **************************************
skipping: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
```

15. Modify the **apache.yml** playbook so that it uses the **myapache** role, setting the **apache_enable** variable to **true**. Test the **myapache** role and confirm that it works properly.

15.1. Use a text editor to modify the **apache.yml** playbook. It should look like the following:

```
---
- name: setup apache on serverb.lab.example.com
  hosts: serverb.lab.example.com
  roles:
    - role: myapache
      apache_enable: true
```

15.2. Run the **apache.yml** playbook. Check the **ansible-playbook** output to make sure the tasks ran properly.

```
[student@workstation lab-roles]$ ansible-playbook apache.yml

PLAY [setup apache on serverb.lab.example.com] *********************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]

TASK [myapache : install apache package] ***************************************
ok: [serverb.lab.example.com]

TASK [myapache : install firewalld package] ************************************
ok: [serverb.lab.example.com]

TASK [myapache : template out apache configuration file] ***********************
changed: [serverb.lab.example.com]

TASK [myapache : template out apache index.html] *******************************
changed: [serverb.lab.example.com]

TASK [myapache : start and enable apache daemon] *******************************
ok: [serverb.lab.example.com]

TASK [myapache : start and enable firewalld daemon] ****************************
ok: [serverb.lab.example.com]

TASK [myapache : open http firewall port] **************************************
changed: [serverb.lab.example.com]

RUNNING HANDLER [myapache : restart apache] ************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=9    changed=4    unreachable=0    failed=0
```

15.3. Use a web browser to confirm that **serverb** is serving web content.

```
[student@workstation lab-roles]$ curl -s http://serverb.lab.example.com
<!-- Ansible managed: /home/student/lab-roles/roles/myapache/templates/
apache_indexhtml.j2 modified on 2016-04-11 15:38:19 by student on
 workstation.lab.example.com -->
<h2>Apache is running!</h2>
```

Evaluation

Grade your work by running the **lab ansible-roles-lab grade** command from your **workstation** machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-roles-lab grade
```

Cleanup

Run the **lab ansible-roles-lab cleanup** command to cleanup the managed host.

```
[student@workstation ~]$ lab ansible-roles-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Roles organize Ansible tasks in a way that allows reuse and sharing.
- Role variables should be defined in **`defaults/main.yml`** when they will be used as parameters, otherwise they should be defined in **`vars/main.yml`**.
- Role dependencies can be defined in the **`dependencies`** section of the **`meta/main.yml`** file of a role.
- Tasks that are to be applied before and after roles are included with the **`pre_tasks`** and **`post_tasks`** tasks in a playbook.
- Ansible roles are referenced in playbooks in the **`roles`** section.
- Default role variables can be overwritten when a role is used in a playbook.
- *Ansible Galaxy* [https://galaxy.ansible.com] is a public library of Ansible roles written by Ansible users.
- The **`ansible-galaxy`** command can search for, display information about, install, list, remove, or initialize roles.
- The **`ansible-galaxy init --offline`** command creates the well-defined directory structure that Ansible roles have.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 8

OPTIMIZING ANSIBLE

Overview	
Goal	Configure connection types, delegations, and parallelism
Objectives	• Configure a connection type and an environment in a playbook • Configure delegation in a playbook • Configure parallelism in Ansible
Sections	• Configuring Connection Types (and Guided Exercise) • Configuring Delegation (and Guided Exercise) • Configuring Parallelism (and Guided Exercise)
Lab	• Optimizing Ansible

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuring Connection Types

Objectives

After completing this section, students should be able to:

- Configure a connection type and an environment in a playbook.

Ansible's connection types

Connection types determine how Ansible connects to managed hosts in order to communicate with them. By default, Ansible uses the SSH protocol for remote connection to target nodes. Assuming **`pipelining`** is not enabled, Ansible uses this SSH connection to transfer modules and template files, to run remote commands, and to run the plays in the playbook on managed hosts. Thus having a fast, stable, and secure SSH connection is of paramount importance to Ansible. Besides **`ssh`**, Ansible offers other connection types which can be used to connect to managed hosts, including **`paramiko`**, and **`local`**. The connection type can be specified in the Ansible configuration file (**`ansible.cfg`**), in the inventory on a host or group basis, inside a playbook, or on the command-line interface.



Note

The **`pipelining`** feature reduces the number of SSH operations required to execute a module on the remote server, by executing many ansible modules without actual file transfer. To support pipelining, the **`requiretty`** feature must be disabled in the **`sudoers`** file on the managed hosts. Pipelining is enabled by setting **`pipelining`** to **`true`** in the **`[ssh_connection]`** section of the Ansible configuration file.

Connection types

Various connection types supported by Ansible include:

- **`smart`**: This connection type checks if the locally installed SSH client supports the **`ControlPersist`** feature. If **`ControlPersist`** is supported, Ansible will use the local SSH client. If the SSH client does not support **`ControlPersist`**, the **`smart`** connection type will fall back to using **`paramiko`**. The **`smart`** connection type is the default.

  The **`ControlPersist`** feature is available in OpenSSH 5.6 or later. It allows SSH connections to persist so that when frequent commands run over SSH, they do not have to repeatedly go through the initial handshake. When the **`ControlPersist`** timeout is reached, the SSH connection goes through the TCP handshake again. This setting can be set in the OpenSSH configuration, but is better done in the Ansible settings. In **`/etc/ansible/ansible.cfg`**, the default setting is listed in a comment:

  ```
  #ssh_args = -o ControlMaster=auto -o ControlPersist=60s
  ```

  The 60 second default value may be changed to something much higher to accept connections, even though the initial connection has exited and the connection is in an idle state for a longer time. This value should be set depending on the length of time it takes for playbooks to run and the frequency of running playbooks. A value of 30 minutes may be more appropriate:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ssh_args = -o ControlMaster=auto -o ControlPersist=30m
```

- **paramiko**: Paramiko is a Python implementation of the SSHv2 protocol. Ansible supports it as the connection type for backward compatibility for RHEL6 and earlier, which did not have support for the **ControlPersist** setting in their OpenSSH version.
- **local**: The **local** connection runs commands locally, instead of running over SSH.
- **ssh**: The **ssh** connection uses an OpenSSH-based connection, which supports **ControlPersist** technology.
- **docker**: Ansible offers a new **docker** connection type. It connects to containers using the **docker exec** command.

Ansible ships with other connection type plug-ins like **chroot**, **libvirt_lxc**, **jail** for FreeBSD **jail** environments, and the recently added **winrm**, which uses the Python **pywinrm** module to communicate to Windows remote hosts that do not have SSH but support PowerShell remoting. Ansible connection types are pluggable and extensible and additional connection types are planned for future releases.

Configuring connection types

Ansible configuration
The connection type can be specified in the **ansible.cfg** file under the **[defaults]** section of the file using the **transport** key.

```
[defaults]

# some basic default values...
... Output omitted ...
#transport = smart
```

The **transport** global value can be overridden by using the **-c *TRANSPORTNAME*** option while running an ad hoc command with **ansible** or running a playbook with **ansible-playbook**.

The default settings related to each individual connection type appear as comments beneath their respective connection headings in **/etc/ansible/ansible.cfg**. They can be overridden specifying that directive with a different value in the controlling **ansible.cfg** file. The settings for the **ssh** connection type are specified under the **[ssh_connection]** section. Likewise, the settings for **paramiko** can be found under the **[paramiko_connection]** section.

The following example shows some of the default **ssh** connection type options.

```
[ssh_connection]
... Output omitted ...
# control_path = %(directory)s/%%h-%%r
#control_path = %(directory)s/ansible-ssh-%%h-%%p-%%r
#pipelining = False
#scp_if_ssh = True
#sftp_batch_mode = False
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Important

Administrators should usually leave the connection type set to **smart** in **ansible.cfg**, and configure playbooks or inventory files to choose an alternative connection setting when necessary.

Inventory file

The connection type can also be specified in inventory files on a per-host basis by using the **ansible_connection** variable. The following example shows an inventory file using the **local** connection type for **localhost**, and the **ssh** connection type for **demo**.

```
[targets]
localhost  ansible_connection=local
demo.lab.example.com  ansible_connection=ssh
```

Playbooks

Connection types can also be specified when running an entire playbook. Specify the **--connection=*CONNECTIONTYPE*** option to the **ansible-playbook** command as shown in the following example.

```
[user@host ~]$ ansible-playbook playbook.yml --connection=local
```

Alternatively, the connection type can be set with the **connection: *CONNECTIONTYPE*** parameter in a playbook as shown in the following example.

```
---
- name: Connection type in playbook
  hosts: 127.0.0.1
  connection: local
```

Setting environment variables in playbooks

Sometimes a managed host needs to configure some shell environment variables before executing commands. For example, a proxy server may need to be specified through an environment variable before installing packages or downloading files; a special **$PATH** variable may be needed; or certain other environment variables may need to be set. Ansible can make these types of environment settings by using the **environment:** parameter in a playbook.

Some Ansible modules, such as **get_url**, **yum**, and **apt**, use environment variables to set their proxy server. The following example shows how to set the proxy server for package installation using the **environment:** parameter which sets the **$http_proxy** environment variable to **http://demo.lab.example.com**.

```
---
- hosts: devservers
  tasks:
     - name: download a file using demo.lab.example.com as proxy
       get_url:
         url: http://materials.example.copm/file.tar.gz
         dest: ~/Downloads
       environment:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
          http_proxy: http://demo.lab.example.com:8080
```

An environment hash can be defined in a variable using the **vars** keyword in a playbook, or using the **group_vars** keyword. These variables can then be referenced inside a playbook using the **environment** parameter.

The following example shows how to define an environment variable and reference it using the **environment** keyword for an entire **block** of tasks.

```
---
- name: Demonstrate environment variable in block
  hosts: localhost
  connection: local
  gather_facts: false
  vars:
    myname: ansible

  tasks:
  - block:
    - name: Print variable
      shell: "echo $name"
      register: result
    - debug: var=result.stdout
    environment:
      name: "{{ myname }}"
```

When using Ansible roles, the environment variables can be passed at a playbook level. The following example shows how to pass the **$http_proxy** environment variable when using an Ansible role.

```
---
- hosts: demohost
  roles:
    - php
    - nginx
  environment:
    http_proxy: http://demo.lab.example.com:8080
```



References

Remote Connection Information — Getting Started — Ansible Documentation
http://docs.ansible.com/ansible/intro_getting_started.html#remote-connection-information

Local Playbooks — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#local-playbooks

Setting the Environment (and Working With Proxies) — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_environment.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Configuring Connection Types

In this exercise, you will configure connection type and an environment using Ansible playbook. The playbook will copy a template file using the connection type defined in the inventory file, either using **local** or **ssh** connection type. The template file needs to be copied to managed hosts **servera** and **workstation** in the **/tmp** directory.

Outcomes

You should be able to:

- Configure connection types in playbook.
- Configure environment variables in a playbook.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab configure-connection-types setup** script. The setup script checks that Ansible is installed on **workstation**, creates a directory structure for the lab environment, and creates the **inventory** directory.

```
[student@workstation ~]$ lab configure-connection-types setup
```

Steps

1. From **workstation**, as the **student** user, change to the directory **~/connection-types**.

```
[student@workstation ~]$ cd ~/connection-types
[student@workstation connection-types]$
```

2. Create an inventory file named **hosts** under **~/connection-types/inventory** with two different host names for **workstation**, **workstation** and **workstation.lab.example.com**, which would use two different connection types, **local** and **ssh** respectively. Also add an inventory line for **servera.lab.example.com** that uses the connection type **ssh**.

   2.1. Create an inventory file named **hosts** under **~/connection-types/inventory** with the host name **workstation** using connection type **local**. The **hosts** file should contain the following line:

```
workstation ansible_connection=local
```

   2.2. Add a new inventory line in the **hosts** file. Add an inventory line for **workstation.lab.example.com** using the connection type **ssh** and setting the host as **localhost**. The file should now contain the following:

```
workstation ansible_connection=local
workstation.lab.example.com ansible_connection=ssh ansible_host=localhost
```

   2.3. Add another line to the **hosts** file. Add an inventory line for **servera.lab.example.com** that uses the connection type **ssh**. The file should now contain the following:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
workstation ansible_connection=local
workstation.lab.example.com ansible_connection=ssh ansible_host=localhost
servera.lab.example.com ansible_connection=ssh
```

3. Create a new directory called **templates**. Below it, create a file named **template.j2**. This file should be copied to the managed host and changed based on the environment variables specified in the playbook. This template file replaces the variables to create an inventory file on the respective host under **/tmp/*hostname***.

   3.1. Create the **templates** subdirectory.

   ```
   [student@workstation connection-types]$ mkdir templates
   ```

   3.2. Create the template, named **template.j2**, in the **templates** directory. Insert a line that replaces the **$GROUPNAME** environment variable (defined in the playbook later) with the inventory group name. In the file add the following lines and continue editing the file **template.j2**:

   ```
   # this is the template file
   [ {{ ansible_env['GROUPNAME'] }} ]
   ```

   3.3. In the **templates/template.j2** file, insert several lines to define the **inventory_hostname**, the **ansible_host**, and the **connection_type** variables. Set these using magic variables and predefined variables. The file should contain the following:

   ```
   # this is the template file
   [ {{ ansible_env['GROUPNAME'] }} ]
   inventory_hostname = {{ inventory_hostname }}
   ansible_host = {{ ansible_host }}
   connection_type = {{ ansible_connection }}
   ```

   Save and close the file.

4. Create a playbook named **playbook.yml** that copies the template **template.j2** to the destination host under **/tmp/** and names the file as that of its host name specified in the inventory file (for example, **/tmp/*hostname***).

   The playbook should also define the environment variable named **$GROUPNAME** and set its value to **testservers**, to define the group name in the inventory file when copied to respective managed hosts using the template **template.j2**. The **playbook.yml** file should contain the following:

   ```
   ---
   # playbook.yml
   - name: testing connection types
     hosts: all
     user: devops
     environment:
       GROUPNAME: testservers
     tasks:
       - name: template out hostfile
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      template:
         src: template.j2
         dest: "/tmp/{{ inventory_hostname }}"
```

5. Execute the playbook **playbook.yml** from its the working directory **~/connection-types**.

```
[student@workstation connection-types]$ ansible-playbook playbook.yml

PLAY [testing connection types] ************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]
ok: [workstation]
ok: [workstation.lab.example.com]

TASK [template out hostfile] ***************************************************
changed: [workstation]
changed: [workstation.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
workstation                : ok=2    changed=1    unreachable=0    failed=0
workstation.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

6. Verify the files created on **workstation.lab.example.com** under the **/tmp** directory as **/tmp/workstation** and **/tmp/workstation.lab.example.com**. These files should contain the connection type used when executing the playbook and specified in the inventory file **hosts**.

   6.1. Verify the content of the file **/tmp/workstation** created on **workstation**.

```
[student@workstation connection-types]$ cat /tmp/workstation
# this is the template file
[ testservers ]
inventory_hostname = workstation
ansible_host = workstation
connection_type = local
```

   6.2. Verify the content of the file **/tmp/workstation.lab.example.com** created on **workstation**.

```
[student@workstation connection-types]$ cat \
> /tmp/workstation.lab.example.com
# this is the template file
[ testservers ]
inventory_hostname = workstation.lab.example.com
ansible_host = localhost
connection_type = ssh
```

7. Verify the file created on **servera.lab.example.com** under **/tmp** directory as **/tmp/servera.lab.example.com**. This file should contain the connection type used when executing the playbook and specified in the inventory file **hosts**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation connection-types]$ ansible servera.lab.example.com -a 'cat /
tmp/servera.lab.example.com'
servera.lab.example.com | SUCCESS | rc=0 >>
# this is the template file
[ testservers ]
inventory_hostname = servera.lab.example.com
ansible_host = servera.lab.example.com
connection_type = ssh
```

Cleanup

Run the **lab configure-connection-types cleanup** command to cleanup the lab.

```
[student@workstation ~]$ lab configure-connection-types cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuring Delegation

Objectives

After completing this section, students should be able to:

- Configure delegation in a playbook.

Configuring delegation

In order to complete some configuration tasks, it may be necessary for actions to be taken on a different server than the one being configured. Some examples of this might include an action that requires waiting for the server to be restarted, adding a server to a load balancer or a monitoring server, or making changes to the DHCP or DNS database needed for the server being configured.

Delegation can help by performing necessary actions for tasks on hosts other than the managed host being targeted by the play in the inventory. Some scenarios that delegation can handle include:

- Delegating a task to the local machine
- Delegating a task to a host outside the play
- Delegating a task to a host that exists in the inventory
- Delegating a task to a host that does not exist in the inventory

Delegating tasks to the local machine

When any action needs to be performed on the node running Ansible, it can be delegated to **`localhost`** by using **`delegate_to: localhost`**.

Here is a sample playbook which runs the command **`ps`** on the **`localhost`** using the **`delegate_to`** keyword, and displays the output using the **`debug`** module:

```
---
- name: delegate_to:localhost example
  hosts: dev
  tasks:
    - name: remote running process
      command: ps
      register: remote_process

    - debug: msg="{{ remote_process.stdout }}"

    - name: Running Local Process
      command: ps
      delegate_to: localhost
      register: local_process

    - debug:
        msg: "{{ local_process.stdout }}"
```

The **`local_action`** keyword is a shorthand syntax replacing **`delegate_to: localhost`**, and can be used on a per-task basis.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The previous playbook can be re-written using this shorthand syntax to delegate the task to the node running Ansible (**localhost**):

```
---
- name: local_action example
  hosts: dev
  tasks:
    - name: remote running process
      command: ps
      register: remote_process

    - debug: msg="{{ remote_process.stdout }} "

    - name: Running Local Process
      local_action: command 'ps'
      register: local_process

    - debug:
        msg: "{{ local_process.stdout }}"
```



Important

Ansible 1.5 introduced the *implicit localhost* feature, which allows actions using "**hosts: localhost**" to succeed on the local machine when **localhost** is not defined in the inventory. In essence, with this feature the **localhost** managed host is implicitly defined by Ansible when no entry exists for it in the inventory.

The effect of this feature is evident in the resolution of the **localhost** host pattern even when no **localhost** entry is defined in the inventory.

```
[student@demo ~]$ cat inventory
[student@demo ~]$ ansible localhost --list-hosts
  hosts (1):
    localhost
```

It is important to note that **hosts: all** will *not* match the implicit localhost entry.

It is also important to note that the connection type for the implicit localhost definition is **local_connection**. This is different from the default **ssh** connection type which would be used if **localhost** was explicitly defined in the inventory.

One side effect of this difference in connection type is that connections made using implicit localhost will ignore the **remote_user** setting since there is no login process involved. If the **remote_user** setting is defined to be different than the user that executes an Ansible command, administrators may experience different outcomes using implicit versus explicit **localhost** definitions.

For example. if privilege escalation with **sudo** is used, implicit localhost would be escalating privileges from the user account that executed the Ansible command, while explicit localhost would do so using the **remote_user** account. If these two accounts were configured with different **sudo** privileges, the privilege escalation attempts may have different outcomes.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Delegating task to a host outside the play

Ansible can be configured to run a task on a host other than the one that is part of the play with **delegate_to**. The delegated module will still run once for every machine, but instead of running on the target machine, it will run on the host specified by **delegate_to**. The facts available will be the ones applicable to the original host and not the host the task is delegated to. The task has the context of the original target host, but it gets executed on the host the task is delegated to.

The following example shows Ansible code that will delegate a task to an outside machine (in this case, **loadbalancer-host**). This example runs a command on the local balancer host to remove the managed hosts from the load balancer before deploying the latest version of the web stack. After that task is finished, a script is run to add the managed hosts back into the load balancer pool.

```
- hosts: webservers
  tasks:
    - name: Remove server from load balancer
      command: remove-from-lb {{ inventory_hostname }}
      delegate_to: loadbalancer-host

    - name: deploy the latest version of web stack
      git:repo=git://foosball.example.org/path/to/repo.git dest=/srv/checkout

    - name: Add server to load balancer pool
      command: add-to-lb {{ inventory_hostname }}
      delegate_to: loadbalancer-host
```

Delegating a task to a host that exists in the inventory

When delegating to a host listed in the inventory, the inventory data will be used when creating the connection to the delegation target. This would include settings for **ansible_connection**, **ansible_host**, **ansible_port**, **ansible_user** and so on. Only the connection-related variables are used; the rest are read from the managed host originally targeted.

Delegating a task to a host that does not exist in the inventory

When delegating a task to a host that does not exist in the inventory, Ansible will use the same connection type and details used for the managed host to connect to the delegating host. To adjust the connection details, use the **add_host** module to create an ephemeral host in your inventory with connection data defined.

```
- name: test play
  hosts: localhost
  tasks:

    - name: add delegation host
      add_host: name=demo ansible_host=172.25.250.10 ansible_user=devops

    - name: echo Hello
      command: echo "Hello from {{ inventory_hostname }}"
      delegate_to: demo
      register: output

    - debug:
        msg: "{{ output.stdout }}"
```

When the preceding playbook is executed, Ansible will use the connection details for the ephemeral host **demo** while executing the task **echo Hello** on the **delegate_to** host. The

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

inventory_hostname will be read from the targeted managed host which in this case is **localhost**.

The following example shows the playbook execution. Note that the output contains the **add_host** line and the variable expansion to **localhost**.

```
[student@demo ~]$ ansible-playbook test.yml -vvv
... Output omitted ...
TASK [add delegation host] *****************************************************
task path: /home/student/ansible/dev-optimizing-ansible/inventory/test.yml:6
creating host via 'add_host': hostname=demo
changed: [localhost] => {"add_host": {"groups": [], "host_name": "demo",
"host_vars": {"ansible_host": "172.25.250.10",
"ansible_user": "devops"}}, "changed": true, "invocation":
{"module_args": {"ansible_host": "172.25.250.10",
"ansible_user": "devops", "name": "demo"}, "module_name": "add_host"}}
... Output omitted ...
TASK [silly echo] **************************************************************
... Output omitted ...
changed: [localhost -> 172.25.250.10] => {"changed": true, "cmd": ["echo", "Hello from
 localhost"],
```

Task execution concurrency with delegation

Delegated tasks run for each managed host targeted. But Ansible tasks can run on multiple managed hosts in parallel. This can create issues with race conditions on the delegated host. This is particularly likely when using conditionals in the task, or when multiple concurrent tasks are run in parallel. This can also create a "thundering herd" problem where too many connections are being opened at once on the delegated host. SSH servers have a **MaxStartups** configuration option that can limit the number of concurrent connections allowed.

Delegated Facts

Any facts gathered by a delegated task are assigned by default to the **delegate_to** host, instead of the host which actually produced the facts. The following example shows a task file that will loop through a list of inventory servers to gather facts.

```
- hosts: app_servers
  tasks:
    - name: gather facts from app servers
      setup:
      delegate_to: "{{item}}"
      with_items: "{{groups['lb_servers']}}"

    - debug: var=ansible_eth0['ipv4']['address']
```

```
#inventory file
[app_servers]
demo.lab.example.com
[lb_servers]
workstation.lab.example.com
```

When the previous playbook is run, the output shows the gathered facts of **workstation.lab.example.com** as the task delegated to the host from the **lb_servers** inventory group instead of **demo.lab.example.com**.

```
[student@demo ~]$ ansible-playbook delegatefacts.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
... Output omitted ...
TASK [gather facts from app servers] ******************************************
ok: [demo.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

TASK [debug] ******************************************************************
ok: [demo.lab.example.com] => {
    "ansible_eth0['ipv4']['address']": "172.25.250.254"
}

PLAY RECAP ********************************************************************
demo.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
```

The **delegate_facts** directive can be set to **True** to assign the gathered facts from the task to the delegated host instead of the current host.

```
- hosts: app_servers
  tasks:
    - name: gather facts from db servers
      setup:
      delegate_to: "{{item}}"
      delegate_facts: True
      with_items: "{{groups['lb_servers']}}"

    - debug: var=ansible_eth0['ipv4']['address']
```

On running the above playbook, the output now shows the facts gathered from **demo.lab.example.com** instead of the facts from the current managed host.

```
[student@demo ~]$ ansible-playbook delegatefacts.yml
... Output omitted ...
TASK [gather facts from db servers] *******************************************
ok: [demo.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

TASK [debug] ******************************************************************
ok: [demo.lab.example.com] => {
    "ansible_eth0['ipv4']['address']": "172.25.250.10"
}

PLAY RECAP ********************************************************************
demo.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
```



References

Delegation — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#delegation

Delegated facts — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#delegated-facts

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Configuring Delegation

In this exercise, you will configure the delegation of tasks in an Ansible playbook. The playbook will configure **workstation** as a proxy server and **servera** as an Apache web server. During the deployment of the website on **servera**, you will delegate the task of stopping the traffic coming to **servera** to **workstation** proxy server and later after deployment you will start the traffic coming to **servera** by delegating task to **workstation**.

Outcomes

You should be able to:

- Configure the delegation of a task in a playbook.

Before you begin

Log in to **workstation** as **student** using **student** as the password. Run the **lab configure-delegation setup** command.

```
[student@workstation ~]$ lab configure-delegation setup
```

The setup script confirms that Ansible is installed on **workstation** and creates a directory structure for the lab environment.

Steps

1. From **workstation**, as the **student** user, change to the **~/configure-delegation** directory.

```
[student@workstation ~]$ cd ~/configure-delegation
[student@workstation configure-delegation]$
```

2. Create an inventory file named **hosts** under **~/configure-delegation/inventory**. The inventory file should have two groups defined: **webservers** and **proxyservers**. The **servera.lab.example.com** host should be part of the **webservers** group and **workstation.lab.example.com** should be part of the **proxyservers** group.

```
[webservers]
servera.lab.example.com

[proxyservers]
workstation.lab.example.com
```

3. Move the **servera.lab.example.com-httpd.conf.j2**, template file that configures the virtual host to the **~/configure-delegation/templates** directory. Later you will use an Ansible variable (`inventory_hostname`) to list the source of this file.

```
[student@workstation configure-delegation]$ mv servera.lab.example.com-httpd.conf.j2
 ~/configure-delegation/templates
```

4. Move the **~/configure-delegation/workstation.lab.example.com-httpd.conf.j2** template file for configuring reverse proxy to the **~/configure-delegation/templates** directory.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation configure-delegation]$ mv workstation.lab.example.com-
httpd.conf.j2 ~/configure-delegation/templates
```

5. Create a template file, named **index.html.j2**, for the website to be hosted on **servera.lab.example.com** under the **templates** directory. The file should contain the following content:

```
The webroot is {{ ansible_fqdn }}.
```

6. Create a playbook named **site.yml** in the main project directory, **~/configure-delegation**. The playbook should define a play to install and configure *httpd*. The play should contain the following tasks:
   - Install the *httpd* package and start and enable the service to **all** hosts defined in the inventory.
   - Configure firewall to accept incoming **http** traffic.
   - Copy the respective **httpd.conf** configuration file to the hosts serving as the web and proxy server. The resulting file should be named **myconfig.conf** under the **/etc/httpd/conf.d/myconfig.conf** directory.

   6.1. Create a **site.yml** playbook under the lab project directory, **~/configure-delegation/**. Define a play inside the playbook that will execute the tasks on **all** hosts. Use **devops** for remote connection and use privilege escalation to **root** using **sudo**.

```
---
- name: Install and configure httpd
  hosts: all
  remote_user: devops
  become: true
```

   6.2. Continue editing the **site.yml** playbook. Define a task to install the *httpd* package and start and enable the **httpd** service on all hosts.

```
  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: installed
    - name: Start and enable httpd
      service:
        name: httpd
        state: started
        enabled: yes
```

   6.3. In the **site.yml** playbook, define a task to enable the firewall to allow web traffic on **servera.lab.example.com**.

```
    - name: Install firewalld
      yum:
        name: firewalld
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        state: installed
    - name: Start and enable firewalld
      service:
        name: firewalld
        state: started
        enabled: yes
    - name: Enable firewall
      firewalld:
        zone: public
        service: http
        permanent: true
        state: enabled
        immediate: true
      when: inventory_hostname in groups['webservers']
```

6.4. Define a task in the **site.yml** playbook to copy the **workstation.lab.example.com-httpd.conf.j2** and **servera.lab.example.com-httpd.conf.j2** template files to the **/etc/httpd/conf.d/myconfig.conf** directory on their respective hosts. After copying the configuration file, use the **notify:** keyword to invoke the **restart httpd** handler defined in the next step.

```
    - name: template server configs
      template:
        src: "templates/{{ inventory_hostname }}-httpd.conf.j2"
        dest: /etc/httpd/conf.d/myconfig.conf
        owner: root
        group: root
        mode: 0644
      notify:
        - restart httpd
```

6.5. In the **site.yml** playbook, define a handler to restart the **httpd** service when it is invoked.

```
  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted
```

6.6. Review the **site.yml** playbook. The file should contain the following:

```
---
- name: Install and configure httpd
  hosts: all
  remote_user: devops
  become: true
  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: installed
    - name: Start and enable httpd
      service:
        name: httpd
        state: started
        enabled: yes
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    - name: Install firewalld
      yum:
        name: firewalld
        state: installed
    - name: Start and enable firewalld
      service:
        name: firewalld
        state: started
        enabled: yes
    - name: Enable firewall
      firewalld:
        zone: public
        service: http
        permanent: true
        state: enabled
        immediate: true
      when: inventory_hostname in groups['webservers']
    - name: template server configs
      template:
        src: "templates/{{ inventory_hostname }}-httpd.conf.j2"
        dest: /etc/httpd/conf.d/myconfig.conf
        owner: root
        group: root
        mode: 0644
      notify:
        - restart httpd

  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted
```

7. Add another play to the **site.yml** playbook. The play should contain the following tasks:
   - Stop the proxy server on **workstation.lab.example.com** using delegation.
   - Deploy the web page by copying the **index.html.j2** to the **/var/www/html/index.html** directory on **servera.lab.example.com**.
   - Start the proxy server on **workstation.lab.example.com** using delegation.

   7.1. In the **site.yml** playbook, define another play that will have tasks to deploy a website that needs to be run on **servera.lab.example.com** of the **webservers** inventory group.

```
- name: Deploy web service and disable proxy server
  hosts: webservers
  remote_user: devops
  become: true
```

   7.2. In the **site.yml** playbook, define a task to stop the incoming web traffic to **servera.lab.example.com** by stopping the proxy server running on **workstation.lab.example.com**.

   Since the **hosts** keyword of the play points to **webservers** host group, use **delegate_to** keyword to delegate this task to **workstation.lab.example.com** of the **proxyservers** host group.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  tasks:
    - name: Stop Apache proxy server
      service:
        name: httpd
        state: stopped
      delegate_to: "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
```

7.3. In the **site.yml** playbook, define a task to copy the **index.html.j2** template present under **templates** directory to **/var/www/html/index.html** on **servera.lab.example.com**, part of the **webservers** group. Change the owner and group to **apache** and file permission to **0644**.

```
    - name: Deploy webpages
      template:
        src: templates/index.html.j2
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: 0644
```

7.4. Add another task to the **site.yml** playbook that starts the proxy server by delegating the task to **workstation.lab.example.com**.

```
    - name: Start Apache proxy server
      service:
        name: httpd
        state: started
      delegate_to: "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
```

7.5. Review the **site.yml** playbook. The file should now contain the following additional content:

```
... Previous play content omitted ...

- name: Deploy web service and disable proxy server
  hosts: webservers
  remote_user: devops
  become: true
  tasks:
    - name: Stop Apache proxy server
      service:
        name: httpd
        state: stopped
      delegate_to: "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
    - name: Deploy webpages
      template:
        src: templates/index.html.j2
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: 0644
    - name: Start Apache proxy server
      service:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        name: httpd
        state: started
      delegate_to: "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
```

8. Check the syntax of the **site.yml** playbook. Resolve any syntax errors you find.

```
[student@workstation configure-delegation]$ ansible-playbook --syntax-check site.yml

playbook: site.yml
```

9. Execute the **site.yml** playbook. Watch for the delegation tasks in the command output.

```
[student@workstation configure-delegation]$ ansible-playbook site.yml

PLAY [Install httpd and configure] *********************************************
... Output omitted ...

PLAY [Deploy web service and disable proxy server] *****************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Stop Apache proxy server] ************************************************
changed: [servera.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

TASK [Deploy webpages] *********************************************************
changed: [servera.lab.example.com]

TASK [Start Apache proxy server] ***********************************************
changed: [servera.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=10   changed=8    unreachable=0    failed=0
workstation.lab.example.com : ok=5    changed=2    unreachable=0    failed=0
```

10. Verify the website by browsing the **http://workstation.lab.example.com/external** link.

```
[student@workstation configure-delegation]$ curl http://workstation.lab.example.com/
external
The webroot is servera.lab.example.com.
```

Cleanup

Run the **lab configure-delegation cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab configure-delegation cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuring Parallelism

Objectives

After completing this section, students should be able to:

- Configure parallelism in Ansible

Configure parallelism in Ansible using forks

Ansible allows much more control over the execution of the playbook by running the tasks in parallel on all hosts. By default, Ansible only fork up to five times, so it will run a particular task on five different machines at once. This value is set in the Ansible configuration file **`ansible.cfg`**.

```
[student@demo ~]$ grep forks /etc/ansible/ansible.cfg
#forks          = 5
```

When there are a large number of managed hosts (more than five), the **`forks`** parameter can be changed to something more suitable for the environment. The default value can be either overridden in the configuration file by specifying a new value for the **`forks`** key, or the value can be changed using the **`--forks`** option for the **`ansible-playbook`** or **`ansible`** commands.

Running tasks in parallel

For any specific play, you can use the **`serial`** keyword in a playbook to temporarily reduce the number of machines running in parallel from the fork count specified in the Ansible configuration file. The **`serial`** keyword is primarily used to control rolling updates.

Rolling updates

If there is a website being deployed on 100 web servers, only 10 of them should be updated at the same time. The **`serial`** key can be set to 10 in the playbook to reduce the number of simultaneous deployments (assuming that the **`fork`** key was set to something higher). The **`serial`** keyword can also be specified as a percentage which will be applied to the total number of hosts in the play. If the number of hosts does not divide equally into the number of passes, the final pass will contain the modulus. Regardless of the percentage, the number of hosts per pass will always be 1 or greater.

```
---
- name: Limit the number of hosts this play runs on at the same time
  hosts: appservers
  serial: 2
```

Ansible, regardless of the number of forks set, only spins up the tasks based on the current number of hosts in a play.

Asynchronous tasks

There are some system operations that take a while to complete. For example, when downloading a large file or rebooting a server, such tasks takes a long time to complete. Using parallelism and forks, Ansible starts the command quickly on the managed hosts, then polls the hosts for status until they are all finished.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

To run an operation in parallel, use the **async** and **poll** keywords. The **async** keyword triggers Ansible to run the job in the background and can be checked later, and its value will be the maximum time that Ansible will wait for the command to complete. The value of **poll** indicates to Ansible how often to poll to check if the command has been completed. The default **poll** value is **10** seconds.

In the example, the **get_url** module takes a long time to download a file and **async: 3600** instructs Ansible to wait for **3600** seconds to complete the task and **poll: 10** is the polling time in seconds to check if the download is complete.

```
---
- name: Long running task
  hosts: demoservers
  remote_user: devops
  tasks:
    - name: Download big file
      get_url: url=http://demo.example.com/bigfile.tar.gz
      async: 3600
      poll: 10
```

Deferring asynchronous tasks

Long running operations or maintenance scripts can be carried out with other tasks, whereas checks for completion can be deferred until later using the **wait_for** module. To configure Ansible to not wait for the job to complete, set the value of **poll** to **0** so that Ansible starts the command and instead of polling for its completion it moves to the next tasks.

```
---
- name: Restart and wait until the server is rebooted
  hosts: demoservers
  remote_user: devops
  tasks:
    - name: restart machine
      shell: sleep 2 && shutdown -r now "Ansible updates triggered"
      async: 1
      poll: 0
      become: true
      ignore_errors: true

    - name: waiting for server to come back
      local_action:
        wait_for:
          host: "{{ inventory_hostname }}"
          state: started
          delay: 30
          timeout: 300
      become: false
```



Note

The shorthand syntax for delegating to **127.0.0.1**, can be also be written by use of the **local_action** keyword.

For the running tasks that take an extremely long time to run, you can configure Ansible to wait for the job as long as it takes. To do this, set the value of **async** to **0**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Asynchronous task status

While an asynchronous task is running, you can also check its completion status by using Ansible **`async_status`** module. The module requires the job or task identifier as its parameter.

```
---
# Async status - fire-forget.yml
- name: Async status with fire and forget task
  hosts: demoservers
  remote_user: devops
  become: true
  tasks:

    - name: Download big file
      get_url: url=http://demo.example.com/bigfile.tar.gz
      async: 3600
      poll: 0
      register: download_sleeper

    - name: Wait for download to finish
      async_status: "jid={{ download_sleeper.ansible_job_id }}"
      register: job_result
      until: job_result.finished
      retries: 30
```

The output of the playbook when executed:

```
[student@demo ~]$ ansible-playbook fire-forget.yml

PLAY [Async status with fire and forget task] **********************************

TASK [setup] *******************************************************************
ok: [demo.example.com]

TASK [Download big file] *******************************************************
ok: [demo.example.com]

TASK [Wait for download to finish] *********************************************
FAILED - RETRYING: TASK: Wait for download to fins (29 retries left). Result
 was: {u'ansible_job_id': u'963772827414.1563', u'started': 1, u'changed': False,
 u'finished': 0, u'results_file': u'/root/.ansible_async/963772827414.1563',
 'invocation': {'module_name': u'async_status', u'module_args': {u'jid':
 u'963772827414.1563', u'mode': u'status'}}}
... Output omitted ...
changed: [demo.example.com]

PLAY RECAP *********************************************************************
demo.example.com    : ok=3    changed=1    unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Rolling Update Batch Size — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#rolling-update-batch-size

Asynchronous Actions and Polling — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_async.html

async_status - Obtain status of asynchronous task — Ansible Documentation
http://docs.ansible.com/ansible/async_status_module.html

Ansible Performance Tuning (For Fun and Profit)
https://www.ansible.com/blog/ansible-performance-tuning

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Configuring Parallelism

In this exercise, you will run a playbook which uses a script to performs a long-running process on **servera.lab.example.com** using an asynchronous task. Instead of waiting for the task to get completed, will check the status using the **async_status** module.

Outcomes

You should be able to:

- Configure parallelism using an asynchronous task in playbook.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab configure-async setup** script. The setup script checks that Ansible is installed on **workstation**, creates a directory structure for the lab environment and an inventory file named **hosts**.

```
[student@workstation ~]$ lab configure-async setup
```

Steps

1. From **workstation**, as the **student** user, change to the directory **~/configure-async**.

```
[student@workstation ~]$ cd ~/configure-async
[student@workstation configure-async]$
```

2. Create a script file named **longfiles.j2** under the **~/configure-async/templates** directory, with the following content.

```
#!/bin/bash
echo "emptying $2" > $2
for i in {00..30}; do
  echo "run $i, $1"
  echo "run $i for $1" >> $2
  sleep 1
done
```

3. Create a playbook named **async.yml** under the lab's project directory. In the playbook use the **webservers** inventory host group, the **devops** remote user, and for privilege escalation use the **root** user and **sudo** as the method.

```
# async.yml
- name: longfiles async playbook
  hosts: webservers
  remote_user: devops
  become: true
```

4. In the playbook file **async.yml**, define a task to:
   - Copy the **longfiles.j2** script under the **~/configure-async/templates** directory to the managed host under **/usr/local/bin/longfiles**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Change the file and group ownership to **root**, and change permission of the script to **0755**.

```
...output-omitted...
  tasks:
    - name: template longfiles script
      template:
        src: templates/longfiles.j2
        dest: /usr/local/bin/longfiles
        owner: root
        group: root
        mode: 0755
```

5. Define a task and use the **async** keyword in the playbook file **async.yml** to:
   - Run the **longfiles** file copied previously under **/usr/local/bin/longfiles** with arguments *foo*, *bar*, and *baz* and the their corresponding output files **/tmp/foo.file**, **/tmp/bar.file** and **/tmp/baz.file** respectively. For example: **/usr/local/bin/longfiles foo /tmp/foo.file**

   - Use the **async** keyword to wait the for the task until **110** second and set the value of **poll** to **0** so that Ansible starts the command then runs in the background. Register a `script_sleeper` variable to store the completion status of the command started.

```
...output-omitted...
    - name: run longfiles script
      command: "/usr/local/bin/longfiles {{ item }} /tmp/{{ item }}.file"
      async: 110
      poll: 0
      with_items:
        - foo
        - bar
        - baz
      register: script_sleeper
```

6. In the playbook **async.yml**, define a task to switch on the debug mode to see the value stored in the variable `script_sleeper`.

```
...output-omitted...
    - name: show script_sleeper value
      debug:
        var: script_sleeper
```

7. In the playbook **async.yml**, define a task in the playbook that will keep checking the status of the async task:
   - Use the **async_status** module to check the status of **async** task triggered previously using the variable **script_sleeper.result**.

   - Set the maximum retries to **30**.

```
...output-omitted...
    - name: check status of longfiles script
      async_status: "jid={{ item.ansible_job_id }}"
      register: job_result
      until: job_result.finished
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      retries: 30
      with_items: "{{ script_sleeper.results }}"
```

8. The completed **async.yml** playbook should have the following content:

```
# async.yml
- name: longfiles async playbook
  hosts: webservers
  remote_user: devops
  become: true

  tasks:
    - name: template longfiles script
      template:
        src: templates/longfiles.j2
        dest: /usr/local/bin/longfiles
        owner: root
        group: root
        mode: 0755

    - name: run longfiles script
      command: "/usr/local/bin/longfiles {{ item }} /tmp/{{ item }}.file"
      async: 110
      poll: 0
      with_items:
        - foo
        - bar
        - baz
      register: script_sleeper

    - name: show script_sleeper value
      debug:
        var: script_sleeper

    - name: check status of longfiles script
      async_status: "jid={{ item.ansible_job_id }}"
      register: job_result
      until: job_result.finished
      retries: 30
      with_items: "{{ script_sleeper.results }}"
```

Check the syntax of the **async.yml** playbook. Correct any errors that you find.

```
[student@workstation configure-async]$ ansible-playbook --syntax-check async.yml

playbook: async.yml
```

9. Run the playbook **async.yml**.

Observe the job ids listed as **ansible_job_id** to each job running on **servera.lab.example.com**, that were run in parallel using the **async** keyword. The task *check status of longfiles script* retries to see if the started jobs are complete using the **async_status** Ansible module.

```
[student@workstation configure-async]$ ansible-playbook async.yml

...output-omitted...
TASK [run a longfiles script] ************************************************
ok: [servera.lab.example.com] => (item=foo)
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
ok: [servera.lab.example.com] => (item=bar)
ok: [servera.lab.example.com] => (item=baz)

TASK [show script_sleeper value] ********************************************
ok: [servera.lab.example.com] => {
    "script_sleeper": {
        "changed": false,
        "msg": "All items completed",
        "results": [
            {
                "_ansible_no_log": false,
                "ansible_job_id": "54096600518.30283",
                "item": "foo",
                "results_file": "/root/.ansible_async/54096600518.30283",
                "started": 1
            },
            {
                "_ansible_no_log": false,
                "ansible_job_id": "971932031343.30382",
                "item": "bar",
                "results_file": "/root/.ansible_async/971932031343.30382",
                "started": 1
            },
            {
                "_ansible_no_log": false,
                "ansible_job_id": "885500386192.30482",
                "item": "baz",
                "results_file": "/root/.ansible_async/885500386192.30482",
                "started": 1
            }
        ]
    }
}

TASK [check status of longfiles script] *************************************
FAILED - RETRYING: TASK: check status of longfiles script (29 retries left). Result
 was: {u'ansible_job_id': u'54096600518.30283', u'started': 1, u'changed': False,
 u'finished': 0, u'results_file': u'/root/.ansible_async/54096600518.30283',
 'invocation': {'module_name': u'async_status', u'module_args': {u'jid':
 u'54096600518.30283', u'mode': u'status'}}}
...output-omitted...
FAILED - RETRYING: TASK: check status of longfiles script (24 retries left). Result
 was: {u'ansible_job_id': u'54096600518.30283', u'started': 1, u'changed': False,
 u'finished': 0, u'results_file': u'/root/.ansible_async/54096600518.30283',
 'invocation': {'module_name': u'async_status', u'module_args': {u'jid':
 u'54096600518.30283', u'mode': u'status'}}}
changed: [servera.lab.example.com] => (item={u'started': 1, 'item': u'foo',
 u'results_file': u'/root/.ansible_async/54096600518.30283', '_ansible_no_log':
 False, u'ansible_job_id': u'54096600518.30283'})
changed: [servera.lab.example.com] => (item={u'started': 1, 'item': u'bar',
 u'results_file': u'/root/.ansible_async/971932031343.30382', '_ansible_no_log':
 False, u'ansible_job_id': u'971932031343.30382'})
changed: [servera.lab.example.com] => (item={u'started': 1, 'item': u'baz',
 u'results_file': u'/root/.ansible_async/885500386192.30482', '_ansible_no_log':
 False, u'ansible_job_id': u'885500386192.30482'})

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=5    changed=2    unreachable=0    failed=0
```

Cleanup

Run the **`lab configure-async cleanup`** command to clean up the lab.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab configure-async cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Optimizing Ansible

In this lab, you will deploy an upgraded web page to add a new feature using the **serial** keyword for rolling updates, on two web servers **serverb.lab.example.com** and **tower.lab.example.com** running behind a load balancer.

The HAProxy load balancer and the web servers are preconfigured on **workstation.lab.example.com**, **serverb.lab.example.com**, and **tower.lab.example.com** respectively.

After the upgrade of the web page, the web servers needs to be rebooted one at a time before adding them back to load balancer pool, without affecting the site availability using **delegation**.

Outcomes
You should be able to write playbooks with tasks:
- To delegate tasks to other hosts.
- To asynchronously run jobs in parallel.
- To use rolling updates.

Before you begin
Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab optimize-ansible-lab setup** script. It checks if Ansible is installed on **workstation** and creates a directory structure for the lab environment with an inventory file. The script preconfigures **serverb.lab.example.com** and **tower.lab.example.com** as web servers and configures **workstation.lab.example.com** as the load balancer server using a round-robin algorithm. The script also creates a **templates** directory under the lab's working directory.

The inventory file **/home/student/lab-optimizing-ansible/inventory/hosts** points to **serverb.lab.example.com** and **tower.lab.example.com** as hosts of **[webservers]** group and **workstation.lab.example.com** as part of the **[lbserver]** group.

```
[student@workstation ~]$ lab optimize-ansible-lab setup
```

Steps
1. Log in to **workstation** as the **student** user and change to the **~/lab-optimizing-ansible** directory.

```
cd ~/lab-optimizing-ansible
[student@workstation lab-optimize-ansible]$
```

2. As the web servers are preconfigured as part of the lab setup, use **curl** to browse **http://workstation.lab.example.com**. Run the **curl** command twice to see the web content from the web servers running on **serverb** and **tower** and **workstation** serving as the load balancer server.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3. You will update the web site, shown previously, by adding a new line to the web content. Create a new web page template, named **index-ver1.html.j2**, under the **templates** directory.

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to {{ inventory_hostname }}.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

4. Create a playbook named **upgrade_webserver.yml** under **~/lab-optimizing-ansible**. The playbook should use privilege escalation using the remote user **devops** and run on hosts from the **webservers** group.

   The updates should be pushed to *one* server at a time.

5. Create a task in the **upgrade_webserver.yml** playbook to remove the web server from the load balancer pool. Use the **haproxy** Ansible module to remove from the **haproxy** load balancer. The task needs to delegated to server from the **[lbserver]** inventory group.

   The **haproxy** module is used to disable a back end server from HAProxy using socket commands. To disable a back end server from the back end pool named **app**, socket path as **/var/lib/haproxy/stats**, and **wait=yes** to wait until the server reports a status of maintenance, use the following:

```
haproxy: state=disabled backend=app host={{ inventory_hostname }} socket=/var/lib/
haproxy/stats wait=yes
```

6. Create a task in the **upgrade_webserver.yml** playbook to copy the updated page template from the lab working directory **templates/index-ver1.html.j2** to the web server's document root as **/var/www/html/index.html**. Also register a variable, **pageupgrade**, which will be used later to invoke other tasks.

7. Create a task in the **upgrade_webserver.yml** playbook to restart the web servers using an *asynchronous* task that will not wait more than **1** second for it to complete. The task should not be polled for completion.

   Use **ignore_errors** as **true** and execute the task if the previously registered **pageupgrade** variable has changed.

8. Create a task in the **upgrade_webserver.yml** playbook to wait for the web server to be rebooted. Use delegation to delegate the task to the local machine. The task should be executed when the variable **pageupgrade** has changed. Privilege escalation is not required for this task.

   Use the **wait_for** module to wait for the web server to be rebooted and specify the **host** as **inventory_hostname**, **state** as **started**, **delay** as **25**, and **timeout** as **200**.

9. Create a task in the **upgrade_webserver.yml** playbook to wait for the web server port to be started. As in the preconfigured web server, the **httpd** service is configured to start at

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

boot time. Specify the **host** as **inventory_hostname**, **port** as **80**, **state** as started, and **timeout** as **20**.

10. Create a final task in the **upgrade_webserver.yml** playbook to add the web server after the upgrade of the web page back to the load balancer pool. Use the **haproxy** Ansible module to add the server back to the **HAProxy** load balancer pool. The task needs to be delegated to all members of the **[lbserver]** inventory group.

    The **haproxy** module is used to enable a back end server from **HAProxy** using socket commands. To enable a back end server from the back end pool named **app**, socket path as **/var/lib/haproxy/stats**, and **wait=yes** to wait until the server reports healthy status, use the following:

```
haproxy: state=enabled backend=app host={{
inventory_hostname }} socket=/var/lib/haproxy/stats wait=yes
```

11. Review the **upgrade-webserver.yml** playbook file.

12. Check the syntax of the playbook **upgrade-webserver.yml**. In case of syntax errors resolve them before proceeding to the next step.

```
[student@workstation lab-optimizing-ansible]$ ansible-playbook --syntax-check
 upgrade_webserver.yml

playbook: upgrade_webserver.yml
```

13. Run the playbook **upgrade-webserver.yml** to upgrade the server.

    The restart task will take several minutes, so move on to the next step when it gets to that point in executing the playbook.



Important

The playbook will wait at the **wait for server restart** task up to 2 minutes to allow the web server to reboot.

14. From **workstation.lab.example.com**, use **curl** to view the web link **http://workstation.lab.example.com**.

    Run the **curl** command several times while the playbook is executing to verify the webserver is still reachable, and that the host changes after the playbook moves on to reboot the other machine.

15. Wait until the remaining tasks from the playbook complete on both **serverb** and **tower**.

16. Verify by browsing the website using the link **http://workstation.lab.example.com**. Rerun the **curl** command to see the updated pages from two different web servers, **serverb.lab.example.com** and **tower.lab.example.com**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Evaluation

From **workstation**, run the **lab optimize-ansible-lab** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab optimize-ansible-lab grade
```

Cleanup

Run the **lab optimize-ansible-lab cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab optimize-ansible-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will deploy an upgraded web page to add a new feature using the **serial** keyword for rolling updates, on two web servers **serverb.lab.example.com** and **tower.lab.example.com** running behind a load balancer.

The HAProxy load balancer and the web servers are preconfigured on **workstation.lab.example.com**, **serverb.lab.example.com**, and **tower.lab.example.com** respectively.

After the upgrade of the web page, the web servers needs to be rebooted one at a time before adding them back to load balancer pool, without affecting the site availability using **delegation**.

Outcomes

You should be able to write playbooks with tasks:

- To delegate tasks to other hosts.
- To asynchronously run jobs in parallel.
- To use rolling updates.

Before you begin

Log in to **workstation** as **student**, using **student** as the password.

On **workstation**, run the **lab optimize-ansible-lab setup** script. It checks if Ansible is installed on **workstation** and creates a directory structure for the lab environment with an inventory file. The script preconfigures **serverb.lab.example.com** and **tower.lab.example.com** as web servers and configures **workstation.lab.example.com** as the load balancer server using a round-robin algorithm. The script also creates a **templates** directory under the lab's working directory.

The inventory file **/home/student/lab-optimizing-ansible/inventory/hosts** points to **serverb.lab.example.com** and **tower.lab.example.com** as hosts of **[webservers]** group and **workstation.lab.example.com** as part of the **[lbserver]** group.

```
[student@workstation ~]$ lab optimize-ansible-lab setup
```

Steps

1. Log in to **workstation** as the **student** user and change to the **~/lab-optimizing-ansible** directory.

```
[student@workstation ~]$ cd ~/lab-optimizing-ansible
[student@workstation lab-optimize-ansible]$
```

2. As the web servers are preconfigured as part of the lab setup, use **curl** to browse **http://workstation.lab.example.com**. Run the **curl** command twice to see the web content from the web servers running on **serverb** and **tower** and **workstation** serving as the load balancer server.

```
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
<head><title>My Page</title></head>
<body>
<h1>
Welcome to serverb.lab.example.com.
</h1>
</body>
</html>
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to tower.lab.example.com.
</h1>
</body>
</html>
```

3. You will update the web site, shown previously, by adding a new line to the web content. Create a new web page template, named **index-ver1.html.j2**, under the **templates** directory.

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to {{ inventory_hostname }}.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

4. Create a playbook named **upgrade_webserver.yml** under **~/lab-optimizing-ansible**. The playbook should use privilege escalation using the remote user **devops** and run on hosts from the **webservers** group.

   The updates should be pushed to *one* server at a time.

   4.1. Create a playbook named **upgrade_webserver.yml** under **~/lab-optimizing-ansible**. Use the hosts that are part of the **webservers** inventory group and use privilege escalation using remote user **devops**. As the updates needs to pushed to one server at a time, use the **serial** keyword with its value as **1**.

   The contents of the **upgrade_webserver.yml** file should be as follows:

```
---
- name: Upgrade Webservers
  hosts: webservers
  remote_user: devops
  become: yes
  serial: 1
```

5. Create a task in the **upgrade_webserver.yml** playbook to remove the web server from the load balancer pool. Use the **haproxy** Ansible module to remove from the **haproxy** load balancer. The task needs to delegated to server from the **[lbserver]** inventory group.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The **haproxy** module is used to disable a back end server from HAProxy using socket commands. To disable a back end server from the back end pool named **app**, socket path as **/var/lib/haproxy/stats**, and **wait=yes** to wait until the server reports a status of maintenance, use the following:

```
haproxy: state=disabled backend=app host={{ inventory_hostname }} socket=/var/lib/
haproxy/stats wait=yes
```

5.1. Create a task in the **upgrade_webserver.yml** playbook to remove the web server from the load balancer pool. The **haproxy** Ansible module can be used to remove the web server from the **haproxy** load balancer. The task needs to be delegated to the server from the **[lbserver]** inventory group. The contents of the **upgrade_webserver.yml** file should appear as follows:

```
  tasks:
    - name: disable the server in haproxy
      haproxy:
        state: disabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

6. Create a task in the **upgrade_webserver.yml** playbook to copy the updated page template from the lab working directory **templates/index-ver1.html.j2** to the web server's document root as **/var/www/html/index.html**. Also register a variable, **pageupgrade**, which will be used later to invoke other tasks.

```
  - name: upgrade the page
    template:
      src: "templates/index-ver1.html.j2"
      dest: "/var/www/html/index.html"
    register: pageupgrade
```

7. Create a task in the **upgrade_webserver.yml** playbook to restart the web servers using an *asynchronous* task that will not wait more than **1** second for it to complete. The task should not be polled for completion.

   Use **ignore_errors** as **true** and execute the task if the previously registered **pageupgrade** variable has changed.

   7.1. Create a task in the **upgrade_webserver.yml** playbook to reboot the web server by adding a task to playbook. Use the **command** module to shut down the machine.

```
    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
```

   7.2. Continue editing the **restart machine** task in the **upgrade_webserver.yml** playbook. Use the **async** keyword to wait for 1 second for the completion and set **poll** to **0** to disable polling. Use **ignore_errors** as **true** and execute the task if the

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

previously registered **pageupgrade** variable has changed. Add the lines in bold to the playbook.

```
    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
      async: 1
      poll: 0
      ignore_errors: true
      when: pageupgrade.changed
```

8. Create a task in the **upgrade_webserver.yml** playbook to wait for the web server to be rebooted. Use delegation to delegate the task to the local machine. The task should be executed when the variable **pageupgrade** has changed. Privilege escalation is not required for this task.

   Use the **wait_for** module to wait for the web server to be rebooted and specify the **host** as **inventory_hostname**, **state** as **started**, **delay** as **25**, and **timeout** as **200**.

   8.1. Create a task in the **upgrade_webserver.yml** playbook. Delegate the task to the **localhost**, and use the **wait_for** module to wait for the server to restart. Specify the **host** as the **inventory_hostname** variable, **port** as **22**, **state** as **started**, **delay** as **25**, and **timeout** as **200**. The task should be executed when the variable **pageupgrade** has changed.

   Privilege escalation is not required for this task. The task in the **upgrade_webserver.yml** playbook should read as follows:

```
    - name: wait for webserver to restart
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 22
        state: started
        delay: 25
        timeout: 200
      become: False
      delegate_to: 127.0.0.1
      when: pageupgrade.changed
```

9. Create a task in the **upgrade_webserver.yml** playbook to wait for the web server port to be started. As in the preconfigured web server, the **httpd** service is configured to start at boot time. Specify the **host** as **inventory_hostname**, **port** as **80**, **state** as started, and **timeout** as **20**.

```
    - name: wait for webserver to come up
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 80
        state: started
        timeout: 20
```

10. Create a final task in the **upgrade_webserver.yml** playbook to add the web server after the upgrade of the web page back to the load balancer pool. Use the **haproxy** Ansible module to add the server back to the **HAProxy** load balancer pool. The task needs to be delegated to all members of the **[lbserver]** inventory group.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The **haproxy** module is used to enable a back end server from **HAProxy** using socket commands. To enable a back end server from the back end pool named **app**, socket path as **/var/lib/haproxy/stats**, and **wait=yes** to wait until the server reports healthy status, use the following:

```
haproxy: state=enabled backend=app host={{
inventory_hostname }} socket=/var/lib/haproxy/stats wait=yes
```

Include the following content in the **upgrade_webserver.yml** playbook:

```
    - name: enable the server in haproxy
      haproxy:
        state: enabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

11. Review the **upgrade-webserver.yml** playbook file.

The following should be the contents of **upgrade-webserver.yml**:

```
---
- name: Upgrade Webservers
  hosts: webservers
  remote_user: devops
  become: yes
  serial: 1

  tasks:
    - name: disable the server in haproxy
      haproxy:
        state: disabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"

    - name: upgrade the page
      template:
        src: "templates/index-ver1.html.j2"
        dest: "/var/www/html/index.html"
      register: pageupgrade

    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
      async: 1
      poll: 0
      ignore_errors: true
      when: pageupgrade.changed

    - name: wait for webserver to restart
      wait_for:
        host: "{{ inventory_hostname }}"
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
        port: 22
        state: started
        delay: 25
        timeout: 200
      become: False
      delegate_to: 127.0.0.1
      when: pageupgrade.changed

    - name: wait for webserver to come up
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 80
        state: started
        timeout: 20

    - name: enable the server in haproxy
      haproxy:
        state: enabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

12. Check the syntax of the playbook **`upgrade-webserver.yml`**. In case of syntax errors resolve them before proceeding to the next step.

    Check the syntax of the playbook **`upgrade-webserver.yml`**. In case of syntax errors, either resolve them or download the playbook from **`http://materials.example.com/playbooks/upgrade_webserver.yml`** before proceeding to the next step.

```
[student@workstation lab-optimizing-ansible]$ ansible-playbook --syntax-check
 upgrade_webserver.yml

playbook: upgrade_webserver.yml
```

13. Run the playbook **`upgrade-webserver.yml`** to upgrade the server.

    The restart task will take several minutes, so move on to the next step when it gets to that point in executing the playbook.



Important

The playbook will wait at the **`wait for server restart`** task up to 2 minutes to allow the web server to reboot.

```
[student@workstation lab-optimizing-ansible]$ ansible-playbook upgrade_webserver.yml

PLAY [Upgrade webservers] ******************************************************

TASK [setup] *******************************************************************
ok: [serverb.lab.example.com]

TASK [disable the server in haproxy] *******************************************
changed: [serverb.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [upgrade the page] ***********************************************************
changed: [serverb.lab.example.com]

TASK [restart machine] ************************************************************
ok: [serverb.lab.example.com]

TASK [wait for webserver to restart] ******************************************
```

14. From **workstation.lab.example.com**, use **curl** to view the web link **http://workstation.lab.example.com**.

    Run the **curl** command several times while the playbook is executing to verify the webserver is still reachable, and that the host changes after the playbook moves on to reboot the other machine.

```
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to tower.lab.example.com.
</h1>
</body>
</html>
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to serverb.lab.example.com.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

15. Wait until the remaining tasks from the playbook complete on both **serverb** and **tower**.

```
... Output omitted ...
TASK [wait for webserver to come up] ******************************************
ok: [serverb.lab.example.com]

TASK [enable the server in haproxy] *******************************************
changed: [serverb.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

PLAY [Upgrade webservers] *****************************************************

TASK [setup] ******************************************************************
ok: [tower.lab.example.com]

TASK [disable the server in haproxy] ******************************************
changed: [tower.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

TASK [upgrade the page] ***********************************************************
changed: [tower.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [restart machine] ***************************************************
ok: [tower.lab.example.com]

TASK [wait for server to restart] ****************************************
ok: [tower.lab.example.com -> localhost]

TASK [wait for webserver to come up] *************************************
ok: [tower.lab.example.com]

TASK [enable the server in haproxy] **************************************
changed: [tower.lab.example.com -> workstation.lab.example.com] =>
 (item=workstation.lab.example.com)

PLAY RECAP ***************************************************************
serverb.lab.example.com    : ok=7    changed=3    unreachable=0    failed=0
tower.lab.example.com      : ok=7    changed=3    unreachable=0    failed=0
```

16. Verify by browsing the website using the link **http://workstation.lab.example.com**. Rerun the **curl** command to see the updated pages from two different web servers, **serverb.lab.example.com** and **tower.lab.example.com**.

```
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to serverb.lab.example.com.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
[student@workstation lab-optimizing-ansible]$ curl http://
workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to tower.lab.example.com.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

Evaluation

From **workstation**, run the **lab optimize-ansible-lab** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab optimize-ansible-lab grade
```

Cleanup

Run the **lab optimize-ansible-lab cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab optimize-ansible-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- *Connection types* determine how Ansible connects to managed hosts in order to communicate with them
- By default, Ansible will use the **`smart`** connection type which will use the SSH protocol to connect with remote hosts, but other connection types are available
- Tasks can use the **`environment`** directive to set shell environment variables when executing tasks on the managed host
- Ansible can *delegate* tasks to run on a different host than the targeted managed host if actions need to be taken on a different server to configure the managed host
- Ansible normally attempts to run tasks in parallel on multiple hosts at the same time. The number of simultaneous connections can be controlled in the configuration file and in the playbook
- The **`async`** keyword triggers Ansible to run the job in the background, and the **`poll`** keyword controls how often Ansible will check to see if the task has finished

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 9

IMPLEMENTING ANSIBLE VAULT

Overview	
Goal	Manage encryption with Ansible Vault.
Objectives	• Create, edit, rekey, encrypt, and decrypt files. • Run a playbook with Ansible Vault.
Sections	• Configuring Ansible Vault (and Guided Exercise) • Executing with Ansible Vault (and Guided Exercise)
Lab	• Implementing Ansible Vault

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuring Ansible Vault

Objectives

After completing this section, students should be able to:

- Use Ansible Vault to create a new encrypted file or encrypt an existing file.
- Use Ansible Vault to view, edit, or change the password on an existing encrypted file.
- Remove encryption from a file that has been encrypted with Ansible Vault.

Ansible Vault

Ansible may need access to sensitive data such as passwords or API keys in order to configure remote servers. Normally, this information might be stored as plain text in inventory variables or other Ansible files. But in that case, any user with access to the Ansible files or a version control system which stores the Ansible files would have access to this sensitive data. This poses an obvious security risk.

There are two primary ways to store this data more securely:

- Use **Ansible Vault**, which is included with Ansible and can encrypt and decrypt any structured data file used by Ansible.
- Use a third-party key management service to store the data in the cloud, such as **Vault** by HashiCorp, Amazon's **AWS Key Management Service**, or Microsoft Azure **Key Vault**.

In this part of the course, you will learn how to use Ansible Vault.

To use Ansible Vault, a command line tool called **`ansible-vault`** is used to create, edit, encrypt, decrypt, and view files. Ansible Vault can encrypt any structured data file used by Ansible. This might include inventory variables, included variable files in a playbook, variable files passed as an argument when executing the playbook, or variables defined in Ansible roles.



Important

Ansible Vault does not implement its own cryptographic functions but uses an external Python toolkit. Files are protected with symmetric encryption using AES256 with a password as the secret key. Note that the way this is done has not been formally audited by a third party.

Create an encrypted file

To create a new encrypted file use the command **`ansible-vault create`** ***`filename`***. The command will prompt for the new vault password and open a file using the default editor. This is **`vim`**, but may be changed to **`vi`** in Ansible 2.1. A different editor may be used by setting and exporting the **`$EDITOR`** variable. For example, to set the default editor to **`nano`**, **`export EDITOR=nano`**.

```
[student@demo ~]$ ansible-vault create secret.yml
New Vault password: redhat
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
Confirm New Vault password: redhat
```

Instead of entering the vault password through standard input, a vault password file can be used to store the vault password. This file will need to be carefully protected through file permissions and other means.

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

The cipher used to protect files is AES256 in recent versions of Ansible, but files encrypted with older versions may still use 128-bit AES.

Edit an existing encrypted file

To edit an existing encrypted file, Ansible Vault provides the command **`ansible-vault edit filename`**. This command will decrypt the file to a temporary file and allows you to edit the file. When saved, it copies the content and removes the temporary file.

```
[student@demo ~]$ ansible-vault edit secret.yml
Vault password: redhat
```



Note

The **`edit`** subcommand always rewrites the file, so it should only be used when making changes. This can have implications when the file is kept under version control. The **`view`** subcommand should always be used to see the file's contents without making changes.

Change the password for an encrypted file

The vault password can be changed using the command **`ansible-vault rekey filename`**. This command can rekey multiple data files at once. It will ask for the original password and the new password.

```
[student@demo ~]$ ansible-vault rekey secret.yml
Vault password: redhat
New Vault password: RedHat
Confirm New Vault password: RedHat
Rekey successful
```

When a using vault password file, use the **`--new-vault-password-file`** option:

```
[student@demo ~]$ ansible-vault rekey \
> --new-vault-password-file=NEW_VAULT_PASSWORD_FILE secret.yml
```

Encrypting an existing file

To encrypt a file that already exists, use the command **`ansible-vault encrypt filename`**. This command can take the names of multiple files to be encrypted as arguments.

```
[student@demo ~]$ ansible-vault encrypt secret1.yml secret2.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Use the **--output=OUTPUT_FILE** option to save the encrypted file with a new name. At most one input file may be used with the **--output** option.

Viewing an encrypted file

Ansible Vault allows you to view the encrypted file using the command **ansible-vault view *filename***, without opening it for editing.

```
[student@demo ~]$ ansible-vault view secret1.yml
Vault password: secret
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
my_secret: "yJJvPqhsiusmmPPZdnjndkdnYNDjdj782meUZcw"
```

Decrypting an existing file

An already existing encrypted file can be permanently decrypted by using the command **ansible-vault decrypt *filename***. The **--output** option can be used when decrypting a single file to save the decrypted file under a different name.

```
[student@demo ~]$ ansible-vault decrypt secret1.yml --output=secret1-decrypted.yml
Vault password: redhat
Decryption successful
```



References

ansible-vault(1) man page

> Vault — Ansible Documentation
> http://docs.ansible.com/ansible/playbooks_vault.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Configuring Ansible Vault

In this exercise, you will create a new encrypted file, edit the file, and change the password on an existing encrypted file. You would also learn how to encrypt and decrypt an existing file.

Outcomes
You should be able to :

- Create a new encrypted file.
- Edit an encrypted file.
- View content of an encrypted file.
- Change the password of an encrypted file.
- Encrypt and decrypt an existing file.

Before you begin
Log into **workstation** as **student** using **student** as a password.

On **workstation**, run the **lab configure-ansible-vault setup** script. The setup script checks that Ansible is installed on **workstation**, and creates a directory structure for the lab environment.

```
[student@workstation ~]$ lab configure-ansible-vault setup
```

Steps

1. From **workstation**, as the **student** user, jump to the directory **~/conf-ansible-vault**.

```
[student@workstation ~]$ cd ~/conf-ansible-vault
[student@workstation conf-ansible-vault]$
```

2. Create an encrypted file named **super-secret.yml** under **~/conf-ansible-vault**. Use **redhat** as the vault password.

   2.1. Create an encrypted file named **super-secret.yml** under **~/conf-ansible-vault**. Enter **redhat** as the vault password when prompted, and confirm.

```
[student@workstation conf-ansible-vault]$ ansible-vault create super-secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

   2.2. Enter the following content into the file. Save and exit the file when you are finished.

```
This is encrypted.
```

3. Attempt to view the content of the encrypted file **super-secret.yml**.

```
[student@workstation conf-ansible-vault]$ cat super-secret.yml
$ANSIBLE_VAULT;1.1;AES256
30353232636462623438613666393263393238613363333735626661646265376566653765633565
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
3663386561393538333864306136316265636632386535330a653764616133343630303633323831
3365313631393363663623431646634636661333762393764396135333236316338656338383933
3635646662316335370a363264366138333434626261363465636331333539323734643363326138
3462656535383166633365313932396537633563313231316261383861356139646232303731313 2
32643865313538623962333239636131393436353235325 32346538
```

As the file **super-secret.yml** is an encrypted file, you cannot view the content in plain text. The default cipher used is *AES* (which is shared-secret based).

4. To view the content of the Ansible Vault encrypted file, use the command **ansible-vault view super-secret.yml**. When prompted, enter the vault password as **redhat**.

```
[student@workstation conf-ansible-vault]$ ansible-vault view super-secret.yml
Vault password: redhat
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
This is encrypted.
```

5. Now edit the encrypted file **super-secret.yml** to add some new content. Use **redhat** as the vault password.

   5.1. Open the **super-secret.yml** encrypted file for editing.

```
[student@workstation conf-ansible-vault]$ ansible-vault edit super-secret.yml
Vault password: redhat
```

   5.2. Enter the following content into the file. Save and exit the file when you are finished.

```
This is also encrypted.
```

6. Verify by viewing the content of **super-secret.yml**, using **ansible-vault view super-secret.yml**. Use the vault password as **redhat**.

```
[student@workstation conf-ansible-vault]$ ansible-vault view super-secret.yml
Vault password: redhat
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
This is encrypted.
This is also encrypted.
```

7. Download the encrypted **passwd.yml** file from **http://materials.example.com/playbooks/passwd.yml**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation conf-ansible-vault]$ wget http://materials.example.com/
playbooks/passwd.yml
```

8. Change the vault password of the **passwd.yml** file, using the command **ansible-vault rekey passwd.yml**. The current password for the **passwd.yml** file is **redhat**. Change this password to **ansible**.

```
[student@workstation conf-ansible-vault]$ ansible-vault rekey passwd.yml
Vault password: redhat
New Vault password: ansible
Confirm New Vault password: ansible
Rekey successful
```

9. Decrypt the encrypted file **passwd.yml** and save the file as **passwd-decrypted.yml**. Use the **ansible-vault decrypt** subcommand with the *--output* option. Enter the vault password as **ansible**.

```
[student@workstation conf-ansible-vault]$ ansible-vault decrypt passwd.yml --
output=passwd-decrypted.yml
Vault password: ansible
Decryption successful
```

10. Verify the file **passwd-decrypted.yml** is decrypted by viewing its content using **cat**.

```
[student@workstation conf-ansible-vault]$ cat passwd-decrypted.yml
user_pw: 5pjsBJxAWUs6pRWD5itO/
```

11. Encrypt the existing file **passwd-decrypted.yml** and save the file as **passwd-encrypted.yml**. Use the **ansible-vault encrypt** subcommand with the *--output*. Enter the vault password as **redhat** and confirm by re-entering the password.

```
[student@workstation conf-ansible-vault]$ ansible-vault encrypt passwd-decrypted.yml
 --output=passwd-encrypted.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

Evaluation

From **workstation**, run the **lab configure-ansible-vault** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab configure-ansible-vault grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Executing with Ansible Vault

Objectives

After completing this section, students should be able to:

- Run a playbook referencing files encrypted with Ansible Vault.

Playbooks and Ansible Vault

In order to run a playbook that accesses files encrypted with Ansible Vault, the encryption password needs to be provided to the **ansible-playbook** command. If the command is run without doing this, it will return an error:

```
[student@demo ~]$ ansible-playbook site.yml
ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

To provide the vault password interactively, use the **--ask-vault-pass** option.

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml
Vault password: redhat
```

Alternatively, a file that stores the encryption password in plain text can be used by specifying it with the **--vault-password-file** option. The password should be a string stored as a single line in the file. Since that file contains the sensitive plain text password, it is vital that it be protected through file permissions and other security measures.

```
[student@demo ~]$ ansible-playbook --vault-password-file=vault-pw-file site.yml
```

The default location of the password file can also be specified by using the **$ANSIBLE_VAULT_PASSWORD_FILE** environment variable.



Important

All files protected by Ansible Vault that are used by a playbook must be encrypted using the same password.

Recommended practices for variable file management

To simplify management, it makes sense to set up your Ansible project so that sensitive variables and all other variables are kept in separate files. The file or files containing the sensitive variables can then be protected with the **ansible-vault** command.

Remember that the preferred way to manage group variables and host variables is to create directories at the playbook level. The **group_vars** directory normally contains variable files with names matching host groups to which they apply. The **host_vars** directory normally contains variable files with names matching hostnames of managed hosts to which they apply.

However, instead of using files in **group_vars** or **host_vars**, you can use *directories* for each host group or managed host. Those directories can then contain multiple variable files, all of which are used by the host group or managed host. For example, in the following project

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

directory for **playbook.yml**, members of the **webservers** host group will use variables in the **group_vars/webservers/vars** file, and **demo.example.com** will use the variables in both **host_vars/demo.example.com/vars** and **host_vars/demo.example.com/vault**:

```
.
├── ansible.cfg
├── group_vars
│   └── webservers
│       └── vars
├── host_vars
│   └── demo.example.com
│       ├── vars
│       └── vault
├── inventory
└── playbook.yml
```

In this scenario, the advantage is that most variables for **demo.example.com** can be placed in **vars**, but sensitive variables can be placed in **vault**. Then the administrator can use **ansible-vault** to encrypt **vault**, while leaving **vars** as plain text.

There is nothing special about the file names being used in this example inside the **host_vars/demo.example.com** directory. That directory could contain more files, some encrypted by Ansible Vault and some which are not.

Playbook variables (as opposed to inventory variables) can also be protected with Ansible Vault. Sensitive playbook variables can be placed in a separate file which is encrypted with Ansible Vault and which is included into the playbook through a **vars_files** directive. This can be useful, since playbook variables take precedence over inventory variables.

Speeding up Vault operations

By default, Ansible uses functions from the *python-crypto* package to encrypt and decrypt vault files. If there are many encrypted files, decrypting them at start-up may cause a perceptible delay. To speed this up, install the *python-cryptography* package:

```
[student@demo ~]$ sudo yum install python-cryptography
```

The *python-cryptography* package provides a Python library which exposes cryptographic recipes and primitives. The default Ansible installation uses **PyCrypto** for these cryptographic operations.

Demonstration: Executing with Ansible Vault

1. Log in to **workstation** as the **student** user. Jump to the **~/exec-ansible-vault** directory.

   ```
   [student@workstation ~]$ cd ~/exec-ansible-vault
   ```

2. Create an encrypted file named **secret.yml** in **~/exec-ansible-vault/vars/** which will contain sensitive playbook variables. Provide a password of **redhat** for the vault and confirm it. This will open a new file in the default text editor, **vim**.

   ```
   [student@workstation exec-ansible-vault]$ ansible-vault create vars/secret.yml
   New Vault password: redhat
   Confirm New Vault password: redhat
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3. Once in the text editor, define an associative array variable called `newusers`. Each entry should have two keys: **`name`** for the username and **`pw`** for the password.

   Define one user with a name of **`demouser1`** and a password of **`redhat`**. Define a second user with a name of **`demouser2`** and a password of **`RedHat`**.

```
newusers:
  - name: demouser1
    pw: redhat
  - name: demouser2
    pw: RedHat
```

   Save the changes and exit the editor. This will create **`vars/secret.yml`**.

```
[student@workstation exec-ansible-vault]$ tree
.
├── ansible.cfg
├── createusers.yml
├── inventory
│   └── hosts
└── vars
    └── secret.yml

2 directories, 4 files
[student@workstation exec-ansible-vault]$ file vars/secret.yml
vars/secret.yml: ASCII text
```

4. Display the contents of the **`create_users.yml`** playbook. Note how it references **`vars/secret.yml`** as an external playbook variables file.

```
---
- name: create user accounts for all our servers
  hosts: devservers
  remote_user: devops
  become: yes
  vars_files:
    - vars/secret.yml
  tasks:
    - name: Creating users from secret.yml
      user:
        name: "{{ item.name }}"
        password: "{{ item.pw | password_hash('sha512') }}"
      with_items: "{{ newusers }}"
```

5. Use **`ansible-playbook --syntax-check`** to check the syntax of the **`create_users.yml`** playbook,

```
[student@workstation exec-ansible-vault]$ ansible-playbook --syntax-check \
> create_users.yml
ERROR! Decryption failed
```

   It failed because it was unable to decrypt **`vars/secret.yml`** to check its syntax. Add the **`--ask-vault-pass`** option to prompt for the vault password while decrypting **`vars/secret.yml`**. In case of any syntax error, resolve before continuing further.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation exec-ansible-vault]$ ansible-playbook --syntax-check \
> --ask-vault-pass create_users.yml
Vault password: redhat

playbook: create_users.yml
```

6. Create a password file, called **vault-pass**, to use for the playbook execution instead of asking for a password. Store the vault password **redhat** as plain text. Change the permission of the file to **0600**.

```
[student@workstation exec-ansible-vault]$ echo 'redhat' > vault-pass
[student@workstation exec-ansible-vault]$ chmod 0600 vault-pass
```

7. Execute the Ansible playbook, this time using the vault password file. This creates the **demouser1** and **demouser2** users on the managed hosts using the passwords stored as the **pw** fields in **secret.yml**.

```
[student@workstation exec-ansible-vault]$ ansible-playbook \
> --vault-password-file=vault-pass create_users.yml
```

8. Verify that both users (**demouser1** and **demouser2**) were created properly by the playbook. Connect to **servera.lab.example.com** via SSH as those users.

   The **-o PreferredAuthentications=password** option must be used, because **servera** has been configured with SSH keys that permit authentication to the system without a password. In this case, we want to test out the password, so force SSH to ignore the SSH key.

   - Log in to **servera.lab.example.com** as the **demouser1** user, using the password of **redhat**. Exit when you are finished.

```
[student@workstation exec-ansible-vault]$ ssh \
> -o PreferredAuthentications=password demouser1@servera.lab.example.com
demouser1@servera.lab.example.com's password: redhat
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to the
 list of known hosts.
[demouser1@servera ~]$ exit
```

   - Log in to **servera.lab.example.com** as the **demouser2** user, using the password of **RedHat**. Exit when you are finished.

```
[student@workstation exec-ansible-vault]$ ssh \
> -o PreferredAuthentications=password demouser2@servera.lab.example.com
demouser2@servera.lab.example.com's password: RedHat
Last login: Fri Apr  8 10:30:58 2016 from workstation.lab.example.com
[demouser2@servera ~]$ exit
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

`ansible-playbook`(1) and **`ansible-vault`**(1) man pages

> Running a Playbook With Vault — Ansible Documentation
> http://docs.ansible.com/ansible/playbooks_vault.html#running-a-playbook-with-vault

> Variables and Vaults — Ansible Documentation
> http://docs.ansible.com/ansible/playbooks_best_practices.html#best-practices-for-variables-and-vaults

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Executing with Ansible Vault

In this exercise, you will use Ansible Vault to encypt the file containing passwords on the local system and use that in a playbook to create users on the **servera.lab.example.com** remote system.

Outcomes
You should be able to:
- Use the variables defined in the encrypted file to execute a playbook.

Before you begin
Log into **workstation** as **student** using **student** as a password.

On **workstation**, run the **lab execute-ansible-vault setup** script. This script ensures that Ansible is installed on **workstation** and creates a directory structure for the lab environment. This directory structure includes an inventory file that points to **servera.lab.example.com** as a managed host, which is part of the **devservers** group.

```
[student@workstation ~]$ lab execute-ansible-vault setup
```

Steps
1. From **workstation**, as the **student** user, jump to the directory **~/exec-ansible-vault**.

   ```
   [student@workstation ~]$ cd ~/exec-ansible-vault
   ```

2. Create an encrypted file named **secret.yml** in the **~/exec-ansible-vault/** directory. This file will define the password variables and store the passwords to be used in the playbook.

   Use the associative array variable **newusers** to define user and password with name variable as **ansibleuser1** and **ansibleuser2** and pw variable as **redhat** and **Re4H1T** respectively.

   Set the vault password to **redhat**.

   2.1. Create an encrypted file named **secret.yml** in **~/exec-ansible-vault/**. Provide a password of **redhat** for the vault and confirm it. This will open a file in the default editor **vim**.

   ```
   [student@workstation exec-ansible-vault]$ ansible-vault create secret.yml
   New Vault password: redhat
   Confirm New Vault password: redhat
   ```

   2.2. Add a associative array variable, named newusers, containing key value pair of user name and password as follows:

   ```
   newusers:
     - name: ansibleuser1
       pw: redhat
     - name: ansibleuser2
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    pw: Re4H1T
```

Save the file.

3. Create a playbook which will use the variables defined in the **secret.yml** encrypted file. Name the playbook **create_users.yml** and create it under the **~/exec-ansible-vault/** directory.

   Configure the playbook to use the **devservers** host group, which was defined by the lab setup script in the inventory file. Run this playbook as the **devops** user on the remote managed host. Configure the playbook to create the **ansibleuser1** and **ansibleuser2** users.

   The password stored as plain text in the variable, pw, should be converted into password hash using hashing filters **password_hash** to get **SHA512** hashed password and passed as an argument to the **user** module. For example,

```
user:
  name: user1
  password: "{{ 'passwordsaresecret' | password_hash('sha512') }}"
```

   The content of the **create_users.yml** should be:

```
---
- name: create user accounts for all our servers
  hosts: devservers
  become: True
  remote_user: devops
  vars_files:
    - secret.yml
  tasks:
    - name: Creating users from secret.yml
      user:
        name: "{{ item.name }}"
        password: "{{ item.pw | password_hash('sha512') }}"
      with_items: "{{ newusers }}"
```

4. Check the syntax of the **create_users.yml** using **ansible-playbook --syntax-check**. Use the *--ask-vault-pass* option to prompt for the vault password set on **secret.yml**. In case of syntax error, resolve before continuing further.

```
[student@workstation exec-ansible-vault]$ ansible-playbook --syntax-check --ask-
vault-pass create_users.yml
Vault password: redhat

playbook: create_users.yml
```

5. Create a password file to use for the playbook execution instead of asking for a password. The file should be called **vault-pass** and it should store the **redhat** vault password as a plain text. Change the permission of the file to **0600**.

```
[student@workstation exec-ansible-vault]$ echo 'redhat' > vault-pass
[student@workstation exec-ansible-vault]$ chmod 0600 vault-pass
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6. Execute the Ansible playbook, using the vault password file to create the **ansibleuser1** and **ansibleuser2** users on a remote system using the passwords stored as variables in the **secret.yml** Ansible Vault encrypted file. Use the vault password file **vault-pass**.

```
[student@workstation exec-ansible-vault]$ ansible-playbook --vault-password-
file=vault-pass create_users.yml

PLAY [create user accounts for all our servers] ******************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [Creating users from secret.yml] ******************************************
changed: [servera.lab.example.com] => (item={u'name': u'ansibleuser1', u'pw':
 u'redhat'})
changed: [servera.lab.example.com] => (item={u'name': u'ansibleuser2', u'pw':
 u'Re4H1T'})

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

7. Verify that both users were created properly by the playbook by connecting via SSH to **servera.lab.example.com**. Since we want to test the password, force SSH to skip the SSH key. Use the **-o PreferredAuthentications=password** option to SSH, because **servera** has been configured with an SSH key that will authenticate without a password.

   7.1. Login to **servera.lab.example.com** as the **ansibleuser1** user, using the password of **redhat**, to verify the user was created properly by the playbook. Exit when you are finished.

```
[student@workstation exec-ansible-vault]$ ssh -o
 PreferredAuthentications=password ansibleuser1@servera.lab.example.com
ansibleuser1@servera.lab.example.com's password: redhat
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to
 the list of known hosts.
[ansibleuser1@servera ~]$ exit
```

   7.2. Login to **servera.lab.example.com** as the **ansibleuser2** user using the password of **Re4H1T** to verify the user was created properly by the playbook. Exit when you are finished.

```
[student@workstation exec-ansible-vault]$ ssh -o
 PreferredAuthentications=password ansibleuser2@servera.lab.example.com
ansibleuser2@servera.lab.example.com's password: Re4H1T
Last login: Fri Apr  8 10:30:58 2016 from workstation.lab.example.com
[ansibleuser2@servera ~]$ exit
```

Evaluation

From **workstation**, run the **lab execute-ansible-vault** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab execute-ansible-vault grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Cleanup

Run the **`lab execute-ansible-vault cleanup`** command to **`cleanup`** the lab.

```
[student@workstation ~]$ lab execute-ansible-vault cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Ansible Vault

In this lab, you will encrypt and decrypt the YAML file containing variables for LUKS encryption which are sensitive. Use the encrypted file containing variables in a playbook to execute remote tasks on **serverb.lab.example.com** to create a LUKS encrypted partition on **/dev/vdb**. Edit the encrypted role variable file to add the path of the new 256-bit key file and add tasks to insert this key to an available key slot on the encrypted device **/dev/vdb** on **serverb.lab.example.com**.

Outcomes
You should be able to:

- Encrypt a file.
- View an encrypted file.
- Run playbooks using an encrypted file.
- Change the password of an existing encrypted file.
- Edit and decrypt an existing encrypted file.

Before you begin
Log into **workstation** as **student**, using **student** as a password. Run the **lab ansible-vault-lab setup** command.

```
[student@workstation ~]$ lab ansible-vault-lab setup
```

This command confirms that Ansible is installed on **workstation** and it creates a directory structure for the lab environment. It also creates the **~student/lab-ansible-vault/inventory/hosts** inventory file that includes **serverb.lab.example.com** as a managed host that is part of the **[prodservers]** host group.

Steps

1. Log in as the **student** user on **workstation**. Change to the **~/lab-ansible-vault** project directory.
2. Use the **ansible-galaxy** command to create a role named **encryptdisk** and its directory structure.
3. Edit the role variable file to add the following variables:

Variable name	Variable value
luks_dev	/dev/vdb
luks_name	crypto
luks_pass	Re4H1TAns1BLe

```
---
# vars file for encryptdisk
luks_dev: /dev/vdb
luks_name: crypto
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
luks_pass: Re4H1TAns1BLe
```

4. Encrypt the role variable file. Use **redhat** as the the vault password.

5. Create task to encrypt a block device as specified using the *luks_dev* variable.

   Download the task file from **http://materials.example.com/playbooks/encryptdisk-tasks.yml** and configure this file as a task for your playbook. These tasks will encrypt a block device using the **luks_*** variables defined in the role's variables file.

6. Create a playbook named **encrypt.yml** directly under the project directory that calls the **encryptdisk** role. Apply the role to the **prodservers** inventory host group.

7. Check the syntax of the playbook and ensure the roles are defined correctly. Use the **redhat** vault password when prompted.

   If the playbook passed the syntax check, run the playbook to create an encrypted disk using the **/dev/vdb** block device on **serverb** and mount it under **/crypto**.

8. Verify the Ansible playbook created **/dev/vdb** as a LUKS disk partition and mounted it as **/crypto** on **serverb.lab.example.com**.

9. Download **keyfile-encrypted.j2** from **http://materials.example.com/playbooks/keyfile-encrypted.j2**. and rekey it. The original password for the file is **RedHat**. Change this password to **redhat**.

10. Permanently decrypt the Ansible Vault encrypted key file, **keyfile-encrypted.j2**, and name the new file **keyfile.j2**. This file will be added to an available key slot on the LUKS encrypted device on **serverb**. The vault password for this template file is **redhat**.

    Store the decrypted **keyfile.j2** in the **encryptdisk** role's **templates** directory.

11. Edit the encrypted role variable file to add a new **luks_key** variable which points to the decrypted **keyfile.j2** file created in the previous step. Use the vault password of **redhat**.

12. Add a default **addkey** variable in **roles/encryptdisk/defaults/main.yml** and set the value to **no**. This variable would be used as a conditional for adding a key file to the LUKS encrypted device.

13. Add a new task to the **encryptdisk** role to:
    - Copy the decrypted **keyfile.j2** key file from the role **templates** directory to **serverb.lab.example.com** as **/root/keyfile**. Set the owner to **root**, the group to **root**, and the mode to **0600**.
    - Use the **luks_key** variable to substitute the path of the key file stored in the encrypted role's variables file. This task should be executed when the *addkey: yes* variable is defined when calling the **encryptdisk** role.

14. Add another task to the **encryptdisk** role to:
    - Use a command similar to the following example to add the key file to an available key slot on the encrypted disk on **serverb.lab.example.com**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Use the **luks_pass** and **luks_dev** variables to substitute the passphrase used earlier to encrypt the disk and the name of the device. This task should be invoked when the *addkey=yes* parameter is passed as an argument when running the playbook.

The following example shows how to add a key file to an available key slot on an encrypted LUKS disk. Replace *password* with the proper password for the device, *devicename* with the proper encrypted device name, and */path/to/keyfilename* with the proper path to the keyfile.

```
echo password | cryptsetup luksAddKey devicename /path/to/keyfilename
```

15. Edit the **~/lab-ansible-vault/encrypt.yml** playbook to specify the **addkey** variable to the **encryptdisk** role. Set the value of **addkey** to **yes**.

16. Check the syntax of the playbook and ensure roles are defined correctly. Use the **redhat** vault password when prompted.

    If the playbook passed the syntax check, run the playbook to add the key file to the encrypted device on **serverb.lab.example.com**.

17. Verify the play worked correctly by accessing the LUKS encrypted file, using the key file added to the encrypted disk on **serverb.lab.example.com**. Use the following commands on **serverb** to sequentially verify the encrypted disk using the **cryptsetup** command:

    17.1. Dump the header information of a LUKS device using the **cryptsetup luksDump /dev/vdb** command.

    17.2. Unmount the mounted volume to close the encrypted device.

    17.3. Remove the existing **crypto** mapping and wipe the key from kernel memory.

    17.4. Open the **/dev/vdb** LUKS device and set up a **crypto** mapping after successful verification of the supplied **/root/keyfile** key file.

    17.5. Mount the filesystems found in **/etc/fstab**.

    17.6. List information about all available block devices.

    Exit **serverb.lab.example.com** when finished.

Evaluation

Run the **lab ansible-vault-lab** command on **workstation**, with the *grade* argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-vault-lab grade
```

Cleanup

Run the **lab ansible-vault-lab cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab ansible-vault-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will encrypt and decrypt the YAML file containing variables for LUKS encryption which are sensitive. Use the encrypted file containing variables in a playbook to execute remote tasks on **serverb.lab.example.com** to create a LUKS encrypted partition on **/dev/vdb**. Edit the encrypted role variable file to add the path of the new 256-bit key file and add tasks to insert this key to an available key slot on the encrypted device **/dev/vdb** on **serverb.lab.example.com**.

Outcomes

You should be able to:

- Encrypt a file.
- View an encrypted file.
- Run playbooks using an encrypted file.
- Change the password of an existing encrypted file.
- Edit and decrypt an existing encrypted file.

Before you begin

Log into **workstation** as **student**, using **student** as a password. Run the **lab ansible-vault-lab setup** command.

```
[student@workstation ~]$ lab ansible-vault-lab setup
```

This command confirms that Ansible is installed on **workstation** and it creates a directory structure for the lab environment. It also creates the **~student/lab-ansible-vault/inventory/hosts** inventory file that includes **serverb.lab.example.com** as a managed host that is part of the **[prodservers]** host group.

Steps

1. Log in as the **student** user on **workstation**. Change to the **~/lab-ansible-vault** project directory.

```
[student@workstation ~]$ cd ~/lab-ansible-vault
[student@workstation lab-ansible-vault]$
```

2. Use the **ansible-galaxy** command to create a role named **encryptdisk** and its directory structure.

```
[student@workstation lab-ansible-vault]$ ansible-galaxy init --offline -p roles/
 encryptdisk
- encryptdisk was created successfully
```

3. Edit the role variable file to add the following variables:

Variable name	Variable value
luks_dev	/dev/vdb
luks_name	crypto

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Variable name	Variable value
luks_pass	Re4H1TAns1BLe

Edit the **encryptdisk** role variable file, **~/lab-ansible-vault/roles/encryptdisk/vars/main.yml**, to add the following variables:

```
---
# vars file for encryptdisk
luks_dev: /dev/vdb
luks_name: crypto
luks_pass: Re4H1TAns1BLe
```

4. Encrypt the role variable file. Use **redhat** as the the vault password.

   Use **ansible-vault** to encrypt the **roles/encryptdisk/vars/main.yml** role variable file.

```
[student@workstation lab-ansible-vault]$ ansible-vault encrypt roles/encryptdisk/
vars/main.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

   Use **ansible-vault view** to display the encrypted file and confirm that the vault password works.

```
[student@workstation lab-ansible-vault]$ ansible-vault view roles/encryptdisk/vars/
main.yml
Vault password: redhat
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
---
# vars file for encryptdisk
luks_dev: /dev/vdb
luks_name: crypto
luks_pass: Re4H1TAns1BLe
```

5. Create task to encrypt a block device as specified using the *luks_dev* variable.

   Download the task file from **http://materials.example.com/playbooks/encryptdisk-tasks.yml** and configure this file as a task for your playbook. These tasks will encrypt a block device using the **luks_*** variables defined in the role's variables file.

   5.1. Download the task file from **http://materials.example.com/playbooks/encryptdisk-tasks.yml**.

```
[student@workstation lab-ansible-vault]$ wget http://materials.example.com/
playbooks/encryptdisk-tasks.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5.2. Move the task file to **roles/encryptdisk/tasks/main.yml**. This task will encrypt a block using the **luks_*** variables.

```
[student@workstation lab-ansible-vault]$ mv encryptdisk-tasks.yml roles/
encryptdisk/tasks/main.yml
```

6. Create a playbook named **encrypt.yml** directly under the project directory that calls the **encryptdisk** role. Apply the role to the **prodservers** inventory host group.

Create the **~/lab-ansible-vault/encrypt.yml** playbook with the following content:

```
---
- name: Encrypt disk on serverb using LUKS
  hosts: prodservers
  remote_user: devops
  become: yes
  roles:
    - encryptdisk
```

7. Check the syntax of the playbook and ensure the roles are defined correctly. Use the **redhat** vault password when prompted.

```
[student@workstation lab-ansible-vault]$ ansible-playbook --syntax-check --ask-
vault-pass encrypt.yml
Vault password: redhat

playbook: encrypt.yml
```

If the playbook passed the syntax check, run the playbook to create an encrypted disk using the **/dev/vdb** block device on **serverb** and mount it under **/crypto**.

```
[student@workstation lab-ansible-vault]$ ansible-playbook --ask-vault-pass
 encrypt.yml
Vault password: redhat

PLAY [Encrypt disk on serverb using LUKS] ***********************************

TASK [setup] ****************************************************************
ok: [serverb.lab.example.com]

TASK [encryptdisk : Check if device already unlocked.] ************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : Umount volumes] *****************************************
skipping: [serverb.lab.example.com]

TASK [encryptdisk : Close disk... because of crypting/action requested] ********
skipping: [serverb.lab.example.com]

TASK [encryptdisk : Check if device already unlocked.] ************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : encrypt disk] *******************************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : open encrypted disk] ************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
changed: [serverb.lab.example.com]

TASK [encryptdisk : Create directory /crypto] *********************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : mkfs on /dev/vdb] *****************************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : create cryptab file] **************************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : mount the /dev/mapper/crypto] *****************************
changed: [serverb.lab.example.com]

PLAY RECAP ********************************************************************
serverb.lab.example.com    : ok=9    changed=8    unreachable=0    failed=0
```

8. Verify the Ansible playbook created **/dev/vdb** as a LUKS disk partition and mounted it as **/crypto** on **serverb.lab.example.com**.

```
[student@workstation lab-ansible-vault]$ ansible prodservers -a 'lsblk'
serverb.lab.example.com | SUCCESS | rc=0 >>
NAME     MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
vda      253:0    0   40G  0 disk
└─vda1   253:1    0   40G  0 part  /
vdb      253:16   0    1G  0 disk
└─crypto 252:0    0 1022M  0 crypt /crypto
```

9. Download **keyfile-encrypted.j2** from **http://materials.example.com/playbooks/keyfile-encrypted.j2**. and rekey it. The original password for the file is **RedHat**. Change this password to **redhat**.

```
[student@workstation lab-ansible-vault]$ wget http://materials.example.com/
playbooks/keyfile-encrypted.j2
... Output omitted ...
[student@workstation lab-ansible-vault]$ ansible-vault rekey keyfile-encrypted.j2
Vault password: RedHat
New Vault password: redhat
Confirm New Vault password: redhat
Rekey successful
```

10. Permanently decrypt the Ansible Vault encrypted key file, **keyfile-encrypted.j2**, and name the new file **keyfile.j2**. This file will be added to an available key slot on the LUKS encrypted device on **serverb**. The vault password for this template file is **redhat**.

    Store the decrypted **keyfile.j2** in the **encryptdisk** role's **templates** directory.

```
[student@workstation lab-ansible-vault]$ ansible-vault decrypt keyfile-encrypted.j2
 --output=roles/encryptdisk/templates/keyfile.j2
Vault password: redhat
Decryption successful
```

11. Edit the encrypted role variable file to add a new **luks_key** variable which points to the decrypted **keyfile.j2** file created in the previous step. Use the vault password of **redhat**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Edit the encrypted role variable file, **roles/encryptdisk/vars/main.yml**, to add a new variable **luks_key: templates/keyfile.j2**.

```
[student@workstation lab-ansible-vault]$ ansible-vault edit roles/encryptdisk/vars/
main.yml
Vault password: redhat
```

```
---
# vars file for encryptdisk
luks_dev: /dev/vdb
luks_name: crypto
luks_pass: Re4H1TAns1BLe
luks_key: templates/keyfile.j2
```

12. Add a default **addkey** variable in **roles/encryptdisk/defaults/main.yml** and set the value to **no**. This variable would be used as a conditional for adding a key file to the LUKS encrypted device.

```
---
# defaults file for encryptdisk
addkey: no
```

13. Add a new task to the **encryptdisk** role to:
    - Copy the decrypted **keyfile.j2** key file from the role **templates** directory to **serverb.lab.example.com** as **/root/keyfile**. Set the owner to **root**, the group to **root**, and the mode to **0600**.
    - Use the **luks_key** variable to substitute the path of the key file stored in the encrypted role's variables file. This task should be executed when the *addkey: yes* variable is defined when calling the **encryptdisk** role.

    Edit the role's task file, **roles/encryptdisk/tasks/main.yml**, to include the new task. The contents of the file should look like the following:

```
---
# tasks file for Ansible Vault lab
... Output omitted ...
  - name: copying the key file
    template:
      src: "{{ luks_key }}"
      dest: /root/keyfile
      owner: root
      group: root
      mode: 0600
    when: addkey
```

14. Add another task to the **encryptdisk** role to:
    - Use a command similar to the following example to add the key file to an available key slot on the encrypted disk on **serverb.lab.example.com**.
    - Use the **luks_pass** and **luks_dev** variables to substitute the passphrase used earlier to encrypt the disk and the name of the device. This task should be invoked when the *addkey=yes* parameter is passed as an argument when running the playbook.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The following example shows how to add a key file to an available key slot on an encrypted LUKS disk. Replace *password* with the proper password for the device, *devicename* with the proper encrypted device name, and */path/to/keyfilename* with the proper path to the keyfile.

```
echo password | cryptsetup luksAddKey devicename /path/to/keyfilename
```

Add the following lines to the **roles/encryptdisk/tasks/main.yml** task file:

```
---
# tasks file for Ansible Vault lab
... Output omitted ...
  - name: add new keyslot to encrypted disk
    shell: echo {{ luks_pass }} | cryptsetup luksAddKey {{ luks_dev }} /root/keyfile
    when: addkey
```

15. Edit the **~/lab-ansible-vault/encrypt.yml** playbook to specify the **addkey** variable to the **encryptdisk** role. Set the value of **addkey** to **yes**.

```
---
- name: Encrypt disk on serverb using LUKS
  hosts: prodservers
  remote_user: devops
  become: yes
  roles:
    - role: encryptdisk
      addkey: yes
```

16. Check the syntax of the playbook and ensure roles are defined correctly. Use the **redhat** vault password when prompted.

```
[student@workstation lab-ansible-vault]$ ansible-playbook --syntax-check --ask-
vault-pass encrypt.yml
Vault password: redhat

playbook: encrypt.yml
```

If the playbook passed the syntax check, run the playbook to add the key file to the encrypted device on **serverb.lab.example.com**.

```
[student@workstation lab_ansible_vault]$ ansible-playbook --ask-vault-pass
 encrypt.yml
Vault password: redhat
... Output omitted ...
TASK [encryptdisk : copying the key file] ***********************************
changed: [serverb.lab.example.com]

TASK [encryptdisk : add new keyslot to encrypted disk] ************************
changed: [serverb.lab.example.com]

PLAY RECAP ******************************************************************
serverb.lab.example.com    : ok=13   changed=10   unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

17. Verify the play worked correctly by accessing the LUKS encrypted file, using the key file added to the encrypted disk on **serverb.lab.example.com**. Use the following commands on **serverb** to sequentially verify the encrypted disk using the **cryptsetup** command:

    17.1. Dump the header information of a LUKS device using the **cryptsetup luksDump /dev/vdb** command.

```
[student@workstation lab-ansible-vault]$ ssh root@serverb
[root@serverb ~]# cryptsetup luksDump /dev/vdb
... Output omitted ...
Key Slot 0: ENABLED
 Iterations:          319599
 Salt:                71 ba be b7 8a 1d db 4f c7 64 d0 58 3e 73 a8 48
                        84 a3 4b 12 54 96 59 09 4d 3c 4e 24 d2 67 e7 a2
 Key material offset: 8
 AF stripes:             4000
Key Slot 1: ENABLED
 Iterations:          332899
 Salt:                55 30 60 ce 4c e5 a0 a6 49 fe e0 86 c0 56 da 2d
                        56 0c 88 54 1c 07 27 27 18 b9 ec 22 99 e8 ab 2e
 Key material offset: 264
 AF stripes:             4000
... Output omitted ...
```

    17.2. Unmount the mounted volume to close the encrypted device.

```
[root@serverb ~]# umount /crypto
```

    17.3. Remove the existing **crypto** mapping and wipe the key from kernel memory.

```
[root@serverb ~]# cryptsetup close crypto
```

    17.4. Open the **/dev/vdb** LUKS device and set up a **crypto** mapping after successful verification of the supplied **/root/keyfile** key file.

```
[root@serverb ~]# cryptsetup open /dev/vdb crypto -d /root/keyfile
```

    17.5. Mount the filesystems found in **/etc/fstab**.

```
[root@serverb ~]# mount -a
```

    17.6. List information about all available block devices.

```
[root@serverb ~]# lsblk
NAME     MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
vda      253:0    0   40G  0 disk
└─vda1   253:1    0   40G  0 part  /
vdb      253:16   0    1G  0 disk
└─crypto 252:0    0 1022M  0 crypt /crypto
```

    Exit **serverb.lab.example.com** when finished.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[root@serverb ~]# exit
[student@workstation lab-ansible-vault]$
```

Evaluation

Run the **lab ansible-vault-lab** command on **workstation**, with the *grade* argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-vault-lab grade
```

Cleanup

Run the **lab ansible-vault-lab cleanup** command to cleanup after the lab.

```
[student@workstation ~]$ lab ansible-vault-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- *Ansible Vault* is one way to protect sensitive data like password hashes and private keys for deployment using Ansible playbooks
- Ansible Vault can use symmetric encryption (normally AES-256) to encrypt and decrypt any structured data file used by Ansible
- Ansible Vault can be used to create and encrypt a text file if it does not already exist or encrypt and decrypt existing files.
- All Vault files used by a playbook need to use the same password
- It is recommended that users keep most variables in a normal file and sensitive variables in a second file protected by Ansible Vault
- Ansible Vault operations can be run faster on Red Hat Enterprise Linux or CentOS by installing the *python-cryptography* package.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 10

TROUBLESHOOTING ANSIBLE

Overview	
Goal	Troubleshoot playbooks and managed hosts.
Objectives	• Troubleshoot playbooks • Troubleshoot managed hosts
Sections	• Troubleshooting Playbooks (and Guided Exercise) • Troubleshooting Ansible Managed Hosts (and Guided Exercise)
Lab	• Troubleshooting Ansible

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Troubleshooting Playbooks

Objectives

After completing this section, students should be able to:

- Discuss common problems with playbooks
- Discuss tips to troubleshoot playbook issues
- Discuss recommended practices for playbook management

Log Files in Ansible

By default, Ansible is not configured to log its output to any log file. It provides a built-in logging infrastructure that can be configured through the **log_path** parameter in the **default** section of the **ansible.cfg** configuration file, or through the **$ANSIBLE_LOG_PATH** environment variable. If any or both are configured, Ansible will store output from both the **ansible** and **ansible-playbook** commands in the log file configured either through the **ansible.cfg** configuration file, or the **$ANSIBLE_LOG_PATH** environment variable.

If Ansible log files are to be kept in the default log file directory, **/var/log**, then the playbooks must be run as **root** or the permissions on **/var/log** must be opened up. More frequently log files are created in the local playbook directory.



Note

Red Hat recommends that you configure **logrotate** to manage Ansible's log file.

The Debug module

One of the modules available for Ansible, the *debug* module, provides a better insight into what is happening on the control node. This module can provide the value for a certain variable at playbook execution time. This feature is key to managing tasks that use variables to communicate with each other (for example, using the output of a task as the input to the following one). The following examples make use of the *msg* and *var* statements inside of the *debug* statement, to show the value at execution time of both the **ansible_memfree_mb** fact and the **output** variable.

```
- debug: msg="The free memory for this system is {{ ansible_memfree_mb }}"
```

```
- debug: var=output verbosity=2
```

Managing errors

There are several issues than can occur during a playbook run, mainly related to the syntax of either the playbook or any of the templates it uses, or due to connectivity issues with the managed hosts (for example, an error in the host name of the managed host in the inventory file). Those errors are issued by the **ansible-playbook** command at execution time. The

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

--syntax-check option checks the YAML syntax for the playbook. If a playbook has a high number of tasks, it may be useful to use either the **--step**, or the **--start-at-task** options in the **ansible-playbook** command. The **--step** option executes tasks interactively, asking if it should execute that task. The **--start-at-task** option starts the execution from a given task and avoid repeating all the previous tasks execution.

```
[student@demo ~]#ansible-playbook play.yml --step
```

```
[student@demo ~]#ansible-playbook play.yml --start-at-task="start httpd service"
```

```
[student@demo ~]#ansible-playbook play.yml --syntax-check
```

Debugging with ansible-playbook

The output given by a playbook run with the **ansible-playbook** command is a good starting point to start troubleshooting issues related to hosts managed by Ansible. For example, a playbook is executed and the following output appears:

```
PLAY [playbook] ****************************************************************
... Output omitted ...
TASK: [Install a service] ******************************************************
ok: [demoservera]
ok: [demoserverb]

PLAY RECAP *********************************************************************
demoservera                : ok=2    changed=0    unreachable=0    failed=0
demoserverb                : ok=2    changed=0    unreachable=0    failed=0
```

The previous output shows a **PLAY** header with the name of the play to be executed, then one or more **TASK** headers are included. Each of these headers represent their associated *task* in the playbook, and it will be executed in all the managed hosts belonging to the group included in the playbook in the *hosts* parameter.

As each managed host executes the tasks, the host is displayed under the corresponding **TASK** header, along with the task state on that managed host, which can be set to **ok**, **fatal**, or **changed**. In the bottom of the playbook, at the **PLAY RECAP** section shows the number of tasks executed for each managed host as its output state.

The default output provided by the **ansible-playbook** command does not provide enough detail to troubleshoot issues that may appear on a managed host. The **ansible-playbook -v** command provides additional debugging information, with up to four total levels.

Verbosity configuration	
Option	**Description**
-v	The output data is displayed.
-vv	Both the output and input data are displayed.
-vvv	Includes information about connections to managed hosts.
-vvvv	Adds extra verbosity options to the connection plug-ins, including the users being used in the managed hosts to execute scripts, and what scripts have been executed.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Recommended Practices for playbook management

Although the previously discussed tools can help to identify and fix issues in playbooks, when developing those playbooks it is important to keep in mind some recommended practices that can help ease the process to troubleshoot issues. Here are some of the recommended practices for playbook development.

- Always name tasks, providing a description in the **name** of the task's purpose. This **name** is displayed when the playbook is executed.
- Include comments to add additional inline documentation about tasks.
- Make use of vertical whitespace effectively. YAML syntax is mostly based on spaces, so avoid the usage of tabs in order to avoid errors.
- Try to keep the playbook as simple as possible. Only use the features that you need.



References

log_path (Configuration file — Ansible Documentation)
http://docs.ansible.com/ansible/intro_configuration.html#log-path

debug — Print statements during execution — Ansible Documentation
http://docs.ansible.com/ansible/debug_module.html

Best Practices — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_best_practices.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Troubleshooting Playbooks

In this exercise, a playbook has errors that need to be corrected. The project structure from previous units is going to be reused. The playbook for this exercise is the **samba.yml** playbook that configures a Samba service on **servera.lab.example.com**.

Outcomes

You should be able to:

- Troubleshoot playbooks.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab troubleshoot-playbooks setup** script. It checks if Ansible is installed on **workstation**. It also creates the **/home/student/troubleshooting/** directory, and downloads on this directory the **inventory**, **samba.yml**, and **samba.conf.j2** files from **http://materials.example.com/troubleshooting/**.

```
[student@workstation ~]$ lab troubleshoot-playbooks setup
```

Steps

1. On workstation, change to the **/home/student/troubleshooting/** directory.

   ```
   [student@workstation ~]$ cd ~/troubleshooting/
   ```

2. Create a file named **ansible.cfg** in the current directory as follows, configuring the **log_path** parameter for Ansible to start logging to the **/home/student/troubleshooting/ansible.log** file, and the **inventory** parameter to use the **/home/student/troubleshooting/inventory** file deployed by the lab script. When you are finished, **ansible.cfg** should have the following contents:

   ```
   [defaults]
   log_path = /home/student/troubleshooting/ansible.log
   inventory = /home/student/troubleshooting/inventory
   ```

3. Run the playbook. This playbook sets up a Samba server if everything is correct. The run will fail due to missing double quotes on the **random_var** variable definition. The error message is very indicative of the issue with the run. Notice the variable **random_var** is assigned a value that contains a colon and is not quoted.

   ```
   [student@workstation troubleshooting]$ ansible-playbook samba.yml
   ERROR! Syntax Error while loading YAML.


   The error appears to have been in '/home/student/troubleshooting/samba.yml': line 8,
    column 30, but may
   be elsewhere in the file depending on the exact syntax problem.

   The offending line appears to be:

       install_state: installed
       random_var: This is colon: test
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
                              ^ here
```

4. Check that the error has been properly logged to the **/home/student/troubleshooting/ansible.log** file.

```
[student@workstation troubleshooting]$ tail ansible.log
The error appears to have been in '/home/student/troubleshooting/samba.yml': line 8,
 column 30, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

    install_state: installed
    random_var: This is colon: test
                              ^ here
```

5. Edit the playbook and correct the error, and add quotes to the entire value being assigned to **random_var**. The corrected edition of **samba.yml** should contain the following content:

```
... Output omitted ...
  vars:
    install_state: installed
    random_var: "This is colon: test"
... Output omitted ...
```

6. Run the playbook using the **--syntax-check** option. An error is issued due to the extra space in the indentation on the last task, **deliver samba config**.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml --syntax-check
ERROR! Syntax Error while loading YAML.


The error appears to have been in '/home/student/troubleshooting/samba.yml': line
 43, column 4, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:


   - name: deliver samba config
   ^ here
```

7. Edit the playbook and remove the extra space for all lines in that task. The corrected playbook content should look like the following:

```
... Output omitted ...
  - name: configure firewall for samba
    firewalld:
      state: enabled
      permanent: true
      immediate: true
      service: samba

  - name: deliver samba config
    template:
      src: templates/samba.conf.j2
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      dest: /etc/samba/smb.conf
      owner: root
      group: root
      mode: 0644
```

8. Run the playbook using the **--syntax-check** option. An error is issued due to the **install_state** variable being used as a parameter in the **install samba** task. It is not quoted.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml --syntax-check
ERROR! Syntax Error while loading YAML.


The error appears to have been in '/home/student/troubleshooting/samba.yml': line
 14, column 15, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

      name: samba
      state: {{ install_state }}
             ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```

9. Edit the playbook and correct the **install samba** task. The reference to the **install_state** variable should be in quotes. The resulting file content should look like the following:

```
... Output omitted ...
  tasks:
  - name: install samba
    yum:
      name: samba
      state: "{{ install_state }}"
... Output omitted ...
```

10. Run the playbook using the **--syntax-check** option. It should not show any additional syntax errors.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml --syntax-check
playbook: samba.yml
```

11. Run the playbook. An error, related to SSH, will be issued, recommending that you run **ansible-playbook** with the highest level of verbosity enabled, using the **-vvvv** option.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PLAY [Install a samba server] ***************************************************

TASK [setup] ********************************************************************
fatal: [servera.lab.exammple.com]: UNREACHABLE! => {"changed": false, "msg": "SSH
 encountered an unknown error during the connection. We recommend you re-run the
 command using -vvvv, which will enable SSH debugging output to help diagnose the
 issue", "unreachable": true}
 to retry, use: --limit @samba.retry

PLAY RECAP **********************************************************************
servera.lab.exammple.com       : ok=0    changed=0    unreachable=1    failed=0
```

12. Ensure the managed host **servera.lab.example.com** is running, using the **ping** command.

```
[student@workstation troubleshooting]$ ping -c3 servera.lab.example.com
PING servera.lab.example.com (172.25.250.10) 56(84) bytes of data.
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=1 ttl=64 time=0.247
 ms
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=2 ttl=64 time=0.329
 ms
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=3 ttl=64 time=0.320
 ms

--- servera.lab.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.247/0.298/0.329/0.041 ms
```

13. Ensure that you can connect to the managed host **servera.lab.example.com** as the *devops* user using SSH, and that the correct SSH keys are in place. Log out again when you have finished.

```
[student@workstation troubleshooting]$ ssh devops@servera.lab.example.com
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to the
 list of known hosts.
... Output omitted ...
[devops@servera ~]$ exit
Connection to servera.lab.example.com closed.
```

14. Rerun the playbook with *-vvvv* to get more information about the run. An error is issued because the **servera.lab.example.com** managed host is not reachable.

```
[student@workstation troubleshooting]$ ansible-playbook -vvvv samba.yml
Using /etc/ansible/ansible.cfg as config file
Loaded callback default of type stdout, v2.0
1 plays in samba.yml

PLAY [Install a samba server] ***************************************************

TASK [setup] ********************************************************************
<servera.lab.exammple.com> ESTABLISH SSH CONNECTION FOR USER: devops
... Output omitted ...
fatal: [servera.lab.exammple.com]: UNREACHABLE! => {...Could not resolve hostname
 servera.lab.exammple.com: Temporary failure in name resolution\r\n", "unreachable":
 true}
... Output omitted ...
PLAY RECAP **********************************************************************
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
servera.lab.exammple.com       : ok=0    changed=0    unreachable=1    failed=0
```

15. When using the highest level of verbosity with Ansible, the Ansible log file is a better option to check output than the console. Check the output from the previous command in the **/home/student/troubleshooting/ansible.log** file.

```
[student@workstation troubleshooting]$ tail ansible.log
... Output omitted ...
2016-05-03 17:36:19,609 p=5762 u=student |  1 plays in samba.yml
2016-05-03 17:36:19,613 p=5762 u=student |  PLAY [Install a samba server]
 ********************************************
2016-05-03 17:36:19,636 p=5762 u=student |  TASK [setup]
 *******************************************************************
2016-05-03 17:36:19,672 p=5762 u=student |  fatal: [servera.lab.exammple.com]:
 UNREACHABLE! => {...Could not resolve hostname servera.lab.exammple.com:...",
 "unreachable": true}
2016-05-03 17:36:19,673 p=5762 u=student |   to retry, use: --limit @samba.retry

2016-05-03 17:36:19,673 p=5762 u=student |  PLAY RECAP
 *******************************************************************
2016-05-03 17:36:19,673 p=5762 u=student |  servera.lab.exammple.com   : ok=0
 changed=0    unreachable=1    failed=0
```

16. Investigate the **inventory** file for errors. Notice the **[samba_servers]** group has misspelled **servera.lab.example.com**. Correct it and make the file look like the following:

```
... Output omitted ...
[samba_servers]
servera.lab.example.com
... Output omitted ...
```

17. Run the playbook again. The *debug install_state variable* task returns the message *The state for the samba service is installed*. This task makes use of the *debug* module, and displays the value of the *install_state* variable. An error is also shown in the *deliver samba config* task, since no **samba.j2** file is available in the working directory **/home/student/troubleshooting/**.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml

PLAY [Install a samba server] ************************************************
... Output omitted ...
TASK [debug install_state variable] ******************************************
ok: [servera.lab.example.com] => {
    "msg": "The state for the samba service is installed"
}
... Output omitted ...
TASK [deliver samba config] **************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failed":
 true, "msg": "IOError: [Errno 2] No such file or directory: u'/home/student/
troubleshooting/samba.j2'"}
... Output omitted ...
PLAY RECAP *******************************************************************
servera.lab.example.com    : ok=5    changed=3    unreachable=0    failed=1
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

18. Edit the playbook, and correct the **src** parameter in the *deliver samba config* task to be **samba.conf.j2**. When you are finished it should look like the following:

```
... Output omitted ...
  - name: deliver samba config
    template:
      src: samba.conf.j2
      dest: /etc/samba/smb.conf
      owner: root
... Output omitted ...
```

19. Run the playbook again. Execute the playbook using the **--step** option. It should run without errors.

```
[student@workstation troubleshooting]$ ansible-playbook samba.yml --step

PLAY [Install a samba server] **************************************************
Perform task: TASK: setup (y/n/c): y
... Output omitted ...
Perform task: TASK: install samba (y/n/c): y
... Output omitted ...
Perform task: TASK: install firewalld (y/n/c): y
... Output omitted ...
Perform task: TASK: debug install_state variable (y/n/c): y
... Output omitted ...
Perform task: TASK: start samba (y/n/c): y
... Output omitted ...
Perform task: TASK: start firewalld (y/n/c): y
... Output omitted ...
Perform task: TASK: configure firewall for samba (y/n/c): y
... Output omitted ...
Perform task: TASK: deliver samba config (y/n/c): y
... Output omitted ...
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=1    unreachable=0    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Troubleshooting Ansible Managed Hosts

Objectives

After completing this section, students should be able to:

- Use the **`ansible-playbook --check`** command
- Use modules to probe service status on the managed hosts
- Use ad hoc commands to check issues on the managed hosts

Check Mode as a Testing Tool

You can use the **`ansible-playbook --check`** command to run smoke tests on a playbook. This option executes the playbook without making any change to the managed hosts' configuration. If a module used within the playbook supports *check mode* the changes that would be made in the managed hosts are displayed. If that mode is not supported by a module those changes will not be displayed.

```
[student@demo ~]# ansible-playbook --check playbook.yml
```

The **`ansible-playbook --check`** command is used when all the tasks included in a playbook have to be executed in *check mode*, but if just some of those tasks need to be executed in *check mode*, the **`always_run`** option is a better solution. Each task can have a **`always_run`** clause associated. The value for this clause is *true* if the task has to be executed in check mode or *false* if it is not. The following task is executed in check mode.

```
tasks:
  - name: task in check mode
    shell: uname -a
    always_run: yes
```

The following task is not executed in check mode.

```
tasks:
  - name: task in check mode
    shell: uname -a
    always_run: false
```



Note

The **`ansible-playbook --check`** command may not work properly if tasks make use of conditionals.

Ansible also provides the **`--diff`** option. This option reports the changes done to the template files on managed hosts. If used with the **`--check`** option, those changes are displayed and not actually made.

```
[student@demo ~]# ansible-playbook --check --diff playbook.yml
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Modules for testing

Some modules can provide additional information about what the status of a managed host is. The following list includes some of the Ansible modules that can be used to test and debug issues on managed hosts.

- The **uri** module provides a way to check that a RESTful API is returning the required content.

```
tasks:
  - action: uri url=http://api.myapp.com return_content=yes
    register: apiresponse

  - fail: msg='version was not provided'
    when: "'version' not in apiresponse.content"
```

- The **script** module supports the execution of a script on a managed host, failing if the return code for that script is non-zero. The script must be on the control node and will be transferred (and executed) on the managed host.

```
tasks:
  - script: check_free_memory
```

- The **stat** module can check that files and directories not managed directly by Ansible are present. Check the usage of the **assert** module in the following example checking that a file exists in the managed host.

```
tasks:
  - stat: path=/var/run/app.lock
    register: lock

  - assert:
      that:
        - lock.stat.exists
```

Using ad hoc commands for testing

The following examples illustrate some of the checks that can be done on a managed host through the use of ad hoc commands. Those examples use modules such as *yum* in order to perform checks on the managed nodes. This example checks that the *httpd* package is currently installed in the **demohost** managed host.

```
[student@demo ~]# ansible demohost -u devops -b -m yum -a 'name=httpd state=present'
```

This example returns the currently available space on the disks configured in the **demohost** managed host.

```
[student@demo ~]# ansible demohost -a 'lsblk'
```

This example returns the currently available free memory on the **demohost** managed host.

```
[student@demo ~]# ansible demohost -a 'free -m'
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The correct level of testing

Ansible ensures that the configuration included in playbooks and performed by its modules is correctly done. It monitors all modules for reported failures, and stops the playbook immediately any failure is encountered. This ensures that any task performed before the failure has no errors. Because of this, there is no need to check if the result of a task managed by Ansible has been correctly applied on the managed hosts. It makes sense to add some health checks either to playbooks, or run those directly as ad hoc commands, when more direct troubleshooting is required.



References

Check Mode ("Dry Run") -- Ansible Documentation
http://docs.ansible.com/ansible/playbooks_checkmode.html

Testing Strategies -- Ansible Documentation
http://docs.ansible.com/ansible/test_strategies.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Troubleshooting Ansible Managed Hosts

In this exercise, the SMTP service is deployed in a managed host. The playbook for this exercise is the **mailrelay.yml** playbook that configures an SMTP service on **servera.lab.example.com**.

Outcomes

You should be able to:

- Troubleshoot managed hosts.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab troubleshoot-managedhosts setup** script. It checks if Ansible is installed on **workstation**. It also downloads the **inventory**, **mailrelay.yml**, and **postfix-relay-main.conf.j2** files from **http://materials.example.com/troubleshooting/** to the **/home/student/troubleshooting/** directory.

```
[student@workstation ~]$ lab troubleshoot-managedhosts setup
```

Steps

1. On workstation, change to the **/home/student/troubleshooting/** directory.

```
[student@workstation ~]$ cd ~/troubleshooting/
```

2. Run the **mailrelay.yml** playbook using check mode.

```
[student@workstation troubleshooting]$ ansible-playbook mailrelay.yml --check
PLAY [create mail relay servers] ***********************************************
...
TASK [check main.cf file] ******************************************************
ok: [servera.lab.example.com]

TASK [verify main.cf file exists] **********************************************
ok: [servera.lab.example.com]  => {
    "msg": "The main.cf file exists"
}
...
TASK [email notification of always_bcc config] *********************************
fatal: [servera.lab.example.com]: FAILED! => {"failed": true, "msg": "The
 conditional check 'bcc_state.stdout != 'always_bcc ='' failed. ...
...
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=6    changed=1    unreachable=0    failed=1
```

The *verify main.cf file exists* task uses the *stat* module. It confirmed that **main.cf** exists in **servera.lab.example.com**.

The *email notification of always_bcc config* task failed. It did not receive output from the *check for always_bcc* task, because the playbook was executed using check mode.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

3. Using an ad hoc command, check the header for the **/etc/postfix/main.cf** file.

```
[student@workstation troubleshooting]$ ansible servera.lab.example.com -u devops -b
 -a "head /etc/postfix/main.cf"
servera.lab.example.com | SUCCESS | rc=0 >>
# Global Postfix configuration file. This file lists only a subset
# of all parameters. For the syntax, and for a complete parameter
# list, see the postconf(5) manual page (command: "man 5 postconf").
#
# For common configuration examples, see BASIC_CONFIGURATION_README
# and STANDARD_CONFIGURATION_README. To find these documents, use
# the command "postconf html_directory readme_directory", or go to
# http://www.postfix.org/.
#
# For best results, change no more than 2-3 parameters at a time,
```

It is the original configuration file provided by the *postfix* package. It does not contain the header line stating that this file is managed by Ansible, because the playbook was executed using check mode.

4. Run the playbook again, but without specifying check mode. The error in the *email notification of always_bcc config* task should disappear.

```
[student@workstation troubleshooting]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ***********************************************
...
TASK [check for always_bcc] ****************************************************
changed: [servera.lab.example.com]

TASK [email notification of always_bcc config] *********************************
skipping: [servera.lab.example.com]

RUNNING HANDLER [restart postfix] **********************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=3    unreachable=0    failed=0
```

5. Using an ad hoc command, display the top of the **/etc/postfix/main.cf** file.

```
[student@workstation troubleshooting]$ ansible servera.lab.example.com -u devops -b
 -a "head /etc/postfix/main.cf"
servera.lab.example.com | SUCCESS | rc=0 >>
# Ansible managed: /home/student/troubleshooting/postfix-relay-main.conf.j2 modified
 on 2016-04-19 21:07:46 by student on workstation.lab.example.com
#
# Global Postfix configuration file. This file lists only a subset
# of all parameters. For the syntax, and for a complete parameter
# list, see the postconf(5) manual page (command: "man 5 postconf").
#
# For common configuration examples, see BASIC_CONFIGURATION_README
# and STANDARD_CONFIGURATION_README. To find these documents, use
# the command "postconf html_directory readme_directory", or go to
# http://www.postfix.org/.
```

Now it starts with a line that contains the string, "Ansible managed". This file was updated and is now managed by Ansible.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6. Add a task to enable the *smtp* service through the firewall.

```
[student@workstation troubleshooting]$ vim mailrelay.yml
...
    - name: postfix firewalld config
      firewalld:
        state: enabled
        permanent: true
        immediate: true
        service: smtp
...
```

7. Run the playbook. The *postfix firewalld config* should have been executed with no errors.

```
[student@workstation troubleshooting]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ***********************************************
...
TASK [postfix firewalld config] ************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=2    unreachable=0    failed=0
```

8. Using an ad hoc command, check that the *smtp* service is now configured in the firewall at **`servera.lab.example.com`**.

```
[student@workstation troubleshooting]$ ansible servera.lab.example.com -u devops -b
 -a "firewall-cmd --list-services"
servera.lab.example.com | SUCCESS | rc=0 >>
dhcpv6-client smtp ssh
```

9. Test the SMTP service, listening on port *TCP/25*, on **`servera.lab.example.com`** with **`telnet`**. Disconnect when you are finished.

```
[student@workstation troubleshooting]$ telnet servera.lab.example.com 25
Trying 172.25.250.10...
Connected to servera.lab.example.com.
Escape character is '^]'.
220 servera.lab.example.com ESMTP Postfix
quit
Connection closed by foreign host.
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Troubleshooting Ansible

In this lab, the control node and the managed hosts have errors that need to be corrected. The **secure-web.yml** playbook configures Apache with SSL on **serverb.lab.example.com**.

Outcomes
You should be able to:
- Troubleshoot playbooks.
- Troubleshoot managed hosts.

Before you begin
Log in to **workstation** as **student** using **student** as the password. Run the **lab troubleshoot-lab setup** command.

```
[student@workstation ~]$ lab troubleshoot-lab setup
```

This script checks if Ansible is installed on **workstation**, creates the **~student/troubleshooting-lab/** directory, and the **html** subdirectory on it. It also downloads from **http://materials.example.com/troubleshooting/** the **ansible.cfg**, **inventory-lab**, **secure-web.yml**, and **vhosts.conf** files to the **/home/student/troubleshooting-lab/** directory, and the **index.html** file to the **/home/student/troubleshooting-lab/html/** directory.

Steps
1. From the **~/troubleshooting-lab** directory, run the *secure-web.yml* playbook. It uses by default the **inventory-lab** file configured in the **inventory** parameter of the **ansible.cfg** file. This playbook sets up Apache with SSL. Solve any syntax issues in the variables definition.
2. Rerun the playbook and solve any issues related to the indentation in the *secure-web* playbook.
3. Rerun the playbook and solve any issues related to variable quotation in the *secure-web* playbook.
4. Rerun the playbook and solve any issues related to the inventory.
5. Rerun the playbook and solve any issues related to the user used to connect to the managed hosts.
6. Rerun the playbook and solve any issues related to the ability of the *devops* user to escalate the privilege to root.
7. Ensure that the Apache service playbook has been executed successfully in **serverb.lab.example.com**. Try to run the playbook first using check mode and check the state of the **httpd** service on **serverb.lab.example.com** using an ad hoc command. Run the playbook again and recheck the service.

Evaluation
From **workstation**, run the **lab troubleshoot-lab grade** script to confirm success on this exercise.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation troubleshooting-lab]$ lab troubleshoot-lab grade
```

Cleanup

From **workstation**, run the **lab troubleshoot-lab cleanup** script to confirm success on this exercise.

```
[student@workstation troubleshooting-lab]$ lab troubleshoot-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, the control node and the managed hosts have errors that need to be corrected. The **secure-web.yml** playbook configures Apache with SSL on **serverb.lab.example.com**.

Outcomes
You should be able to:
- Troubleshoot playbooks.
- Troubleshoot managed hosts.

Before you begin
Log in to **workstation** as **student** using **student** as the password. Run the **lab troubleshoot-lab setup** command.

```
[student@workstation ~]$ lab troubleshoot-lab setup
```

This script checks if Ansible is installed on **workstation**, creates the **~student/troubleshooting-lab/** directory, and the **html** subdirectory on it. It also downloads from **http://materials.example.com/troubleshooting/** the **ansible.cfg**, **inventory-lab**, **secure-web.yml**, and **vhosts.conf** files to the **/home/student/troubleshooting-lab/** directory, and the **index.html** file to the **/home/student/troubleshooting-lab/html/** directory.

Steps
1. From the **~/troubleshooting-lab** directory, run the *secure-web.yml* playbook. It uses by default the **inventory-lab** file configured in the **inventory** parameter of the **ansible.cfg** file. This playbook sets up Apache with SSL. Solve any syntax issues in the variables definition.

   1.1. On workstation, change to the **/home/student/troubleshooting-lab** project directory.

   ```
   [student@workstation ~]$ cd ~/troubleshooting-lab/
   ```

   1.2. Run the *secure-web* playbook, included in the **secure-web.yml** file. This playbook sets up Apache with SSL if everything is correct.

   ```
   [student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml
   ERROR! Syntax Error while loading YAML.
   ...
   The error appears to have been in '/home/student/Ansible-course/troubleshooting-
   lab/secure-web.yml': line 7, column 30, but may
   be elsewhere in the file depending on the exact syntax problem.

   The offending line appears to be:

     vars:
       random_var: This is colon: test
                                ^ here
   ```

   1.3. Correct the syntax issue in the variables definition by adding double quotes to the **This is colon: test** string. The resulting change should look like the following:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
... Output omitted ...
  vars:
    random_var: "This is colon: test"
... Output omitted ...
```

2. Rerun the playbook and solve any issues related to the indentation in the *secure-web* playbook.

   2.1. Rerun the *secure-web* playbook, included in the **secure-web.yml** file, using the **--syntax-check** option.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml --
syntax-check
ERROR! Syntax Error while loading YAML.


The error appears to have been in '/home/student/Ansible-course/troubleshooting-
lab/secure-web.yml': line 43, column 10, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:


         - name: start and enable web services
         ^ here
```

   2.2. Correct any syntax issues in the indentation. Remove the extra space at the beginning of the *start and enable web services* task elements. The resulting change should look like the following:

```
... Output omitted ...
        - name: start and enable web services
          service:
            name: httpd
            state: started
            enabled: yes
          tags:
            - services
... Output omitted ...
```

3. Rerun the playbook and solve any issues related to variable quotation in the *secure-web* playbook.

   3.1. Rerun the *secure-web* playbook, included in the **secure-web.yml** file, using the **--syntax-check** option.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml --
syntax-check
ERROR! Syntax Error while loading YAML.


The error appears to have been in '/home/student/Ansible-course/troubleshooting-
lab/secure-web.yml': line 13, column 20, but may
be elsewhere in the file depending on the exact syntax problem.
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
The offending line appears to be:

          yum:
            name: {{ item }}
                  ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```

3.2. Correct the *item* variable in the **install web server packages** task. Add double quotes to **{{ item }}**. The resulting change should look like the following:

```
... Output omitted ...
        - name: install web server packages
          yum:
            name: "{{ item }}"
            state: latest
          notify:
            - restart services
          tags:
            - packages
          with_items:
            - httpd
            - mod_ssl
            - crypto-utils
... Output omitted ...
```

3.3. Rerun the playbook using the **--syntax-check** option. It should not show any additional syntax errors.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml --
syntax-check
playbook: secure-web.yml
```

4. Rerun the playbook and solve any issues related to the inventory.

4.1. Rerun the *secure-web* playbook, included in the **secure-web.yml** file.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml
PLAY [create secure web service] ***********************************************

TASK [setup] *******************************************************************
fatal: [serverb.lab.example.com]: UNREACHABLE! => {"changed": false, "msg": "SSH
 encountered an unknown error during the connection. We recommend you re-run the
 command using -vvvv, which will enable SSH debugging output to help diagnose
 the issue", "unreachable": true}
 to retry, use: --limit @secure-web.retry

PLAY RECAP *********************************************************************
serverb.lab.example.com        : ok=0    changed=0    unreachable=1    failed=0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4.2. Rerun the *secure-web* playbook again with *-vvvv* parameter to add verbosity. Notice that the connection appears to be to **`tower.lab.example.com`**, instead of **`serverb.lab.example.com`**.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml -vvvv
TASK [setup] *******************************************************************
tower.lab.example.com ESTABLISH SSH CONNECTION FOR USER: students
tower.lab.example.com SSH: EXEC ssh -C -vvv -o ControlMaster=auto
 -o ControlPersist=60s -o Port=22 -o KbdInteractiveAuthentication=no
 -o PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey
 -o PasswordAuthentication=no -o User=students -o ConnectTimeout=10
 -o ControlPath=/home/student/.ansible/cp/ansible-ssh-%C -tt
 tower.lab.example.com
 '/bin/sh -c '"'"'( umask 22 && mkdir -p "` echo $HOME/.ansible/tmp/
 ansible- tmp-1460241127.16-3182613343880 `" && echo "` echo $HOME/.ansible/
 tmp/ansible-tmp-1460241127.16-3182613343880 `" )'"'"''
```

4.3. Correct the inventory. Delete the **`ansible_host`** host variable so the file looks like the following:

```
[webservers]
serverb.lab.example.com
```

5. Rerun the playbook and solve any issues related to the user used to connect to the managed hosts.

5.1. Rerun the *secure-web* playbook, included in the **`secure-web.yml`** file.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml -vvvv
...
TASK [setup] *******************************************************************
serverb.lab.example.com ESTABLISH SSH CONNECTION FOR USER: students
serverb.lab.example.com EXEC ssh -C -vvv -o ControlMaster=auto
 -o ControlPersist=60s -o Port=22 -o KbdInteractiveAuthentication=no
 -o PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey
 -o PasswordAuthentication=no -o User=students -o ConnectTimeout=10
 -o ControlPath=/home/student/.ansible/cp/ansible-ssh-%C -tt
 serverb.lab.example.com '/bin/sh -c '"'"'( umask 22 && mkdir -p "`
 echo $HOME/.ansible/tmp/ansible-tmp-1460241127.16-3182613343880 `" &&
 echo "` echo $HOME/.ansible/tmp/ansible-tmp-1460241127.16-3182613343880
 `" )'"'"''
... Output omitted ...
```

5.2. Edit the *secure-web* playbook to specify the **devops** user. The first lines of the playbook should look like the following:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  user: devops
... Output omitted ...
```

6. Rerun the playbook and solve any issues related to the ability of the *devops* user to escalate the privilege to root.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6.1. Rerun the *secure-web* playbook, included in the **secure-web.yml** file.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml -vvvv
... Output omitted ...
failed: [serverb.lab.example.com] => (item=[u'httpd', u'mod_ssl',
 u'crypto-utils']) => {"changed": true, "failed": true, "invocation":
 {"module_args": {"conf_file": null, "disable_gpg_check": false, "disablerepo":
 null, "enablerepo": null, "exclude": null, "install_repoquery": true,
 "list": null, "name": ["httpd", "mod_ssl", "crypto-utils"], "state": "latest",
 "update_cache": false}, "module_name": "yum"}, "item": ["httpd", "mod_ssl",
 "crypto-utils"], "msg": "You need to be root to perform this command.\n",
 "rc": 1, "results": ["Loaded plugins: langpacks, search-disabled-repos\n"]}
... Output omitted ...
```

6.2. Edit the playbook to include the **become** parameter. The resulting change should look like the following:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  user: devops
  become: true
... Output omitted ...
```

7. Ensure that the Apache service playbook has been executed successfully in **serverb.lab.example.com**. Try to run the playbook first using check mode and check the state of the **httpd** service on **serverb.lab.example.com** using an ad hoc command. Run the playbook again and recheck the service.

7.1. Rerun the **secure-web.yml** playbook in check mode. The **install web server packages** task reflects the changes that would be done. The **httpd_conf_syntax variable** task shows the current value for the **httpd_conf_syntax** variable. The **restart services** handler fails because the *httpd* package has not been installed and no change has been made to **serverb.lab.example.com**.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml --
check
PLAY [create secure web service] ***********************************************
...
TASK [install web server packages] *********************************************
changed: [serverb.lab.example.com] => (item=[u'httpd', u'mod_ssl', u'crypto-
utils'])
...
TASK [httpd_conf_syntax variable] **********************************************
ok: [serverb.lab.example.com] => {
    "msg": "The httpd_conf_syntax variable value is {u'msg': u'remote module
 does not support check mode', u'skipped': True, u'changed': False}"
}
...
RUNNING HANDLER [restart services] *********************************************
fatal: [serverb.lab.example.com]: FAILED! => {"changed": false, "failed": true,
 "msg": "systemd could not find the requested service \"'httpd'\": "}
...
PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=6    changed=4    unreachable=0    failed=1
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

7.2. Using an ad hoc command, check the state of the *httpd* service in **serverb.lab.example.com**. The *httpd* service is not installed in **serverb.lab.example.com**.

```
[student@workstation troubleshooting-lab]$ ansible all -u devops -b -a
 'systemctl status httpd'
serverb.lab.example.com | FAILED | rc=3 >>
● httpd.service
   Loaded: not-found (Reason: No such file or directory)
   Active: inactive (dead)
```

7.3. Rerun the *secure-web* playbook, included in the **secure-web.yml** file.

```
[student@workstation troubleshooting-lab]$ ansible-playbook secure-web.yml
PLAY [create secure web service] ***********************************************
...
TASK [install web server packages] *********************************************
changed: [serverb.lab.example.com] => (item=[u'httpd', u'mod_ssl', u'crypto-
utils'])
...
TASK [httpd_conf_syntax variable] **********************************************
ok: [serverb.lab.example.com] => {
    "msg": "The httpd_conf_syntax variable value is {u'changed': True, u'end':
 u'2016-05-03 19:43:37.612170', u'stdout': u'', u'cmd': [u'/sbin/httpd', u'-t'],
 'failed': False, u'delta': u'0:00:00.033463', u'stderr': u'Syntax OK', u'rc':
 0, 'stdout_lines': [], 'failed_when_result': False, u'start': u'2016-05-03
 19:43:37.578707', u'warnings': []}"
}
...
RUNNING HANDLER [restart services] *********************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=10   changed=7    unreachable=0    failed=0
```

7.4. Using an ad hoc command, check the state of the *httpd* service in **serverb.lab.example.com**. The *httpd* service should be now running in **serverb.lab.example.com**.

```
[student@workstation troubleshooting-lab]$ ansible all -u devops -b -a
 'systemctl status httpd'
serverb.lab.example.com | SUCCESS | rc=0 >>
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Tue 2016-05-03 19:43:39 CEST; 12s ago
... Output omitted ...
```

Evaluation

From **workstation**, run the **lab troubleshoot-lab grade** script to confirm success on this exercise.

```
[student@workstation troubleshooting-lab]$ lab troubleshoot-lab grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Cleanup

From **workstation**, run the **lab troubleshoot-lab cleanup** script to confirm success on this exercise.

```
[student@workstation troubleshooting-lab]$ lab troubleshoot-lab cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible provides built-in logging. This feature is not enabled by default.
- The **log_path** parameter in the **default** section of the **ansible.cfg** configuration file specifies the location of the log file to which all Ansible output is redirected.
- The **debug** module provides additional debugging information while running a playbook (for example, current value for a variable).
- The **-v** option of the **ansible-playbook** command provides several levels of output verbosity. This is useful for debugging Ansible tasks when running a playbook.
- The **--check** option enables Ansible modules with check mode support to display changes to be performed, instead of applying those changes to the managed hosts.
- Additional checks can be executed on the managed hosts using ad hoc commands.
- There is no need to double-check the configuration performed by Ansible as long as the playbook completes successfully.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 11

IMPLEMENTING ANSIBLE TOWER

Overview	
Goal	Implement Ansible Tower
Objectives	• Describe Ansible Tower architecture and features • Deploy Ansible Tower • Configure users and privileges in Ansible Tower • Manage hosts with Ansible Tower • Manage jobs in Ansible Tower
Sections	• Describing Ansible Tower (and Quiz) • Deploying Ansible Tower (and Guided Exercise) • Configuring Users in Ansible Tower (and Guided Exercise) • Managing Hosts with Ansible Tower (and Guided Exercise) • Managing Jobs in Ansible Tower (and Guided Exercise)
Lab	• Implementing Ansible Tower

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Describing Ansible Tower

Objectives

After completing this section, students should be able to:

- Describe Ansible Tower architecture and features.

Why Ansible Tower?

Ansible Tower is a web-based UI that provides an enterprise solution for IT automation. It has a user-friendly dashboard to manage deployments and monitor resources. Ansible Tower complements Ansible, adding automation, visual management, and monitoring capabilities.

Tower provides user access control to administrators. It facilitates the use of SSH credentials for administration, while at the same time preventing direct access to, or the transfer of, those credentials. The Ansible inventory can be graphically managed or synchronized with a wide variety of cloud sources, including Red Hat OpenStack Platform. Tower's dashboard is the user's entry point, providing a high-level overview of the hosts managed by Tower. This includes recent activity on those hosts, and problems that may have arisen.

These automation capabilities allow the implementation of continuous delivery and configuration management, while integrating their management in one tool. Through Tower's System Tracking it is possible to audit and manage the history of the hosts managed by Tower, and to compare the state of different hosts. Tower's Remote Command Execution allows atomic operations to be run against hosts, making use of the Tower's role-based access control engine to control who can execute operations, and logging any action taken.

Ansible Tower includes the Tower Installation Wizard, a deployment tool that eases the deployment of Ansible Tower. After selecting Tower configuration to be deployed using the wizard, a playbook is used to install Tower.

Ansible Tower architecture

Ansible Tower is based on a Django web application that requires that Ansible be installed. It also relies on a database back-end that uses PostgreSQL. The default installation installs all components required by Ansible Tower on a single machine.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

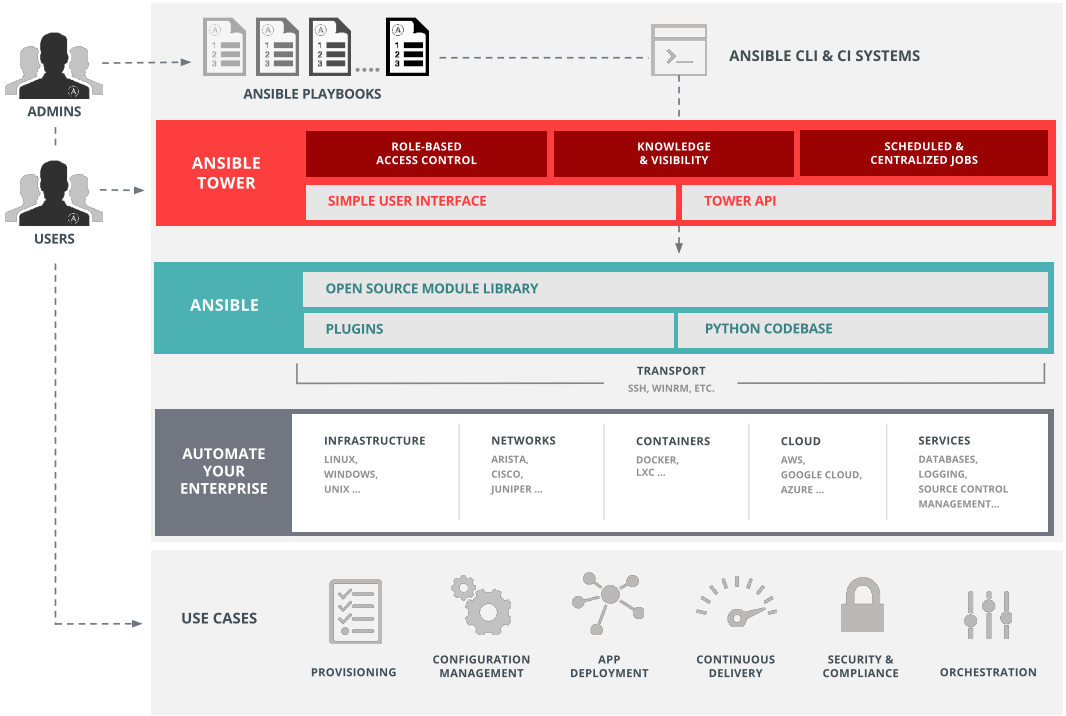


Figure 11.1: Ansible Tower architecture

Ansible Tower can be installed in three scenarios depending on the scale of deployment and number of hosts to be managed:

- Single machine installation with an internal database: This is a deployment where the web interface, REST API back end, and database are all running on a single machine. It uses **PostgreSQL** provided by the operating system and configures Tower to use it as its database.

- Single machine with an external database: This is a deployment with the web interface and REST API on one machine and an external **PostgreSQL** database on a second machine. The replication and failover of this database is not configured by Ansible Tower, but is managed separately.

- Active/Passive redundancy using multiple machines: In this deployment mode, the web interface and REST API are active on one *primary node* at a time, but one or more passive *secondary nodes* are on standby and can take over if the primary node fails. The external **PostgreSQL** or **MongoDB** database must run on a machine that is not a primary or secondary node.

Ansible Tower features

Tower provides features such as role-based access control, push-button deployment, centralized logging, and a RESTful API. LDAP or Active Directory can also be used to manage users with Ansible Tower. Tower does not support all the functionality provided by the Ansible CLI, including playbook development support, where Ansible's CLI is the best option. The following list includes some of the most relevant features offered by Ansible Tower:

- Support for automation of playbook runs, inventory synchronization and project source control, so those can be scheduled to be executed at a given time, being able to monitor the state of a managed host.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- A RESTful API which supports all the functionality provided through the Tower's dashboard. A CLI supporting this RESTful API is also provided by Tower.
- Improved playbook execution by providing a user-friendly way of supplying playbook variables and selecting credentials, monitoring the playbook while it is being executed, and visualizing the results with many different views, including a graphical inventory with the history of every managed host.
- Role-based access control, allowing different teams or users to manage Tower resources based on their requirements.



References

Ansible Tower Administration Guide
http://docs.ansible.com/ansible-tower/latest/html/administration/index.html

Variables — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_variables.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Quiz: Describing Ansible Tower

Choose the correct answer to the following questions:

1. Which of the following features are provided by Ansible Tower? (Choose three.)

   a. Role-based access control
   b. Playbook creator wizard
   c. Centralized logging
   d. RESTful API
   e. Ansible Tower is only a web based UI.

2. Which of the following enhancements are ***additions*** to Ansible provided by Ansible Tower? (Choose three.)

   a. Playbook development
   b. Remote execution
   c. Automation
   d. Monitoring
   e. Version Control
   f. Visual management

3. Which services are included in Ansible Tower? (Choose three.)

   a. RESTful API
   b. Python API
   c. Tower's CLI
   d. Ansible CLI
   e. Tower deployment tool
   f. Dashboard

4. Which of the following architectures are supported by the Tower Installation Wizard? (Choose three.)

   a. All-in-one (single machine)
   b. Single machine with an external database
   c. Active/active redundancy multi-machine with an external database
   d. Active/passive redundancy multi-machine with an external database

5. Which of the following use cases are supported by Ansible Tower? (Choose two.)

   a. Data analytics
   b. Playbook development
   c. Continuous delivery
   d. Bare-metal OS deployment
   e. Configuration management

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

Choose the correct answer to the following questions:

1. Which of the following features are provided by Ansible Tower? (Choose three.)

   a. Role-based access control
   b. Playbook creator wizard
   c. Centralized logging
   d. RESTful API
   e. Ansible Tower is only a web based UI.

2. Which of the following enhancements are ***additions*** to Ansible provided by Ansible Tower? (Choose three.)

   a. Playbook development
   b. Remote execution
   c. Automation
   d. Monitoring
   e. Version Control
   f. Visual management

3. Which services are included in Ansible Tower? (Choose three.)

   a. RESTful API
   b. Python API
   c. Tower's CLI
   d. Ansible CLI
   e. Tower deployment tool
   f. Dashboard

4. Which of the following architectures are supported by the Tower Installation Wizard? (Choose three.)

   a. All-in-one (single machine)
   b. Single machine with an external database
   c. Active/active redundancy multi-machine with an external database
   d. Active/passive redundancy multi-machine with an external database

5. Which of the following use cases are supported by Ansible Tower? (Choose two.)

   a. Data analytics
   b. Playbook development
   c. Continuous delivery
   d. Bare-metal OS deployment
   e. Configuration management

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Deploying Ansible Tower

Objectives

After completing this section, students should be able to:

- Deploy Ansible Tower

Deploying Ansible Tower

When installing Ansible Tower, the control node should have the latest version of Ansible installed. Tower is a browser-based application and the installation process installs several dependencies such as **PostgreSQL**, **Django**, **Apache HTTPD**, and others. It is required that Ansible Tower is installed on a stand-alone server and should not be co-located with any other applications.

Ansible Tower requirements

Ansible Tower is supported on Linux based systems on the x86-64 architecture, including Red Hat Enterprise Linux 6, Red Hat Enterprise Linux 7, CentOS 6, CentOS 7, Ubuntu 12.04 LTS, and Ubuntu 14.04 LTS. This course will focus on the installation procedure for Red Hat Enterprise Linux 7 or CentOS 7. A minimum of 2 GB RAM and 20 GB of hard disk space is required for the installation.

Bundled installation setup

Use the bundled installation program if you do not have access to online Extra Packages for Enterprise Linux (EPEL) repositories for the dependencies. The installer will still require the packages found in standard Red Hat Enterprise Linux or CentOS distributions. Note that the bundled installer only supports installation on Red Hat Enterprise Linux or CentOS.

Administrators can get a free trial of Ansible Tower by visiting the following URL: **`https://www.ansible.com/tower-trial`**. Once contact information has been provided and the end user license agreement has been read and agreed to, the bundled installer can be downloaded for installation and product evaluation.

Ansible Tower installation

The first step to install Ansible Tower is to unpack the setup bundle and run a configuration script that will set up a playbook and inventory file that will be used to install Tower. You can run both the configuration step and the playbook from a remote control node or locally.

Unpacking the tarball

First the setup bundle which comes in archive as a compressed **gzip** file needs to extracted.

```
[student@demo ~]$ tar xvf ansible-tower-setup-bundle-1.el7.tar.gz
```

Tower machine configuration

The extracted archive contains a **`configure`** script which is used to setup the playbook and inventory file depending on the parameters passed when the script is executed. The same playbook can be used several times if the same set up needs to be replicated to multiple machines, there is no need to run the **`configure`** script again.

```
[student@demo ansible-tower-setup-bundle-2.4.5-1.el7]$ ./configure
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The **configure** script takes the following options:

Option	Description
-l, **--local**	Install Ansible Tower on the local machine with an internal PostgreSQL database.
--no-secondary-prompt	Skip prompts regarding secondary Tower nodes to be added.
-A, **--no-autogenerate**	Disable auto-generation of passwords for PostgreSQL, and instead prompts the user for passwords.
-o *FILE*, **--option-file=*FILE***	Use *FILE* as the source of answers for configuration.

The configuration playbook is created under the setup bundle directory with the name **tower_setup_conf.yml**. This file contains needed Tower passwords, database connection information, and SSH connection information.

The inventory file lists the machines that the setup playbook will use. The machines are grouped as **[primary]** and **[secondary]** groups of managed hosts.

Running the Tower setup
After the **tower_setup_conf.yml** playbook and inventory file have been created, the **setup.sh** script in the unpacked setup bundle is run to execute the playbook and installs Ansible Tower.

```
[student@demo ansible-tower-setup-bundle-2.4.5-1.el7]$ ./setup.sh
```

The **setup.sh** script takes the following options:

Option	Description
-c *FILE*	Specify the file that stores the Tower configuration. The default file is **tower_setup_conf.yml** in the setup bundle directory.
-i *FILE*	Specify the file that is to be used for the host inventory. The default file is **inventory** in the setup bundle directory.
-p	Require Ansible to prompt for SSH passwords when connecting to remote machines.
-s	Require Ansible to prompt for **sudo** passwords on remote machines when installing Tower.
-u	Require Ansible to prompt for **su** passwords on remote machines when installing Tower.
-e	Set additional Ansible variables for Tower to use during installation.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Option	Description
-b	Perform a database backup instead of installing Tower.
-r *BACKUP_FILE*	Perform a database restore from the file specified, instead of installing Tower.

Ansible Tower Dashboard

When logged in to Ansible Tower using the web UI, the administrator can view a graph that shows all the recent job activity, the number of managed hosts, and quick pointers to lists of hosts with problems. The dashboard also displays real time data about the execution of tasks completed in playbooks.

In the following examples, the text will assume that **`http://demo.lab.example.com/`** is a URL for an Ansible Tower web UI.

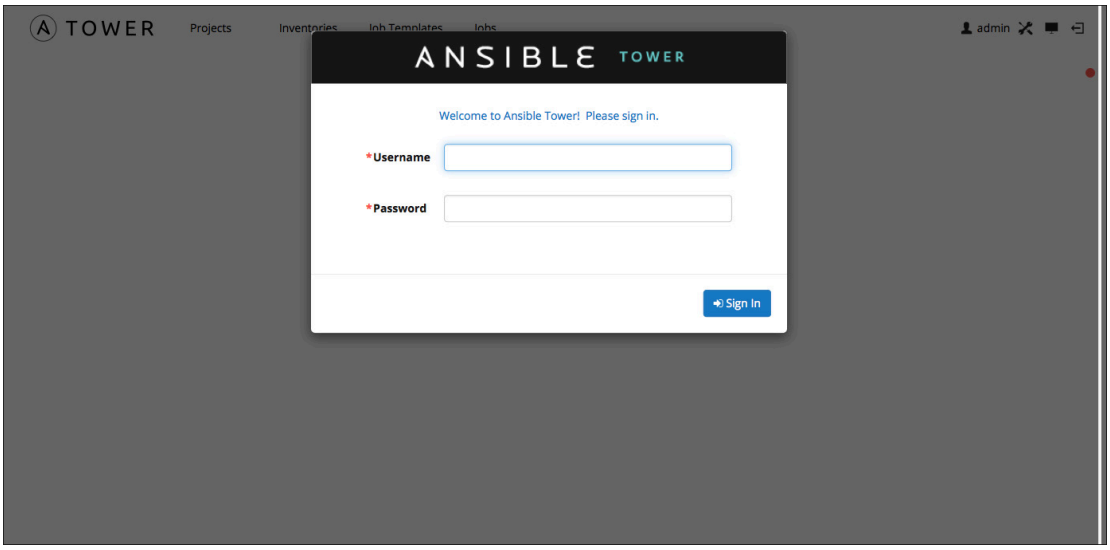


Figure 11.2: Ansible Tower login screen

Tower also configures a minimal **Munin** instance to monitor itself, allowing graphs to show things like memory consumption and CPU usage, as well as the number of queued and active jobs. The **Munin** dashboard can be accessed using **`http://demo.lab.example.com/munin`** and requires the **`admin`** user name and password provided during the setup.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

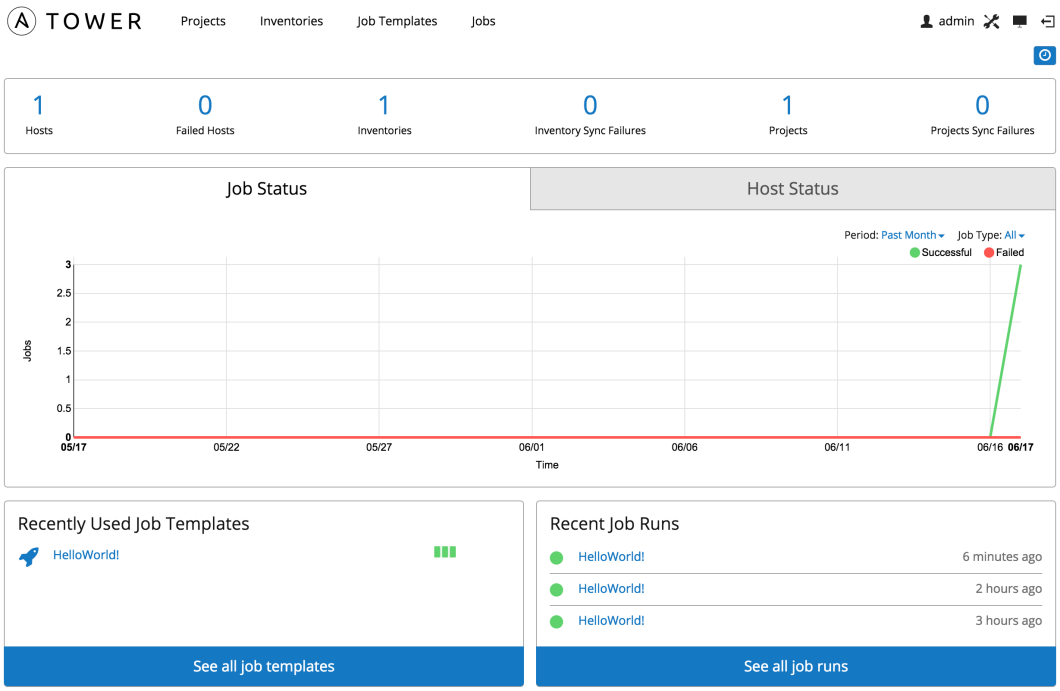


Figure 11.3: Ansible Tower dashboard

The administrator password for Ansible Tower can be changed after the installation process is complete using the command **`tower-manage`**. The command needs to run as **`root`** on the Tower server:

```
[root@tower ~]# tower-manage changepassword admin
```

Ansible Tower SSL Certificate

Ansible Tower uses a self signed certificate for HTTPS communication, **`/etc/tower/awx.cert`** is the certificate file and **`/etc/tower.awx.key`** is the key file. These files can be replaced with the company owned CA certificate as needed, but the file names must be the same.

Ansible Tower REST APIs

Ansible Tower exposes REST APIs to provide access to its resources such as projects, jobs, inventories, credentials and so on using a URI path. These APIs can accessed via a browser using the link **`http://demo.lab.example.com/api/`**.

Clicking on various links in the **REST API** page allows you to explore related resources. The following screenshot provides a simple view that reports very basic information about the Tower instance. The other information such as users, projects, credentials and so on require admin credentials to access it.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

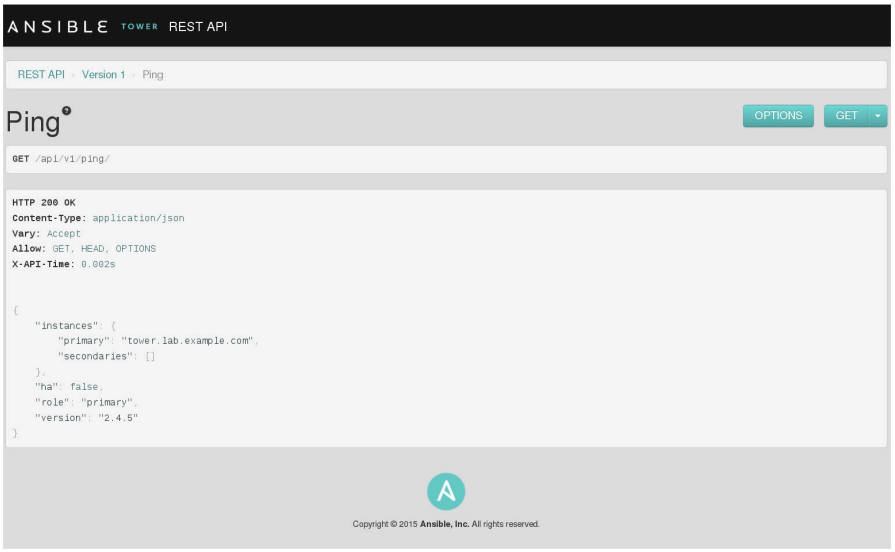


Figure 11.4: Ansible REST API interface

To access the information using the command line:

```
[student@demo ~]$ curl -s http://demo.lab.example.com/api/v1/ping/ | json_reformat
{
    "instances": {
        "primary": "demo.lab.example.com", "secondaries": [

        ]
    },
    "ha": false,
    "role": "primary",
    "version": "2.4.5"
}
```



References

The Tower Installation Wizard – Ansible Tower Installation and Reference Guide
http://docs.ansible.com/ansible-tower/latest/html/installandreference/tower_install_wizard.html

Tower Installation Scenarios – Ansible Tower Installation and Reference Guide
http://docs.ansible.com/ansible-tower/latest/html/installandreference/install_scenarios.html

Introduction to the Tower API – Ansible Tower API Guide
https://docs.ansible.com/ansible-tower/latest/html/towerapi/intro.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Deploying Ansible Tower

In this exercise, you will deploy Ansible Tower with **tower.lab.example.com** as the primary server and the deployment mode would be single machine with an internal database. The installation will be triggered from the control host **workstation.lab.example.com**.

Outcomes
You should be able to:
- Deploy Ansible Tower.
- Log in and explore the Ansible Tower web interface.

Before you begin
Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-tower-install setup** script. The setup script checks that Ansible is installed on **workstation**, and creates a directory structure for the lab environment. The script also copies the Ansible Tower setup bundle and license file in the lab's environment directory.

```
[student@workstation ~]$ lab ansible-tower-install setup
```

Steps
1. From **workstation**, as the **student** user, change to the directory **~/ansible-tower-install**.

```
[student@workstation ~]$ cd ~/ansible-tower-install
[student@workstation ansible-tower-install]$
```

2. In the directory **~/ansible-tower-install** locate the Ansible Tower setup bundle and license text file.

```
[student@workstation ansible-tower-install]$ ls -l
total 180464
-rw-r--r--. 1 student student       436 Apr 22 08:32 Ansible-Tower-license.txt
-rw-r--r--. 1 student student 184789299 Apr 22 08:32 ansible-tower-setup-
bundle-1.el7.tar.gz
```

3. Unpack the setup bundle **ansible-tower-setup-bundle-1.el7.tar.gz** present under the directory **~/ansible-tower-install**.

```
[student@workstation ansible-tower-install]$ tar zxvf ansible-tower-setup-
bundle-1.el7.tar.gz
```

4. From the directory **ansible-tower-setup-bundle-2.4.5-1.el7** created after unpacking the Ansible Tower installation tarball, run the **configure** command. The **configure** command creates a file called **tower_setup_conf.yml** that contains the configuration for Ansible Tower. Provide the following parameters to the **configure** command when prompted:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Parameter	Value
hostname	tower.lab.example.com
database	internal
Ansible Tower admin password	redhat
Munin password	redhat
SSH user	devops
SSH keys	yes
SSH key location	/home/student/.ssh/lab_rsa

4.1. Change to the **ansible-tower-setup-bundle-2.4.5-1.el7** directory created after unpacking the Tower installation tarball.

```
[student@workstation ansible-tower-install]$ cd ansible-tower-setup-
bundle-2.4.5-1.el7/
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$
```

4.2. Execute the **configure** command under **ansible-tower-setup-bundle-2.4.5-1.el7** directory, to create the **tower_setup_conf.yml** configuration playbook.

```
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$ ./configure
```

4.3. As **tower.lab.example.com** is the primary server for this Ansible Tower deployment, use **tower.lab.example.com** as the hostname to configure Ansible Tower.

```
-----------------------------------------
Welcome to the Ansible Tower Install Wizard
-----------------------------------------

This wizard will guide you through the setup process.

PRIMARY TOWER MACHINE
Tower can be installed (or upgraded) on this machine, or onto a remote machine
that is reachable by SSH.

...output-omitted...

Enter the hostname or IP to configure Ansible Tower
(default: localhost): tower.lab.example.com
```

4.4. Provide the database to be used for Ansible Tower. Press **i** to use an internal database on the tower host.

```
...output-omitted...
DATABASE
Tower can use an internal database installed on the Tower machine, or an
external PostgreSQL database. An external database could be a hosted database,
such as Amazon's RDS.

...output-omitted...
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
Will this installation use an (i)nternal or (e)xternal database? i
```

4.5. Provide the Tower administrator password as **redhat**. When prompted, provide the password as **redhat**, to be used by Munin, a built-in monitoring system to monitor the Tower sever.

```
...output-omitted...
PASSWORDS
For security reasons, since this is a new install, you must specify the
following application passwords.

Enter the desired Ansible Tower admin user password: redhat
Enter the desired Munin password: redhat
```

4.6. Enter the user information for connecting to **tower.lab.example.com** using SSH.

In this deployment we would be using **devops** user to connect to **tower.lab.example.com** and use **sudo** as the method for privilege escalation and no password is required for **sudo** access.

```
...output-omitted...
CONNECTION INFORMATION
Enter the SSH user to connect with (default: root): devops
Root access is required to install Tower.
Will you use (1) sudo or (2) su? 1
Will sudo require a password (y/N)? N
```

4.7. For key based SSH authentication press **Y** when prompted and specify the path of the private SSH key as **/home/student/.ssh/lab_rsa**.

```
...output-omitted...
Will you be using SSH keys (Y/n)? Y
Please specify the path to the SSH private key: /home/student/.ssh/lab_rsa
```

4.8. Press **y** to confirm the settings provided are correct.

```
...output-omitted...
REVIEW
You selected the following options:

The primary Tower machine is: tower.lab.example.com
Tower will operate on an INTERNAL database.
Using SSH user: devops

Are these settings correct (y/n)? y
```

4.9. The inputs provided are saved in a playbook file named **tower_step_conf.yml**. This file can be used later on to bypass the configuration step if you plan to install Tower using the same information and passwords.

```
...output-omitted...
FINISHED!
You have completed the setup wizard. You may execute the installation of
Ansible Tower by issuing the following command:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
# Add your SSH key to SSH agent.
# You may be asked to enter your SSH unlock key password to do this.
ssh-agent bash
ssh-add /home/student/.ssh/lab_rsa
./setup.sh
```

5. Due to a known bug in Ansible Tower version *2.4.5* with the Tower configuration wizard, the playbook created by **configure** command fails when executed.To work around this issue, edit the **inventory** file found in the **ansible-tower-setup-bundle-2.4.5-1.el7** directory and replace **ansible_sudo=True** with **ansible_become=True**. The inventory file should appear as follows:

```
[primary]
tower.lab.example.com

[all:vars]
ansible_ssh_user=devops
ansible_become=True
```

6. Run the setup playbook with **sudo** using **setup.sh** in the **~/ansible-tower-install/ansible-tower-setup-bundle-2.4.5-1.el7** directory. This script uses the **tower_setup_conf.yml** and **inventory** files written by the installation configuration wizard.

```
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$ sudo ./setup.sh
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [tower.lab.example.com]
...output-omitted...
PLAY RECAP *********************************************************************
tower.lab.example.com      : ok=118  changed=67   unreachable=0    failed=0

The setup process completed successfully.
[warn] /var/log/tower does not exist.
Setup log saved to setup.log.
```

7. From workstation, use a browser to open the Ansible Tower web UI using the web link **https://tower.lab.example.com**. Add security exception as the certificate used is a self-signed certificate.

   Log in to Ansible Tower as the user **admin** using the password **redhat**.

8. In the **Update License** page, copy and paste the license from the license text file **~/ansible-tower-install/Ansible-Tower-license.txt** into the **License File** window. Accept the **End User License Agreement** by selecting the checkbox and press **Submit**.

   When you see the **License Accepted** window, click **OK**.

9. Open the **Setup** page by clicking on the wrench and screwdriver icon in the top right corner of the portal. Click the View Your License link to open the License page. Note the values mentioned in **Tower Version** and **Time Remaining** text boxes.

10. The license provided for Ansible Tower for this lab can be used to manage **10** managed hosts. Verify this by opening the **Managed Hosts** page.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Exercise Summary

Ansible Tower is now installed on **`tower.lab.example.com`** using the control host **`workstation.lab.example.com`**, and can be used to manage up to **`10`** hosts, but no users or hosts are configured yet.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Configuring Users in Ansible Tower

Objectives

After completing this section, students should be able to:

- Create users in Ansible Tower
- Define an Ansible inventory

Ansible Tower user privileges

Administrators can manage users in Ansible Tower with various privileges. There are three types of users in Ansible:

- *Normal user*: Normal users have read and write access to the inventory and projects they have been assigned.
- *Organization Administrator*: Organization administrators have all the rights normal users have, as well as read and write access over the entire organization, and all of its inventories and projects. Organization administrators do not have any access to the content that belongs to other organizations. Organization administrators can create and manage users within their own organization.
- *Superuser*: Superusers have read and write permissions over the entire Tower installation. Superusers are typically systems administrators who are responsible for managing Ansible Tower, and delegating responsibilities to organization administrators.



Note

Upon installation, the initial Ansible Tower user that is created inherits the **`superuser`** privileges.

Managing users in Ansible Tower

From the web interface, users with the **`superuser`** and **`organization administrators`** privileges can manage users by defining an user name, an email address, a login as well as a password. The following screenshot shows the view for managing users:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

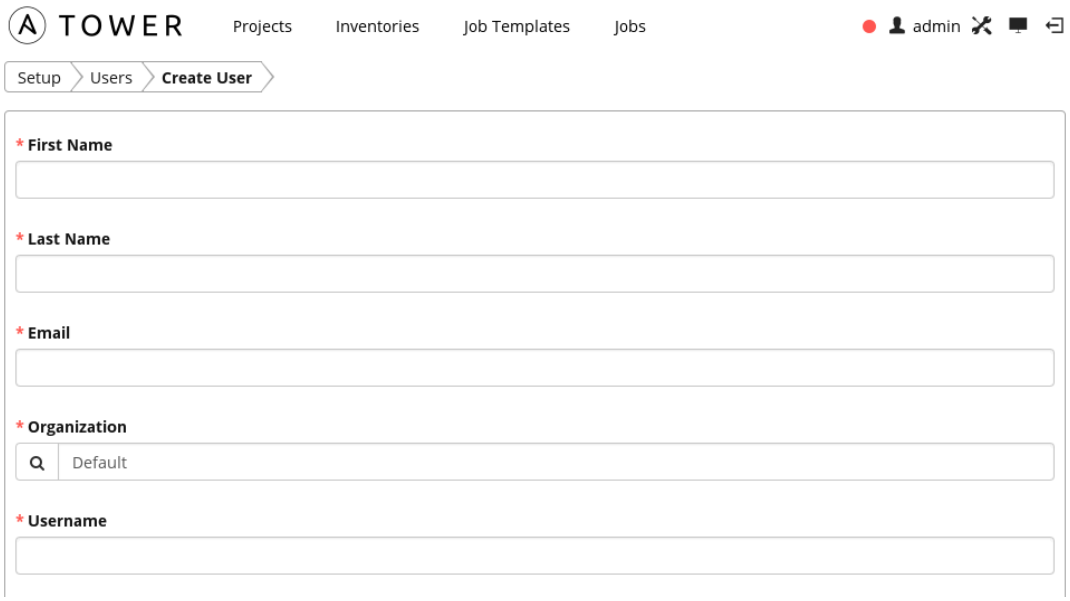


Figure 11.5: Ansible Tower user view

Upon creation, further user management is available for administrators. Such management includes editing user:

- Credentials
- Permissions
- Privileges
- Organizations
- Teams

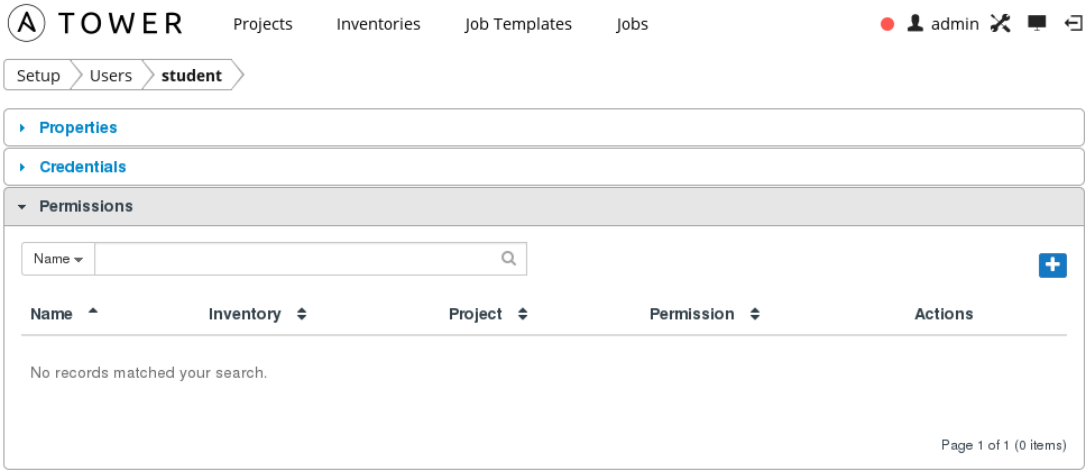


Figure 11.6: Ansible Tower editing user view

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

User credentials

Credentials are used by Ansible Tower for various authentication tasks, such as launching jobs against managed hosts, synchronizing with inventories, as well as importing project content from version control systems. The following screenshot shows how user credentials can be managed:

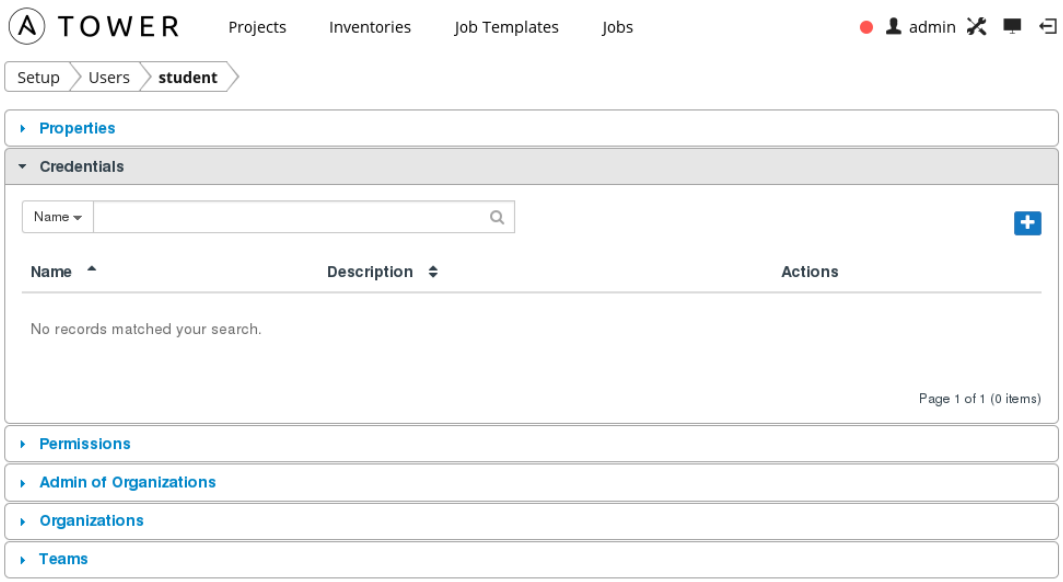


Figure 11.7: Ansible Tower user credentials

User permissions

Permissions define the set of privileges assigned to users or user teams. They determine users' ability to read, modify, and administer projects, inventories, and job templates.

There are two permission types available which can be assigned to users and teams, each with their own set of permissions:

- **Inventory**: This permission type grants permission to users to manage inventories, groups, and hosts; it implements the following levels:
  - **`read`**: Allows viewing of groups and hosts within a specified inventory.
  - **`write`**: Allows creation, modification, and removal of groups and hosts within a specified inventory. This level of privilege does not grant permission to modify the inventory settings, but does grant read permission on those settings.
  - **`admin`**: Allows modification of the settings for the specified inventory. This permission level grants both read and write permissions.
  - **`execute commands`**: Allows the user to execute commands on the inventory.
- **Job Template**: This permission type grants permission to launch jobs from the specified project against the specified inventory; it implements the following levels:
  - **`create`**: Allows users or teams to create job templates. This role requires both **`run`** and **`check`** permissions.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- **run**: Allows users or teams to run jobs. Giving a user or a team this level also grants the **check** permissions.
- **check**: Allows users or teams to launch jobs in a dry-run mode. Items that would be changed are reported, but the changes are not actually made.

The following screenshot shows how user permissions can be managed by Tower administrators:

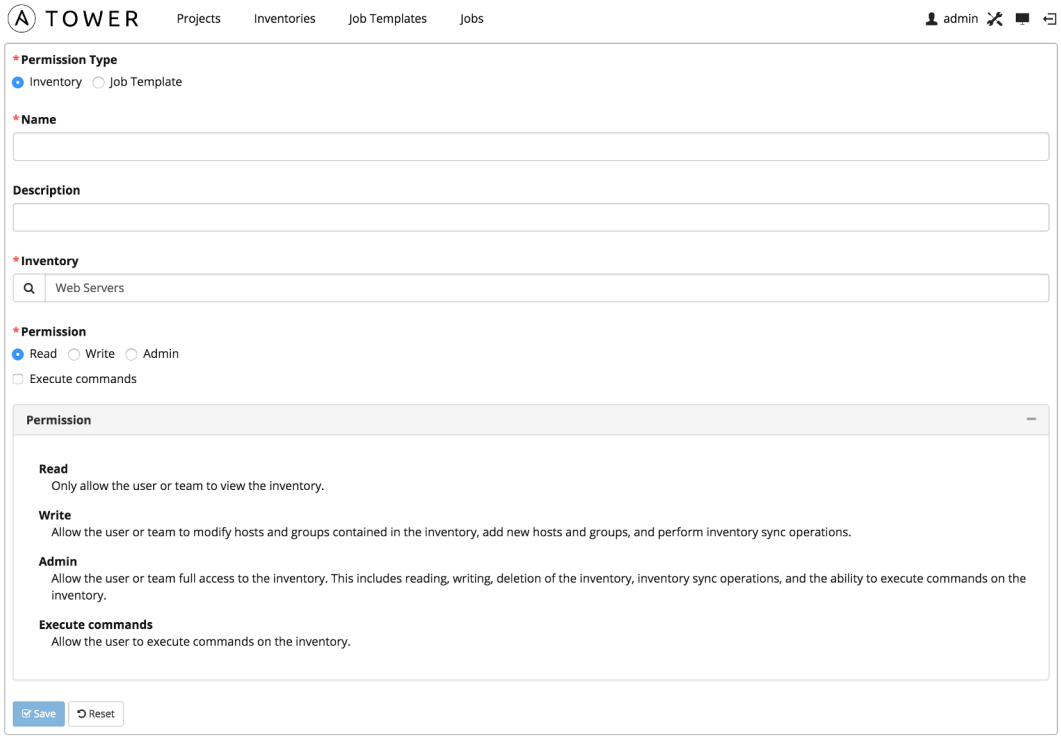


Figure 11.8: Ansible Tower user permissions

Organization managers

Users can be configured as organization managers, giving them the ability to manage organization inventories and projects. The following screenshot shows how to define organizations for users:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

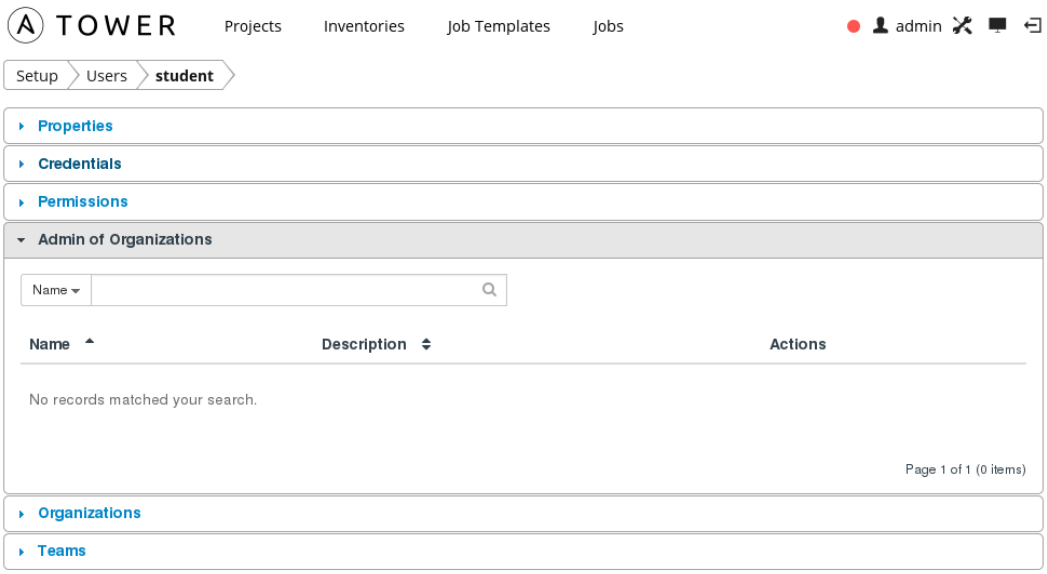


Figure 11.9: Ansible Tower users organization management

Organizations

Ansible Tower ships with the **`default`** organization, but administrators can define their own organizations for logical grouping of users. The following screenshot shows how to attach a user to an organization:

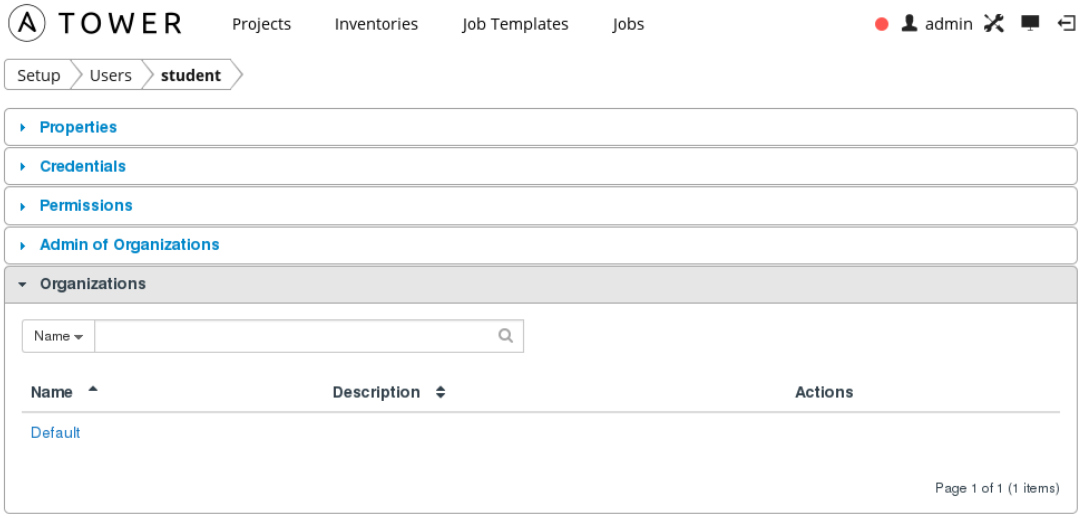


Figure 11.10: Ansible Tower users organization

User teams

Teams are a subdivision of an organization and have associated users, projects, credentials, as well as permissions. Teams provide a means to implement role-based access control schemes, and delegate responsibilities across organizations. Teams allow managers to easily assign the same permissions to a set of users, instead of manually setting permissions for each user individually. The following screenshot shows how to attach a user to a team:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

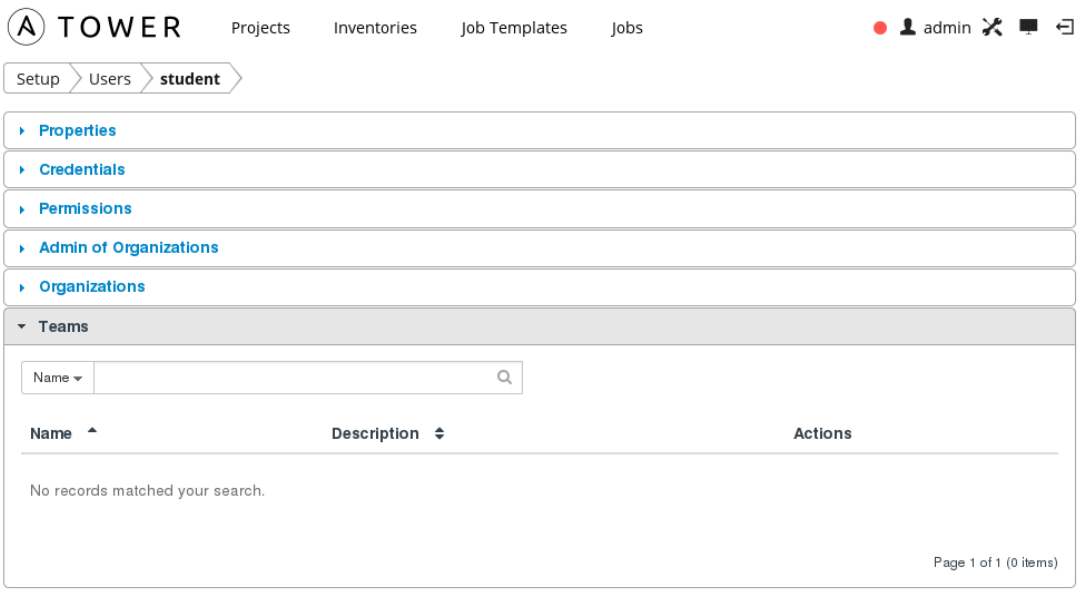


Figure 11.11: Ansible Tower users team



References

Users – Ansible Tower User Guide
http://docs.ansible.com/ansible-tower/2.2.0/html/userguide/users.html#users

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Configuring Users in Ansible Tower

In this exercise, you will define an Ansible inventory and create a user in Ansible Tower.

Outcomes
You should be able to:
- Configure users and privileges in Ansible Tower.

Before you begin
From **`workstation.lab.example.com`**, ensure the Ansible Tower interface is reachable at **`https://tower.lab.example.com`**. Make sure you can log in as the administrative user, using **`admin`** as the user name and **`redhat`** as the password.

Steps
1. From **`workstation.lab.example.com`**, open a browser and connect to the Ansible Tower page, located at **`https://tower.lab.example.com`**. Log in to the Tower interface using **`admin`** as the username, **`redhat`** as the password.

2. To manage the permissions for a specific user, an inventory must be first defined in Ansible Tower. To do so, click the **`Inventories`** link at the top of the page, then click **plus** icon on the upper right to open the **`Create Inventory`** assistant.

   From the next screen, create an inventory with the following parameters:

Field	Value
Name	**`Classroom`**
Description	**`Classroom environment`**
Organization	**`Default`**

   When you have filled in all the details, click **Save**.

3. Create a new user by opening the **Setup** page and clicking the wrench and screwdriver icon in the top right.

   3.1. Click the **`Users`** link in the middle of the page to manage users.

   3.2. Click the **plus** icon in the middle right to add a new user.

   3.3. On the next screen, fill in the details as follows:

Field	Value
First Name	**`Student`**
Last Name	**`Training`**
Email	**`student@lab.example.com`**
Organization	**`Default`**
Username	**`student`**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Field	Value
Password/ Confirm Password	**redhat123**
Superuser	cleared

Click **Save** to create the new user.

4. After creating the user, permissions for the user can be managed. From the next screen, the **Permissions** tab should be active. Click the **plus** icon located in the right, then fill in the details as follows:

Field	Value
Permission Type	**Inventory**
Name	**Classroom Privileges**
Description	**Privileges for the classroom environment**

For the inventory to attach to the user, click the magnifier icon under the **Inventory** field. From the popup that appears, select the **Classroom** inventory and click **Select**.

Set the permission type to **Administrate Inventory**, then click **Save** to confirm the changes.

5. To confirm the creation of the user, log out of the administrative account by clicking the **Sign Out** button located on the top right of the screen.

After logging out, log in using **student** as the **Username** and **redhat123** as the **Password**. Click **Sign In** to log in.

6. Click **Inventories** at the top of the screen and confirm that the **Classroom** environment is present.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Hosts with Ansible Tower

Objectives

After completing this section, students should be able to:

- Manage hosts with Ansible Tower

Tower inventories

Administrators can manage Ansible inventories directly in the web interface of Ansible Tower. Remember that an inventory is a collection of hosts on which Ansible (or Ansible Tower) can run jobs. The following table shows available fields that can be specified when creating an inventory:

Tower Inventory Fields

Field	Description
Name	The name of the inventory to create. This field is mandatory.
Description	More description information about the inventory. This field is optional.
Organization	The organization that the inventory belongs to. This controls which users have access to this inventory.
Variables	Host-specific variables, in JSON or YAML format, which can be used in Tower projects.

An existing inventory can be edited by clicking the pencil icon next to its name. The following screenshot shows the Create Inventory window:



Figure 11.12: Tower Create Inventory window

Administrators can run ad hoc commands against an inventory by clicking the rocket icon. The assistant lists the supported modules and the module options that can be passed as arguments. The following screenshot shows the assistant for ad hoc commands:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Figure 11.13: Tower ad hoc commands

Managing host groups with Ansible Tower

Once an inventory is created, hosts and host groups can be created in it. Groups allow logical organization of hosts, and can also be used for dynamic host discovery from a cloud provider or other information source. The following table shows available options for inventory groups:

Tower Host Group Options

Option	Description
Name	The name of the inventory to create. This field is mandatory.
Description	The optional, detailed description for the inventory.
Variables	Host-specific variables, in JSON or YAML format, which can be used in Tower projects.
Source	The source to manage hosts from. The source can be a cloud or infrastructure provider.

The following screenshot shows an inventory overview for host and group management:

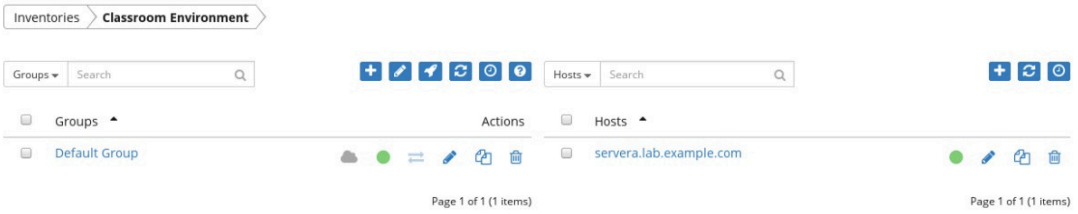


Figure 11.14: Tower inventory overview

A group can be edited or copied. A group can also be moved to be a subgroup of an existing group within an organization. Host groups in Ansible Tower follow the same principles as regular host groups created within inventory files.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

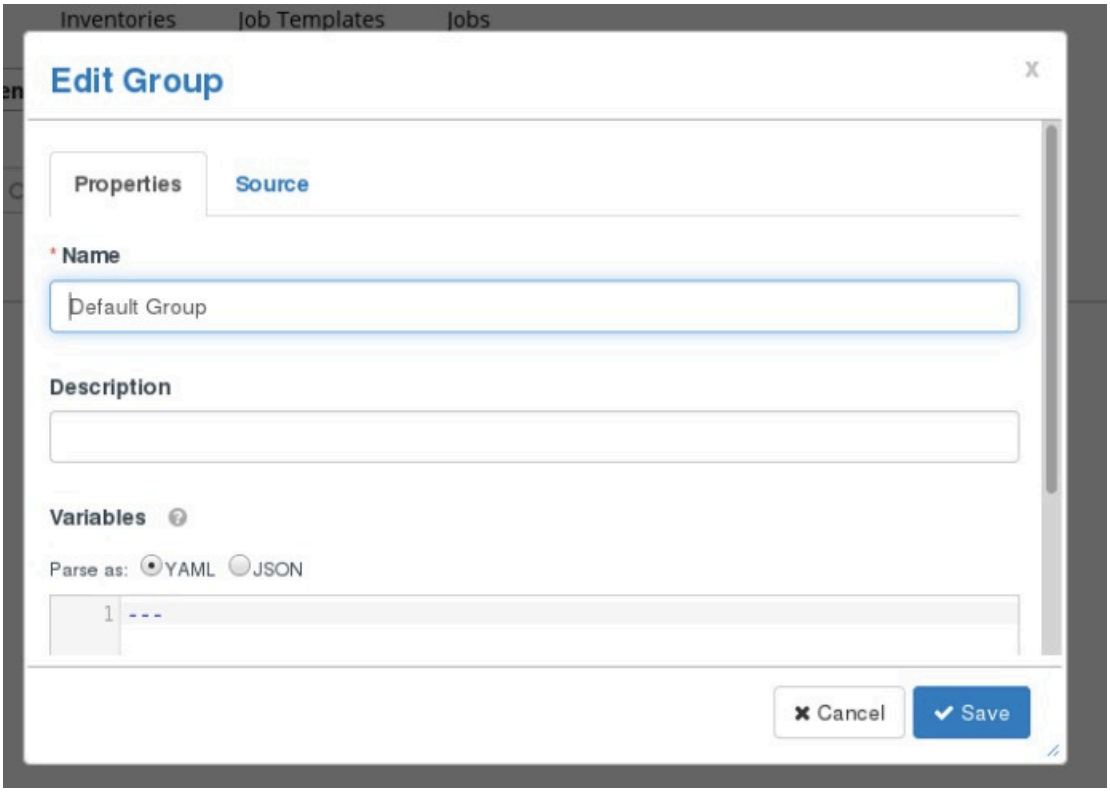


Figure 11.15: Tower host group properties

Administrators can choose various sources for host provisioning within an inventory. Before being able to automatically discover hosts from providers, a set of credentials must be defined for a team or a user. Credentials will be used to connect to the provider endpoint.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

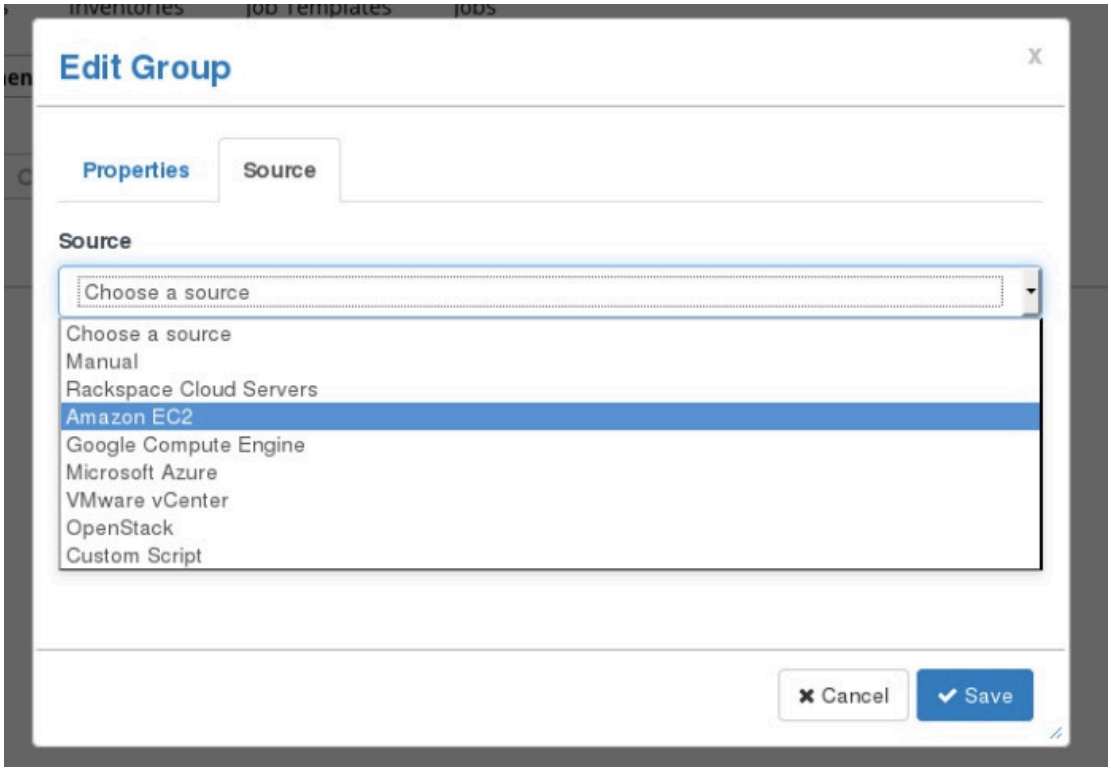


Figure 11.16: Tower host group source option

When a user attempts to delete a host group that has hosts, Ansible Tower will warn the user and will ask whether the hosts should be removed or moved to another group.

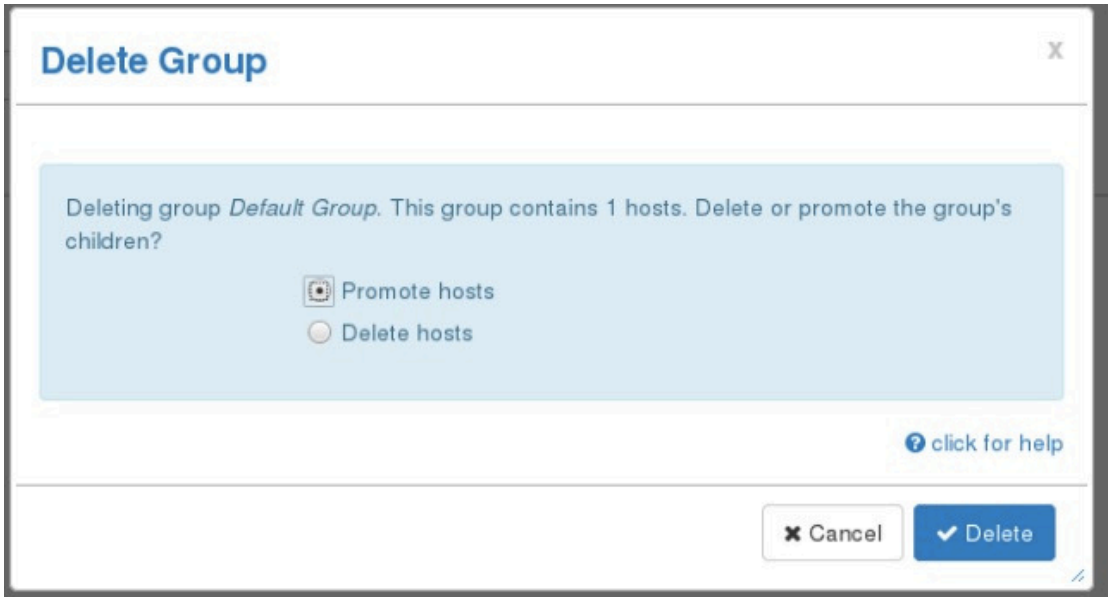


Figure 11.17: Tower host groups removal

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing hosts with Ansible Tower

Upon the creation of an inventory, hosts can either be added manually or they can be automatically discovered from a platform or cloud provider. Examples of such providers include Amazon EC2, Microsoft Azure, OpenStack, Rackspace Public Cloud, and others. The following table shows the available fields for inventory hosts in Ansible Tower:

Ansible control node components

Field	Description
Host Name	The fully-qualified domain name, or IP address, of the host to manage. This field is mandatory.
Description	The description of the host. This field is optional.
Enabled	This checkbox determines whether to include or exclude a host from plays.
Variables	Host-specific variables, in JSON or YAML format, which can be used in Tower projects.

After a host has been created, it can be removed, copied, or moved to another inventory. The circle next to each host indicates the status of jobs run against the host. An empty circle means no jobs have been run against the host. A green circle indicates the last job run against the host succeeded. A red circle means the last job run against the host failed. The following screenshot shows various states for managed hosts:

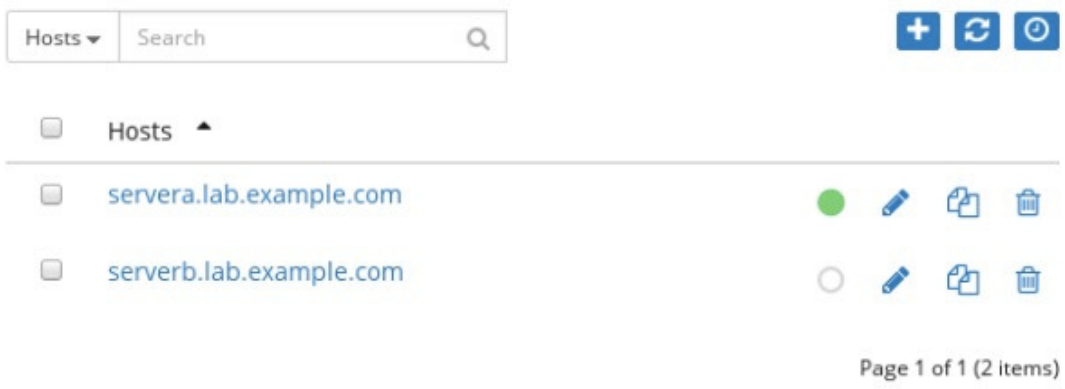


Figure 11.18: Tower host job status

In the previous screenshot, the last job that ran on **servera** completed successfully, but no jobs have been run on **serverb**.



References

Inventories — Ansible Tower User Guide
http://docs.ansible.com/ansible-tower/2.2.0/html/userguide/inventories.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Hosts with Ansible Tower

In this exercise, you will add managed hosts to an inventory by editing as the **admin** user. Then run ad hoc commands against the inventory with the non-admin account.

Outcomes
You should be able to:
- Add managed hosts with Ansible Tower.

Before you begin
Log in to **workstation** as **student**, with a password of **student**. Run the **ansible-tower-managenodes** lab setup script.

```
[student@workstation ~]$ lab ansible-tower-managenodes setup
```

This command confirms the Ansible Tower web interface is reachable at **https://tower.lab.example.com**. It also checks if the **student** Tower user exists.

Steps
1. From **workstation.lab.example.com**, open a browser and connect to the Ansible Tower page, located at **https://tower.lab.example.com**.

   Log in to the Tower interface using **admin** as the user name, **redhat** as the password.

2. Click the **Inventories** link at the top of the page, and edit the inventory by clicking the **Classroom** link.

3. Add a group class **webservers** to the **Classroom** inventory.

   3.1. On top of the **Groups** section, click the **plus** (+) to add the group, and then click **Save**.

4. Add **servera.lab.example.com** as a managed host to the **webservers** group.

   4.1. Open the **webservers** group by clicking on the **webservers** link.

   4.2. Click the **+** icon that corresponds to the hosts in the group, in the middle right of the page.

   In the **Host Properties** window enter the **Host Name** as **servera.lab.example.com**, and ensure the **Enabled** checkbox is selected.

5. Create machine credentials, which are used by Ansible Tower to connect to the managed nodes.

   5.1. Click the wrench and screwdriver icon in the upper right of the page to open the setup page.

   5.2. Click the **Credentials** link, and then the **+** button to open the **Create Credential** page. Enter the following information to create the credential:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Field	Value
Name	devops credential
Description	DevOps operator credentials for classroom, SSH key and sudo
Does this belong to a team or user	user
User that owns the credential	student
Type	Machine
Username	devops
Password	Leave blank
Private Key	Paste the content from the **/home/student/.ssh/lab_rsa** file.
Privilege Escalation	Sudo
Privilege Escalation Username	root
Privilege Escalation Password	Leave blank
Vault Password	Leave blank

6. Log out from the **admin** account on **https://tower.lab.example.com** and log in using the **student** account using the password **redhat123**.

7. Click the **Inventories** page and select the **Classroom** inventory by clicking on the name. Verify that **servera.lab.example.com** is visible.

8. Verify by running the **uptime** ad hoc command on **servera.lab.example.com** using the **command** Ansible module.

   8.1. Click the **Inventories** page and select the **Classroom** inventory by clicking on the name.

   8.2. Select the **webservers** group by clicking on the name.

   8.3. Click the rocket icon, third one from the left for the **webservers** group. This will open the **Run Command** page. Enter the following information:

Field	Value
Module	command
Argument	uptime
Host Pattern	all
Machine Credential	devops credential

   8.4. Click the **Launch** button to execute the ad hoc command. The output will be available on **Standard Output** window. Click refresh button on the right hand corner to view the output.

```
Identity added: /tmp/ansible_tower_10bzbU/ad_hoc_credential
    (/tmp/ansible_tower_10bzbU/ad_hoc_credential) servera.lab.example.com |
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      SUCCESS | rc=0 >>  16:31:26 up 55 min,  1 user,  load average: 0.00, 0.01,
0.01
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Managing Jobs in Ansible Tower

Objectives

After completing this section, students should be able to:

- Manage jobs in Ansible Tower

Manage jobs in Ansible Tower

Ansible Tower allows jobs to be used to execute ad hoc commands or to run playbooks. A logical collection of playbooks is organized into a *Project*. Tower allows *Job Templates* to be created that can predefine how to run a playbook from a project as a job, and which can be reused repeatedly and shared with other users.

Projects

Projects are logical collections of Ansible playbooks in Ansible Tower. These playbooks either reside on the Ansible Tower instance, or in a source code version control system supported by Tower, such as Git, Subversion, or Mercurial. The default project directory where Ansible Tower looks for playbooks is **`/var/lib/awx/projects`**, but can be modified by using the environment variable **$PROJECT_BASE**. The project directories and files should be owned by the **awx** user, which runs the Tower service.

Add a new project

To create a new project, click **+** in the **Projects** page. If you are using a local directory on the Tower instance, use the **SCM Type** as **Manual**, after manually creating the project's directory under **`/var/lib/awx/projects`** which is owned by the **awx** user.

```
[student@demo ~]$ sudo mkdir /var/lib/awx/projects/demoproject
[student@demo ~]$ sudo cp demo.yml /var/lib/awx/projects/demoproject
[student@demo ~]$ sudo chown -R awx /var/lib/awx/projects/demoproject
```



Figure 11.19: Create Projects in Ansible Tower

Job Templates

A job template contains the playbook and related parameters needed for Tower to execute a job. After creating a job template it can be reused for future jobs and shared between *Teams*.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Create a new job template

To create a new job template, click **+** in the **Job Templates** page. The Create Job Templates page has the following fields:

- Name
  The name to associate with the job template.
- Job Type
  Can be **Run** (to execute the playbook when launched), **Check** (checks only the syntax and the environment setup but does not execute the playbook), or **Scan** (gathers system information).
- Inventory
  The inventory to use when executing this job template.
- Project
  Project to be used with this job template.
- Playbook
  The playbook to be launched with this job template.
- Credential
  Which of the user's credentials to use with this job template in order to authenticate to the managed hosts.
- Forks
  The number of parallel or simultaneous processes to use while executing the playbook.
- Limit
  A host pattern to further constrain the list of hosts managed or affected by the playbook.
- Job Tags
  A comma-separated list of playbook tags to constrain what parts of the playbooks are executed.
- Extra Variables
  Pass extra command line variables to the playbook. This is the `-e` or `--extra-vars` command line parameter for **`ansible-playbook`** command.
- Prompt for Extra Variables
  If this is checked, the user is prompted for **Extra Variables** at job execution.
- Enable Survey
  Set extra variables through a user-friendly wizard when the job template is used to start a job. The survey will also validate user input before using it. *Surveys* are only available to customers with *Enterprise* edition licenses.'
- Allow Callbacks
  Allows a host to use the Tower API to launch a job from this job template through a *provisioning callback*. This is used for a managed host to initiate a playbook run against itself, so it does not need to wait for a user to launch a job from the Tower console.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

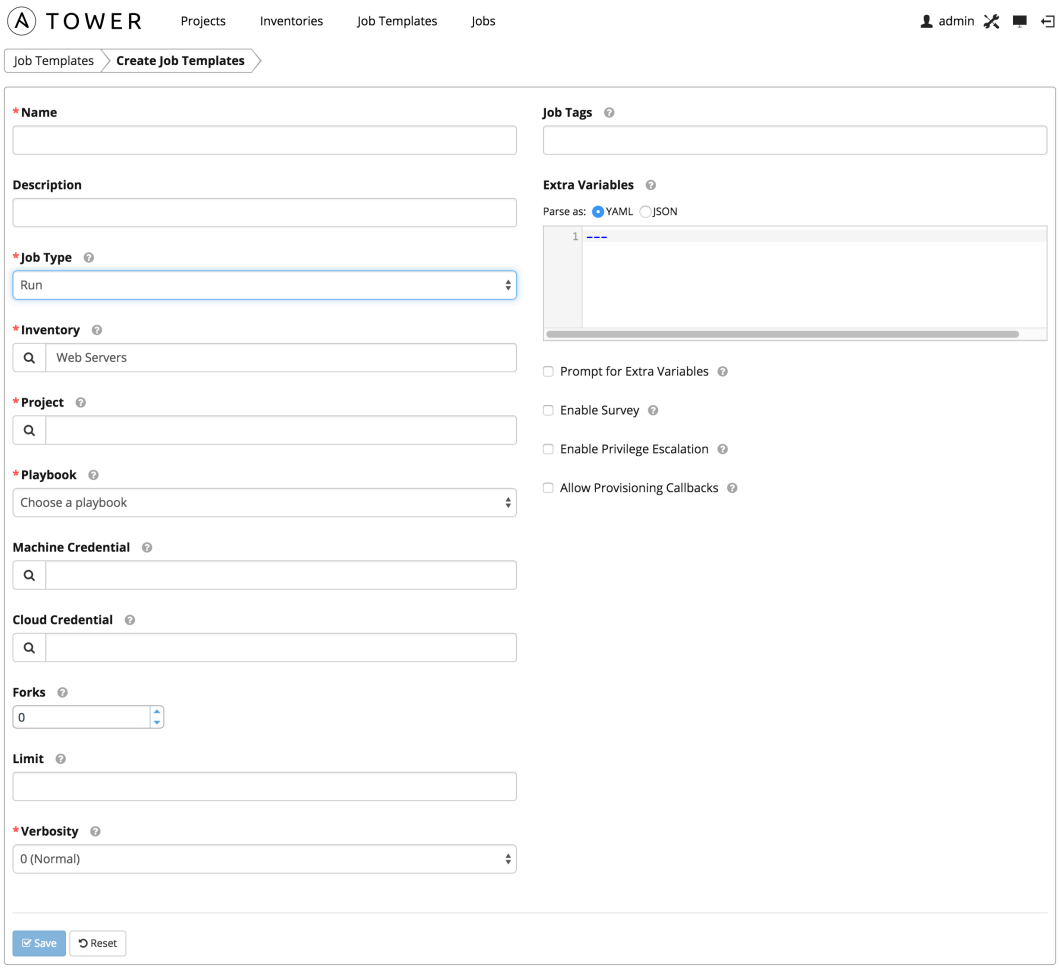


Figure 11.20: Create Job Templates in Ansible Tower

Launching jobs

After creating a job template for a job that contains all the parameters normally passed to **`ansible-playbook`**, it can be used to launch a *push-button* deployment of that job.

Launch a job template

To launch a Job Template, click rocket button under the **Actions** column of the job template to be launched. A job may require additional information to run. The following data may be requested at launch:

- Credentials
- Password or passphrase to connect to remote manage hosts, if set as *Ask*.
- Survey questions, if configured for the job templates.
- Extra variables, if requested by the job template.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

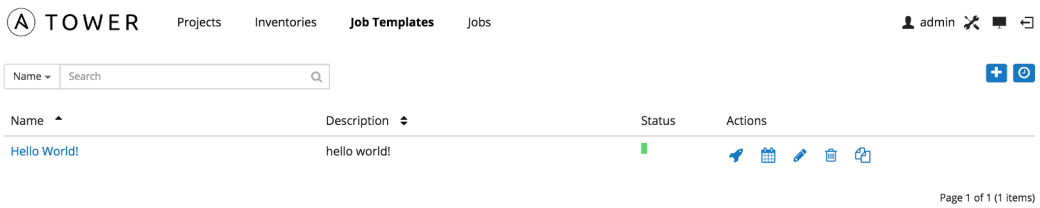


Figure 11.21: Launch a job template in Ansible Tower

Scheduling a job

Ansible Tower can schedule jobs to run at a particular time or on a repeating basis. For example, it may be that any available updates need to be applied during a maintenance window at 8 AM on Tuesdays. The **Add Schedule** page can be used to set a schedule on which to run a job template.

Schedule jobs

To schedule a job, click the calendar icon under the **Actions** column for the job template that needs to be scheduled. Use the **+** icon button to add a new schedule. The **Add Schedule** page has the following fields:

- Name: Schedule name
- Start Date: Start date of the schedule.
- State Time: Start time of the schedule.
- Local Time Zone: Time zone to be used.
- Repeat frequency: Frequency of the execution of the scheduled job.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

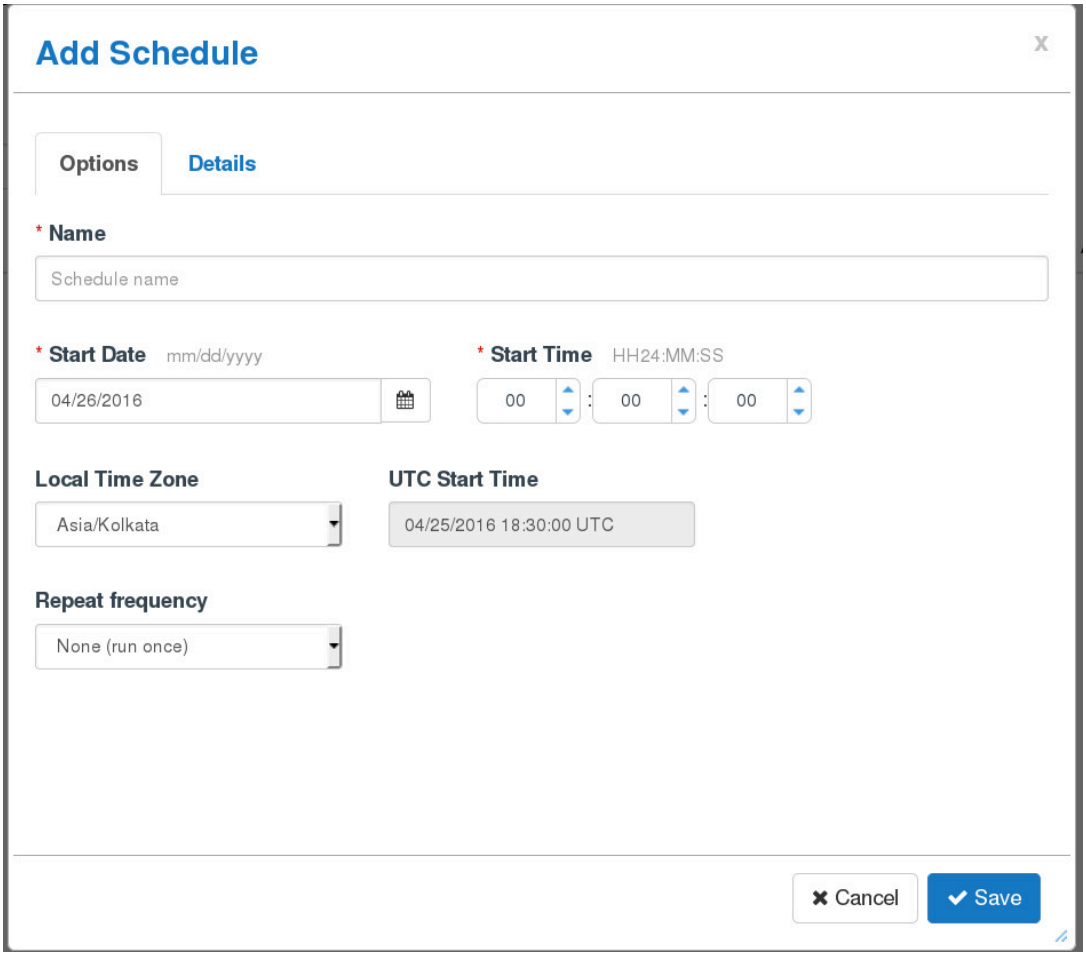


Figure 11.22: Schedule job in Ansible Tower

Checking the job status

The **Jobs** page for playbook *Run* jobs shows details of all the tasks and events for that playbook run. The **Jobs** page consists of multiple areas: *Status*, *Plays*, *Tasks*, *Host Events*, *Events Summary*, and *Hosts Summary*.

- Status: The *Status* area shows status of the job, which can be either *Running*, *Pending*, *Successful*, or *Failed*.
- Plays: The *Plays* area shows the plays that were run as part of the playbook. For each play, Tower shows the start time for the play, the elapsed time of the play, the play **Name**, and whether the play succeeded or failed. Clicking a specific play filters the **Tasks** and **Host Events** area to only display tasks and hosts relative to that play.
- Tasks: The **Tasks** area shows the tasks run as part of plays in the playbook. For each task, apart from the other details it shows summary of host status for that task. The host status can be *Success*, *Changed*, *Failure*, *Unreachable*, or *Skipped*.
- Host Events: The **Host Events** area shows hosts affected by the selected play and task.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Event Summary: The **Events Summary** area shows a summary of events for all hosts affected by this playbook. For each host, the **Events Summary** area shows the host name and the number of completed tasks for that host, sorted by status.
- Host Summary: The **Host Summary** area shows a graph summarizing the status of all hosts affected by this playbook run.

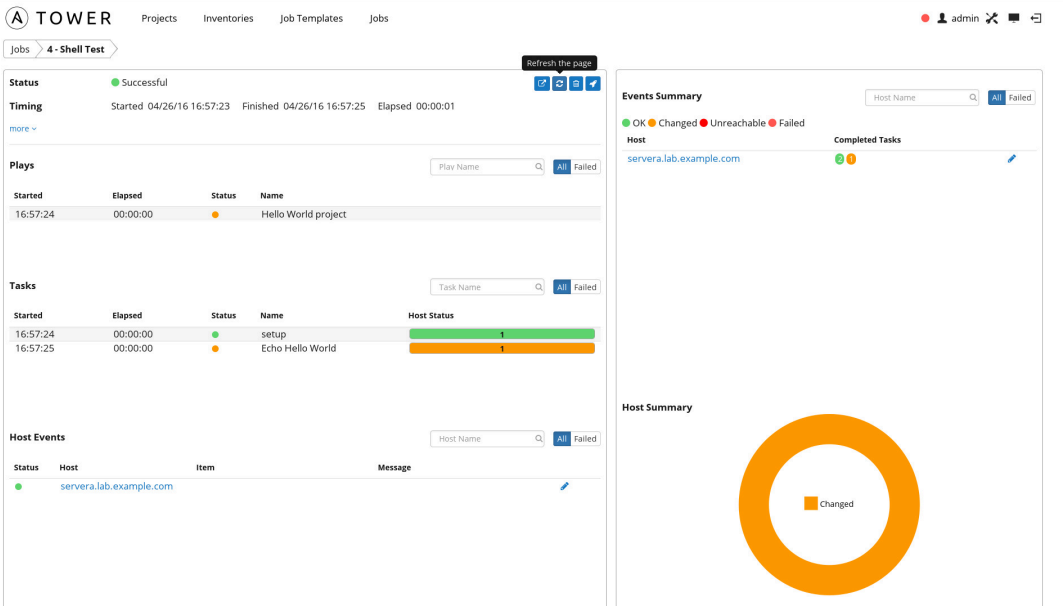


Figure 11.23: Job status in Ansible Tower



References

Projects — Ansible Tower User Guide
http://docs.ansible.com/ansible-tower/latest/html/userguide/projects.html

Job Templates — Ansible Tower User Guide
http://docs.ansible.com/ansible-tower/latest/html/userguide/job_templates.html

Jobs — Ansible Tower User Guide
http://docs.ansible.com/ansible-tower/latest/html/userguide/jobs.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Managing Jobs in Ansible Tower

In this exercise, you will create projects and job templates associated with the playbook stored in the local directory of **tower.lab.example.com**. Schedule the same job to run at scheduled time every week.

Outcomes
You should be able to:
- Create and execute jobs using playbooks stored in the local directory.
- Schedule a job to run at a given time.

Before you begin
Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-tower-managejobs setup** script, which ensures the Ansible Tower interface is reachable at **http://tower.lab.example.com**. It also checks if the required user named **student**, credential, and inventory are created from the previous guided exercises.

```
[student@workstation ~]$ lab ansible-tower-managejobs setup
```

Steps
1. On **workstation**, create a playbook called towertest.yml and add the following content:

```
---
- name: Tower Test
  hosts: all
  tasks:

    - name: Current date
      shell: date "+DATE- %Y-%m-%d%nTime- %H:%M:%S"
```

2. Copy the **towertest.yml** file to **tower.lab.example.com:/tmp** as the **devops** user.

```
[student@workstation ~]$ scp towertest.yml devops@tower.lab.example.com:/tmp
```

3. From **workstation**, remotely log in to **tower.lab.example.com** as the **devops** user.

```
[student@workstation ~]$ ssh devops@tower.lab.example.com
Last login: Fri Apr 22 19:23:57 2016 from localhost
[devops@tower ~]$
```

4. Create the **/var/lib/awx/projects/managing-jobs** directory on **tower** using **sudo**.

```
[devops@tower ~]$ sudo mkdir /var/lib/awx/projects/managing-jobs
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5. Copy **/tmp/towertest.yml** to the directory **/var/lib/awx/projects/managing-jobs** using **sudo**.

```
[devops@tower ~]$ sudo cp /tmp/towertest.yml /var/lib/awx/projects/managing-jobs
```

6. Ensure that the **/var/lib/awx/projects/managing-jobs** and file **towertest.yml** are owned by the same user and group that the Tower service runs as (**awx**).

```
[devops@tower ~]$ sudo chown -R awx: /var/lib/awx/projects/managing-jobs
```

7. From **workstation.lab.example.com**, open a browser and connect to the Ansible Tower page, located at **https://tower.lab.example.com**. Log in to the Tower interface using **admin** as the user name, and **redhat** as the password.

8. Create a new project by clicking **Projects** link at the top of the page. Use the **+** button on the upper right side of the page to open the **Create Project** page. Enter the following information to create the new project:

Field	Value
Name	Tower Test
SCM Type	Manual
Playbook Directory	managing-jobs

9. Add permission to the student user to create job templates and execute them. Go to the setup page by clicking the wrench and screwdriver in the upper right of the page. Click **Users** to open the users page.

   Edit the **student** user by clicking the pencil icon under **Actions** in **student** row. Add permissions in the **Permissions** page by clicking the **+** icon. Provide the following values for the fields in the page:

Field	Value
Permission Type	Job Template
Name	Job Template Privileges
Project	Tower Test
Inventory	Classroom
Permission	Create a Job Template

   Click **Save**.

10. Log out as the **admin** user from the Ansible Tower portal and log in as **student** using the password **redhat123**.

11. Add a new job template by clicking the **Job Templates** page and then the **+** button on the upper right corner. Enter the following details in the fields of **Create Job Templates** page.

Field	Value
Name	Tower Test

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Field	Value
Job Type	Run
Inventory	Classroom
Project	Tower Test
Playbook	towertest.yml
Machine Credentials	devops credential

Click **Save**.

12. Verify the project and job template created, by running the job by clicking rocket icon under **Actions** column in **Tower Test** job template row. The **Status** shows **Running** with a flashing green dot when refreshed.

    Refresh the page to see the **Status** for the tasks and plays; green indicates OK and yellow indicates changes made on the managed hosts.

13. Click on the **Job Templates** link to go to the **Job Templates** page. Schedule the **Tower Test** job template to run at the scheduled time. Click the calendar icon, under the **Actions** column, in the **Tower Test** row in the **Job Templates** page. Add a new schedule by clicking **+**.

    Enter the following values in the **Add Schedule** window:

Field	Value
Name	Tower Test Weekly Job
Start Date	Use the current date on **`tower.lab.example.com`**
Start Time	Calculate two minutes from the current time on **`tower.lab.example.com`**
Repeat frequency	1 weeks
On Days	Mon, Tue, Wed, Thu, Fri
End	Never

    Click **Save**.

14. Verify the successful execution of the scheduled playbook tasks. Select the **Jobs** tab from the top menubar. In the **Jobs** page, you can view the time when the **`Tower Test`** job ran.

    If the job did not run, check whether the **Start Time** was set to some point before the job was saved.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Ansible Tower

In this lab, you will continue working with the Ansible Tower instance that was installed earlier in the unit. Playbooks will use **serverb.lab.example.com** and **workstation.lab.example.com**.

Outcomes
You should be able to:

- Configure users and privileges.
- Manage managed hosts.
- Manage jobs.

Before you begin
Log in to **workstation** as **student**, using **student** as the password. Run the **lab ansible-tower-lab setup** command.

```
[student@workstation ~]$ lab ansible-tower-lab setup
```

It confirms that Ansible Tower is installed on **tower**.

Steps

1. Create a new user with the following details:
   - Name: Ansible Operator
   - Email: student@workstation.lab.example.com
   - Organization: Default
   - User: operator
   - Password: redhat123
2. Create a new inventory with the following parameters:
   - Name: **lab**
   - Description: **lab**
   - Organization: **Default**
   - Leave variables empty for the time being.
3. Create a new permission type for the **operator** user, in the **lab** inventory, with the following details:
   - Permission Type: **Inventory**
   - Name: **lab-perms**
   - Description: **lab permissions**
   - Inventory: **lab**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Permission: **`Read Inventory`**
- Execute Commands on the Inventory: checked

4. Create a new group called *website* in the *lab* inventory, and add **`serverb.lab.example.com`** to it. Use the following details:
   - Name: *website*
   - Description: *website*
   - Leave variables empty for the time being.
5. Create a new machine credential, with the following details:
   - Name: **`Second Ansible operator, SSH key and sudo`**
   - Description: **`Second Ansible operator credentials for the website group, SSH key and sudo.`**
   - Does this belong to a team or user: **`user`**
   - User that owns the credential: **`operator`**
   - Type: **`Machine`**
   - Username: **`devops`**
   - Password: Leave blank
   - Private Key: Paste in the contents from the **`/home/student/.ssh/lab_rsa`** file
   - Privilege Escalation: **`Sudo`**
   - Privilege Escalation Username: **`root`**
   - Privilege Escalation Password: Leave blank
6. Run an ad hoc command as the **`operator`** user, on all the managed hosts of the **`website`** group of the **`lab`** inventory. The ad hoc command will use the following details:
   - Module: **`setup`**
   - Host Pattern: **`all`**
   - Machine Credential: **`Second Ansible operator, SSH key and sudo`**
   - Verbosity: **`0 (Normal)`**
7. Create a second project using the project files for the **`ansibletower-lab`** project, available at **`http://materials.example.com/tower/ansibletower-lab.tar`**. Name the project **`Web Site`**.
8. Create and execute a job template using the following details:
   - Name: **`Web Site Template`**
   - Job Type: **`Run`**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Inventory: **lab**
- Project: **Web Site**
- Playbook: **systemnotice.yml**
- Machine Credential: **Second Ansible operator, SSH key and sudo**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will continue working with the Ansible Tower instance that was installed earlier in the unit. Playbooks will use **serverb.lab.example.com** and **workstation.lab.example.com**.

Outcomes
You should be able to:
- Configure users and privileges.
- Manage managed hosts.
- Manage jobs.

Before you begin
Log in to **workstation** as **student**, using **student** as the password. Run the **lab ansible-tower-lab setup** command.

```
[student@workstation ~]$ lab ansible-tower-lab setup
```

It confirms that Ansible Tower is installed on **tower**.

Steps
1. Create a new user with the following details:
   - Name: Ansible Operator
   - Email: student@workstation.lab.example.com
   - Organization: Default
   - User: operator
   - Password: redhat123

   1.1. Log in to the web interface as the **admin** user, using **redhat** as the password.

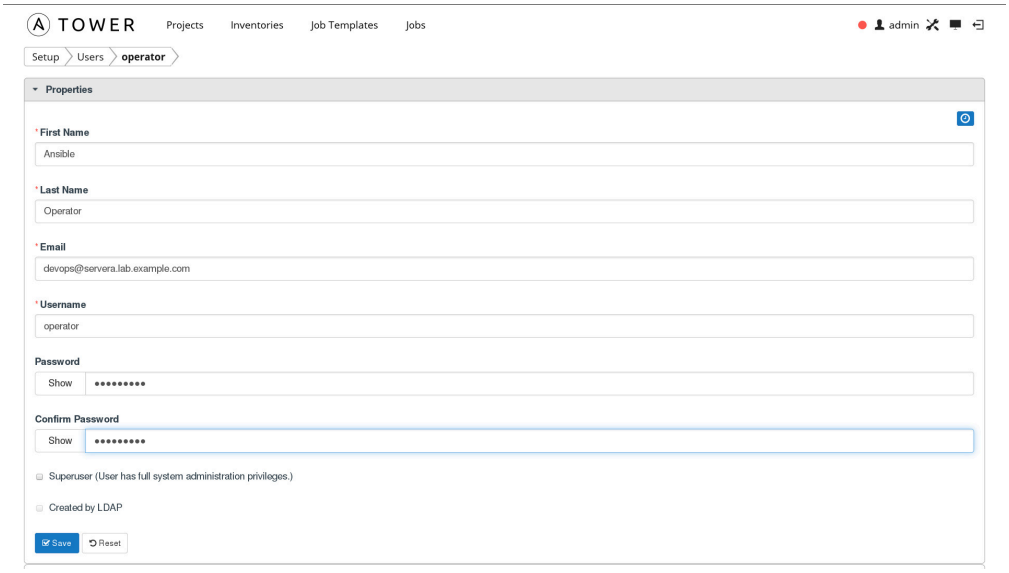
   1.2. Open the **Setup** page by clicking on the wrench and screwdriver icon in the upper right.

   1.3. Click **Users** link in the middle left of the page.

   1.4. Click the plus icon in the middle right side of the page to create a new user.

   1.5. Fill in the details as follows:
      - First Name: **Ansible**
      - Last Name: **Operator**
      - Email: **student@workstation.lab.example.com**
      - Organization: **Default**
      - Username: **operator**
      - Password: **redhat123**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Creating the user

1.6. Click the **Save** button to create the **`operator`** user in Ansible Tower.

2. Create a new inventory with the following parameters:
   - Name: **`lab`**
   - Description: **`lab`**
   - Organization: **`Default`**
   - Leave variables empty for the time being.

   2.1. Click the **Inventories** link at the top of the page, then click the plus icon on the top right to open the **Create Inventory** wizard.

   Create an inventory with the following parameters:
   - Name: **`lab`**
   - Description: **`lab`**
   - Organization: **`Default`**
   - Leave variables empty for the time being.

   2.2. Click the **Save** button in order to create the **`lab`** inventory in Ansible Tower.

3. Create a new permission type for the **`operator`** user, in the **`lab`** inventory, with the following details:
   - Permission Type: **`Inventory`**
   - Name: **`lab-perms`**
   - Description: **`lab permissions`**
   - Inventory: **`lab`**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Permission: **`Read Inventory`**
- Execute Commands on the Inventory: checked

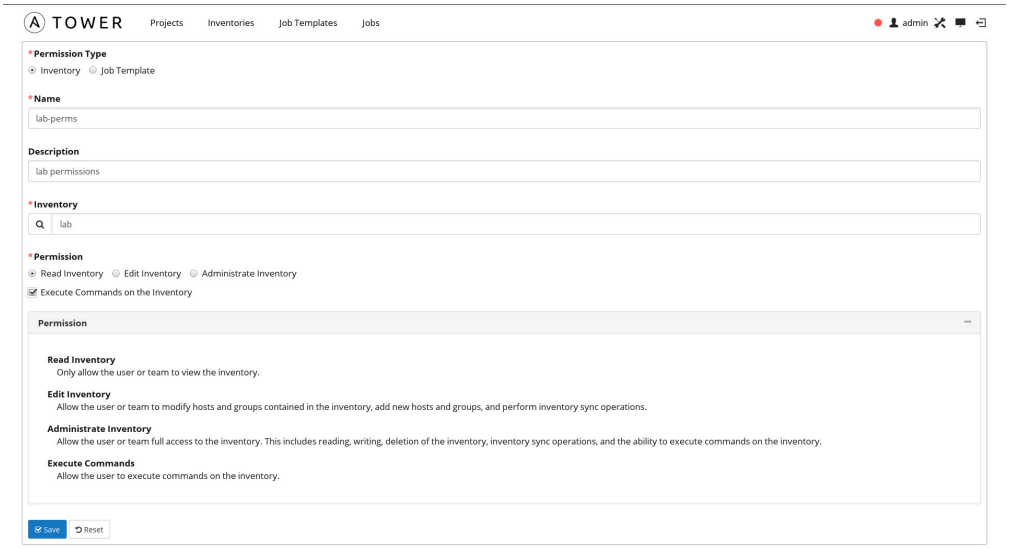
3.1. Add permissions for the new user. Navigate to the user by clicking on the wrench and screwdriver icon in the top right of the screen. Next click on the **Users** link and select the **`operator`** user. Open the **Permissions** section by clicking on it.

3.2. Create a new permission by first creating a **Permission Type**. Click the plus button in the upper right of the **Permissions** section for the *operator* user page.

Fill in the details as follows:

- Name: *lab-perms*
- Description: *lab permissions*
- Inventory: *lab*
- Permission: *Read Inventory*
- Execute Commands on the Inventory: *selected*

Save the Permission.



Creating a new permission type for operator

4. Create a new group called *website* in the *lab* inventory, and add **`serverb.lab.example.com`** to it. Use the following details:
   - Name: *website*
   - Description: *website*
   - Leave variables empty for the time being.

   4.1. Open the **Inventories** page by clicking on the link at the top of the page.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

4.2. Edit the *lab* inventory by clicking on the *lab* name itself.

4.3. At this point, groups and hosts can be added to the inventory. Add the *website* group, clicking plus button on the **Groups** section.

Fill in the details as follows:

- Name: *website*
- Description: *website*
- Leave variables empty for the time being.

4.4. Add the **serverb.lab.example.com** host to the **website** group. Open the **website** group and click on the plus icon that corresponds to the **Hosts** section in the group. Use **serverb.lab.example.com** as the **Host Name**, and keep the default values for the other fields.

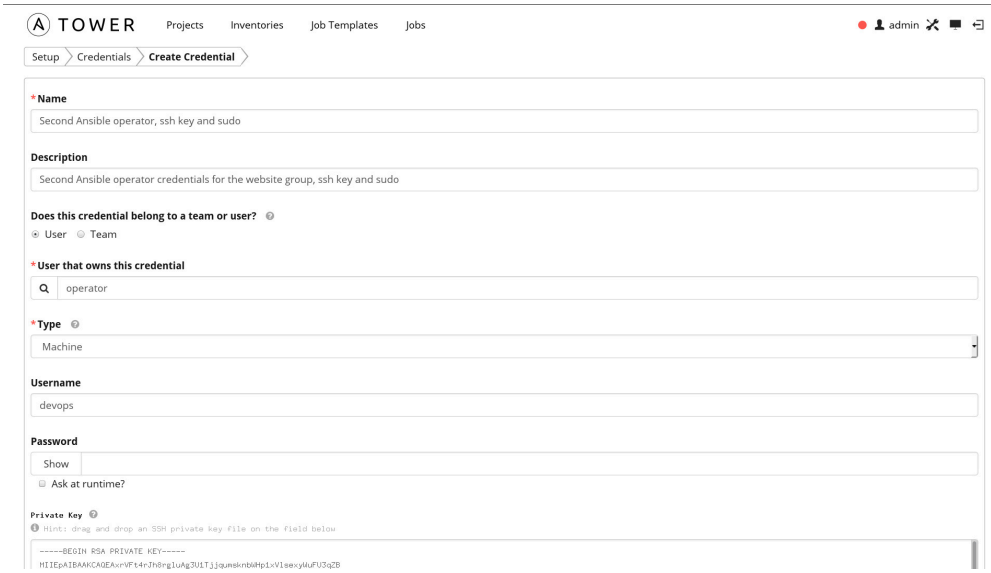
5. Create a new machine credential, with the following details:
   - Name: **Second Ansible operator, SSH key and sudo**
   - Description: **Second Ansible operator credentials for the website group, SSH key and sudo.**
   - Does this belong to a team or user: **user**
   - User that owns the credential: **operator**
   - Type: **Machine**
   - Username: **devops**
   - Password: Leave blank
   - Private Key: Paste in the contents from the **/home/student/.ssh/lab_rsa** file
   - Privilege Escalation: **Sudo**
   - Privilege Escalation Username: **root**
   - Privilege Escalation Password: Leave blank

   5.1. Click on the wrench and screwdriver icon in the upper right of the page. Click on the **Credentials** link. Click the plus button to launch the credentials wizard.

   Fill in the **Create Credential** wizard dialog as follows:
   - Name: **Second Ansible operator, SSH key and sudo**
   - Description: **Second Ansible operator credentials for the website group, SSH key and sudo**
   - Does this belong to a team or user: **user**
   - User that owns the credential: **operator**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Type: **Machine**
- Username: **devops**
- Password: Leave blank
- Private Key: Paste in the contents from the **/home/student/.ssh/lab_rsa** file in **workstation**
- Privilege Escalation: **Sudo**
- Privilege Escalation Username: **root**
- Privilege Escalation Password: Leave blank



Creating the Second Ansible operator, SSH key and sudo credential

5.2. Click **Save** to save the credential.

6. Run an ad hoc command as the **operator** user, on all the managed hosts of the **website** group of the **lab** inventory. The ad hoc command will use the following details:
   - Module: **setup**
   - Host Pattern: **all**
   - Machine Credential: **Second Ansible operator, SSH key and sudo**
   - Verbosity: **0 (Normal)**

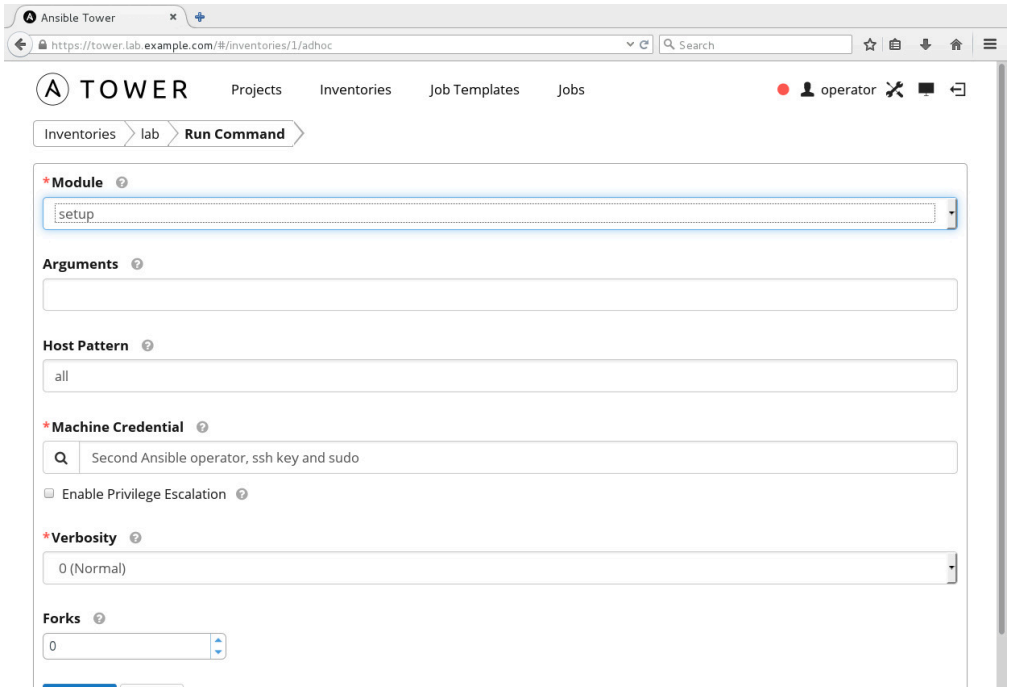
   6.1. Log out from the **admin** account, using the **Sign Out** button at the upper right, and log in with the **operator** account, using **redhat123** as the password. Go to the **Inventories** page and select the **lab** inventory by clicking the name.

   6.2. Select the **website** group by clicking the name.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

6.3. Click the rocket icon, third one from the left, for the **website** group. This will open the **Run Command** wizard screen. Fill in details as follows:

- Module: **setup**
- Host Pattern: **all**
- Machine Credential: **Second Ansible operator, SSH key and sudo**
- Verbosity: **0 (Normal)**

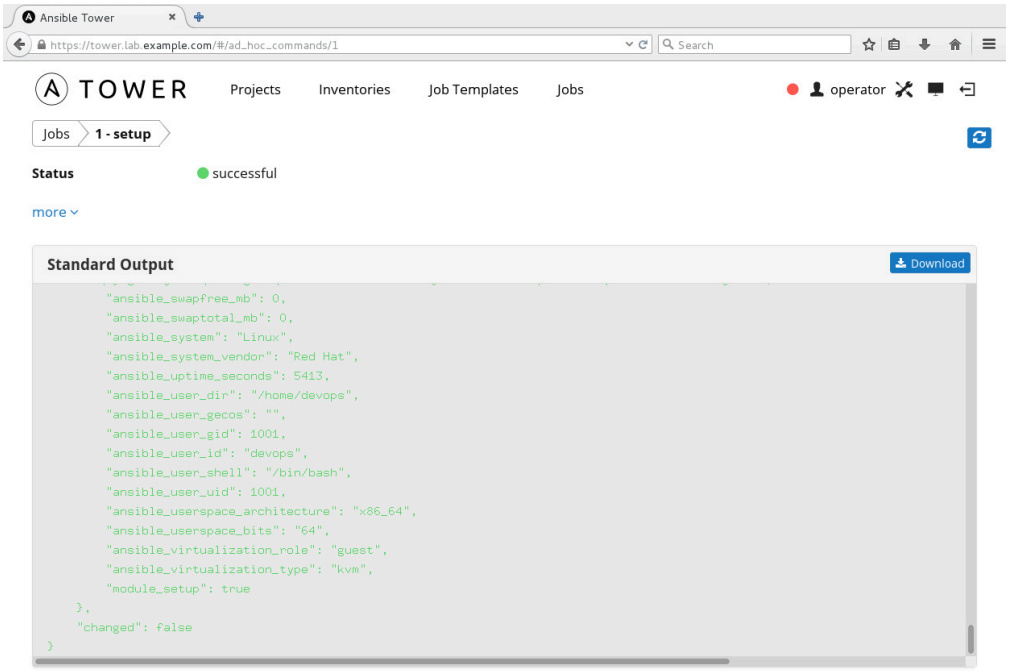


Configuring the ad hoc command

6.4. Click **Launch** to run the command.

6.5. Verify that the ad hoc command has been properly executed by checking the **Standard Output** section for the output of the ad hoc command. It should display the facts for the **serverb.lab.example.com** managed host.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Verifying the ad hoc command execution

7. Create a second project using the project files for the **ansibletower-lab** project, available at **http://materials.example.com/tower/ansibletower-lab.tar**. Name the project **Web Site**.

   7.1. Create a second project.

```
[student@workstation ~]$ wget http://materials.example.com/tower/ansibletower-
lab.tar
[student@workstation ~]$ tar xvf ansibletower-lab.tar
```

   7.2. Copy the **ansibletower-lab** directory onto **tower.lab.example.com** using the **devops** user. Log out when done.

```
[student@workstation ~]$ scp -r ansibletower-lab devops@tower.lab.example.com:/
tmp/
[student@workstation ~]$ ssh devops@tower.lab.example.com
[devops@tower ~]$ sudo cp -r /tmp/ansibletower-lab /var/lib/awx/projects/
[devops@tower ~]$ sudo chown -R awx: /var/lib/awx/projects/ansibletower-lab
[devops@tower ~]$ logout
```

   7.3. Log in to Ansible Tower as the **admin** user, using **redhat** as the password.

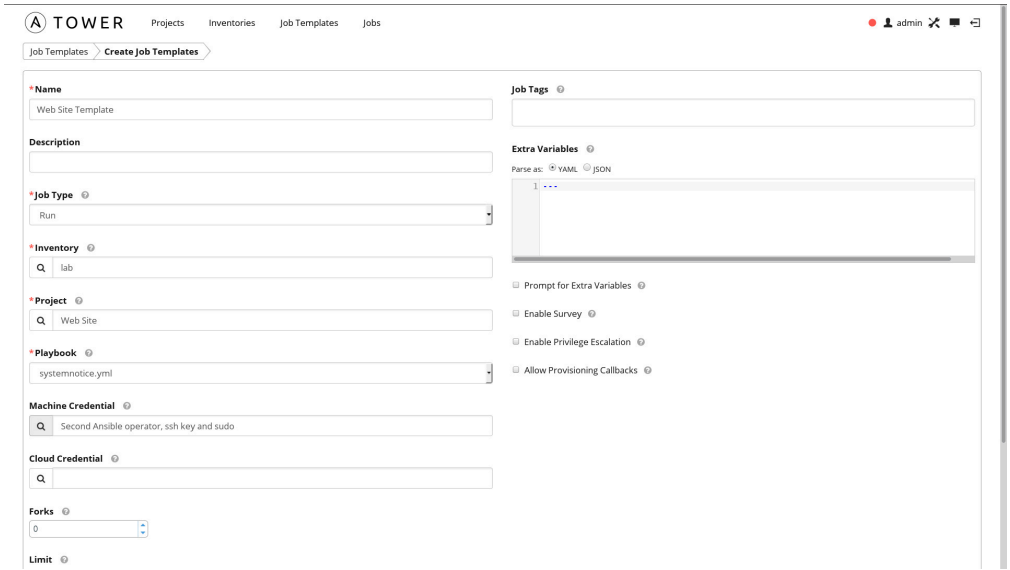
   7.4. Start the **Create Project** wizard by clicking the **Projects** link at the top of the page, and then the plus button on the top right side of the page. Create the project as follows:
   - Name: **Web Site**
   - SCM Type: **Manual**
   - Playbook Directory: **ansibletower-lab**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

7.5. Click **Save** to create the **`Web Site`** project.

8. Create and execute a job template using the following details:
   - Name: **`Web Site Template`**
   - Job Type: **`Run`**
   - Inventory: **`lab`**
   - Project: **`Web Site`**
   - Playbook: **`systemnotice.yml`**
   - Machine Credential: **`Second Ansible operator, SSH key and sudo`**

   8.1. Create a job template by clicking the **Job Templates** link at the top of the page, and then the plus button on the upper right. Create the template as follows:
      - Name: **`Web Site Template`**
      - Job Type: **`Run`**
      - Inventory: **`lab`**
      - Project: **`Web Site`**
      - Playbook: **`systemnotice.yml`**
      - Machine Credential: **`Second Ansible operator, SSH key and sudo`**



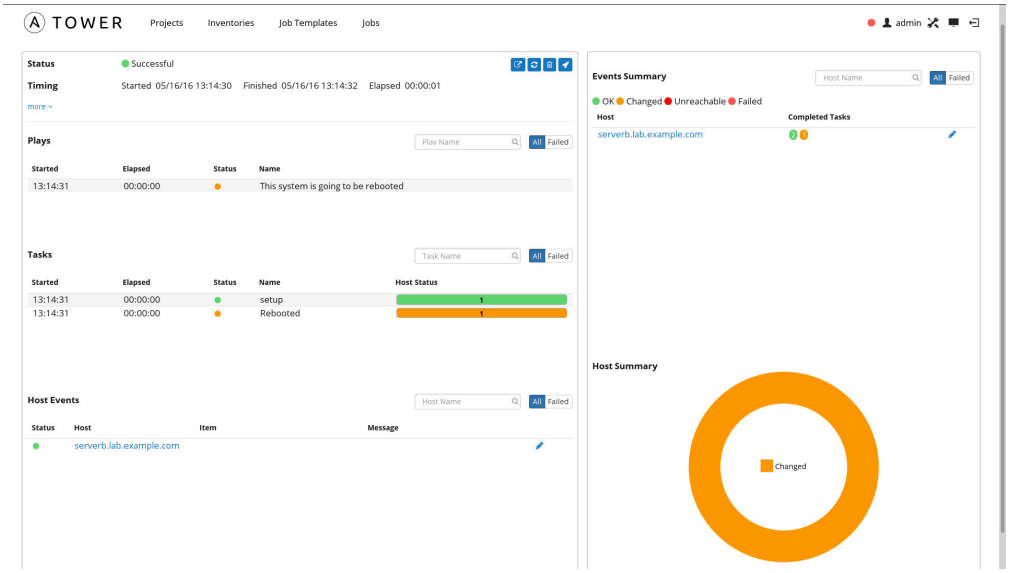
Creating the Web Site Template job template

8.2. Save the project.

8.3. Execute the job template by clicking the rocket button under the **Actions** column of the **Web Site Template** row.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

8.4. Verify in the previous screen that the job template has been successfully executed, checking the **Status** for the **`Web Site Template`** job template is **`Successful`**. Click **refresh** to update the job template status.



Verifying the Web Site Template job template execution

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible Tower provides features such as role-based access control, push-button deployment, centralized logging, and a RESTful API.
- Ansible Tower supports the automation of playbook runs, which can be scheduled for execution at a given time.
- Ansible Tower also configures a minimal Munin instance to monitor itself, allowing graphs to show information such as memory consumption and CPU usage, as well as the number of queued and active jobs.
- *Credentials* are used by Ansible Tower for various authentication, such as launching jobs against managed hosts, synchronizing with inventories, as well as importing projects content from version control systems.
- User permissions define the set of privileges assigned to users as well as teams, which uses role-based access control (RBAC). These permissions provide the ability to read, modify, and administer projects, inventories, and job templates.
- *Inventories* allows to define host and host groups.
- Ansible Tower hosts can be either added manually, or automatically discovered from a platform or cloud provider.
- *Projects* are logical collections of Ansible playbooks in Ansible Tower.
- The project directories and files should be owned by the **awx** user and group that the Ansible Tower service runs as.
- *Job Templates* contains the playbook along with its related set of parameters for running an Ansible job.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 12

IMPLEMENTING ANSIBLE IN A DEVOPS ENVIRONMENT

Overview	
Goal	Implement Ansible in a DevOps environment using Vagrant.
Objectives	• Describe Ansible 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 Ansible in a DevOps Environment

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Provisioning Vagrant Machines

Objectives

After completing this section, students should be able to:

- Discuss Ansible in a DevOps environment.
- Provision a Vagrant machine.

DevOps in the enterprise

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 derives its name from the 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 (IaC)*. 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 Ansible software has become a popular tool for managing infrastructure code. Ansible allows tasks such as software installations and server configurations to be automated. This automation removes the introduction of human errors resulting from manual administration. It also allows for the repeatability of routine tasks and removes the complexity of these tasks so they can be easily performed even by entry level administrators. Ansible's use can greatly improve an organization's operational efficiency and simultaneously enhance the predictability of successful outcomes.

Ansible's architecture allows for the 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.

Ansible'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, Ansible also provides administrators the ability to audit system configurations and detect when they deviate from the desired state.

While configuration management tools like Ansible simplify and improve infrastructure management, they also introduce the dilemma of how this new infrastructure code should be managed. To overcome this challenge, administrators could follow the example 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

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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.

Using Vagrant

When testing code for production deployment, results are only relevant and valid if a test is conducted in a development environment that 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-effective 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.

The Vagrant software overcomes this challenge by streamlining the creation and configuration of virtual development environments. Vagrant has its own domain-specific language which is used to create a set of instructions for managing virtualization software such as Virtualbox, KVM, and VMware. It can also interact with configuration management software such as Ansible, Puppet, Salt, and Chef. Vagrant 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 with different team members. Because 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:

Vagrant: The Vagrant software automates the build and configuration of virtual machines. A command line interface is provided to administrators for the management of Vagrant projects. Using the **`vagrant`** command provided, administrators can instantiate, remove, and manage Vagrant machines. Vagrant is currently not packaged with Red Hat Enterprise Linux but is available for download from the Vagrant website, **`https://www.vagrantup.com`**.

Box: A box is a **`tar`** file that 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.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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

The creator of Vagrant started the HashiCorp company in 2012 to further the development of Vagrant. 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`**. Preconfigured RHEL 7.1 Vagrant box images for libvirt and Virtual box are available from the Red Hat Container Development Kit. This course will use a preconfigured Red Hat Enterprise Linux box image because 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]# vim 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`** when it is created.

```
Vagrant.configure(2) do |config|
  config.vm.box = "rhel7.1"
  config.vm.box_url = "http://content.example.com/ansible2.0/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.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[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 is dynamically assigned an IP address by the virtualization provider.

When 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**'.

Having an identical username/password combination goes against good security practice. However, Vagrant environments are meant to be standalone virtual environments hosted on a physical system for development purposes. Having a standard account and password for the management of Vagrant machines also facilitates the use of Vagrant projects by multiple developers.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Vagrant also offers a helpful feature called *synced folders*. By default, Vagrant uses the synchronized 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. Even though 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.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Vagrant offers a variety of provisioners to suit the provisioning method preferred by administrators. The most basic provisioner is the *shell provisioner* which uses shell commands to perform provisioning tasks.

The following example demonstrates how Vagrant's shell provisioner can be used to automate the configuration of yum repositories 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
  SHELL

... Configuration omitted ...

end
```



References

Boxes — Vagrant Documentation
https://docs.vagrantup.com/v2/boxes.html

Providers — Vagrant Documentation
https://docs.vagrantup.com/v2/providers/index.html

Vagrantfile — Vagrant Documentation
https://docs.vagrantup.com/v2/vagrantfile/index.html

Vagrant Command-Line Interface — Vagrant Documentation
https://docs.vagrantup.com/v2/cli/index.html

Vagrant Provisioning — Vagrant Documentation
https://docs.vagrantup.com/v2/provisioning/index.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Provisioning Vagrant Machines

In this exercise, you will provision a Vagrant machine for use in a DevOps environment.

Outcomes
You should be able to deploy a Vagrant machine and configure it using the shell provisioner.

Before you begin
Reset your **tower** server.

Steps

1. Log in to **workstation** as **student** and run the **ansible-vagrant-practice** lab setup script. The script will make the Vagrant software available on **tower**.

```
[student@workstation ~]$ lab ansible-vagrant-practice setup
```

2. Log in to **tower** as the **root** user. Create a work directory, **/root/vagrant**, for Vagrant work. Also create a subdirectory, **webapp**, inside the Vagrant work directory.

```
[root@tower ~]# mkdir -p vagrant/webapp
```

3. In the **/root/vagrant/webapp** directory, create a Vagrant configuration file, **Vagrantfile** by copying it from **/var/tmp/Vagrantfile**. This file will be used to 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@tower ~]# cd vagrant/webapp
[root@tower webapp]# cp /var/tmp/Vagrantfile .
[root@tower webapp]# cat Vagrantfile
Vagrant.require_version ">= 1.7.0"

Vagrant.configure(2) do |config|

  # Identify which Vagrant box to use
  config.vm.box = "rhel7.1"
  config.vm.box_url = "http://content.example.com/ansible2.0/x86_64/dvd/vagrant/
rhel-server-libvirt-7.1-1.x86_64.box"

  # Define host settings
  config.vm.hostname = "dev.lab.example.com"

  # Define sync folder(s)

  # Define shell provisioner

  # Define ansible provisioner

end
```

4. Test the deployment of the virtual machine with the new configuration.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[root@tower webapp]# 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
==> default: Creating image (snapshot of base box volume).
==> default: Creating domain with the following settings...
...output omitted...
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Rsyncing folder: /root/vagrant/webapp/ => /home/vagrant/sync
```

5. Check to see which Yum repositories are available in the Vagrant box.

```
[root@tower webapp]# vagrant ssh
[vagrant@dev ~]$ yum repolist
Loaded plugins: product-id, subscription-manager
repolist: 0
```

6. Exit from and shut down the Vagrant machine. Note that your IP address may be different from the one shown in the example.

```
[vagrant@dev ~]$ exit
logout
Connection to 192.168.121.238 closed.

[root@tower webapp]# vagrant destroy
==> default: Removing domain...
```

7. Because there are no Yum repositories available, configure Vagrant so that the necessary Yum repository configuration files are created when the Vagrant machine is deployed.

   7.1. Create a directory within the Vagrant work directory to host the necessary Yum repository configuration file and then change to that directory.

   ```
   [root@tower webapp]# mkdir -p /root/vagrant/webapp/etc/yum.repos.d
   ```

   7.2. Copy the **/etc/yum.repos.d/rhel_dvd.repo** Yum repository configuration file to this newly created directory.

   ```
   [root@tower webapp]# cp /etc/yum.repos.d/rhel_dvd.repo /root/vagrant/webapp/etc/
   yum.repos.d/
   ```

   7.3. In the **/root/vagrant/webapp** directory, create a shell script, **provisioner.sh**, which copies Yum repo configuration from the sync directory to the **/etc/yum.repos.d** directory on the Vagrant box. It should have the following contents:

   ```
   #!/bin/bash

   # Install yum config file
   sudo cp /home/vagrant/sync/etc/yum.repos.d/rhel_dvd.repo /etc/yum.repos.d/
   rhel_dvd.repo
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The copy must be executed using sudo because it requires **root** privileges.

7.4. Modify **Vagrantfile** so that the **provisioner.sh** script is executed by the shell provisioner during provisioning to install the Yum repository configuration file to **/etc/yum.repos.d/rhel_dvd.repo**. 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.

```
  # Define shell provisioner
  config.vm.provision "shell", path: "provisioner.sh"
```

8. Provision the Vagrant machine using the newly modified configuration.

```
[root@tower 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/ => /home/vagrant/sync
==> default: Running provisioner: shell...
    default: Running: /tmp/vagrant-shell20160427-14951-pb7qkg.sh
```

9. Verify that a Yum repository is now available.

```
[root@tower 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                 4,620
repolist: 4,620
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Deploying Vagrant in a DevOps Environment

Objectives

After completing this section, students should be able to:

- Deploy Vagrant in a DevOps environment using Ansible

Integrating Vagrant with Ansible

In the previous section, deploying a Vagrant machine with a base operating system was shown. Vagrant's provisioning feature and the shell provisioner were demonstrated for the management of configurations on Vagrant machines after their creation from the base operating system image.

In addition to the shell provisioner, Vagrant offers a variety of provisioners to suit the provisioning methodology preferred by an organization. Two types of provisioners are offered by the Vagrant software for administrators who want to manage their Vagrant machines with Ansible.

The *ansible* provisioner performs its work by executing Ansible on the Vagrant host. When using this provisioner, Ansible is installed and executed on the Vagrant host which serves the role of the control node and the Vagrant machines are the managed hosts.

In contrast, the *ansible_local* provisioner performs its work by executing Ansible on the Vagrant machines. To use this provisioner, Ansible is installed and executed on the Vagrant machine. The Vagrant machine serves as the control node and manages itself as the managed node.

When using Vagrant's ansible 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 through the execution of playbooks. As long as the same playbook used to provision a production system is executed against a Vagrant machine, both systems can be configured identically. The Vagrant machine can 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 synchronized 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 synchronized folders in the **`Vagrantfile`** configuration file. Vagrant offers various mechanisms for keeping these folders synchronized. Vagrant's default synchronization folder type is **`VirtualBox`**. This folder type offers a bidirectional synchronization of the contents on the host and the Vagrant machine folder, and will continuously synchronize contents as changes are introduced to folder contents on either the host or the Vagrant machine.

`VirtualBox` synchronized folders are only available when using the **`VirtualBox`** provider. When using another provider, a different synchronization folder type is required. Consult the documentation to understand how each synchronized folder type works, because they can behave differently.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Because this course does not use the **VirtualBox** provider, the **rsync** synchronized folder type will be used instead. Unlike **VirtualBox** synchronized folders, **rsync** synchronized 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 only copies 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
```

Configuring Vagrant for Ansible provisioning

Like the shell provisioner, the Vagrant Ansible provisioners are employed through the use of the **config.vm.provision** method call in the **Vagrantfile** configuration file. The following example demonstrates how Vagrant's **ansible** provisioner can be used in conjunction with an Ansible playbook, **playbook.yml**, located in the Vagrant project directory, to automate the configuration of a Vagrant machine after it is deployed with a base operating system.

```
Vagrant.configure(2) do |config|

... Configuration omitted ...

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = 'playbook.yml'
  end

... Configuration omitted ...

end
```

Because the **ansible** provisioner is used, **ansible-playbook** is executed on the Vagrant host. Therefore, the Ansible software must be already be installed on the Vagrant host when this Vagrant Ansible provisioner is used.

In contrast with the **ansible_local** provisioner, because Ansible is executed on the Vagrant machine, the Ansible software must already be installed on the Vagrant machine prior to the execution of the provisioner. This can be accomplished by installing the Ansible software using another provisioner prior to execution of the **ansible_local** provisioner.

The following example demonstrates how Vagrant's **ansible_local** provisioner can be used in conjunction with an Ansible playbook, **playbook.yml**, located in the Vagrant project directory, to automate the configuration of a Vagrant machine after it is deployed with a base operating system. Because the Ansible software is a prerequisite for the successful execution of the **ansible_local** provisioner, the example uses the **shell** provisioner to first configure a Yum repository and install the Ansible software on the Vagrant machine.

```
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 ansible
    SHELL
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
... Configuration omitted ...

    config.vm.provision "ansible_local" do |ansible|
      ansible.playbook = 'playbook.yml'
    end

... Configuration omitted ...

end
```

Creating a Vagrant development environment

When a Vagrant machine has been provisioned using a Vagrant Ansible provisioner, it should have all the software needed for its intended purpose. For example, when deploying a Vagrant machine for the development of a web server, one would expect the Ansible playbook used by the Ansible provisioner to install Apache, start the web service, and configure the firewall for HTTP access.

In addition to configuration management, the creation of a development environment can be automated further by incorporating application source code installation into the provisioning playbook. Ansible offers several source control modules to work with version control software, such as Git and Subversion.

The following example shows how the contents of a web application's **DocumentRoot** folder can be installed by calling the **git** module in an Ansible playbook. The task uses the **git** module to clone the contents of the Git repository, **webapp.git**, on **demo.example.com** into the **/var/www/** directory on the Vagrant machine. The **repo** option defines the address of the Git repository. The **dest** option defines the target location for the installation of the source code on the Vagrant machine. The **accept_host** option is useful when accessing a Git repository using the SSH protocol. It automatically adds the host key for the repository URL.

```
... Configuration omitted ...
  - name: get source
    git:
      repo: ssh://student@workstation/home/student/git/webapp.git
      dest: /var/www/html
      accept_hostkey: yes
... Configuration omitted ...
```

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
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



Note

Configuration changes made to **`Vagrantfile`** will not have any effect on an running Vagrant machine. Changes will take effect when the machine is halted with **`vagrant halt`** or **`vagrant destroy`**, and then launched again 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 re-executed when **`vagrant reload`** is issued.

Creating a reusable Vagrant development environment

When a Vagrant project is configured to use Ansible in conjunction with a playbook which configures Vagrant machines according to an organization's build standard and also installs application source code, 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. Because 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 preconfigured 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, configuration files, playbooks, and so on). The **`vagrant up`** recreates the Vagrant development environment using Ansible and the retrieved components.

Because 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 make 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 accomplished quickly and easily by implementing the production playbook changes to the playbook used for the provisioning of the Vagrant development machines. The Ansible provisioner applies the new configuration when new Vagrant machines are instantiated. This has little to no impact on the developers, who are free to continue with development tasks.

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. Because the operations team have control over development environments' resemblance to the production environment, code deployments to production should be uneventful.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



References

Ansible and Vagrant — Vagrant Documentation
https://www.vagrantup.com/docs/provisioning/ansible_intro.html

Ansible Provisioner — Vagrant Documentation
https://www.vagrantup.com/docs/provisioning/ansible.html

Ansible Local Provisioner — Vagrant Documentation
https://www.vagrantup.com/docs/provisioning/ansible_local.html

Vagrant Rsync Synced Folder — Vagrant Documentation
https://docs.vagrantup.com/v2/synced-folders/rsync.html

Vagrant Forwarded Ports — Vagrant Documentation
https://docs.vagrantup.com/v2/networking/forwarded_ports.html

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Guided Exercise: Deploying Vagrant in a DevOps Environment

In this exercise, you will create a development environment using Vagrant and Ansible.

Outcomes
You should be able to use Ansible to deploy a Vagrant machine as a development environment for a website.

Before you begin
`tower` should have a working Vagrant machine configured with a YUM repository as covered in a previous exercise.

Steps

1. Log in to **`workstation`** and run **`lab ansible-devops-practice setup`**. The lab setup script will install and configure the Git on the repository server and on **`tower`**. It also downloads the playbook needed for this exercise.

```
[student@workstation ~]$ lab ansible-devops-practice setup
```

2. Log in to **`tower`** as the **`root`** user. Change directory to **`/root/vagrant/webapp`** and verify the status of the Vagrant environment.

   2.1. The Vagrant machine from the previous exercise should not be running.

```
[root@tower ~]# cd vagrant/webapp
[root@tower webapp]# vagrant status
Current machine states:

default                   not created (libvirt)

The Libvirt domain is not created. Run `vagrant up` to create it.
```

   2.2. If the Vagrant machine from the previous exercise is running, terminate the instance.

```
[root@tower webapp]# vagrant status
Current machine states:

default                   running (libvirt)

The Libvirt domain is running. To stop this machine, you can run
`vagrant halt`. To destroy the machine, you can run `vagrant destroy`.

[root@tower webapp]# vagrant destroy
==> default: Removing domain...
==> default: Running cleanup tasks for 'shell' provisioner...
```

3. Create a playbook, **`intranet-dev.yml`** by copying it from **`/var/tmp/intranet-dev.yml`**. This playbook will be used with the Vagrant **`ansible`** provisioner to provision the Vagrant machine as a website development environment.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[root@tower webapp]# cp /var/tmp/intranet-dev.yml .
```

4. Review the downloaded playbook to determine the tasks that will be performed. The playbook will ensure that the latest versions of the *httpd* and *firewalld* packages are installed and that **`firewalld`** was configured to allow incoming connections for the http service. It also installs SSH private key and Git configuration files which will then be used to obtain the source code for the intranet website.

```
- name: intranet services
  hosts: all
  become: yes
  become_user: root
  tasks:
  - name: latest httpd version installed
    yum:
      name: httpd
      state: latest
  - name: latest firewalld version installed
    yum:
      name: firewalld
      state: latest
  - name: httpd enabled and running
    service:
      name: httpd
      enabled: true
      state: started
  - name: firewalld enabled and running
    service:
      name: firewalld
      enabled: true
      state: started
  - name: firewalld permits http service
    firewalld:
      service: http
      permanent: true
      state: enabled
    notify:
      - restart firewalld
  - name: create .ssh directory
    file:
      path: /root/.ssh
      state: directory
      mode: 0700
  - name: install private key file
    copy:
      src: /root/.ssh/id_rsa
      dest: /root/.ssh/id_rsa
      mode: 0600
  - name: install git configuration
    copy:
      src: /root/.gitconfig
      dest: /root/.gitconfig
      mode: 0644
  - name: get source
    git:
      repo: ssh://student@workstation/home/student/git/webapp.git
      dest: /var/www/html
      accept_hostkey: yes
  handlers:
    - name: restart firewalld
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    service:
      name: firewalld
      state: restarted
```

5. Modify the Vagrant configuration file to use the **ansible** provisioner to configure the Vagrant machine to host a website development environment.

   5.1. Modify the Vagrant configuration file so that the **ansible** provisioner is used to provision the Vagrant machine using the downloaded playbook, **intranet-dev.yml**.

```
# Define ansible provisioner
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "intranet-dev.yml"
end
```

   5.2. Modify the Vagrant configuration file host settings so port **80** of the Vagrant machine can be accessed on 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
```

6. Install Ansible on the Vagrant host, **tower**, so that it is available for use by Vagrant's **ansible** provisioner.

```
[root@tower webapp]# yum install -y ansible
```

7. Bring the Vagrant machine up. It will use the newly introduced changes.

```
[root@tower 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: ansible...

PLAY [intranet services] *******************************************************

TASK [setup] *******************************************************************
ok: [default]

TASK [latest httpd version installed] ******************************************
changed: [default]

TASK [latest firewalld version installed] **************************************
changed: [default]

TASK [httpd enabled and running] ***********************************************
changed: [default]

TASK [firewalld enabled and running] *******************************************
changed: [default]

TASK [firewalld permits http service] ******************************************
changed: [default]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [install SSH keys] ***************************************************
changed: [default]

TASK [set directory permission] *******************************************
changed: [default]

TASK [set private key file permission] ************************************
changed: [default]

TASK [set pub key file permission] ****************************************
ok: [default]

TASK [get source] *********************************************************
changed: [default]

RUNNING HANDLER [restart firewalld] ***************************************
changed: [default]

PLAY RECAP ****************************************************************
default                    : ok=12   changed=10   unreachable=0    failed=0
```

8. Verify that the web application is working properly on the Vagrant machine using port 8000 on **`localhost`**.

```
[root@tower webapp]# curl http://localhost:8000
Welcome to Web App 1.0
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Implementing Ansible in a DevOps Environment

In this lab, you will modify and publish web content using Vagrant and Ansible.

Outcomes
You should be able modify web content on a development website running on a Vagrant machine and deploy the changes to a production server using Ansible.

Before you begin
The development team you just joined is hosting their intranet development web server on a Vagrant machine. Vagrant uses the **ansible** provisioner in conjunction with the **intranet-dev.yml** file to provision this Vagrant machine.

In addition, to streamline the process of pushing web content to the intranet production web server, **servera**, the team is making use of an **ansible.cfg** configuration file, **inventory** inventory file, and a **intranet-prod.yml** playbook. They are storing everything in a Git repository so that a new team member can just check out the project and have everything they need to work on the development web server and then push code to the production web server.

You have been assigned the task of updating the **/var/www/html/index.html** so that it contains the message "**Welcome to Web App 2.0**".

Before you begin, reset **tower**. Then, log in to **workstation** and run **lab ansible-vagrant-lab setup** to install the Vagrant software, create the Git repository, as well as install and configure the Git software.

```
[student@workstation ~]$ lab ansible-vagrant-lab setup
```

Steps
1. Log in to **tower** as the **root** user and install the Ansible software so it will be available for use by the Vagrant ansible provisioner.

2. Create a working directory for the Vagrant web application environment.

3. Clone the Vagrant intranet development environment from the Git repository on **workstation** using the following command.

   ```
   [root@tower webapp]# git clone student@workstation:/var/git/vagrantwebapp.git .
   ```

   Start the Vagrant machine that hosts the intranet development web server.

   3.1. Clone the Vagrant environment from the **vagrantwebapp.git** Git repo.

   3.2. Start the Vagrant machine.

4. Connect to and verify the contents on the development intranet website from the Vagrant host, **tower**.

   ```
   [root@tower webapp]# curl http://localhost:8000
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
Welcome to Web App 1.0
```

5. Update the development intranet website by modifying the **/var/www/html/index.html** file on the Vagrant machine. Commit the change and push the changes to the Git repository using the following commands executed from **/var/www/html** directory on the Vagrant machine.

```
[root@dev html]# git commit -am 'New web app version'
[root@dev html]# git push origin master
```

Exit the Vagrant machine after the changes are pushed.

6. Connect to and verify the changes made on the development intranet website from the Vagrant host, **tower**.

```
[root@tower webapp]# curl http://localhost:8000
Welcome to Web App 2.0
```

7. Deploy the changes to the production intranet website by executing the playbook, **intranet-prod.yml** using the **ansible.cfg** configuration and **inventory** file.

8. Connect to and verify that the changes made on the development intranet website have been propagated to the production intranet website running on **servera**.

9. Run **lab ansible-vagrant-lab grade** on **workstation** to grade your work.

```
[student@workstation ~]$ lab ansible-vagrant-lab grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this lab, you will modify and publish web content using Vagrant and Ansible.

Outcomes
You should be able modify web content on a development website running on a Vagrant machine and deploy the changes to a production server using Ansible.

Before you begin
The development team you just joined is hosting their intranet development web server on a Vagrant machine. Vagrant uses the **ansible** provisioner in conjunction with the **intranet-dev.yml** file to provision this Vagrant machine.

In addition, to streamline the process of pushing web content to the intranet production web server, **servera**, the team is making use of an **ansible.cfg** configuration file, **inventory** inventory file, and a **intranet-prod.yml** playbook. They are storing everything in a Git repository so that a new team member can just check out the project and have everything they need to work on the development web server and then push code to the production web server.

You have been assigned the task of updating the **/var/www/html/index.html** so that it contains the message "**Welcome to Web App 2.0**".

Before you begin, reset **tower**. Then, log in to **workstation** and run **lab ansible-vagrant-lab setup** to install the Vagrant software, create the Git repository, as well as install and configure the Git software.

```
[student@workstation ~]$ lab ansible-vagrant-lab setup
```

Steps
1. Log in to **tower** as the **root** user and install the Ansible software so it will be available for use by the Vagrant ansible provisioner.

```
[root@tower ~]# yum install -y ansible
Loaded plugins: langpacks, search-disabled-repos
Resolving Dependencies
--> Running transaction check
...output omitted
Dependency Installed:
  libgnome-keyring.x86_64 0:3.8.0-3.el7       perl-Error.noarch 1:0.17020-2.el7
    perl-Git.noarch 0:1.8.3.1-5.el7
  perl-TermReadKey.x86_64 0:2.30-20.el7       python-crypto.x86_64 0:2.6-4.el7
    python-ecdsa.noarch 0:0.11-3.el7ost
  python-httplib2.noarch 0:0.7.7-3.el7        python-keyczar.noarch 0:0.71c-2.el7
    python-paramiko.noarch 0:1.15.1-1.el7
  python-pyasn1.noarch 0:0.1.6-2.el7          sshpass.x86_64 0:1.05-1.el7.rf

Complete!
```

2. Create a working directory for the Vagrant web application environment.

```
[root@tower ~]# mkdir -p vagrant/webapp
[root@tower ~]# cd vagrant/webapp
```

3. Clone the Vagrant intranet development environment from the Git repository on **workstation** using the following command.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[root@tower webapp]# git clone student@workstation:/var/git/vagrantwebapp.git .
```

Start the Vagrant machine that hosts the intranet development web server.

3.1. Clone the Vagrant environment from the **vagrantwebapp.git** Git repo.

```
[root@tower webapp]# git clone student@workstation:/var/git/vagrantwebapp.git .
Cloning into 'vagrantwebapp'...
The authenticity of host 'workstation (172.25.250.254)' can't be established.
ECDSA key fingerprint is 84:fc:5e:82:a8:4f:bb:f3:c0:06:61:77:ad:a5:e5:b9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'workstation,172.25.250.254' (ECDSA) to the list of
 known hosts.
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 16 (delta 1), reused 16 (delta 1)
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (1/1), done.
```

3.2. Start the Vagrant machine.

```
[root@tower webapp]# 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
RUNNING HANDLER [restart firewalld] ********************************************
changed: [default]

PLAY RECAP *********************************************************************
default                    : ok=12   changed=10   unreachable=0    failed=0
```

4. Connect to and verify the contents on the development intranet website from the Vagrant host, **tower**.

```
[root@tower webapp]# curl http://localhost:8000
Welcome to Web App 1.0
```

5. Update the development intranet website by modifying the **/var/www/html/index.html** file on the Vagrant machine. Commit the change and push the changes to the Git repository using the following commands executed from **/var/www/html** directory on the Vagrant machine.

```
[root@dev html]# git commit -am 'New web app version'
[root@dev html]# git push origin master
```

Exit the Vagrant machine after the changes are pushed.

5.1. Log in to the Vagrant machine.

```
[root@tower webapp]# vagrant ssh
Last login: Fri Apr 29 10:30:04 2016 from 192.168.121.1
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

5.2. Become the **root** user.

```
[vagrant@dev ~]$ sudo -i
```

5.3. Change to the development website's document root directory.

```
[root@dev ~]# cd /var/www/html
```

5.4. Modify the **index.html** file so it contains the message "**Welcome to Web App 2.0**".

```
Welcome to Web App 2.0
```

5.5. Commit and push the change to the Git repository.

```
[root@dev html]# git commit -am 'New web app version'
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@dev html]# git push origin master
Counting objects: 5, done.
Writing objects: 100% (3/3), 262 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://student@workstation/var/git/webapp.git
   1d0ad93..6f75595  master -> master
```

5.6. Exit the Vagrant machine.

```
[root@dev html]# exit
logout
[vagrant@dev ~]$ exit
logout
Connection to 192.168.121.56 closed.
```

6. Connect to and verify the changes made on the development intranet website from the Vagrant host, **tower**.

```
[root@tower webapp]# curl http://localhost:8000
Welcome to Web App 2.0
```

7. Deploy the changes to the production intranet website by executing the playbook, **intranet-prod.yml** using the **ansible.cfg** configuration and **inventory** file.

```
[root@tower webapp]# cat ansible.cfg
[defaults]
inventory = inventory
host_key_checking = False
[root@tower webapp]# cat inventory
servera.lab.example.com
[root@tower webapp]# ansible-playbook intranet-prod.yml

PLAY [internet services] *******************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
TASK [latest httpd version installed] ***************************************
ok: [servera.lab.example.com]

TASK [latest firewalld version installed] ***********************************
ok: [servera.lab.example.com]

TASK [httpd enabled and running] ********************************************
ok: [servera.lab.example.com]

TASK [firewalld enabled and running] ****************************************
ok: [servera.lab.example.com]

TASK [firewalld permits http service] ***************************************
ok: [servera.lab.example.com]

TASK [get source] ***********************************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [restart firewalld] *****************************************
changed: [servera.lab.example.com]

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=8    changed=1    unreachable=0    failed=0
```

8. Connect to and verify that the changes made on the development intranet website have been propagated to the production intranet website running on **servera**.

```
[root@tower webapp]# curl http://servera
Welcome to Web App 2.0
```

9. Run **lab ansible-vagrant-lab grade** on **workstation** to grade your work.

```
[student@workstation ~]$ lab ansible-vagrant-lab grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

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 **`ansible`** provisioner automates the provisioning of a Vagrant machine through execution of Ansible on the Vagrant host.
- Vagrant's **`ansible_local`** provisioner automates the provisioning of a Vagrant machine through execution of Ansible on the Vagrant machine.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.



CHAPTER 13

COMPREHENSIVE REVIEW: AUTOMATION WITH ANSIBLE

Overview	
Goal	Review tasks from *Automation with Ansible*
Objectives	• Review tasks from *Automation with Ansible*
Sections	• Comprehensive Review
Lab	• Lab: Deploying Ansible • Lab: Deploying Ansible Tower and Executing Jobs • Lab: Creating Roles and using Dynamic Inventory • Lab: Optimizing Ansible

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Comprehensive Review

Objectives

After completing this section, students should be able to review and refresh knowledge and skills learned in *Automation with Ansible* .

Reviewing *Automation with Ansible*

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Students can refer to earlier sections in the textbook for extra study.

Chapter 1, Introducing Ansible

Describe the terminology and architecture of Ansible.

- Describe Ansible concepts, reference architecture, and use cases.
- Describe Ansible deployments and orchestration methods.
- Describe Ansible inventory concepts.

Chapter 2, Deploying Ansible

Install Ansible and run ad hoc commands.

- Install Ansible.
- Manage Ansible configuration files.
- Run Ansible ad hoc commands.
- Manage dynamic inventory.

Chapter 3, Implementing Playbooks

Write Ansible plays and execute a playbook.

- Write YAML files.
- Implement Ansible playbooks.
- Write and execute a playbook.

Chapter 4, Managing Variables and Inclusions

To describe variable scope and precedence, manage variables and facts in a play, and manage inclusions.

- Manage variables in Ansible projects
- Manage facts in playbooks
- Include variables and tasks from external files into a playbook

Chapter 5, Implementing Task Control

Manage task control, handlers, and tags in Ansible playbooks

- Construct conditionals and loops in a playbook
- Implement handlers in a playbook

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Implement tags in a playbook
- Resolve errors in a playbook

Chapter 6, Implementing Jinja2 Templates
Implement a Jinja2 template
- Describe Jinja2 templates
- Implement Jinja2 templates

Chapter 7, Implementing Roles
Create and manage roles
- Describe the structure and behavior of a role
- Create a role
- Deploy roles with Ansible Galaxy

Chapter 8, Optimizing Ansible
Configure connection types, delegations, and parallelism
- Configure a connection type and an environment in a playbook
- Configure delegation in a playbook
- Configure parallelism in Ansible

Chapter 9, Implementing Ansible Vault
Manage encryption with Ansible Vault.
- Create, edit, rekey, encrypt, and decrypt files.
- Run a playbook with Ansible Vault.



Important

The material on Ansible Vault is not revisited in this chapter. Students wishing to review it, especially in preparation for the EX407 certification exam, should look at the exercises in *Chapter 9, Implementing Ansible Vault*.

Chapter 10, Troubleshooting Ansible
Troubleshoot playbooks and managed hosts.
- Troubleshoot playbooks
- Troubleshoot managed hosts

Chapter 11, Implementing Ansible Tower
Implement Ansible Tower
- Describe Ansible Tower architecture and features
- Deploy Ansible Tower
- Configure users and privileges in Ansible Tower
- Manage hosts with Ansible Tower

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

- Manage jobs in Ansible Tower

Chapter 12, Implementing Ansible in a DevOps Environment

Implement Ansible in a DevOps environment using Vagrant.

- Describe Ansible in a DevOps environment and provision Vagrant machines.
- Deploy Vagrant in a DevOps environment.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Deploying Ansible

In this review, you will install Ansible on **workstation** and use it as a control node and configure it for connections to the managed hosts **servera** and **serverb**. Use ad hoc commands to perform actions on the managed hosts.

Outcomes
You should be able to:
- Install Ansible.
- Use ad hoc commands to perform actions on managed hosts.

Before you begin
Save any work that you want to keep on your machines. When you have saved any data you want to keep, reset all of the virtual machines.

If you did not reset all of your virtual machines at the end of the last lab, do so now. This may take a few minutes.

Log in as the **student** user on **workstation** and run **lab ansible-deploy-cr setup**. This script ensures that the managed hosts, **servera** and **serverb**, are reachable on the network. The script creates a directory structure for the lab in the student's home directory.

```
[student@workstation ~]$ lab ansible-deploy-cr setup
```

Instructions
Install and configure Ansible on **workstation**, and ensure that you meet the following criteria. Demonstrate that you can construct the ad hoc commands specified on the list of criteria in order to modify the managed hosts and verify that the modifications worked.

- Install Ansible on **workstation** so that it can serve as the control node.
- On the control node, create an inventory file, **/home/student/ansible-deploy-cr/inventory/hosts**, which contains a group called **dev**. This group should consist of the managed hosts **servera.lab.example.com** and **serverb.lab.example.com**.
- Create the Ansible configuration file in **/home/student/ansible-deploy-cr/ansible.cfg**. The configuration file should point to the inventory file created in **/home/student/ansible-deploy-cr/inventory**.
- Execute an ad hoc command using privilege escalation to modify the contents of the **/etc/motd** file on **servera** and **serverb** so that it contains the string **Managed by Ansible\n**. Use **devops** as the remote user.
- Execute an ad hoc command to verify that the contents of the **/etc/motd** file on **servera** and **serverb** are identical.

Evaluation
From **workstation**, run the **lab ansible-deploy-cr** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab ansible-deploy-cr grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this review, you will install Ansible on **workstation** and use it as a control node and configure it for connections to the managed hosts **servera** and **serverb**. Use ad hoc commands to perform actions on the managed hosts.

Outcomes
You should be able to:

- Install Ansible.
- Use ad hoc commands to perform actions on managed hosts.

Before you begin
Save any work that you want to keep on your machines. When you have saved any data you want to keep, reset all of the virtual machines.

If you did not reset all of your virtual machines at the end of the last lab, do so now. This may take a few minutes.

Log in as the **student** user on **workstation** and run **lab ansible-deploy-cr setup**. This script ensures that the managed hosts, **servera** and **serverb**, are reachable on the network. The script creates a directory structure for the lab in the student's home directory.

```
[student@workstation ~]$ lab ansible-deploy-cr setup
```

Instructions
Install and configure Ansible on **workstation**, and ensure that you meet the following criteria. Demonstrate that you can construct the ad hoc commands specified on the list of criteria in order to modify the managed hosts and verify that the modifications worked.

- Install Ansible on **workstation** so that it can serve as the control node.
- On the control node, create an inventory file, **/home/student/ansible-deploy-cr/inventory/hosts**, which contains a group called **dev**. This group should consist of the managed hosts **servera.lab.example.com** and **serverb.lab.example.com**.
- Create the Ansible configuration file in **/home/student/ansible-deploy-cr/ansible.cfg**. The configuration file should point to the inventory file created in **/home/student/ansible-deploy-cr/inventory**.
- Execute an ad hoc command using privilege escalation to modify the contents of the **/etc/motd** file on **servera** and **serverb** so that it contains the string **Managed by Ansible\n**. Use **devops** as the remote user.
- Execute an ad hoc command to verify that the contents of the **/etc/motd** file on **servera** and **serverb** are identical.

Steps
1. Install Ansible on **workstation** so that it can serve the control node.

```
[student@workstation ~]$ sudo yum -y install ansible
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for student: student
```

2. On the control node, create an inventory file, **/home/student/ansible-deploy-cr/inventory/hosts**, which contains a group called **dev**. This group should consist of the managed hosts **servera.lab.example.com** and **serverb.lab.example.com**.

   2.1. Use the **vim** text editor to create and edit the inventory file **/home/student/ansible-deploy-cr/inventory/hosts**.

   ```
   [student@workstation ~]$ cd /home/student/ansible-deploy-cr/inventory/
   [student@workstation inventory]$ vim hosts
   ```

   2.2. Add the following entries to the file to create the **dev** host group and its group members as **servera.lab.example.com** and **serverb.lab.example.com**. Save the changes and exit the text editor.

   ```
   [dev]
   servera.lab.example.com
   serverb.lab.example.com
   ```

3. Create the Ansible configuration file in **/home/student/ansible-deploy-cr/ansible.cfg**. The configuration file should point to the inventory file created in **/home/student/ansible-deploy-cr/inventory**.

   3.1. Use the **vim** text editor to create and edit the ansible configuration file **/home/student/ansible-deploy-cr/ansible.cfg**.

   ```
   [student@workstation inventory]$ cd ..
   [student@workstation ansible-deploy-cr]$ vim ansible.cfg
   ```

   3.2. Add the following entries to the file to point to the inventory directory **/home/student/ansible-deploy-cr/inventory**. Save the changes and exit the text editor.

   ```
   [defaults]
   inventory=/home/student/ansible-deploy-cr/inventory
   ```

4. Execute an ad hoc command using privilege escalation to modify the contents of the **/etc/motd** file on **servera** and **serverb** so that it contains the string **Managed by Ansible \n**. Use **devops** as the remote user.

   4.1. From the directory **/home/student/ansible-deploy-cr**, execute an ad hoc command using privilege escalation to modify the contents of the **/etc/motd** file on **servera** and **serverb**, so that it contains the string **Managed by Ansible\n**. Use **devops** as the remote user.

   ```
   [student@workstation ansible-deploy-cr]$ ansible dev -m copy -a
    'content="Managed by Ansible\n" dest=/etc/motd' -b -u devops
   ```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
serverb.lab.example.com | SUCCESS => {
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1463700139.62-268323083587449/
source",
    "state": "file",
    "uid": 0
}
servera.lab.example.com | SUCCESS => {
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1463700139.63-258952343613886/
source",
    "state": "file",
    "uid": 0
}
```

5. Execute an ad hoc command to verify that the contents of the **/etc/motd** file on **servera** and **serverb** are identical.

```
[student@workstation ansible-deploy-cr]$ ansible dev -m command -a "cat /etc/motd"
servera.lab.example.com | SUCCESS | rc=0 >>
Managed by Ansible

serverb.lab.example.com | SUCCESS | rc=0 >>
Managed by Ansible
```

Evaluation

From **workstation**, run the **lab ansible-deploy-cr** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-deploy-cr grade
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Deploying Ansible Tower and Executing Jobs

In this review, you will deploy Ansible Tower on **tower.lab.example.com** as a primary server and use it to execute playbook on **servera.lab.example.com** and **serverb.lab.example.com**.

Outcomes
You should be able to:

- Deploy Ansible Tower.
- Configure Ansible Tower users and permissions.
- Add managed hosts to an Ansible Tower inventory and configure their access credentials.
- Create Ansible Tower job templates and use them to manage jobs.

Before you begin
Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-tower-cr setup** script. This script ensures that the tower host, **tower.lab.example.com** is reachable on the network. The script also checks that Ansible is installed on **workstation** and creates a directory structure for the lab environment. The script copies the **Ansible Tower** setup bundle under the lab's environment directory.

```
[student@workstation ~]$ lab ansible-tower-cr setup
```

Instructions
Install Ansible Tower on **tower.lab.example.com** and configure it according to the following criteria. You should read through the entire list before you begin.

- Install Ansible Tower on **tower.lab.example.com**. A setup bundle, **ansible-tower-setup-bundle-1.el7.tar.gz**, is available in the directory, **/home/student/ansible-tower-cr**, on **workstation**. The directory is accessible by logging in as the **student** user with the password **student**. A valid license file, **Ansible-Tower-license.txt**, is also located in that directory.

  Firefox is available on **workstation** so that you can access Ansible Tower's web-based user interface when it has been installed.

  The following parameters should be set when creating the **tower_setup_conf.yml** file for installing Ansible Tower. Note that this will set the **admin** password for your Ansible Tower configuration to **redhat**.

Ansible Tower configuration

Parameters	Values
hostname	**tower.lab.example.com**
database	internal
Ansible Tower admin password	**redhat**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Parameters	Values
Munin password	**redhat**
SSH user	**devops**
Root access	**sudo**
sudo password	no
SSH keys	yes
SSH key location	**/home/student/.ssh/lab_rsa**



Important

In the version of Ansible Tower used by this course, the **inventory** file that is created in your unpacked setup bundle directory may need to be edited before you can run the playbook created for installation. If the inventory contains the directive **ansible_sudo=True**, that directive will need to be replaced with **ansible_become=True**. The resulting inventory file should read:

```
[primary]
tower.lab.example.com

[all:vars]
ansible_ssh_user=devops
ansible_become=True
```

- Create a new user in Ansible Tower, **devops**, with the following details.

Field	Value
First Name	Ansible
Last Name	Operator
Email	devops@lab.example.com
Organization	Default
User	**devops**
Password	**redhat123**

- Create a new inventory named **dev** that belongs to the **Default** organization. It should include a machine group named **devservers** which has the managed host **serverb.lab.example.com** as a member. Create a permission type for the **devops** user that grants read permission on the inventory as specified in the following table.

Field	Value
Permission Type	Inventory
Name	Inventory Privilege
Inventory	**dev**
Permission	Read Inventory

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Field	Value
Execute Commands on the Inventory	selected

- Configure Ansible Tower with machine credentials named **devops credentials** and owned by the **devops** user, which can be used by that user to access the managed hosts. Use the remote **devops** username for access, and leave the password blank but use the private key in **/home/student/.ssh/lab_rsa** for authentication. Configure privilege escalation using **sudo** to switch to the user name **root**. Leave the privilege escalation password blank.

- Create a directory on **tower.lab.example.com** that will contain a playbook for a Tower project. The directory should be named **/var/lib/awx/projects/managing-jobs**. It should contain a playbook named **screeninstall.yml** that consists of the following content:

```
---
 - name: Install screen on devservers
   hosts: all
   tasks:

   - name: Install screen
     yum:
       name: screen
       state: latest
```

  The directory and playbook should be owned by user **awx**.

  You can log in to **tower.lab.example.com** as user **devops** using **ssh** from the **student** account on **workstation**. The **devops** user on **tower.lab.example.com** has **sudo** access to **root**.

- Create a new project, **Screen Install**, that has an SCM Type of Manual and a Playbook Directory of **managing-jobs**.

- Create a permission type for the **devops** user that allows them to create a job template using the **dev** inventory for the **Screen Install** project.

- Using the configuration above, log in to Ansible Tower as the **devops** user and create a job template named **Screen install on devservers**, which uses the **Screen Install** project to run the **screeninstall.yml** playbook on the **dev** inventory. Specify machine credentials as **devops credentials** and enable privilege escalation.

Evaluation

Run a job using the **Screen install on devservers** job template, by clicking the rocket icon under the Actions column in the **Screen install on devservers** job template row.

Initially, after refreshing the page, the Status should show Running with a flashing green dot. Continue to refresh the Status page to see Status update for the tasks and plays. Green indicates Ansible returned OK and yellow indicates changes were made on the managed hosts.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this review, you will deploy Ansible Tower on **tower.lab.example.com** as a primary server and use it to execute playbook on **servera.lab.example.com** and **serverb.lab.example.com**.

Outcomes

You should be able to:

- Deploy Ansible Tower.
- Configure Ansible Tower users and permissions.
- Add managed hosts to an Ansible Tower inventory and configure their access credentials.
- Create Ansible Tower job templates and use them to manage jobs.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-tower-cr setup** script. This script ensures that the tower host, **tower.lab.example.com** is reachable on the network. The script also checks that Ansible is installed on **workstation** and creates a directory structure for the lab environment. The script copies the **Ansible Tower** setup bundle under the lab's environment directory.

```
[student@workstation ~]$ lab ansible-tower-cr setup
```

Instructions

Install Ansible Tower on **tower.lab.example.com** and configure it according to the following criteria. You should read through the entire list before you begin.

- Install Ansible Tower on **tower.lab.example.com**. A setup bundle, **ansible-tower-setup-bundle-1.el7.tar.gz**, is available in the directory, **/home/student/ansible-tower-cr**, on **workstation**. The directory is accessible by logging in as the **student** user with the password **student**. A valid license file, **Ansible-Tower-license.txt**, is also located in that directory.

  Firefox is available on **workstation** so that you can access Ansible Tower's web-based user interface when it has been installed.

  The following parameters should be set when creating the **tower_setup_conf.yml** file for installing Ansible Tower. Note that this will set the **admin** password for your Ansible Tower configuration to **redhat**.

Ansible Tower configuration

Parameters	Values
hostname	**tower.lab.example.com**
database	internal
Ansible Tower admin password	**redhat**
Munin password	**redhat**
SSH user	**devops**
Root access	**sudo**

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Parameters	Values
sudo password	no
SSH keys	yes
SSH key location	**/home/student/.ssh/lab_rsa**



Important

In the version of Ansible Tower used by this course, the **inventory** file that is created in your unpacked setup bundle directory may need to be edited before you can run the playbook created for installation. If the inventory contains the directive **ansible_sudo=True**, that directive will need to be replaced with **ansible_become=True**. The resulting inventory file should read:

```
[primary]
tower.lab.example.com

[all:vars]
ansible_ssh_user=devops
ansible_become=True
```

- Create a new user in Ansible Tower, **devops**, with the following details.

Field	Value
First Name	Ansible
Last Name	Operator
Email	devops@lab.example.com
Organization	Default
User	**devops**
Password	**redhat123**

- Create a new inventory named **dev** that belongs to the **Default** organization. It should include a machine group named **devservers** which has the managed host **serverb.lab.example.com** as a member. Create a permission type for the **devops** user that grants read permission on the inventory as specified in the following table.

Field	Value
Permission Type	Inventory
Name	Inventory Privilege
Inventory	**dev**
Permission	Read Inventory
Execute Commands on the Inventory	selected

- Configure Ansible Tower with machine credentials named **devops credentials** and owned by the **devops** user, which can be used by that user to access the managed hosts. Use the remote **devops** username for access, and leave the password blank but use the private key

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

in **/home/student/.ssh/lab_rsa** for authentication. Configure privilege escalation using **sudo** to switch to the user name **root**. Leave the privilege escalation password blank.

- Create a directory on **tower.lab.example.com** that will contain a playbook for a Tower project. The directory should be named **/var/lib/awx/projects/managing-jobs**. It should contain a playbook named **screeninstall.yml** that consists of the following content:

```
---
 - name: Install screen on devservers
   hosts: all
   tasks:

   - name: Install screen
     yum:
       name: screen
       state: latest
```

The directory and playbook should be owned by user **awx**.

You can log in to **tower.lab.example.com** as user **devops** using **ssh** from the **student** account on **workstation**. The **devops** user on **tower.lab.example.com** has **sudo** access to **root**.

- Create a new project, **Screen Install**, that has an SCM Type of Manual and a Playbook Directory of **managing-jobs**.
- Create a permission type for the **devops** user that allows them to create a job template using the **dev** inventory for the **Screen Install** project.
- Using the configuration above, log in to Ansible Tower as the **devops** user and create a job template named **Screen install on devservers**, which uses the **Screen Install** project to run the **screeninstall.yml** playbook on the **dev** inventory. Specify machine credentials as **devops credentials** and enable privilege escalation.

Steps

1. From **workstation**, as the **student** user, change to the directory **/home/student/ansible-tower-cr**.

```
[student@workstation ~]$ cd /home/student/ansible-tower-cr/
```

2. Unpack the setup bundle **ansible-tower-setup-bundle-1.el7.tar.gz** present in the directory **/home/student/ansible-tower-cr**.

```
[student@workstation ansible-tower-cr]$ ls
Ansible-Tower-license.txt ansible-tower-setup-bundle-1.el7.tar.gz
[student@workstation ansible-tower-cr]$ tar zxvf ansible-tower-setup-
bundle-1.el7.tar.gz
```

3. Run the **configure** command from the directory **/home/student/ansible-tower-cr/ansible-tower-setup-bundle-2.4.5-1.el7** to create the **tower_setup_conf.yml** file using the following parameters:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Ansible Tower configuration

Parameters	Values
hostname	**tower.lab.example.com**
database	internal
Ansible Tower admin password	**redhat**
Munin password	**redhat**
SSH user	**devops**
Root access	**sudo**
sudo password	no
SSH keys	yes
SSH key location	**/home/student/.ssh/lab_rsa**

3.1. Change to the **ansible-tower-setup-bundle-2.4.5-1.el7** directory created after unpacking the Tower installation tar file.

```
[student@workstation ansible-tower-cr]$ cd ansible-tower-setup-
bundle-2.4.5-1.el7/
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$
```

3.2. Execute the **configure** command under the **ansible-tower-setup-bundle-2.4.5-1.el7** directory, to create the **tower_setup_conf.yml** configuration playbook.

```
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$ ./configure
```

3.3. Because **tower.lab.example.com** is the primary server for this Ansible Tower deployment, use **tower.lab.example.com** as the host name to configure Ansible Tower.

```
-----------------------------------------
Welcome to the Ansible Tower Install Wizard
-----------------------------------------
...output omitted...
Enter the hostname or IP to configure Ansible Tower
(default: localhost): tower.lab.example.com
```

3.4. Specify the type of database to be used for Ansible Tower. Enter **i** to use an *internal* database on the **tower** host.

```
DATABASE
...output omitted...
Will this installation use an (i)nternal or (e)xternal database? i
```

3.5. When prompted, enter **redhat** for both the Ansible Tower admin user password and for the Munin password. Munin is a built-in monitoring system for monitoring the *Tower* server.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
PASSWORDS
For security reasons, since this is a new install, you must specify the
following application passwords.

Enter the desired Ansible Tower admin user password: redhat
Enter the desired Munin password: redhat
```

3.6. Enter the user information for connecting to **tower.lab.example.com** using SSH. In this deployment we use the **devops** user to connect to **tower.lab.example.com** and use **sudo** as the method for privilege escalation. No password is required for *sudo* access.

```
CONNECTION INFORMATION
Enter the SSH user to connect with (default: root): devops

Root access is required to install Tower.
Will you use (1) sudo or (2) su? 1

Will sudo require a password (y/N)? N
```

3.7. For key based SSH authentication, enter **Y** when prompted and specify the path of the private SSH key as **/home/student/.ssh/lab_rsa**.

```
Will you be using SSH keys (Y/n)? Y

Please specify the path to the SSH private key: /home/student/.ssh/lab_rsa
```

3.8. Enter **y** to confirm the settings provided are correct.

```
REVIEW
You selected the following options:

The primary Tower machine is: tower.lab.example.com
Tower will operate on an INTERNAL database.
Using SSH user: devops

Are these settings correct (y/n)? y
Settings saved to /home/student/ansible-tower-cr/ansible-tower-setup-
bundle-2.4.5-1.el7/tower_setup_conf.yml.

FINISHED!
You have completed the setup wizard. You may execute the installation of
Ansible Tower by issuing the following command:

# Add your SSH key to SSH agent.
# You may be asked to enter your SSH unlock key password to do this.
ssh-agent bash
ssh-add /home/student/.ssh/lab_rsa
./setup.sh
```

3.9. If necessary, edit the **inventory** file created under the **ansible-tower-setup-bundle-2.4.5-1.el7** directory to replace **ansible_sudo=True** with **ansible_become=True**. The inventory file should look like the following example.

```
[primary]
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
tower.lab.example.com

[all:vars]
ansible_ssh_user=devops
ansible_become=True
```

4. Run the setup playbook with **sudo** using the file **setup.sh** in the directory **~/ansible-tower-cr/ansible-tower-setup-bundle-2.4.5-1.el7**.

```
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$ pwd
/home/student/ansible-tower-cr/ansible-tower-setup-bundle-2.4.5-1.el7
[student@workstation ansible-tower-setup-bundle-2.4.5-1.el7]$ sudo ./setup.sh
...output omitted...
PLAY RECAP *********************************************************************
tower.lab.example.com      : ok=118  changed=67   unreachable=0    failed=0

The setup process completed successfully.
[warn] /var/log/tower does not exist.
Setup log saved to setup.log.
```

5. From **workstation**, use **Firefox** to open the Ansible Tower web page at **https://tower.lab.example.com**. Log in as the **admin** the user with the password **redhat**.

6. Update the license from the license text file **~/ansible-tower-cr/Ansible-Tower-license.txt**.

   In the **Update License** page, copy and paste the license from the license text file **~/ansible-tower-cr/Ansible-Tower-license.txt**.

   In the **License File** window, accept the **End User License Agreement** by selecting the checkbox and click **Submit**. You should see a **License Accepted** window confirming the action. Click **OK** to continue.

7. Create a new user with the following details:

Field	Value
First Name	Ansible
Last Name	Operator
Email	devops@lab.example.com
Organization	Default
User	**devops**
Password	**redhat123**

   7.1. Open the **Setup** page by clicking the wrench and screwdriver icon in the upper right.

   7.2. Click the **Users** link in the middle left of the page.

   7.3. Click the **+** icon button in the middle right side of the page to create a new user.

   7.4. On the screen, fill in the details as specified in the table at the start of this step.

   Click Save to create the **devops** user.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

8. Create a new inventory with the following parameters:

Field	Value
Name	**dev**
Description	development
Organization	Default

8.1. Click the **Inventories** link at the top of the page. Then click **+** icon on the upper right to open the **Create Inventory** wizard.

Create an inventory using the parameters specified in the table at the start of this step.

8.2. Click **Save** to save the **dev** inventory in Ansible Tower.

9. Add a group named **devservers** to the **dev** inventory.

Edit the **dev** inventory by clicking the **dev** link. On top of the **Groups** section, click **+** sign to add the group named **devservers**. Click **Save** to save the group details.

10. Add **serverb.lab.example.com** as a managed host to the **devservers** group.

Open the **devservers** group by clicking the **devservers** link. Click the **+** icon that corresponds to the hosts in the group, in the middle right hand side of the page. In the **Host Properties** window enter the **Host Name** as **serverb.lab.example.com**, ensure the **Enabled** check box is selected.

Click **Save** to save the host details.

11. Create machine credentials for Ansible Tower to use to connect to the managed nodes. Use the following information to create the credentials:

Field	Values
Name	**devops credentials**
Description	Devops user credentials for dev
Does this belong to a team or user	User
User that owns the credential	**devops**
Type	Machine
Username	**devops**
Password	Leave it blank
Private Key	Paste the content from **/home/student/.ssh/lab_rsa**
Privilege Escalation	Sudo
Privilege Escalation Username	**root**
Privilege Escalation Password	Leave it blank

Click the wrench and screwdriver in the top right of the page to open the setup page. Click the **Credentials** link, and then the **+** button to open the **Create Credential** page.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Click **Save** to save the credentials.

12. On **tower.lab.example.com**, create a directory named **managing-jobs** under **/var/lib/awx/projects/**.

```
[student@workstation ~]$ ssh devops@tower.lab.example.com
Last login: Wed May 11 18:42:06 2016 from workstation.lab.example.com
[devops@tower ~]$ sudo mkdir /var/lib/awx/projects/managing-jobs/
```

13. Create a playbook named **screeninstall.yml** in the **/var/lib/awx/projects/managing-jobs/** directory. Add the following content to the file **screeninstall.yml**:

```
---
- name: Install screen on devservers
  hosts: all
  tasks:

  - name: Install screen
    yum:
      name: screen
      state: latest
```

14. Change the permissions to ensure that the playbook directory **/var/lib/awx/projects/managing-jobs** and file **screeninstall.yml** are owned by the **awx** user.

```
[devops@tower ~]$ sudo chown -R awx: /var/lib/awx/projects/managing-jobs/
```

15. Create a new project named **Screen Install**. Enter the following information to create the new project:

Field	Value
Name	**Screen Install**
SCM Type	Manual
Playbook Directory	**managing-jobs**

Click the **Projects** link at the top of the page. Click **+** on the upper right side of the page to open the **Create Project** page. Specify the name of the project as **Screen Install**, SCM type as **manual** and playbook directory as **managing-jobs**.

Click **Save** to save the project details.

16. Create a new permission type for the **devops** user in the **dev** inventory, with the following details:

Field	Value
Permission Type	Inventory
Name	Inventory Privilege
Inventory	**dev**
Permission	Read Inventory

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Field	Value
Execute Commands on the Inventory	selected

Go to the setup page by clicking the wrench and screwdriver in the upper right of the page. Click **Users** to open the users page. Edit the **devops** user by clicking the pencil icon under **Actions** in **devops** row. Add permissions in the **Permissions** page by clicking the **+** icon.

Click **Save**.

17. Create a new permission type for the **devops** user to create job templates and execute them for **`Screen Install`** project, with the following details:

Field	Value
Permission Type	Job Template
Name	Job Template Privilege
Project	**`Screen Install`**
Inventory	**`dev`**
Permission	Create a Job Template

Edit the **devops** user by clicking the pencil icon under **Actions** in the devops row. Add permissions in the **Permissions** page by clicking the **+** icon.

Click **Save**.

18. Log out as the **`admin`** user from the Ansible Tower portal and log in as **devops** using the password **`redhat123`**.

Verify the permissions of the **devops** user by adding a new job template. Click the **Job Templates** page and then the **+** icon button on the upper right corner. Enter the following details in the fields of **Create Job Templates** page.

Field	Value
Name	**`Screen install on devservers`**
Job Type	Run
Inventory	**`dev`**
Project	**`Screen Install`**
Playbook	**`screeninstall.yml`**
Machine Credentials	**`devops credentials`**
Enable Privilege Escalation	checked

Click **Save** to save the job template details.

Evaluation

Run a job using the **`Screen install on devservers`** job template, by clicking the rocket icon under the **Actions** column in the **`Screen install on devservers`** job template row.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Initially, after refreshing the page, the **Status** should show **Running** with a flashing green dot. Continue to refresh the **Status** page to see **Status** update for the tasks and plays. Green indicates Ansible returned **OK** and yellow indicates changes were made on the managed hosts.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Creating Roles and using Dynamic Inventory

In this review, you will create a role to configure a *vsftp* server on **serverb.lab.example.com** and verify it by installing an **FTP** client on **servera.lab.example.com** and then connecting to the **FTP** server on **serverb.lab.example.com** as an **anonymous** user to list and download the content.

Outcomes
You should be able to:

- Create a role for configuring vsftpd.
- Use a Jinja2 template for vsftpd server configuration.
- Create and execute a playbook by including the role.
- Use dynamic inventory to execute a playbook.

Before you begin
Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-roles-cr setup** script. This script ensures that the host, **serverb.lab.example.com**, is reachable on the network. The script also checks that Ansible is installed on **workstation** and creates a directory structure for the lab environment and required Ansible configuration files.

```
[student@workstation ~]$ lab ansible-roles-cr setup
```

Instructions
Create a role named **ansible-vsftpd** that will be used with a playbook, **vsftpd-configure.yml**. This playbook will install and configure an FTP server on members of host group **ftpservers**. A second playbook, **ftpclient.yml**, will be needed to install *lftp* on members of host group **ftpclients**. Members of that host group will be determined using a dynamic inventory script.

The **ansible-vsftpd** role should be configured based on the following criteria:

- In the **student** account on **workstation**, under the directory **/home/student/ansible-roles-cr**, create the directory structure for a role named **ansible-vsftpd**.
- Copy the provided **/home/student/ansible-roles-cr/vsftpd.conf.j2** template into the **~/ansible-roles-cr/roles/ansible-vsftpd/templates** directory.
- Define the following variables in **roles/ansible-vsftpd/vars/main.yml**:

Variable	Value
vsftpd_packages	vsftpd
vsftpd_service	vsftpd

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Variable	Value
vsftpd_config_file	/etc/vsftpd/vsftpd.conf

- Define the following default variables in **`roles/ansible-vsftpd/defaults/main.yaml`**:

Variable	Value
vsftpd_anonymous_enable	false
vsftpd_connect_from_port_20	true
vsftpd_listen	true
vsftpd_local_enable	true
vsftpd_setype	public_content_t
vsftpd_syslog_enable	true
vsftpd_write_enable	false
vsftpd_chroot_local_user	true

- Define a handler in **`roles/ansible-vsftpd/handlers/main.yml`** named **`restart vsftpd`** to restart the **`vsftpd`** service, using the variable **`vsftpd_service`** as the name of the service.

- Define a role in **`roles/ansible-vsftpd/tasks/main.yml`** that does four things. (This may require more than four tasks to be defined.)

  1. Ensure all packages needed by **`vsftpd`** are installed by using a loop to iterate over all items defined by the **`vsftpd_packages`** variable.

  2. Ensure the **`vsftpd`** service as defined by the **`vsftpd_service`** variable is started and enabled to start at boot.

  3. Ensure that the **`vsftpd.conf.j2`** template has been deployed to the location defined by the **`vsftpd_config_file`** variable and is owned by user **`root`**, group **`root`**, has octal file permissions **`0600`**, and an SELinux type of **`etc_t`**. The **`restart vsftpd`** handler should be notified when this task is run.

  4. Ensure that **`firewalld`** is installed, started, and enabled. Ensure that the **`ftp`** service ports have been opened in the firewall immediately and persistently ("permanently").

The playbook **`vsftpd-configure.yml`** should be created in **`/home/student/ansible-role-cr/`** on **`workstation`**. It needs to meet the following criteria:

- The playbook runs on the host group **`ftpservers`**.

- Privilege escalation should be configured using the remote user **devops** and the **sudo** method.

- The following variables should be set in the playbook:

Variable	Value
vsftpd_anon_root	/mnt/share
vsftpd_local_root	/mnt/share

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Variable	Value
vsftpd_anonymous_enable	true

- It should invoke the **ansible-vsftpd** role.
- After invoking the **ansible-vsftpd** role, the following post tasks should be run:

  1. Use the **command** module to create a GPT disk label on **/dev/vdb** and a new partition, **/dev/vdb1**, that starts 1MiB from the beginning of the device and ends at the end of the device. Use the **ansible-doc** command to learn how to use the **creates** argument to skip this task if **/dev/vdb1** has already been created. This is to avoid destructive repartitioning of the device. Use the following command to create the partition:

     ```
     parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%
     ```

  2. Ensure a **/mnt/share** directory exists for ftp *anon_root*. Ensure its owner and group are set to **root**, the SELinux type to the value for the variable `vsftpd_setype` as specified in the role's vars file, and octal mode is **0755**.
  3. Use **ansible-doc -l** to find a module that can make a file system on a block device. Use **ansible-doc** to learn how to use that module. Add a task to the playbook that creates an XFS file system on **/dev/vdb1**. Do not force creation of that file system if one exists already.
  4. Add a task that ensures that **/etc/fstab** mounts the device **/dev/vdb1** on **/mnt/share** at boot, and that it is currently mounted. (Use **ansible-doc** to find a module that can help with this.)
  5. Make sure a file named **README** exists in the ftp root directory and contains the content **'Welcome to the FTP server at {{ ansible_fqdn }}\n'**.

The **ftpclient.yml** playbook and the dynamic inventory it uses should be configured in **/home/student/ansible-role-cr/** on **workstation** as follows:

- Download the dynamic inventory script from **http://materials.example.com/comp-review/dynamic/crinventory.py**. Place it in the inventory directory **/home/student/ansible-roles-cr/inventory**. Ensure it is executable.
- The playbook runs on the host group **ftpclients**.
- Privilege escalation should be configured using the remote user **devops** and the **sudo** method.
- Add a task that makes sure the latest version of the *lftp* package is installed.

This exercise can be manually validated by logging in to **servera.lab.example.com** as user **student** and using the **lftp** command to download the **README** file from **serverb.lab.example.com**.

Evaluation

From **workstation**, run the **lab ansible-roles-cr** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab ansible-roles-cr grade
```

Cleanup

Run the **lab ansible-roles-cr cleanup** command to clean up the lab tasks on **servera** and **serverb**.

```
[student@workstation ~]$ lab ansible-roles-cr cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this review, you will create a role to configure a *vsftp* server on **serverb.lab.example.com** and verify it by installing an FTP client on **servera.lab.example.com** and then connecting to the FTP server on **serverb.lab.example.com** as an **anonymous** user to list and download the content.

Outcomes

You should be able to:

- Create a role for configuring vsftpd.
- Use a Jinja2 template for vsftpd server configuration.
- Create and execute a playbook by including the role.
- Use dynamic inventory to execute a playbook.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-roles-cr setup** script. This script ensures that the host, **serverb.lab.example.com**, is reachable on the network. The script also checks that Ansible is installed on **workstation** and creates a directory structure for the lab environment and required Ansible configuration files.

```
[student@workstation ~]$ lab ansible-roles-cr setup
```

Instructions

Create a role named **ansible-vsftpd** that will be used with a playbook, **vsftpd-configure.yml**. This playbook will install and configure an FTP server on members of host group **ftpservers**. A second playbook, **ftpclient.yml**, will be needed to install *lftp* on members of host group **ftpclients**. Members of that host group will be determined using a dynamic inventory script.

The **ansible-vsftpd** role should be configured based on the following criteria:

- In the **student** account on **workstation**, under the directory **/home/student/ansible-roles-cr**, create the directory structure for a role named **ansible-vsftpd**.
- Copy the provided **/home/student/ansible-roles-cr/vsftpd.conf.j2** template into the **~/ansible-roles-cr/roles/ansible-vsftpd/templates** directory.
- Define the following variables in **roles/ansible-vsftpd/vars/main.yml**:

Variable	Value
vsftpd_packages	vsftpd
vsftpd_service	vsftpd
vsftpd_config_file	/etc/vsftpd/vsftpd.conf

- Define the following default variables in **roles/ansible-vsftpd/defaults/main.yaml**:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Variable	Value
vsftpd_anonymous_enable	false
vsftpd_connect_from_port_20	true
vsftpd_listen	true
vsftpd_local_enable	true
vsftpd_setype	public_content_t
vsftpd_syslog_enable	true
vsftpd_write_enable	false
vsftpd_chroot_local_user	true

- Define a handler in **roles/ansible-vsftpd/handlers/main.yml** named **restart vsftpd** to restart the **vsftpd** service, using the variable **vsftpd_service** as the name of the service.
- Define a role in **roles/ansible-vsftpd/tasks/main.yml** that does four things. (This may require more than four tasks to be defined.)
  1. Ensure all packages needed by **vsftpd** are installed by using a loop to iterate over all items defined by the **vsftpd_packages** variable.
  2. Ensure the **vsftpd** service as defined by the **vsftpd_service** variable is started and enabled to start at boot.
  3. Ensure that the **vsftpd.conf.j2** template has been deployed to the location defined by the **vsftpd_config_file** variable and is owned by user **root**, group **root**, has octal file permissions **0600**, and an SELinux type of **etc_t**. The **restart vsftpd** handler should be notified when this task is run.
  4. Ensure that **firewalld** is installed, started, and enabled. Ensure that the **ftp** service ports have been opened in the firewall immediately and persistently ("permanently").

The playbook **vsftpd-configure.yml** should be created in **/home/student/ansible-role-cr/** on **workstation**. It needs to meet the following criteria:

- The playbook runs on the host group **ftpservers**.
- Privilege escalation should be configured using the remote user **devops** and the **sudo** method.
- The following variables should be set in the playbook:

Variable	Value
vsftpd_anon_root	/mnt/share
vsftpd_local_root	/mnt/share
vsftpd_anonymous_enable	true

- It should invoke the **ansible-vsftpd** role.
- After invoking the **ansible-vsftpd** role, the following post tasks should be run:

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

1. Use the **command** module to create a GPT disk label on **/dev/vdb** and a new partition, **/dev/vdb1**, that starts 1MiB from the beginning of the device and ends at the end of the device. Use the **ansible-doc** command to learn how to use the **creates** argument to skip this task if **/dev/vdb1** has already been created. This is to avoid destructive repartitioning of the device. Use the following command to create the partition:

```
parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%
```

2. Ensure a **/mnt/share** directory exists for ftp *anon_root*. Ensure its owner and group are set to **root**, the SELinux type to the value for the variable `vsftpd_setype` as specified in the role's vars file, and octal mode is **0755**.

3. Use **ansible-doc -l** to find a module that can make a file system on a block device. Use **ansible-doc** to learn how to use that module. Add a task to the playbook that creates an XFS file system on **/dev/vdb1**. Do not force creation of that file system if one exists already.

4. Add a task that ensures that **/etc/fstab** mounts the device **/dev/vdb1** on **/mnt/share** at boot, and that it is currently mounted. (Use **ansible-doc** to find a module that can help with this.)

5. Make sure a file named **README** exists in the ftp root directory and contains the content **'Welcome to the FTP server at {{ ansible_fqdn }}\n'**.

The **ftpclient.yml** playbook and the dynamic inventory it uses should be configured in **/home/student/ansible-role-cr/** on **workstation** as follows:

- Download the dynamic inventory script from **http://materials.example.com/comp-review/dynamic/crinventory.py**. Place it in the inventory directory **/home/student/ansible-roles-cr/inventory**. Ensure it is executable.
- The playbook runs on the host group **ftpclients**.
- Privilege escalation should be configured using the remote user **devops** and the **sudo** method.
- Add a task that makes sure the latest version of the *lftp* package is installed.

This exercise can be manually validated by logging in to **servera.lab.example.com** as user **student** and using the **lftp** command to download the **README** file from **serverb.lab.example.com**.

Steps

1. From **workstation**, as the **student** user, change to the directory **/home/student/ansible-roles-cr**.

```
[student@workstation ~]$ cd /home/student/ansible-roles-cr/
```

2. Create the directory structure for a role called **ansible-vsftpd**.

```
[student@workstation ansible-roles-cr]$ ansible-galaxy init --offline -p roles
 ansible-vsftpd
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
- ansible-vsftpd was created successfully
```

3. Make the **vsftpd.conf.j2** file a template for the **ansible-vsftpd** role. The lab setup script copied the file to student's home directory. Move it to the **roles/ansible-vsftpd/templates/** subdirectory.

```
[student@workstation ansible-roles-cr]$ mv vsftpd.conf.j2 roles/ansible-vsftpd/
templates
```

4. Define the `vsftpd_packages`, `vsftpd_service`, and `vsftpd_config_file` variables in vars files for the **ansible-vsftpd** role so it contains the following:

Variable	Value
vsftpd_packages	vsftpd
vsftpd_service	vsftpd
vsftpd_config_file	/etc/vsftpd/vsftpd.conf

Define the variables `vsftpd_packages`, `vsftpd_service`, and `vsftpd_config_file` in the file **roles/ansible-vsftpd/vars/main.yml**.

```
---
# vars file for ansible-vsftpd
vsftpd_packages: vsftpd
vsftpd_service: vsftpd
vsftpd_config_file: /etc/vsftpd/vsftpd.conf
```

5. Define the default variables for the **ansible-vsftpd** role so that it contains the following:

Variable	Value
vsftpd_anonymous_enable	false
vsftpd_connect_from_port_20	true
vsftpd_listen	true
vsftpd_local_enable	true
vsftpd_setype	public_content_t
vsftpd_syslog_enable	true
vsftpd_write_enable	false
vsftpd_chroot_local_user	true

5.1. Define the default variables for the **ansible-vsftpd** role in the file **roles/ansible-vsftpd/defaults/main.yml**.

```
---
# defaults file for ansible-vsftpd
vsftpd_anonymous_enable: false
vsftpd_connect_from_port_20: true
vsftpd_listen: true
vsftpd_local_enable: true
vsftpd_setype: public_content_t
vsftpd_syslog_enable: true
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
vsftpd_write_enable: false
vsftpd_chroot_local_user: true
```

6. Define a handler named **`restart vsftpd`** to restart the **vsftpd** service.

   Define the handler in the file **`roles/ansible-vsftpd/handlers/main.yml`** to restart the *vsftpd* service.

```
---
# handlers file for ansible-vsftpd
- name: restart vsftpd
  service:
    name: '{{ vsftpd_service }}'
    state: restarted
```

7. Create the **`main.yml`** file in the **`tasks`** subdirectory of the role. The role should perform four tasks:
   - Install the *vsftpd* package.
   - Start and enable the **vsftpd** service.
   - Install the template configuration file that configures the **vsftpd** service.
   - Ensure that **`firewalld`** allows incoming connections for the FTP service.

   Use the variables defined in the vars file of **`ansible-vsftpd`** role.

   7.1. Create a task in the **`roles/ansible-vsftpd/tasks/main.yml`** file to install the *vsftpd* package.

```
---
# tasks file for ansible-vsftpd
- name: Install packages
  yum:
    name: '{{ item }}'
    state: installed
  with_items: '{{ vsftpd_packages }}'
  tags: vsftpd
```

   7.2. Create a task to ensure that the **vsftpd** service is started and enabled to run at boot time.

```
...file content omitted...
- name: Ensure service is started
  service:
    name: '{{ vsftpd_service }}'
    state: started
    enabled: true
  tags: vsftpd
```

   7.3. Create a task to copy the template **`vsftpd.conf.j2`** from the **`ansible-vsftpd`** role templates subdirectory to the path defined in `vsftpd_config_file` variable.

```
...file content omitted...
- name: Install configuration file
  template:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    src: vsftpd.conf.j2
    dest: '{{ vsftpd_config_file }}'
    owner: root
    group: root
    mode: '0600'
    setype: etc_t
  notify: restart vsftpd
  tags: vsftpd
```

7.4. Create a task to install, start, enable, and open the **firewalld** port for the **ftp** service.

```
...file content omitted...
- name: Install firewalld
  yum:
    name: firewalld
    state: present

- name: Start and enable firewalld
  service:
    name: firewalld
    state: started
    enabled: yes

- name: Open ftp port in firewall
  firewalld:
    service: ftp
    permanent: true
    state: enabled
    immediate: yes
```

7.5. Review the contents of the file **roles/ansible-vsftpd/tasks/main.yml**.

```
---
# tasks file for ansible-vsftpd
- name: Install packages
  yum:
    name: '{{ item }}'
    state: installed
  with_items: '{{ vsftpd_packages }}'
  tags: vsftpd

- name: Ensure service is started
  service:
    name: '{{ vsftpd_service }}'
    state: started
    enabled: true
  tags: vsftpd

- name: Install configuration file
  template:
    src: vsftpd.conf.j2
    dest: '{{ vsftpd_config_file }}'
    owner: root
    group: root
    mode: '0600'
    setype: etc_t
  notify: restart vsftpd
  tags: vsftpd

- name: Install firewalld
  yum:
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
    name: firewalld
    state: present

- name: Start and enable firewalld
  service:
    name: firewalld
    state: started
    enabled: yes

- name: Open ftp port in firewall
  firewalld:
    service: ftp
    permanent: true
    state: enabled
    immediate: yes
```

8. Create the playbook **vsftpd-configure.yml** in **/home/student/ansible-role-cr/** to be executed on the host group **ftpservers** as the **devops** user. The playbook should use sudo privilege escalation using the **root** user. It should also define the following variables:

Variable	Value
vsftpd_anon_root	/mnt/share
vsftpd_local_root	/mnt/share
vsftpd_anonymous_enable	true

The **vsftpd-configure.yml** file should also define the following post tasks after **ansible-vsftpd** role is invoked:

- Use the **command** module to create a GPT disk label on **/dev/vdb**, and a new partition, **/dev/vdb1**, that starts 1MiB from the beginning of the device and ends at the end of the device. Use the **ansible-doc** command to learn how to use the **creates** argument to skip this task if **/dev/vdb1** has already been created. This is to avoid destructive repartitioning of the device. Use the following command to create the partition:

```
parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%
```

- Ensure a **/mnt/share** directory exists for ftp *anon_root*. Ensure its owner and group is set to **root**, its SELinux type to the value for the variable `vsftpd_setype` as specified in the role's vars file, and octal mode is **0755**.

- Use **ansible-doc -l** to find a module that can make a file system on a block device. Use **ansible-doc** to learn how to use that module. Add a task to the playbook that uses it to create an XFS file system on **/dev/vdb1**. Do not force creation of that file system if one exists already.

- Add a task that ensures that **/etc/fstab** will mount the device **/dev/vdb1** on **/mnt/share** at boot, and that it is currently mounted. (Use **ansible-doc** to find a module that can help with this.)

- Make sure a file named **README** exists in the ftp root directory and contains the content **'Welcome to the FTP server at {{ ansible_fqdn }}\n'**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

8.1. Create the playbook **vsftpd-configure.yml** in **/home/student/ansible-role-cr/** to execute it on the host group ftpservers. It should execute the playbook using **devops** as the remote user, and use sudo privilege escalation using the **root** user .

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  become: true
  remote_user: devops
```

8.2. In the playbook **vsftpd-configure.yml**, set the values for the following variables:

Variable	Value
vsftpd_anon_root	/mnt/share
vsftpd_local_root	/mnt/share
vsftpd_anonymous_enable	true

```
...file content omitted..
  vars:
    vsftpd_anon_root: /mnt/share
    vsftpd_local_root: /mnt/share
    vsftpd_anonymous_enable: true
```

8.3. In the playbook **vsftpd-configure.yml**, invoke the role **ansible-vsftpd**.

```
...file content omitted..
  roles:
    - ansible-vsftpd
```

8.4. In the playbook **vsftpd-configure.yml**, create a post task to create a partition on **/dev/vdb** using the command **parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%**. Use the **creates** argument of the **command** module to skip this task if **/dev/vdb1** has already been created. Note that the **>** character in the example YAML below is allowing the value for **command:** to be wrapped on multiple lines in the playbook, ignoring newlines and indentation.

```
...file content omitted...
  post_tasks:
    - name: Partition disks
      command: >
        creates=/dev/vdb1
        parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%
```

8.5. In the playbook **vsftpd-configure.yml**, create a post task to create a directory **/mnt/share**. Set the owner and group to **root**, the SELinux type to the value for the variable `vsftpd_setype` as specified in the role's vars file, and mode to **0755**.

```
...file content omitted..
  post_tasks:
...file content omitted..
    - name: Ensure anon_root directory exists
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
      file:
        path: '{{ vsftpd_anon_root }}'
        owner: root
        group: root
        state: directory
        setype: '{{ vsftpd_setype }}'
        mode: '0755'
```

8.6. Use **ansible-doc -l** to find a module that can make a file system on a block device. Use *ansible-doc* to learn how to use that module.

Add a task to the playbook **vsftpd-configure.yml** that creates an XFS file system on **/dev/vdb1**. Do not force creation of that file system if one exists already.

```
...file content omitted..
  post_tasks:
...file content omitted..
    - name: Create file system on /dev/vdb
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: no
```

8.7. In the playbook **vsftpd-configure.yml**, create a task to make in entry in **/etc/fstab** and mount the device **/dev/vdb1** on **/mnt/share** using an XFS file system.

```
...file content omitted..
  post_tasks:
...file content omitted..
    - name: Mount the file system
      mount:
        name: '{{ vsftpd_anon_root }}'
        src: /dev/vdb1
        fstype: xfs
        state: mounted
        dump: '1'
        passno: '2'
```

8.8. In the playbook **vsftpd-configure.yml**, create a task to copy a file named **README** having the content *"Welcome to the FTP server at {{ ansible_fqdn }}\n"* to **/mnt/share**. Set the SELinux type to **public_content_t**.

```
...file content omitted..
  post_tasks:
...file content omitted..
    - name: Copy some file to the ftp root
      copy:
        dest: '{{ vsftpd_anon_root }}/README'
        content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"
        setype: '{{ vsftpd_setype }}'
```

8.9. Review the contents of the playbook **vsftpd-configure.yml**.

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  become: true
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
  remote_user: devops
  vars:
    vsftpd_anon_root: /mnt/share
    vsftpd_local_root: /mnt/share
    vsftpd_anonymous_enable: true

  roles:
    - ansible-vsftpd

  post_tasks:
    - name: Partition disks
      command: >
        creates=/dev/vdb1
        parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%

    - name: Ensure anon_root directory exists
      file:
        path: '{{ vsftpd_anon_root }}'
        owner: root
        group: root
        state: directory
        setype: '{{ vsftpd_setype }}'
        mode: '0755'

    - name: Create file system on /dev/vdb
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: no

    - name: Mount the file system
      mount:
        name: '{{ vsftpd_anon_root }}'
        src: /dev/vdb1
        fstype: xfs
        state: mounted
        dump: '1'
        passno: '2'

    - name: Copy some file to the ftp root
      copy:
        dest: '{{ vsftpd_anon_root }}/README'
        content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"
        setype: '{{ vsftpd_setype }}'
```

9. Run the playbook **vsftpd-configure.yml**. Check the **ansible-playbook** output to make sure the tasks ran properly.

```
[student@workstation ansible-roles-cr]$ ansible-playbook vsftpd-configure.yml
```

10. Download the dynamic inventory script located at **http://materials.example.com/comp-review/dynamic/crinventory.py** in the directory **/home/student/ansible-roles-cr/inventory**. The **crinventory.py** script returns the details for the **ftpclients** host group.

    10.1. Download the dynamic inventory script from **http://materials.example.com/comp-review/dynamic/crinventory.py** in the directory **/home/student/ansible-roles-cr/inventory**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ansible-roles-cr]$ wget \
> http://materials.example.com/comp-review/dynamic/crinventory.py \
> -O inventory/crinventory.py
```

10.2. Change the permissions for the **crinventory.py** script to **0755**.

```
[student@workstation ansible-roles-cr]$ chmod 0755 inventory/crinventory.py
```

10.3. Execute the command **ansible ftpclients --list-hosts** to check if **servera.lab.example.com** is returned as the host for the **ftpclients** host group.

```
[student@workstation ansible-roles-cr]$ ansible ftpclients --list-hosts
  hosts (1):
    servera.lab.example.com
```

11. Create a playbook **/home/student/ansible-roles-cr/ftpclient.yml** to install the *lftp* package. This operation will require escalated privileges, so enable privilege escalation using **sudo** and the **root** user. The connection to the managed host should be made with the **devops** user. The hosts group should be **ftpclients**.

```
---
- name: ftp client installed
  hosts: ftpclients
  remote_user: devops
  become: yes
  tasks:
    - name: latest lftp version installed
      yum:
        name: lftp
        state: latest
```

12. Execute the playbook **/home/student/ansible-roles-cr/ftpclient.yml** to install the *lftp* package on managed hosts which are part of dynamic inventory **ftpclients**.

```
[student@workstation ansible-roles-cr]$ ansible-playbook ftpclient.yml

PLAY [ftp client installed] ****************************************************

TASK [setup] *******************************************************************
ok: [servera.lab.example.com]

TASK [latest lftp version installed] *******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

13. Use **lftp** to verify that **servera** can be used to connect the **vsftpd** server running on **serverb**.

```
[student@workstation ansible-roles-cr]$ ssh servera
Warning: Permanently added 'servera' (ECDSA) to the list of known hosts.
Last login: Wed May 11 18:16:39 2016 from workstation.lab.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
Managed by Ansible
[student@servera ~]$ lftp serverb
lftp serverb:~>
```

14. From **servera**, download the **README** file from **serverb.lab.example.com** into the directory **/home/student** using **lftp**. Check the contents of the file **/home/student/README**.

    Exit from **servera** when finished.

```
lftp serverb:~> get README
52 bytes transferred
lftp serverb:/> exit
[student@servera ~]$ cat README
Welcome to the FTP server at serverb.lab.example.com
[student@servera ~]$ exit
[student@workstation ansible-roles-cr]$
```

Evaluation

From **workstation**, run the **lab ansible-roles-cr** script with the **grade** argument, to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-roles-cr grade
```

Cleanup

Run the **lab ansible-roles-cr cleanup** command to clean up the lab tasks on **servera** and **serverb**.

```
[student@workstation ~]$ lab ansible-roles-cr cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Lab: Optimizing Ansible

In this review, you will deploy an updated web page to two web servers running behind a load balancer. You will use the **serial** keyword to push this update to one server at a time, and **delegate_to** to remove web servers from the load balancer pool when being updated and to add them back again when the update completes.

The HAProxy load balancer is preconfigured on **workstation.lab.example.com**, and Apache HTTPD is preconfigured on **servera.lab.example.com**, and **serverb.lab.example.com** respectively.

After the upgrade of the web content, the web servers need to be rebooted one at a time before adding them back to load balancer pool, without affecting the site availability.

Outcomes

You should be able to:

- Delegate tasks to other hosts.
- Asynchronously run jobs in parallel.
- Use rolling updates.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-optimize-cr setup** script.
It checks if Ansible is installed on **workstation** and creates a directory structure for the lab environment with an inventory file. The script preconfigures **servera.lab.example.com** and **serverb.lab.example.com** as web servers and configures **workstation.lab.example.com** as the load balancer server using a round-robin algorithm. The script also creates a **templates** directory under the lab's working directory.

The inventory file **/home/student/ansible-optimize-cr/inventory/hosts** lists **servera.lab.example.com** and **serverb.lab.example.com** as managed hosts which are members of the **[webservers]** group and also lists **workstation.lab.example.com** as part of the **[lbserver]** group.

```
[student@workstation ~]$ lab ansible-optimize-cr setup
```

Instructions

Configure a playbook and supporting materials on **workstation** that meet the following criteria:

- As **student** on **workstation**, create a web page template named **index-ver1.html.j2** in the **/home/student/ansible-optimize-cr/templates/** directory. It should have the following content:

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to {{ inventory_hostname }}.
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

- Create a playbook named **upgrade_webserver.yml** in **/home/student/ansible-optimize-cr/**. The playbook should run on the host group **webservers**. It should configure privilege escalation using the remote user **devops**. It should use the **serial** method to push to one host at a time.

  In addition, the playbook should execute the following tasks in the specified order:

  1. Use the **haproxy** Ansible module to disable the web server in the load balancer pool named **app**. The host should be referred to using the **inventory_hostname** variable. Use the socket **/var/lib/haproxy/stats**. Wait until the server reports a status of **MAINT**. The task needs to be delegated to the server from the **[lbserver]** inventory group.

  2. Deploy the **index-ver1.html.j2** template to **/var/www/html/index.html**. Register the **pageupgrade** variable when this task runs.

  3. Reboot the server. Set an asynchronous delay of one second, do not poll, and ignore errors. Execute this task when **pageupgrade** changes.

  4. Use the **wait_for** module to wait for the server to reboot. Determine this by waiting for the **sshd** port (22) to open. Use the **inventory_hostname** variable to determine the host to wait for. Delay 25 seconds before starting to poll, and time out after 200 seconds. Do not escalate privileges. Delegate this task to 127.0.0.1. Like the previous task, execute this task when **pageupgrade** changes.

  5. Use the **wait_for** module to wait for the webserver to be started. Use the **inventory_hostname** variable to determine the host to wait for, and poll port 80. Time out after 20 seconds.

  6. Use the **haproxy** Ansible module to re-enable the web server in the load balancer pool named **app**. Use the same information as specified for the first task in this playbook.

- Run the completed playbook and manually confirm that everything worked. You can use the **curl** command to retrieve pages through the load balancer at **http://workstation.lab.example.com**.

  Note that the template file you deployed customized the document root for the web site so that it will be clear which back-end server you are viewing through the load balancer on any given page reload.

Evaluation

From **workstation**, run the **lab ansible-optimize-cr** script with the **grade** argument, to assess this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab ansible-optimize-cr grade
```

Cleanup

Run the **lab ansible-optimize-cr cleanup** command to clean up the lab tasks on **servera** and **serverb**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab ansible-optimize-cr cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

Solution

In this review, you will deploy an updated web page to two web servers running behind a load balancer. You will use the **serial** keyword to push this update to one server at a time, and **delegate_to** to remove web servers from the load balancer pool when being updated and to add them back again when the update completes.

The **HAProxy** load balancer is preconfigured on **workstation.lab.example.com**, and Apache HTTPD is preconfigured on **servera.lab.example.com**, and **serverb.lab.example.com** respectively.

After the upgrade of the web content, the web servers need to be rebooted one at a time before adding them back to load balancer pool, without affecting the site availability.

Outcomes

You should be able to:

- Delegate tasks to other hosts.
- Asynchronously run jobs in parallel.
- Use rolling updates.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run the **lab ansible-optimize-cr setup** script.
It checks if Ansible is installed on **workstation** and creates a directory structure for the lab environment with an inventory file. The script preconfigures **servera.lab.example.com** and **serverb.lab.example.com** as web servers and configures **workstation.lab.example.com** as the load balancer server using a round-robin algorithm. The script also creates a **templates** directory under the lab's working directory.

The inventory file **/home/student/ansible-optimize-cr/inventory/hosts** lists **servera.lab.example.com** and **serverb.lab.example.com** as managed hosts which are members of the **[webservers]** group and also lists **workstation.lab.example.com** as part of the **[lbserver]** group.

```
[student@workstation ~]$ lab ansible-optimize-cr setup
```

Instructions

Configure a playbook and supporting materials on **workstation** that meet the following criteria:

- As **student** on **workstation**, create a web page template named **index-ver1.html.j2** in the **/home/student/ansible-optimize-cr/templates/** directory. It should have the following content:

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to {{ inventory_hostname }}.
</h1>
<h2>A new feature added.</h2>
</body>
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
</html>
```

- Create a playbook named **upgrade_webserver.yml** in **/home/student/ansible-optimize-cr/**. The playbook should run on the host group **webservers**. It should configure privilege escalation using the remote user **devops**. It should use the **serial** method to push to one host at a time.

  In addition, the playbook should execute the following tasks in the specified order:

  1. Use the **haproxy** Ansible module to disable the web server in the load balancer pool named **app**. The host should be referred to using the **inventory_hostname** variable. Use the socket **/var/lib/haproxy/stats**. Wait until the server reports a status of **MAINT**. The task needs to be delegated to the server from the **[lbserver]** inventory group.

  2. Deploy the **index-ver1.html.j2** template to **/var/www/html/index.html**. Register the **pageupgrade** variable when this task runs.

  3. Reboot the server. Set an asynchronous delay of one second, do not poll, and ignore errors. Execute this task when **pageupgrade** changes.

  4. Use the **wait_for** module to wait for the server to reboot. Determine this by waiting for the **sshd** port (22) to open. Use the **inventory_hostname** variable to determine the host to wait for. Delay 25 seconds before starting to poll, and time out after 200 seconds. Do not escalate privileges. Delegate this task to 127.0.0.1. Like the previous task, execute this task when **pageupgrade** changes.

  5. Use the **wait_for** module to wait for the webserver to be started. Use the **inventory_hostname** variable to determine the host to wait for, and poll port 80. Time out after 20 seconds.

  6. Use the **haproxy** Ansible module to re-enable the web server in the load balancer pool named **app**. Use the same information as specified for the first task in this playbook.

- Run the completed playbook and manually confirm that everything worked. You can use the **curl** command to retrieve pages through the load balancer at **http://workstation.lab.example.com**.

  Note that the template file you deployed customized the document root for the web site so that it will be clear which back-end server you are viewing through the load balancer on any given page reload.

Steps

1. From **workstation**, as the **student** user, change to the directory **/home/student/ansible-optimize-cr**.

```
[student@workstation ~]$ cd /home/student/ansible-optimize-cr
```

2. Because the web servers are preconfigured as part of the lab setup, use **curl** to browse **http://workstation.lab.example.com**. Run the **curl** command twice to see the web content from the web server running on servera and serverb.

```
[student@workstation ansible-optimize-cr]$ curl http://workstation.lab.example.com
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to servera.lab.example.com.
</h1>
</body>
</html>
```

```
[student@workstation ansible-optimize-cr]$ curl http://workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to serverb.lab.example.com.
</h1>
</body>
</html>
```

3. Create a new web page template named **index-ver1.html.j2** under the **templates** directory of the lab's working directory.

```
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to {{ inventory_hostname }}.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

4. Create a playbook named **upgrade_webserver.yml** under **~/ansible-optimize-cr**. The playbook should use privilege escalation using the remote user **devops** and a **hosts** directive using the **webservers** host group.

   The updates needs to be pushed to one server at a time.

   The contents of the **upgrade_webserver.yml** file should be as follows:

```
---
 - name: Upgrade Webservers
   hosts: webservers
   remote_user: devops
   become: yes
   serial: 1
```

5. Create a task in the **upgrade_webserver.yml** playbook to remove the web server from the load balancer pool. Use the **haproxy** Ansible module to remove from the HAProxy load balancer. The task needs to be delegated to a server from the **[lbserver]** inventory group.

   The **haproxy** module is used to disable a back-end server from HAProxy using socket commands. To disable a back-end server from the back-end pool named app, specify the socket path as **/var/lib/haproxy/stats**, and configure `wait=yes` so that the task waits until the server reports a status of **MAINT**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

The contents of the **upgrade_webserver.yml** file should be as follows:

```
...file content omitted...
  tasks:
    - name: disable the server in haproxy
      haproxy:
        state: disabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

6. Create a task in the **upgrade_webserver.yml** playbook to copy the updated page template from the lab working directory **templates/index-ver1.html.j2** to the web servers' document root directories as the file **/var/www/html/index.html**. Also register a variable pageupgrade, which would be later used to invoke other tasks.

```
...file content omitted...
    - name: upgrade the page
      template:
        src: "templates/index-ver1.html.j2"
        dest: "/var/www/html/index.html"
      register: pageupgrade
```

7. Create a task in the **upgrade_webserver.yml** playbook to restart the web servers using an *asynchronous* task that will not wait more than **1** second for the task to complete and that tasks should not be polled from completion.

   Set **ignore_errors** to **true** and execute the task if the earlier registered pageupgrade variable has changed.

   7.1. Create a task in the **upgrade_webserver.yml** playbook to reboot the web server by adding a task to the playbook. Use the **command** module to shut down the machine.

```
...file content omitted...
    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
```

   7.2. Continue editing the task in the **upgrade_webserver.yml** playbook.

   Use the **async** keyword to specify a 1-second wait for task completion. Disable polling by setting the **poll** parameter to **0**. Set **ignore_errors** to **true** and execute the task if the earlier registered pageupgrade variable has changed. Add the following lines in bold to the playbook:

```
...file content omitted...
    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
      async: 1
      poll: 0
      ignore_errors: true
      when: pageupgrade.changed
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

8. Create a task in the **upgrade_webserver.yml** playbook.

   Delegate the task to **localhost**, and use the **wait_for** module to wait for the server to be restarted. Specify the **host** as **inventory_hostname**, **port** as **22**, **state** as **started**, **delay** as **25**, and **timeout** as **200**. The task should be executed when the variable pageupgrade has changed. Privilege escalation is not required for this task.

   The task in the **upgrade_webserver.yml** playbook should read as follows:

```
...file content omitted...
    - name: wait for webserver to reboot
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 22
        state: started
        delay: 25
        timeout: 200
      become: False
      delegate_to: 127.0.0.1
      when: pageupgrade.changed
```

9. Create a task in the **upgrade_webserver.yml** playbook to wait for the web server port to open. (The **httpd** service should already be enabled so it is started when the machine boots.) Specify the **host** as **inventory_hostname**, **port** as **80**, **state** as **started**, and **timeout** as **20**.

```
    - name: wait for webserver to come up
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 80
        state: started
        timeout: 20
```

10. Create a task in the **upgrade_webserver.yml** playbook to add the web server to the load balancer pool after the upgrade of the page. Use the **haproxy** Ansible module to add the server back to the HAProxy load balancer pool. The task needs to be delegated to the **workstation.lab.example.com** server, which is part of the **[lbserver]** inventory group.

    The **haproxy** module is used to enable a back-end sever from HAProxy using socket commands. To enable a back-end server from the **backend pool** named **app**, specify **/var/lib/haproxy/stats** for **socket path**, and set **wait** to **yes** so that the task waits for the server to report a status of healthy.

```
    - name: enable the server in haproxy
      haproxy:
        state: enabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

11. Review the contents of the playbook **upgrade-webserver.yml**.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
---
- name: Upgrade Webservers
  hosts: webservers
  remote_user: devops
  become: yes
  serial: 1

  tasks:
    - name: disable the server in haproxy
      haproxy:
        state: disabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"

    - name: upgrade the page
      template:
        src: "templates/index-ver1.html.j2"
        dest: "/var/www/html/index.html"
      register: pageupgrade

    - name: restart machine
      command: shutdown -r now "Ansible updates triggered"
      async: 1
      poll: 0
      ignore_errors: true
      when: pageupgrade.changed

    - name: wait for webserver to reboot
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 22
        state: started
        delay: 25
        timeout: 200
      become: False
      delegate_to: 127.0.0.1
      when: pageupgrade.changed

    - name: wait for webserver to come up
      wait_for:
        host: "{{ inventory_hostname }}"
        port: 80
        state: started
        timeout: 20

    - name: enable the server in haproxy
      haproxy:
        state: enabled
        backend: app
        host: "{{ inventory_hostname }}"
        socket: /var/lib/haproxy/stats
        wait: yes
      delegate_to: "{{ item }}"
      with_items: "{{ groups.lbserver }}"
```

12. Check the syntax of the playbook **`upgrade-webserver.yml`**. Resolve any syntax errors before proceeding to the next step.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

You can compare your playbook to the one available for download from **http://materials.example.com/comp-review/playbooks/ansible-optimize/upgrade_webserver.yml** or use the provided playbook in place of your own for the next step. Check the syntax using the **ansible-playbook --syntax-check** command.

```
[student@workstation ansible-optimize-cr]$ ansible-playbook --syntax-check
 upgrade_webserver.yml
```

13. Run the playbook **upgrade-webserver.yml** to upgrade the server.

    The restart task will take several minutes, so move on to the next step when it gets to that point in executing the playbook.

```
[student@workstation ansible-optimize]$ ansible-playbook upgrade_webserver.yml
```

14. From **workstation**, use **curl** to view the web link **http://workstation.lab.example.com**. Run the **curl** command several times while the playbook is executing to verify the webserver is still reachable, and that the host changes when the playbook moves on to reboot the other machine.

15. Wait until the remaining tasks from the playbook complete on both **servera** and **serverb**.

16. Verify the web content by browsing the website using the link **http://workstation.lab.example.com**. Rerun the **curl** command to see the updated pages from two different web servers, **servera.lab.example.com** and **serverb.lab.example.com** with the updated content.

```
[student@workstation ansible-optimize]$ curl http://workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to servera.lab.example.com.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

```
[student@workstation ansible-optimize]$ curl http://workstation.lab.example.com
<html>
<head><title>My Page</title></head>
<body>
<h1>
Welcome to serverb.lab.example.com.
</h1>
<h2>A new feature added.</h2>
</body>
</html>
```

Evaluation

From **workstation**, run the **lab ansible-optimize-cr** script with the **grade** argument, to assess this exercise. Correct any reported failures and rerun the script until successful.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

```
[student@workstation ~]$ lab ansible-optimize-cr grade
```

Cleanup

Run the **lab ansible-optimize-cr cleanup** command to clean up the lab tasks on **servera** and **serverb**.

```
[student@workstation ~]$ lab ansible-optimize-cr cleanup
```

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.

For use by rtc-vtlogin29 Copyright © 2016 Red Hat, Inc.