For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Red Hat
Training and Certification

Student Workbook (ROLE)

Red Hat Enterprise Linux 8.4 RH294

Red Hat Enterprise Linux Automation with Ansible

Edition 1

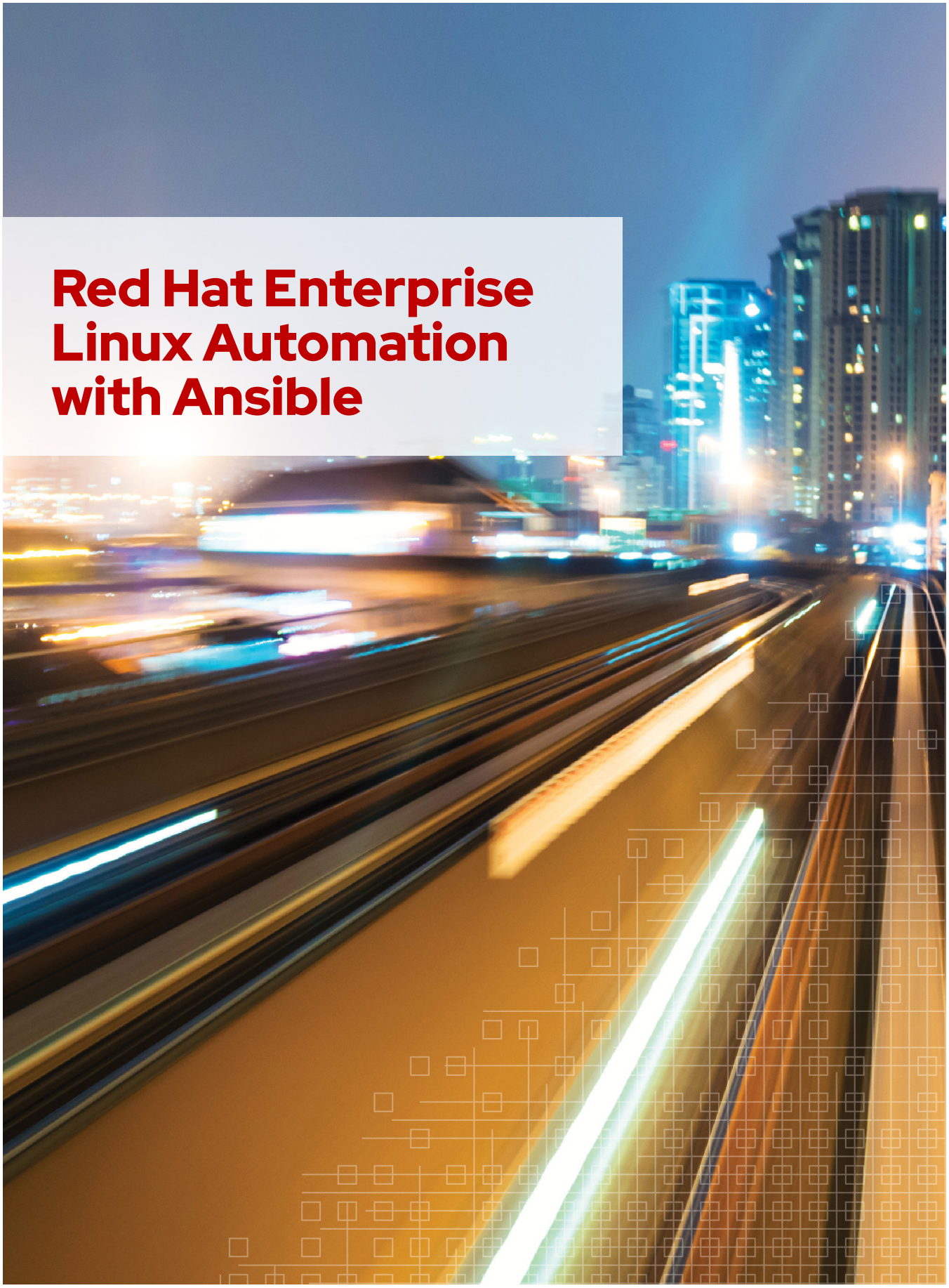
RH294-RHEL8.4-en-1-20210818

Copyright ©2021 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Red Hat Enterprise Linux Automation with Ansible



Copyright ©2021 Red Hat, Inc.

Red Hat Enterprise Linux 8.4 RH294
Red Hat Enterprise Linux Automation with Ansible
Edition 1 20210818
Publication date 20210818

Authors:	Trey Feagle, Hervé Quatremain, Dallas Spohn, Adolfo Vazquez, Morgan Weetman
Course Architect:	Steven Bonneville
DevOps Engineer:	Dan Kolepp
Editor:	Philip Sweany, Seth Kenlon, Jeff Tyson, Nicole Muller, David O'Brien

Copyright © 2021 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2021 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 send email to training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries 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.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

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: James Mighion, Alejandra Ramirez Palacios, Michael Phillips

Portions of this course were adapted from the Ansible Lightbulb project. The material from that project is available from https://github.com/ansible/lightbulb under the MIT License.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Document Conventions

This section describes various conventions and practices used throughout all Red Hat Training courses.

Admonitions

Red Hat Training courses use the following admonitions:

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

Note
These 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 something that makes your life easier.

Important
These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.

Warning
These should not be ignored. Ignoring these admonitions will most likely cause data loss.

Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Introduction

Red Hat Enterprise Linux Automation with Ansible

Red Hat Enterprise Linux Automation with Ansible (RH294) is intended for Linux system administrators and developers who need to automate provisioning, configuration, application deployment, and orchestration.

Students will learn how to install and configure Ansible on a management workstation and prepare managed hosts for automation. Students will write Ansible Playbooks to automate tasks, and run them to ensure servers are correctly deployed and configured. Examples of approaches to automate common Linux system administration tasks will be explored.

Course Objectives

- Install and configure Ansible from Red Hat Ansible Automation Platform on a control node.
- Create and manage inventories of managed hosts, and prepare them for Ansible automation.
- Run individual ad hoc automation tasks from the command line.
- Write Ansible Playbooks to consistently automate multiple tasks and apply them to managed hosts.
- Parameterize playbooks using variables and facts, and protect sensitive data with Ansible Vault.
- Write and reuse existing Ansible roles to simplify playbook creation and reuse code.
- Automate common Red Hat Enterprise Linux system administration tasks using Ansible.

Audience

- Linux system administrators, DevOps engineers, infrastructure automation engineers, and systems design engineers responsible for automation of configuration management, consistent and repeatable application deployment, provisioning and deployment of development, testing, and production servers, and integration with DevOps CI/CD workflows.

Prerequisites

- Red Hat Certified System Administrator (EX200/RHCSA) certification or equivalent Red Hat Enterprise Linux knowledge and experience.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Orientation to the Classroom Environment

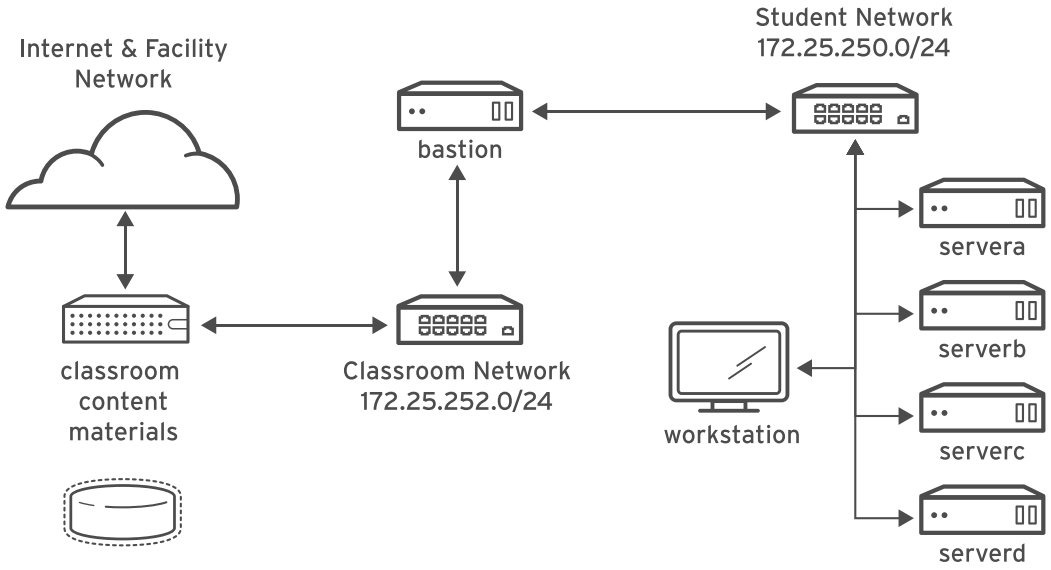


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is `workstation`. Four other machines are also used by students for these activities: `servera`, `serverb`, `serverc`, and `serverd`. All these five 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
bastion.lab.example.com	172.25.250.254	Gateway system to connect student private network to classroom server (must always be running)
workstation.lab.example.com	172.25.250.9	Graphical workstation used for system administration
servera.lab.example.com	172.25.250.10	Host managed with Ansible
serverb.lab.example.com	172.25.250.11	Host managed with Ansible
serverc.lab.example.com	172.25.250.12	Host managed with Ansible
serverd.lab.example.com	172.25.250.13	Host managed with Ansible

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The primary function of `bastion` is that it acts as a router between the network that connects the student machines and the classroom network. If `bastion` is down, other student machines will only be able to access systems on the individual student network.

Several systems in the classroom 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 is provided in the instructions for those activities. These are provided by the `classroom.example.com` virtual machine. Both `classroom` and `bastion` should always be running for proper use of the lab environment.

Controlling Your Systems

`rht-vmctl` Commands

Action	Command
Start `server` machine	`rht-vmctl start server`
View "physical console" to log in and work with the `server` machine	`rht-vmview view server`
Reset `server` machine to its previous state and restart the virtual machine	`rht-vmctl reset server`

Students are assigned remote computers in a Red Hat Online Learning classroom. They are accessed through a web application hosted at rol.redhat.com [http://rol.redhat.com]. Students should log in to this site using their Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the **Online Lab** tab.

Machine States

Virtual Machine State	Description
STARTING	The virtual machine is in the process of booting.
STARTED	The virtual machine is running and available (or, when booting, soon will be).
STOPPING	The virtual machine is in the process of shutting down.
STOPPED	The virtual machine is completely shut down. Upon starting, the virtual machine boots into the same state as when it was shut down (the disk will have been preserved).
PUBLISHING	The initial creation of the virtual machine is being performed.
WAITING_TO_START	The virtual machine is waiting for other virtual machines to start.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Depending on the state of a machine, a selection of the following actions is available.

Classroom/Machine Actions

Button or Action	Description
PROVISION LAB	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE LAB	Delete the ROL classroom. Destroys all virtual machines in the classroom. Caution: Any work generated on the disks is lost.
START LAB	Start all virtual machines in the classroom.
SHUTDOWN LAB	Stop all virtual machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the virtual machine and run commands. In most cases, students should log in to the workstation virtual machine and use ssh to connect to the other virtual machines.
ACTION → Start	Start (power on) the virtual machine.
ACTION → Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION → Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. Caution: Any work generated on the disk is lost.

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION → Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION → Reset**

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE LAB** to remove the entire classroom environment. After the lab has been deleted, you can click **PROVISION LAB** to provision a new set of classroom systems.

Warning
The **DELETE LAB** operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

The Red Hat Online Learning enrollment entitles students to a certain amount of computer time. To help conserve allotted computer time, the ROL classroom has an associated countdown timer, which shuts down the classroom environment when the timer expires.

To adjust the timer, click **MODIFY** to display the **New Autostop Time** dialog box. Set the number of hours and minutes until the classroom should automatically stop.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Click **ADJUST TIME** to apply this change to the timer settings.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 1

Introducing Ansible

Goal

Describe the fundamental Ansible concepts and how it is used, and install Ansible from Red Hat Ansible Automation Platform.

Objectives

- Describe the motivation for automating Linux administration tasks with Ansible, fundamental Ansible concepts, and Ansible's basic architecture.
- Install Ansible on a control node and describe the distinction between community Ansible and Red Hat Ansible Automation Platform.

Sections

- Automating Linux Administration with Ansible (and Quiz)
- Installing Ansible (and Guided Exercise)

Automating Linux Administration with Ansible

Objective

After completing this section, you should be able to describe the motivation for automating Linux administration tasks with Ansible, fundamental Ansible concepts, and Ansible's basic architecture.

Automation and Linux System Administration

For many years, most system administration and infrastructure management has relied on manual tasks performed through graphical or command-line user interfaces. System administrators often work from checklists, other documentation, or a memorized routine to perform standard tasks.

This approach is error-prone. It is easy for a system administrator to skip a step or perform a step mistakenly.

Often there is limited verification that the steps were performed properly or that they result in the expected outcome.

Furthermore, by managing each server manually and independently, it is very easy for many servers that are supposed to be identical in configuration to be different in minor (or major) ways. This can make maintenance more difficult and introduce errors or instability into the IT environment.

Automation can help avoid the problems caused by manual system administration and infrastructure management. As a system administrator, you can use automation to ensure that all your systems are quickly and correctly deployed and configured. This allows you to automate the repetitive tasks in your daily schedule, freeing up your time and allowing you to focus on more critical things. For your organization, this means you can more quickly roll out the next version of an application or updates to a service.

Infrastructure as Code

A good automation system allows you to implement *Infrastructure as Code* practices. Infrastructure as Code means that you can use a machine-readable automation language to define and describe the state you want your IT infrastructure to be in. Ideally, this automation language should also be very easy for humans to read, because then you can easily understand what the state is and make changes to it. This code is then applied to your infrastructure to ensure that it is actually in that state.

If the automation language is represented as simple text files, it can easily be managed in a version control system. The advantage of this is that every change can be checked into the version control system, ensuring that you have a history of the changes you make over time. If you want to revert to an earlier known-good configuration, you can check out that version and apply it to your infrastructure.

This builds a foundation to help you follow best practices in DevOps. Developers can define their desired configuration in the automation language. Operators can review those changes more easily to provide feedback, and use that automation to reproducibly ensure that systems are in the state expected by the developers.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Mitigating Human Error

By reducing the tasks performed manually on servers using automation of tasks and Infrastructure as Code practices, your servers will be in consistent configurations more often. This means that you need to become accustomed to making changes by updating your automation code, rather than manually applying them to your servers. Otherwise, you run the risk of losing manually applied changes the next time you apply changes using automation.

Automation allows you to use code review, peer review by multiple subject matter experts, and documentation of the procedure by the automation itself to reduce your operational risks.

Ultimately, you can enforce that changes to your IT infrastructure must be made through automation in order to mitigate human error.

What is Ansible?

Ansible is an open source automation platform. It is a *simple automation language* that can perfectly describe an IT application infrastructure in Ansible Playbooks. It is also an *automation engine* that runs Ansible Playbooks.

Ansible can manage powerful automation tasks and can adapt to many different workflows and environments. At the same time, new users of Ansible can very quickly use it to become productive.

Ansible Is Simple

Ansible Playbooks provide human-readable automation. This means that playbooks are automation tools that are also easy for humans to read, comprehend, and change. No special coding skills are required to write them. Playbooks execute tasks in order. The simplicity of playbook design makes them usable by every team, which allows people new to Ansible to get productive quickly.

Ansible Is Powerful

You can use Ansible to deploy applications for configuration management, for workflow automation, and for network automation. Ansible can be used to orchestrate the entire application life cycle.

Ansible Is Agentless

Ansible is built around an *agentless architecture*. Typically, Ansible connects to the hosts it manages using OpenSSH or WinRM and runs tasks, often (but not always) by pushing out small programs called *Ansible modules* to those hosts. These programs are used to put the system in a specific desired state. Any modules that are pushed are removed when Ansible is finished with its tasks. You can start using Ansible almost immediately because no special agents need to be approved for use and then deployed to the managed hosts. Because there are no agents and no additional custom security infrastructure, Ansible is more efficient and more secure than other alternatives.

Ansible has a number of important strengths:

- *Cross platform support*: Ansible provides agentless support for Linux, Windows, UNIX, and network devices, in physical, virtual, cloud, and container environments.
- *Human-readable automation*: Ansible Playbooks, written as YAML text files, are easy to read and help ensure that everyone understands what they will do.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- *Perfect description of applications*: Every change can be made by Ansible Playbooks, and every aspect of your application environment can be described and documented.
- *Easy to manage in version control*: Ansible Playbooks and projects are plain text. They can be treated like source code and placed in your existing version control system.
- *Support for dynamic inventories*: The list of machines that Ansible manages can be dynamically updated from external sources in order to capture the correct, current list of all managed servers all the time, regardless of infrastructure or location.
- *Orchestration that integrates easily with other systems*: HP SA, Puppet, Jenkins, Red Hat Satellite, and other systems that exist in your environment can be leveraged and integrated into your Ansible workflow.

Ansible: The Language of DevOps

Figure 1.1: Ansible across the application life cycle

Communication is the key to DevOps. Ansible is the first automation language that can be read and written across IT. It is also the only automation engine that can automate the application life cycle and continuous delivery pipeline from start to finish.

Ansible Concepts and Architecture

There are two types of machines in the Ansible architecture: *control nodes* and *managed hosts*. Ansible is installed and run from a control node, and this machine also has copies of your Ansible project files. A control node could be an administrator's laptop, a system shared by a number of administrators, or a server running Red Hat Ansible Tower.

Managed hosts are listed in an *inventory*, which also organizes those systems into groups for easier collective management. The inventory can be defined in a static text file, or dynamically determined by scripts that get information from external sources.

Instead of writing complex scripts, Ansible users create high-level *plays* to ensure a host or group of hosts are in a particular state. A play performs a series of *tasks* on the hosts, in the order specified by the play. These plays are expressed in YAML format in a text file. A file that contains one or more plays is called a *playbook*.

Each task runs a *module*, a small piece of code (written in Python, PowerShell, or some other language), with specific arguments. Each module is essentially a tool in your toolkit. Ansible ships with hundreds of useful modules that can perform a wide variety of automation tasks. They can act on system files, install software, or make API calls.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

When used in a task, a module generally ensures that some particular aspect of the machine is in a particular state. For example, a task using one module might ensure that a file exists and has particular permissions and contents, while a task using a different module might make certain that a particular file system is mounted. If the system is not in that state, the task should put it in that state. If the system is already in that state, it does nothing. If a task fails, the default Ansible behavior is to abort the rest of the playbook for the hosts that had a failure.

Tasks, plays, and playbooks are designed to be *idempotent*. This means that you can safely run a playbook on the same hosts multiple times. When your systems are in the correct state, the playbook makes no changes when you run it. This means that you should be able to run a playbook on the same hosts multiple times safely. When your systems are in the correct state the playbook should make no changes when you run it. There are a handful of modules that you can use to run arbitrary commands. However, you must use those modules with care to ensure that they run in an idempotent way.

Ansible also uses *plug-ins*. Plug-ins are code that you can add to Ansible to extend it and adapt it to new uses and platforms.

The Ansible architecture is agentless. Typically, when an administrator runs an Ansible Playbook or an ad hoc command, the control node connects to the managed host using SSH (by default) or WinRM. This means that clients do not need to have an Ansible-specific agent installed on managed hosts, and do not need to permit special network traffic to some nonstandard port.

Getting Support for Ansible

Red Hat Ansible Automation Platform is a fully supported version of Ansible that allows enterprises to manage their automation at scale.

It provides:

- Official support for the core Ansible toolset.
- Certified content collections to help you accelerate adoption of Ansible automation with supported code.
- Cloud services to help you discover certified Ansible content, facilitate team collaboration, and provide operational analytics to automate mixed, hybrid environments.
- On-premise tools to help you centralize management of automation tasks.

For example, the automation controller component (formerly called Red Hat Ansible Tower) is an enterprise framework that you can use to control who has access to run playbooks on which hosts, share the use of SSH credentials without allowing users to transfer them or see their contents, log all of your Ansible jobs, and manage inventory, among many other things.

It provides a browser-based user interface (web UI) and a RESTful API. The upstream Ansible community does not automatically include this with core Ansible, but it is developed as open source and is provided and supported as part of the Red Hat Ansible Automation Platform product.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

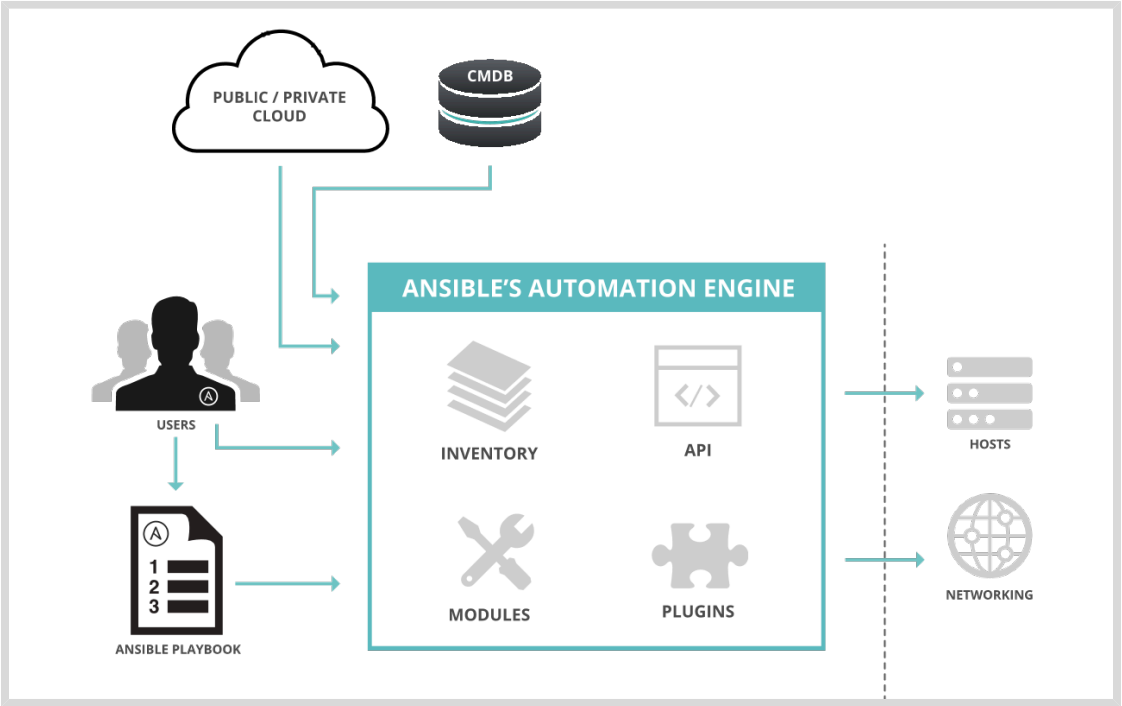


Figure 1.2: Ansible architecture

The Ansible Way

Complexity Kills Productivity

Simpler is better. Ansible is designed so that its tools are simple to use and automation is simple to write and read. You should take advantage of this to strive for simplification in how you create your automation.

Optimize For Readability

The Ansible automation language is built around simple, declarative, text-based files that are easy for humans to read. Written properly, Ansible Playbooks can clearly document your workflow automation.

Think Declaratively

Ansible is a *desired-state engine*. It approaches the problem of how to automate IT deployments by expressing them in terms of the state that you want your systems to be in. Ansible's goal is to put your systems into the desired state, only making changes that are necessary. Trying to treat Ansible like a scripting language is not the right approach.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

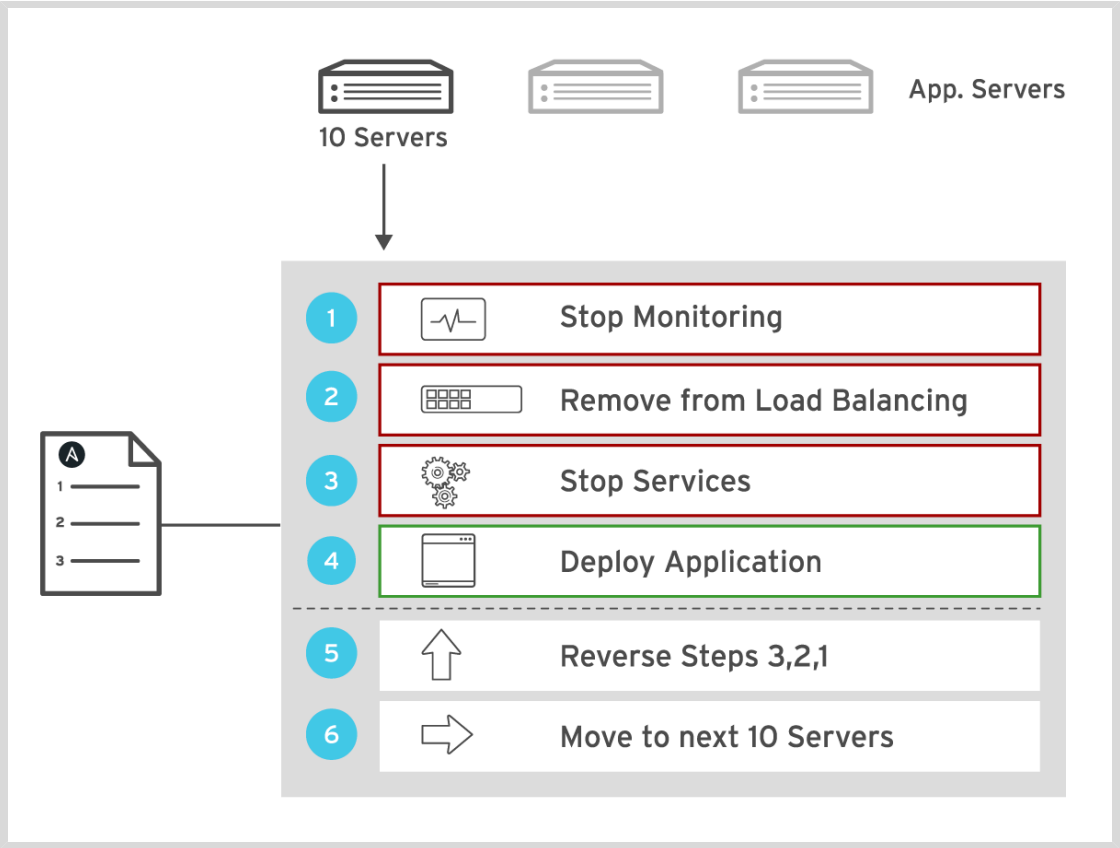


Figure 1.3: Ansible provides complete automation

Use Cases

Unlike some other tools, Ansible combines and unites orchestration with configuration management, provisioning, and application deployment in one easy-to-use platform.

Some use cases for Ansible include:

Configuration Management
Centralizing configuration file management and deployment is a common use case for Ansible, and it is how many power users are first introduced to the Ansible automation platform.

Application Deployment
When you define your application with Ansible, and manage the deployment with Red Hat Ansible Tower, teams can effectively manage the entire application life cycle from development to production.

Provisioning
Applications have to be deployed or installed on systems. Ansible and Red Hat Ansible Tower can help streamline the process of provisioning systems, whether you are PXE booting and kickstarting bare-metal servers or virtual machines, or creating virtual machines or cloud instances from templates. Applications have to be deployed or installed on systems.

Continuous Delivery
Creating a CI/CD pipeline requires coordination and buy-in from numerous teams. You cannot do it without a simple automation platform that everyone in your organization can use. Ansible Playbooks keep your applications properly deployed and managed throughout their entire life cycle.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Security and Compliance
When your security policy is defined in Ansible Playbooks, scanning and remediation of site-wide security policies can be integrated into other automated processes.

Instead of being an afterthought, it is an integral part of everything that is deployed.

Orchestration
Configurations alone do not define your environment. You need to define how multiple configurations interact, and ensure that the disparate pieces can be managed as a whole.

References

Ansible
https://www.ansible.com

How Ansible Works
https://www.ansible.com/how-ansible-works

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Quiz

Automating Linux Administration with Ansible

Choose the correct answer to the following questions:

1. Which of the following terms best describes the Ansible architecture?

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

2. Which network protocol does Ansible use by default to communicate with managed nodes?

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

3. Which of the following files defines the actions that Ansible performs on managed nodes?

a. Host inventory
b. Manifest
c. Playbook
d. Script

4. What syntax is used to define Ansible Playbooks?

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

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Automating Linux Administration with Ansible

Choose the correct answer to the following questions:

1. Which of the following terms best describes the Ansible architecture?

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

2. Which network protocol does Ansible use by default to communicate with managed nodes?

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

3. Which of the following files defines the actions that Ansible performs on managed nodes?

a. Host inventory
b. Manifest
c. Playbook
d. Script

4. What syntax is used to define Ansible Playbooks?

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

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Installing Ansible

Objectives

After completing this section, you should be able to install Ansible on a control node and describe the distinction between community Ansible and Red Hat Ansible Automation Platform.

Ansible and Red Hat Ansible Automation Platform

Red Hat provides a fully supported version of Ansible through Red Hat Ansible Automation Platform. Ansible Automation Platform provides the core Ansible toolset plus additional certified and supported content, tools, and cloud services. Customers with a valid subscription can use the available repository, install the additional tools, and consume certified content from the cloud services.

This course is currently based on Red Hat Ansible Automation Platform 1.2, which includes Ansible 2.9.

Note

Earlier versions of Red Hat Ansible Automation Platform refer to the included version of Ansible as "Red Hat Ansible Engine", and you will see this terminology used in some documentation.

The upstream development community also provides an unsupported version of Ansible. This used to be provided as RPM packages, but is transitioning to be provided only from the Python Package Index (PyPI).

Control Nodes

Ansible is simple to install. The Ansible software only needs to be installed on the control node (or nodes) from which Ansible will be run. Hosts that are managed by Ansible do not need to have Ansible installed.

Installing the core Ansible toolset involves relatively few steps and has minimal requirements. On the other hand, installing the additional components that Red Hat Ansible Automation Platform provides, such as the automation controller (formerly called Red Hat Ansible Tower), requires a Red Hat Enterprise Linux 8.2 or later system, with a minimum of two CPUs, 4 GiB of RAM, and 20 GiB of available disk space.

Python 3 (version 3.5 or later) or Python 2 (version 2.7 or later) needs to be installed on the control node.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

If you are running Red Hat Enterprise Linux 8, Ansible can automatically use the *platform-python* package that supports system utilities that use Python. You do not need to install the *python36* or *python27* package from AppStream.

```
[root@controlnode ~]# yum list installed platform-python
Installed Packages
platform-python.x86_64       3.6.8-37.el         @anaconda
```

You need a valid Red Hat Ansible Automation Platform subscription to install the core toolset on your control node. The installation process is as follows:

If you have activated Simple Content Access for your organization in the Red Hat Customer Portal, then you do not need to attach the subscription to your system. The installation process is as follows:

Warning

You do not need to run these steps in your classroom environment.

- Register your system to Red Hat Subscription Manager.

```
[root@host ~]# subscription-manager register
```

- Enable the Red Hat Ansible Engine repository.

```
[root@host ~]# subscription-manager repos \
> --enable ansible-2-for-rhel-8-x86_64-rpms
```

- Install Red Hat Ansible Engine.

```
[root@host ~]# yum install ansible
```

Managed Hosts

One of the benefits of Ansible is that managed hosts do not need to have a special agent installed. The Ansible control node connects to managed hosts using a standard network protocol to ensure that the systems are in the specified state.

Managed hosts might have some requirements depending on how the control node connects to them and what modules it will run on them.

Linux and UNIX managed hosts need to have Python 2 (version 2.6 or later) or Python 3 (version 3.5 or later) installed for most modules to work.

For Red Hat Enterprise Linux 8, you may be able to depend on the *platform-python* package. You can also enable and install the *python36* application stream (or the *python27* application stream).

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[root@host ~]# yum module install python36
```

If SELinux is enabled on the managed hosts, ensure that the *python3-libselinux* package is installed before using modules that are related to any copy, file, or template functions. (Note that if the other Python components are installed, you can use Ansible modules such as `yum` or `package` to ensure that this package is also installed.)

Important

Some package names may be different in Red Hat Enterprise Linux 7 and earlier because of the ongoing migration to Python 3.

For Red Hat Enterprise Linux 7 and earlier, install the *python* package, which provides Python 2. Instead of *python3-libselinux*, install *libselinux-python* instead.

Some modules might have their own additional requirements. For example, the `dnf` module, which can be used to install packages on current Fedora systems, requires the *python3-dnf* package (*python-dnf* in RHEL 7).

Note

Some modules do not need Python. For example, arguments passed to the Ansible `raw` module are run directly through the configured remote shell instead of going through the module subsystem. This can be useful for managing devices that do not have Python available or cannot have Python installed, or for bootstrapping Python onto a system that does not have it.

However, the `raw` module is difficult to use in a safely idempotent way. If you can use a normal module instead, it is generally better to avoid using `raw` and similar command modules. This is discussed further later in the course.

Microsoft Windows-based Managed Hosts

Ansible includes a number of modules that are specifically designed for Microsoft Windows systems. These are listed in the Windows modules [https://docs.ansible.com/ansible/2.9/modules/list_of_windows_modules.html] section of the Ansible module index.

Most of the modules specifically designed for Microsoft Windows managed hosts require PowerShell 3.0 or later on the managed host rather than Python. In addition, the managed hosts need to have Windows PowerShell remoting configured. Ansible also requires at least .NET Framework 4.0 or later to be installed on Windows managed hosts.

This course uses Linux-based managed hosts in its examples, and does not go into great depth on the specific differences and adjustments needed when managing Microsoft Windows-based managed hosts. More information is available on the Ansible web site at https://docs.ansible.com/ansible/2.9/user_guide/windows.html.

Managed Network Devices

You can also use Ansible automation to configure managed network devices such as routers and switches. Ansible includes a large number of modules specifically designed for this purpose. This includes support for Cisco IOS, IOS XR, and NX-OS; Juniper Junos; Arista EOS; and VyOS-based networking devices, among others.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

You can write Ansible Playbooks for network devices using the same basic techniques that you use when writing playbooks for servers. Because most network devices cannot run Python, Ansible runs network modules on the control node, not on the managed hosts. Special connection methods are also used to communicate with network devices, typically using either CLI over SSH, XML over SSH, or API over HTTP(S).

This course does not cover automation of network device management in any depth. For more information on this topic, see *Ansible for Network Automation* [https://docs.ansible.com/ansible/2.9/network/index.html] on the Ansible community website, or attend our alternative course *Ansible for Network Automation* (DO457) [https://www.redhat.com/en/services/training/do457-ansible-network-automation].

Preparing for Changes to Ansible Release Methods

Both the upstream Ansible community and Red Hat Ansible Automation Platform are going through a transition in how Ansible is packaged and distributed to users.

Ansible 2.9, Ansible in Red Hat Ansible Automation Platform 1.2, and earlier versions of both were provided as an RPM package (*ansible*). This package also included all Ansible modules and plug-ins.

In future versions of Red Hat Ansible Automation Platform, the code that runs automation will be moved to a new package, *ansible-core*, and supported modules and plug-ins will be provided using a new feature, *content collections*. Content collections will be discussed in more detail later in this course. In addition, Ansible Automation Platform 2 will also include enhanced tools and features to run your playbooks, new cloud services features, and enhanced versions of the automation controller (formerly known as Red Hat Ansible Tower) and automation hub.

Future versions of community-built Ansible will provide the executables and a selected set of content through the Python Package Index (PyPI), from which the `pip install ansible` command can install them. However, this selected set of content might be different from what Red Hat supports and certifies in Red Hat Ansible Automation Platform.

The automation code, tools, and techniques you will learn in this course apply directly to future versions of Ansible with little or no modification.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

References

`ansible-doc`(1) man page

Knowledgebase: "How Do I Download and Install Red Hat Ansible Engine?"
https://access.redhat.com/articles/3174981

Simple Content Access
https://access.redhat.com/articles/simple-content-access

Product Documentation for Red Hat Ansible Automation Platform 1.2
https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/1.2/

Windows Guides — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/windows.html

Ansible for Networking Automation — Ansible Documentation
https://docs.ansible.com/ansible/2.9/network/index.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Installing Ansible

In this exercise, you will install Ansible on a control node running Red Hat Enterprise Linux.

Outcomes

You should be able to install Ansible on a control node.

Before You Begin

Log in to `workstation` as `student` using `student` as the password, and run the `lab intro-install start` command. This command configures the control node.

```
[student@workstation ~]$ lab intro-install start
```

Instructions

1. Install Ansible on `workstation` so that you can use that machine as your control node.

```
[student@workstation ~]$ sudo yum install ansible
[sudo] password for student: student
Last metadata expiration check: 0:00:44 ago on Thu 22 Jul 2021 01:27:41 AM EDT.
Dependencies resolved.
...output omitted...
Is this ok [y/d/N]: y
...output omitted...
```

2. Verify that Ansible is installed on the system. Execute the `ansible` command with the `--version` option.

```
[student@workstation ~]$ ansible --version
ansible 2.9.15
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/student/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.8 (default, Mar 18 2021, 08:58:41) [GCC 8.4.1 20200928 (Red
 Hat 8.4.1-1)]
```

3. Invoke the `setup` module on the local host to retrieve the value of the `ansible_python_version` fact.

```
[student@workstation ~]$ ansible -m setup localhost | grep ansible_python_version
    "ansible_python_version": "3.6.8",
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finish

On `workstation`, run the `lab intro-install finish` script to clean up this exercise.

```
[student@workstation ~]$ lab intro-install finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- Automation is a key tool to mitigate human error and quickly ensure that your IT infrastructure is in a consistent, correct state.
- Ansible is an open source automation platform that can adapt to many different workflows and environments.
- Ansible can be used to manage many different types of systems, including servers running Linux, Microsoft Windows, or UNIX, and network devices.
- Ansible Playbooks are human-readable text files that describe the desired state of an IT infrastructure.
- Ansible is built around an agentless architecture in which Ansible is installed on a control node and clients do not need any special agent software.
- Ansible connects to managed hosts using standard network protocols such as SSH, and runs code or commands on the managed hosts to ensure that they are in the state specified by Ansible.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 2

Implementing an Ansible Playbook

Goal

Create an inventory of managed hosts, write a simple Ansible Playbook, and run the playbook to automate tasks on those hosts.

Objectives

- Describe Ansible inventory concepts and manage a static inventory file.
- Describe where Ansible configuration files are located, how Ansible selects them, and edit them to apply changes to default settings.
- Run a single Ansible automation task using an ad hoc command and explain some use cases for ad hoc commands.
- Write a basic Ansible Playbook and run it using the `ansible-playbook` command.
- Write a playbook that uses multiple plays and per-play privilege escalation, and effectively use ansible-doc to learn how to use new modules to implement tasks for a play.

Sections

- Building an Ansible Inventory (and Guided Exercise)
- Managing Ansible Configuration Files (and Guided Exercise)
- Running Ad Hoc Commands (and Guided Exercise)
- Writing and Running Playbooks (and Guided Exercise)
- Implementing Multiple Plays (and Guided Exercise)

Lab

- Implementing Playbooks

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Building an Ansible Inventory

Objectives

After completing this section, you should be able to describe Ansible inventory concepts and manage a static inventory file.

Defining the Inventory

An *inventory* defines a collection of hosts that Ansible will manage. These hosts can also be assigned to *groups*, which can be managed collectively. Groups can contain child groups, and hosts can be members of multiple groups. The inventory can also set variables that apply to the hosts and groups that it defines.

Host inventories can be defined in two different ways. A *static* host inventory can be defined by a text file. A *dynamic* host inventory can be generated by a script or other program as needed, using external information providers.

Specifying Managed Hosts with a Static Inventory

A static inventory file is a text file that specifies the managed hosts that Ansible targets. You can write this file using a number of different formats, including INI-style or YAML. The INI-style format is very common and will be used for most examples in this course.

Note

There are multiple static inventory formats supported by Ansible. In this section, we are focusing on the most common one, INI-style format.

In its simplest form, an INI-style static inventory file is a list of host names or IP addresses of managed hosts, each on a single line:

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

Normally, however, you organize managed hosts into *host groups*. Host groups allow you to more effectively run Ansible against a collection of systems. In this case, each section starts with a host group name enclosed in square brackets ([]). This is followed by the host name or an IP address for each managed host in the group, each on a single line.

In the following example, the host inventory defines two host groups: `webservers` and `db-servers`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com
```

Hosts can be in multiple groups. In fact, recommended practice is to organize your hosts into multiple groups, possibly organized in different ways depending on the role of the host, its physical location, whether it is in production or not, and so on. This allows you to easily apply Ansible plays to specific hosts.

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com

[east-datacenter]
web1.example.com
db1.example.com

[west-datacenter]
web2.example.com
db2.example.com

[production]
web1.example.com
web2.example.com
db1.example.com
db2.example.com

[development]
192.0.2.42
```

Important

Two host groups always exist:

- The `all` host group contains every host explicitly listed in the inventory.
- The `ungrouped` host group contains every host explicitly listed in the inventory that is not a member of any other group.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Defining Nested Groups

Ansible host inventories can include groups of host groups. This is accomplished by creating a host group name with the `:children` suffix. The following example creates a new group called `north-america`, which includes all hosts from the `usa` and `canada` groups.

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

[canada]
ontario01.example.com
ontario02.example.com

[north-america:children]
canada
usa
```

A group can have both managed hosts and child groups as members. For example, in the previous inventory you could add a `[north-america]` section that has its own list of managed hosts. That list of hosts would be merged with the additional hosts that the `north-america` group inherits from its child groups.

Simplifying Host Specifications with Ranges

You can specify ranges in the host names or IP addresses to simplify Ansible host inventories. You can specify either numeric or alphabetic ranges. Ranges have the following syntax:

```
[START:END]
```

Ranges match all values from *START* to *END*, inclusively. Consider the following examples:

- 192.168.[4:7].[0:255] matches all IPv4 addresses in the 192.168.4.0/22 network (192.168.4.0 through 192.168.7.255).
- server[01:20].example.com matches all hosts named server01.example.com through server20.example.com.
- [a:c].dns.example.com matches hosts named a.dns.example.com, b.dns.example.com, and c.dns.example.com.
- 2001:db8::[a:f] matches all IPv6 addresses from 2001:db8::a through 2001:db8::f.

If leading zeros are included in numeric ranges, they are used in the pattern. The second example above does not match `server1.example.com` but does match `server07.example.com`. To illustrate this, the following example uses ranges to simplify the `[usa]` and `[canada]` group definitions from the earlier example:

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

[canada]
ontario[01:02].example.com
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Verifying the Inventory

When in doubt, use the `ansible` command to verify a machine's presence in the inventory:

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

  hosts (0):
```

You can run the following command to list all hosts in a group:

```
[user@controlnode ~]$ ansible canada --list-hosts
  hosts (2):
    ontario01.example.com
    ontario02.example.com
```

Important

If the inventory contains a host and a host group with the same name, the `ansible` command prints a warning and targets the host. The host group is ignored.

There are various ways to deal with this situation, the easiest being to ensure that host groups do not use the same names as hosts in the inventory.

Overriding the Location of the Inventory

The `/etc/ansible/hosts` file is considered the system's default static inventory file. However, normal practice is not to use that file but to define a different location for inventory files in your Ansible configuration file. This is covered in the next section.

The `ansible` and `ansible-playbook` commands that you use to run Ansible ad hoc commands and playbooks later in the course can also specify the location of an inventory file on the command line with the `--inventory` *PATHNAME* or `-i` *PATHNAME* option, where PATHNAME is the path to the desired inventory file.

Defining Variables in the Inventory

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

Describing a Dynamic Inventory

Ansible inventory information can also be dynamically generated, using information provided by external databases. The open source community has written a number of dynamic inventory scripts that are available from the upstream Ansible project. If those scripts do not meet your needs, you can also write your own.

For example, a dynamic inventory program could contact your Red Hat Satellite server or Amazon EC2 account, and use information stored there to construct an Ansible inventory. Because the

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

program does this when you run Ansible, it can populate the inventory with up-to-date information provided by the service as new hosts are added and old hosts are removed.

References

Inventory: Ansible Documentation

http://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Building an Ansible Inventory

In this exercise, you will create a new static inventory containing hosts and groups.

Outcomes

You should be able to create default and custom static inventories.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab deploy-inventory start` command. This start script ensures that the managed hosts, `servera`, `serverb`, `serverc`, and `serverd`, are reachable on the network.

```
[student@workstation ~]$ lab deploy-inventory start
```

Instructions

1. Modify `/etc/ansible/hosts` to include `servera.lab.example.com` as a managed host.

   1.1. Add `servera.lab.example.com` to the end of the default inventory file, `/etc/ansible/hosts`.

```
[student@workstation ~]$ sudo vim /etc/ansible/hosts
...output omitted...
## db-[99:101]-node.example.com

servera.lab.example.com
```

   1.2. Continue editing the `/etc/ansible/hosts` inventory file by adding a `[webservers]` group to the bottom of the file with `serverb.lab.example.com` server as a group member. Save and exit when complete.

```
...output omitted...
## db-[99:101]-node.example.com

servera.lab.example.com

[webservers]
serverb.lab.example.com
```

2. Verify the managed hosts in the `/etc/ansible/hosts` inventory file.

   2.1. Use the `ansible all --list-hosts` command to list all managed hosts in the default inventory file.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ ansible all --list-hosts
  hosts (2):
    servera.lab.example.com
    serverb.lab.example.com
```

2.2. Use the `ansible ungrouped --list-hosts` command to list only managed hosts that do not belong to a group.

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

2.3. Use the `ansible webservers --list-hosts` command to list only managed hosts that belong to the `webservers` group.

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

3. Create a custom static inventory file named `inventory` in the `/home/student/deploy-inventory` working directory.

Information about your four managed hosts is listed in the following table. You will assign each host to multiple groups for management purposes based on the purpose of the host, the city where it is located, and the deployment environment to which it belongs.

In addition, groups for US cities (Raleigh and Mountain View) must be set up as children of the group `us` so that hosts in the United States can be managed as a group.

Server Inventory Specifications

Host name	Purpose	Location	Environment
`servera.lab.example.com`	Web server	Raleigh	Development
`serverb.lab.example.com`	Web server	Raleigh	Testing
`serverc.lab.example.com`	Web server	Mountain View	Production
`serverd.lab.example.com`	Web server	London	Production

3.1. Create the `/home/student/deploy-inventory` working directory, and change into it.

```
[student@workstation ~]$ mkdir ~/deploy-inventory
[student@workstation ~]$ cd ~/deploy-inventory
[student@workstation deploy-inventory]$
```

3.2. Create an `inventory` file in the `/home/student/deploy-inventory` working directory. Use the Server Inventory Specifications table as a guide. Edit the `inventory` file and add the following content:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[webservers]
server[a:d].lab.example.com

[raleigh]
servera.lab.example.com
serverb.lab.example.com

[mountainview]
serverc.lab.example.com

[london]
serverd.lab.example.com

[development]
servera.lab.example.com

[testing]
serverb.lab.example.com

[production]
serverc.lab.example.com
serverd.lab.example.com

[us:children]
raleigh
mountainview
```

4. Use variations of the `ansible` *`host-or-group`* `-i inventory --list-hosts` command to verify the managed hosts and groups in the custom `/home/student/deploy-inventory/inventory` inventory file.

Important

Your `ansible` command must include the `-i inventory` option. This makes `ansible` use your `inventory` file in the current working directory instead of the system `/etc/ansible/hosts` inventory file.

4.1. Use the `ansible all -i inventory --list-hosts` command to list all managed hosts.

```
[student@workstation deploy-inventory]$ ansible all -i inventory --list-hosts
  hosts (4):
    servera.lab.example.com
    serverb.lab.example.com
    serverc.lab.example.com
    serverd.lab.example.com
```

4.2. Use the `ansible ungrouped -i inventory --list-hosts` command to list all managed hosts listed in the inventory file but are not part of a group. There are no ungrouped managed hosts in this inventory file.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation deploy-inventory]$ ansible ungrouped -i inventory \
> --list-hosts
 [WARNING]: No hosts matched, nothing to do

  hosts (0):
```

4.3. Use the `ansible development -i inventory --list-hosts` command to list all managed hosts listed in the `development` group.

```
[student@workstation deploy-inventory]$ ansible development -i inventory \
> --list-hosts
  hosts (1):
    servera.lab.example.com
```

4.4. Use the `ansible testing -i inventory --list-hosts` command to list all managed hosts listed in the `testing` group.

```
[student@workstation deploy-inventory]$ ansible testing -i inventory \
> --list-hosts
  hosts (1):
    serverb.lab.example.com
```

4.5. Use the `ansible production -i inventory --list-hosts` command to list all managed hosts listed in the `production` group.

```
[student@workstation deploy-inventory]$ ansible production -i inventory \
> --list-hosts
  hosts (2):
    serverc.lab.example.com
    serverd.lab.example.com
```

4.6. Use the `ansible us -i inventory --list-hosts` command to list all managed hosts listed in the `us` group.

```
[student@workstation deploy-inventory]$ ansible us -i inventory --list-hosts
  hosts (3):
    servera.lab.example.com
    serverb.lab.example.com
    serverc.lab.example.com
```

4.7. You are encouraged to experiment with other variations to confirm managed host entries in the custom inventory file.

Finish

On `workstation`, run the `lab deploy-inventory finish` script to clean up this exercise.

```
[student@workstation ~]$ lab deploy-inventory finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Ansible Configuration Files

Objectives

After completing this section, you should be able to describe where Ansible configuration files are located, how Ansible selects them, and edit them to apply changes to default settings.

Configuring Ansible

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

Using /etc/ansible/ansible.cfg

The *ansible* package provides a base configuration file located at `/etc/ansible/ansible.cfg`. This file is used if no other configuration file is found.

Using ~/.ansible.cfg

Ansible looks for a `.ansible.cfg` file in the user's home directory. This configuration is used instead of the `/etc/ansible/ansible.cfg` if it exists and if there is no `ansible.cfg` file 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 stored 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 an inventory and 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 the ANSIBLE_CONFIG environment variable

You can use 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 that the variable specifies instead of any of the previously mentioned configuration files.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Configuration File Precedence

The search order for a configuration file is the reverse of the preceding list. The first file located in the search order is the one that Ansible selects. Ansible only uses configuration settings from the first file that it finds.

Any file specified by the `ANSIBLE_CONFIG` environment variable overrides all other configuration files. If that variable is not set, the directory in which the `ansible` command was run is then checked for an `ansible.cfg` file. If that file is not present, the user's home directory is checked for a `.ansible.cfg` file. The global `/etc/ansible/ansible.cfg` file is only used if no other configuration file is found. If the `/etc/ansible/ansible.cfg` configuration file is not present, Ansible contains defaults which it uses.

Because of the multitude of locations in which Ansible configuration files can be placed, it can be confusing which configuration file is being used by Ansible. You can run the `ansible --version` command to clearly identify which version of Ansible is installed, and which configuration file is being used.

```
[user@controlnode ~]$ ansible --version
ansible 2.9.21
  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.

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

Ansible only uses settings from the configuration file with the highest precedence. Even if other files with lower precedence exist, their settings are ignored and not combined with those in the selected configuration file. Therefore, if you choose to create your own configuration file in favor of the global `/etc/ansible/ansible.cfg` configuration file, you need to duplicate all desired settings from that file to your own user-level configuration file. Settings not defined in the user-level configuration file remain set to the built-in defaults, even if they are set to some other value by the global configuration file.

Managing Settings in the 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. For basic operation use the following two sections:

- `[defaults]` sets defaults for Ansible operation
- `[privilege_escalation]` configures how Ansible performs privilege escalation on managed hosts

For example, the following is a typical `ansible.cfg` file:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

The directives in this file are explained in the following table:

Ansible Configuration

Directive	Description
`inventory`	Specifies the path to the inventory file.
`remote_user`	The name of the user to log in as on the managed hosts. If not specified, the current user's name is used.
`ask_pass`	Whether or not to prompt for an SSH password. Can be `false` if using SSH public key authentication.
`become`	Whether to automatically switch user on the managed host (typically to `root`) after connecting. This can also be specified by a play.
`become_method`	How to switch user (typically `sudo`, which is the default, but `su` is an option).
`become_user`	The user to switch to on the managed host (typically `root`, which is the default).
`become_ask_pass`	Whether to prompt for a password for your `become_method`. Defaults to `false`.

Configuring Connections

Ansible needs to know how to communicate with its managed hosts. One of the most common reasons to change the configuration file is to control which methods and users Ansible uses to administer managed hosts. Some of the information needed includes:

- The location of the inventory that lists the managed hosts and host groups
- Which connection protocol to use to communicate with the managed hosts (by default, SSH), and whether or not a nonstandard network port is needed to connect to the server
- Which remote user to use on the managed hosts; this could be `root` or it could be an unprivileged user
- If the remote user is unprivileged, Ansible needs to know if it should try to escalate privileges to `root` and how to do it (for example, by using `sudo`)
- Whether or not to prompt for an SSH password or `sudo` password to log in or gain privileges

Inventory Location

In the `[defaults]` section, the `inventory` directive can point directly to a static inventory file, or to a directory containing multiple static inventory files and dynamic inventory scripts.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[defaults]
inventory = ./inventory
```

Connection Settings

By default, Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the `[defaults]` section.

By default, Ansible attempts to connect to the managed host using the same user name as the local user running the Ansible commands. To specify a different remote user, set the `remote_user` parameter to that user name.

If the local user running Ansible has private SSH keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in. If that is not the case, you can configure Ansible to prompt the local user for the password used by the remote user by setting the directive `ask_pass = true`.

```
[defaults]
inventory = ./inventory

remote_user = root
ask_pass = true
```

Assuming that you are using a Linux control node and OpenSSH on your managed hosts, if you can log in as the remote user with a password then you can probably set up SSH key-based authentication, which would allow you to set `ask_pass = false`.

The first step is to make sure that the user on the control node has an SSH key pair configured in `~/.ssh`. You can run the `ssh-keygen` command to accomplish this.

For a single existing managed host, you can install your public key on the managed host and use the `ssh-copy-id` command to populate your local `~/.ssh/known_hosts` file with its host key, as follows:

```
[user@controlnode ~]$ ssh-copy-id root@web1.example.com
The authenticity of host 'web1.example.com (192.168.122.181)' can't be
 established.
ECDSA key fingerprint is 70:9c:03:cd:de:ba:2f:11:98:fa:a0:b3:7c:40:86:4b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
 out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
 now it is to install the new keys
root@web1.example.com's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@web1.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Note

You can also use an Ansible Playbook to deploy your public key to the `remote_user` account on *all* managed hosts using the `authorized_key` module.

This course has not covered Ansible Playbooks in detail yet. A play that ensures that your public key is deployed to the managed hosts' `root` accounts might read as follows:

```
- name: Public key is deployed to managed hosts for Ansible
  hosts: all

  tasks:
    - name: Ensure key is in root's ~/.ssh/authorized_hosts
      authorized_key:
        user: root
        state: present
        key: '{{ item }}'
      with_file:
        - ~/.ssh/id_rsa.pub
```

Because the managed host would not have SSH key-based authentication configured yet, you would have to run the playbook using the `ansible-playbook` command with the `--ask-pass` option in order for the command to authenticate as the remote user.

Escalating Privileges

For security and auditing reasons, Ansible might need to connect to remote hosts as an unprivileged user before escalating privileges to get administrative access as `root`. This can be set up in the `[privilege_escalation]` section of the Ansible configuration file.

To enable privilege escalation by default, set the directive `become = true` in the configuration file. Even if this is set by default, there are various ways to override it when running ad hoc commands or Ansible Playbooks. (For example, there might be times when you want to run a task or play that does not escalate privileges.)

The `become_method` directive specifies how to escalate privileges. Several options are available, but the default is to use `sudo`. Likewise, the `become_user` directive specifies which user to escalate to, but the default is `root`.

If the `become_method` mechanism chosen requires the user to enter a password to escalate privileges, you can set the `become_ask_pass = true` directive in the configuration file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Note

On Red Hat Enterprise Linux 7, the default configuration of `/etc/sudoers` grants all users in the `wheel` group the ability to use `sudo` to become `root` after entering their password.

One way to enable a user (`someuser` in the following example) to use `sudo` to become `root` without a password is to install a file with the appropriate directives into the `/etc/sudoers.d` directory (owned by `root`, with octal permissions 0400):

```
## password-less sudo for Ansible user
someuser ALL=(ALL) NOPASSWD:ALL
```

Think through the security implications of whatever approach you choose for privilege escalation. Different organizations and deployments might have different trade-offs to consider.

The following example `ansible.cfg` file assumes that you can connect to the managed hosts as `someuser` using SSH key-based authentication, and that `someuser` can use `sudo` to run commands as `root` without entering a password:

```
[defaults]
inventory = ./inventory
remote_user = someuser
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Non-SSH Connections

The protocol used by Ansible to connect to managed hosts is set by default to `smart`, which determines the most efficient way to use SSH. This can be set to other values in a number of ways.

For example, there is one exception to the rule that SSH is used by default. If you do not have `localhost` in your inventory, Ansible sets up an *implicit localhost* entry to allow you to run ad hoc commands and playbooks that target `localhost`. This special inventory entry is not included in the `all` or `ungrouped` host groups. In addition, instead of using the `smart` SSH connection type, Ansible connects to it using the special `local` connection type by default.

```
[user@controlnode ~]$ ansible localhost --list-hosts
[WARNING]: provided hosts list is empty, only localhost is available

  hosts (1):
    localhost
```

The `local` connection type ignores the `remote_user` setting and runs commands directly on the local system. If privilege escalation is being used, it runs `sudo` from the user account that

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

ran the Ansible command, not `remote_user`. This can lead to confusion if the two users have different `sudo` privileges.

If you want to make sure that you connect to `localhost` using SSH like other managed hosts, one approach is to list it in your inventory. But, this includes it in the `all` and `ungrouped` groups, which you may not want to do.

Another approach is to change the protocol used to connect to `localhost`. The best way to do this is to set the `ansible_connection` *host variable* for `localhost`. To do this, in the directory from which you run Ansible commands, create a `host_vars` subdirectory. In that subdirectory, create a file named `localhost`, containing the line `ansible_connection: smart`. This ensures that the `smart` (SSH) connection protocol is used instead of `local` for `localhost`.

You can use this the other way around as well. If you have `127.0.0.1` listed in your inventory, by default you will connect to it using `smart`. You can also create a `host_vars/127.0.0.1` file containing the line `ansible_connection: local` and it will use `local` instead.

Host variables are covered in more detail later in the course.

Note

You can also use *group variables* to change the connection type for an entire host group. This can be done by placing files with the same name as the group in a `group_vars` directory, and ensuring that those files contain settings for the connection variables.

For example, you might want all your Microsoft Windows managed hosts to use the `winrm` protocol and port 5986 for connections. To configure this, you could put all of those managed hosts in group `windows`, and then create a file named `group_vars/windows` containing the following lines:

```
ansible_connection: winrm
ansible_port: 5986
```

Configuration File Comments

There are two comment characters allowed by Ansible configuration files: the hash or number sign (#) and the semicolon (;).

The number sign at the start of a line comments out the entire line. It must not be on the same line with a directive.

The semicolon character comments out everything to the right of it on the line. It can be on the same line as a directive, as long as that directive is to its left.

References

`ansible`(1), `ansible-config`(1), `ssh-keygen`(1), and `ssh-copy-id`(1) man pages

Configuration file: Ansible Documentation
https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Ansible Configuration Files

In this exercise, you will customize your Ansible environment by editing an Ansible configuration file.

Outcomes

You should be able to create a configuration file to configure your Ansible environment with persistent custom settings.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab deploy-manage start` command. This script ensures that the managed host, `servera`, is reachable on the network.

```
[student@workstation ~]$ lab deploy-manage start
```

Instructions

1. Create the `/home/student/deploy-manage` directory, which will contain the files for this exercise. Change to this newly created directory.

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

2. In your `/home/student/deploy-manage` directory, use a text editor to start editing a new file, `ansible.cfg`.

   Create a `[defaults]` section in that file. In that section, add a line which uses the `inventory` directive to specify the `./inventory` file as the default inventory.

```
[defaults]
inventory = ./inventory
```

   Save your work and exit the text editor.

3. In the `/home/student/deploy-manage` directory, use a text editor to start editing the new static inventory file, `inventory`.

   The static inventory should contain four host groups:

   - `[myself]` should contain the host `localhost`.
   - `[intranetweb]` should contain the host `servera.lab.example.com`.
   - `[internetweb]` should contain the host `serverb.lab.example.com`.
   - `[web]` should contain the `intranetweb` and `internetweb` host groups.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3.1. In `/home/student/deploy-manage/inventory`, create the `myself` host group by adding the following lines:

```
[myself]
localhost
```

3.2. In `/home/student/deploy-manage/inventory`, create the `intranetweb` host group by adding the following lines:

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

3.3. In `/home/student/deploy-manage/inventory`, create the `internetweb` host group by adding the following lines:

```
[internetweb]
serverb.lab.example.com
```

3.4. In `/home/student/deploy-manage/inventory`, create the `web` host group by adding the following lines:

```
[web:children]
intranetweb
internetweb
```

3.5. Confirm that your final `inventory` file looks like the following:

```
[myself]
localhost

[intranetweb]
servera.lab.example.com

[internetweb]
serverb.lab.example.com

[web:children]
intranetweb
internetweb
```

Save your work and exit the text editor.

4. Use the `ansible` command with the `--list-hosts` option to test the configuration of your inventory file's host groups. This does not actually connect to those hosts.

```
[student@workstation deploy-manage]$ ansible myself --list-hosts
  hosts (1):
    localhost
[student@workstation deploy-manage]$ ansible intranetweb --list-hosts
  hosts (1):
    servera.lab.example.com
[student@workstation deploy-manage]$ ansible internetweb --list-hosts
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  hosts (1):
    serverb.lab.example.com
[student@workstation deploy-manage]$ ansible web --list-hosts
  hosts (2):
    servera.lab.example.com
    serverb.lab.example.com
[student@workstation deploy-manage]$ ansible all --list-hosts
  hosts (3):
    localhost
    servera.lab.example.com
    serverb.lab.example.com
```

5. Open the `/home/student/deploy-manage/ansible.cfg` file in a text editor. Add a `[privilege_escalation]` section to configure Ansible to automatically use the `sudo` command to switch from `student` to `root` when running tasks on the managed hosts. Ansible should also be configured to prompt you for the password that `student` uses for the `sudo` command.

   5.1. Create the `[privilege_escalation]` section in the `/home/student/deploy-manage/ansible.cfg` configuration file by adding the following entry:

```
[privilege_escalation]
```

   5.2. Enable privilege escalation by setting the `become` directive to `true`.

```
become = true
```

   5.3. Set the privilege escalation to use the `sudo` command by setting the `become_method` directive to `sudo`.

```
become_method = sudo
```

   5.4. Set the privilege escalation user by setting the `become_user` directive to `root`.

```
become_user = root
```

   5.5. Enable prompting for the privilege escalation password by setting the `become_ask_pass` directive to `true`.

```
become_ask_pass = true
```

   5.6. Confirm that the complete `ansible.cfg` file looks like the following:

```
[defaults]
inventory = ./inventory

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

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Save your work and exit the text editor.

6. Run the `ansible --list-hosts` command again to verify that you are now prompted for the `sudo` password.

   When prompted for the `sudo` password, enter `student`, even though it is not used for this dry run.

```
[student@workstation deploy-manage]$ ansible intranetweb --list-hosts
BECOME password: student
  hosts (1):
    servera.lab.example.com
```

Finish

On `workstation`, run the `lab deploy-manage finish` script to clean up this exercise.

```
[student@workstation ~]$ lab deploy-manage finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Running Ad Hoc Commands

Objectives

After completing this section, you should be able to run a single Ansible automation task using an ad hoc command and explain some use cases for ad hoc commands.

Running Ad Hoc Commands with Ansible

An *ad hoc command* is a way of executing a single Ansible task quickly, one that you do not need to save to run again later. They are simple, online operations that can be run without writing a playbook.

Ad hoc commands are useful for quick tests and changes. For example, you can use an ad hoc command to make sure that a certain line exists in the `/etc/hosts` file on a group of servers. You could use another ad hoc command to efficiently restart a service on many different machines, or to ensure that a particular software package is up-to-date.

Ad hoc commands are very useful for quickly performing simple tasks with Ansible. They do have their limits, and in general you will want to use Ansible Playbooks to realize the full power of Ansible. In many situations, however, ad hoc commands are exactly the tool you need to perform simple tasks quickly.

Running Ad Hoc Commands

Use the `ansible` command to run ad hoc commands:

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

The *host-pattern* argument is used to specify the managed hosts on which the ad hoc command should be run. It could be a specific managed host or host group in the inventory. You have already seen this used in conjunction with the `--list-hosts` option, which shows you which hosts are matched by a particular host pattern. You have also already seen that you can use the `-i` option to specify a different inventory location to use than the default in the current Ansible configuration file.

The `-m` option takes as an argument the name of the *module* that Ansible should run on the targeted hosts. Modules are small programs that are executed to implement your task. Some modules need no additional information, but others need additional arguments to specify the details of their operation. The `-a` option takes a list of those arguments as a quoted string.

One of the simplest ad hoc commands uses the `ping` module. This module does not do an ICMP ping, but checks to see if you can run Python-based modules on managed hosts. For example, the following ad hoc command determines whether all managed hosts in the inventory can run standard modules:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@controlnode ~]$ ansible all -m ping
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Performing Tasks with Modules Using Ad Hoc Commands

Modules are the tools that ad hoc commands use to accomplish tasks. Ansible provides hundreds of modules which do different things. You can usually find a tested, special-purpose module that does what you need as part of the standard installation.

The `ansible-doc -l` command lists all modules installed on a system. You can use `ansible-doc` to view the documentation of particular modules by name, and find information about what arguments the modules take as options. For example, the following command displays documentation for the `ping` module:

```
[user@controlnode ~]$ ansible-doc ping
> PING    (/usr/lib/python3.6/site-packages/ansible/modules/system/ping.py)

        A trivial test module, this module always returns `pong' on successful
 contact. It does not make sense in playbooks, but it is useful from `/usr/bin/
ansible' to
        verify the ability to login and that a usable Python is configured. This
 is NOT ICMP ping, this is just a trivial test module that requires Python on the
        remote-node. For Windows targets, use the [win_ping] module instead. For
 Network targets, use the [net_ping] module instead.

  * This module is maintained by The Ansible Core Team
OPTIONS (= is mandatory):

- data
        Data to return for the `ping' return value.
        If this parameter is set to `crash', the module will cause an exception.
        [Default: pong]
        type: str


SEE ALSO:
      * Module net_ping
           The official documentation on the net_ping module.
           https://docs.ansible.com/ansible/2.9/modules/net_ping_module.html
      * Module win_ping
           The official documentation on the win_ping module.
           https://docs.ansible.com/ansible/2.9/modules/win_ping_module.html


AUTHOR: Ansible Core Team, Michael DeHaan
        METADATA:
          status:
          - stableinterface
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
            supported_by: core


EXAMPLES:

# Test we can logon to 'webservers' and execute python with json lib.
# ansible webservers -m ping

# Example from an Ansible Playbook
- ping:

# Induce an exception to see what happens
- ping:
    data: crash


RETURN VALUES:

ping:
    description: value provided with the data parameter
    returned: success
    type: str
    sample: pong
```

To learn more about modules, access the online Ansible documentation at http://docs.ansible.com/ansible/2.9/modules/modules_by_category.html.

The following table lists a number of useful modules as examples. Many others exist.

Ansible Modules

Module category	Modules
Files modules	• `copy`: Copy a local file to the managed host • `file`: Set permissions and other properties of files • `lineinfile`: Ensure a particular line is or is not in a file • `synchronize`: Synchronize content using `rsync`
Software package modules	• `package`: Manage packages using autodetected package manager native to the operating system • `yum`: Manage packages using the YUM package manager • `apt`: Manage packages using the APT package manager • `dnf`: Manage packages using the DNF package manager • `gem`: Manage Ruby gems • `pip`: Manage Python packages from PyPI
System modules	• `firewalld`: Manage arbitrary ports and services using `firewalld` • `reboot`: Reboot a machine • `service`: Manage services • `user`: Add, remove, and manage user accounts

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Module category	Modules
Net Tools modules	• `get_url`: Download files over HTTP, HTTPS, or FTP • `nmcli`: Manage networking • `uri`: Interact with web services

Most modules take arguments. You can find the list of arguments available for a module in the module's documentation. Ad hoc commands pass arguments to modules using the `-a` option. 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.

For example, the following ad hoc command uses the `user` module to ensure that the `newbie` user exists and has UID 4000 on `servera.lab.example.com`:

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "comment": "",
    "createhome": true,
    "group": 4000,
    "home": "/home/newbie",
    "name": "newbie",
    "shell": "/bin/bash",
    "state": "present",
    "system": false,
    "uid": 4000
}
```

Most modules are *idempotent*, which means that they can be run safely multiple times, and if the system is already in the correct state, they do nothing. For example, if you run the previous ad hoc command again, it should report no change:

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "append": false,
    "changed": false
    "comment": "",
    "group": 4000,
    "home": "/home/newbie",
    "move_home": false,
    "name": "newbie",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 4000
}
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Running Arbitrary Commands on Managed Hosts

The `command` module allows administrators to run arbitrary commands on the command line of managed hosts. The command to be run is specified as an argument to the module using the `-a` option. For example, the following command runs the `hostname` command on the managed hosts referenced by the `mymanagedhosts` host pattern.

```
[user@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname
host1.lab.example.com | CHANGED | rc=0 >>
host1.lab.example.com
host2.lab.example.com | CHANGED | 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, showing the name of the managed host that the ad hoc operation ran 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 might find it useful to have a single line of output for each operation performed on a managed host. Use the `-o` option to display the output of Ansible ad hoc commands in a single line format.

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

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.

Note

If an ad hoc command does not specify which module to use with the `-m` option, Ansible uses the `command` module by default.

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

The following example illustrates the difference between the `command` and `shell` modules. If you try to execute the built-in Bash command `set` with these two modules, it only succeeds with the `shell` module.

```
[user@controlnode ~]$ ansible localhost -m command -a set
localhost | FAILED | rc=2 >>
[Errno 2] No such file or directory
[user@controlnode ~]$ ansible localhost -m shell -a set
localhost | CHANGED | rc=0 >>
BASH=/bin/sh
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interact
ive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
...output omitted...
```

Both `command` and `shell` modules require a working Python installation on the managed host. A third module, `raw`, can run commands directly using the remote shell, bypassing the module subsystem. This is useful when managing systems that cannot have Python installed (for example, a network router). It can also be used to install Python on a host.

Important

In most circumstances, it is a recommended practice that you avoid the `command`, `shell`, and `raw` "run command" modules.

Most other modules are idempotent and can perform change tracking automatically. They can test the state of systems and do nothing if those systems are already in the correct state. By contrast, it is much more complicated to use "run command" modules in a way that is idempotent. Depending upon them makes it harder for you to be confident that rerunning an ad hoc command or playbook would not cause an unexpected failure. When a `shell` or `command` module runs, it typically reports a `CHANGED` status based on whether it thinks it affected machine state.

There are times when "run command" modules are valuable tools and a good solution to a problem. If you do need to use them, it is probably best to try to use the `command` module first, resorting to `shell` or `raw` modules only if you need their special features.

Configuring Connections for Ad Hoc Commands

The directives 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, they take precedence over the directive configured in the Ansible configuration file. The following table shows the analogous command-line options for each configuration file directive.

Ansible Command-line Options

Configuration file directives	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 directives using command-line options, their currently defined values can be determined by consulting the output of `ansible --help`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@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

Working with Patterns: Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/intro_patterns.html

Introduction to Ad-Hoc Commands: Ansible Documentation
http://docs.ansible.com/ansible/2.9/user_guide/intro_adhoc.html

Module Index: Ansible Documentation
http://docs.ansible.com/ansible/2.9/modules/modules_by_category.html

command - Executes a command on a remote node: Ansible Documentation
http://docs.ansible.com/ansible/2.9/modules/command_module.html

shell - Execute commands in nodes: Ansible Documentation
http://docs.ansible.com/ansible/2.9/modules/shell_module.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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.

You will execute 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 to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab deploy-adhoc start` command. This script ensures that the managed host `servera` is reachable on the network. It also creates and populates the `/home/student/deploy-adhoc` working directory with materials used in this exercise.

```
[student@workstation ~]$ lab deploy-adhoc start
```

Instructions

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 -l -U devops
...output omitted...
User devops may run the following commands on workstation:
    (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 -l
...output omitted...
User devops may run the following commands on servera:
    (ALL) NOPASSWD: ALL
[devops@servera ~]$ exit
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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

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

```
[student@workstation ~]$ cd ~/deploy-adhoc
[student@workstation deploy-adhoc]$ cat ansible.cfg
[defaults]
inventory=inventory
[student@workstation deploy-adhoc]$ cat inventory
[control_node]
localhost

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

The configuration file uses the directory's `inventory` file as the Ansible inventory. Note that Ansible is not yet configured to use privilege escalation.

3. Using the `all` host group and the `ping` module, execute an ad hoc command that ensures all managed hosts can run Ansible modules using Python.

```
[student@workstation deploy-adhoc]$ ansible all -m ping
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

4. Using the `command` module, execute an ad hoc command on `workstation` to identify the user account that Ansible uses to perform operations on managed hosts. Use the `localhost` host pattern to connect to `workstation` for the ad hoc command execution. Because you are connecting locally, `workstation` is both the control node and managed host.

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

Notice that the ad hoc command was performed on the managed host as the `student` user.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. 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 deploy-adhoc]$ ansible localhost -m command -a 'id' -u devops
localhost | CHANGED | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Notice that the ad hoc command was performed on the managed host as the `devops` user.

6. Using the `copy` module, execute an ad hoc command on `workstation` to change the contents of the `/etc/motd` file so that it consists of the string "Managed by Ansible" followed by a newline. Execute the command using the `devops` account, but do not use the `--become` option to switch to `root`. The ad hoc command should fail due to lack of permissions.

```
[student@workstation deploy-adhoc]$ ansible localhost -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops
localhost | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "msg": "Destination /etc not writable"
}
```

The ad hoc command failed because the `devops` user does not have permission to write to the file.

7. Run the command again using privilege escalation. You could fix the settings in the `ansible.cfg` file, but for this example just use appropriate command-line options of the `ansible` command.

Using the `copy` module, execute the previous command on `workstation` to change the contents of the `/etc/motd` file so that it consists of the string "Managed by Ansible" followed by a newline. Use the `devops` user to make the connection to the managed host, but perform the operation as the `root` user using the `--become` option. The use of the `--become` option is sufficient because the default value for the `become_user` directive is set to `root` in the `/etc/ansible/ansible.cfg` file.

```
[student@workstation deploy-adhoc]$ ansible localhost -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-
tmp-1558954193.0260043-131348380629718/source",
    "state": "file",
    "uid": 0
}
```

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

8. Run the previous ad hoc command again on all hosts using the `all` host group. This ensures that `/etc/motd` on both `workstation` and `servera` consist of the text "`Managed by Ansible`".

```
[student@workstation deploy-adhoc]$ ansible all -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
servera.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "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-1558954250.7893758-136255396678462/source",
    "state": "file",
    "uid": 0
}
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/motd",
    "secontext": "system_u:object_r:etc_t:s0",
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "size": 19,
    "state": "file",
    "uid": 0
}
```

You should see SUCCESS for `localhost` and CHANGED for `servera`. However, `localhost` should report `"changed": false` because the file is already in the correct state. Conversely, `servera` should report `"changed": true` because the ad hoc command updated the file to the correct state.

9. Using the `command` module, execute an ad hoc command to run `cat /etc/motd` to verify that the contents of the file have been successfully modified on both `workstation` and `servera`. Use the `all` host group and the `devops` user to specify and make the connection to the managed hosts. You do not need privilege escalation for this command to work.

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

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

Finish

On `workstation`, run the `lab deploy-adhoc finish` script to clean up this exercise.

```
[student@workstation ~]$ lab deploy-adhoc finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Writing and Running Playbooks

Objectives

After completing this section, you should be able to write a basic Ansible Playbook and run it using the `ansible-playbook` command.

Ansible Playbooks and Ad Hoc Commands

Ad hoc commands can run a single, simple task against a set of targeted hosts as a one-time command. The real power of Ansible, however, is in learning how to use playbooks to run multiple, complex tasks against a set of targeted hosts in an easily repeatable manner.

A *task* is the application of a module to perform a specific unit of work. A *play* is a sequence of tasks to be applied, in order, to one or more hosts selected from your inventory. A *playbook* is a text file containing a list of one or more plays to run in a specific order.

Plays allow you to change a lengthy, complex set of manual administrative tasks into an easily repeatable routine with predictable and successful outcomes. In a playbook, you can save the sequence of tasks in a play into a human-readable and immediately runnable form. The tasks themselves, because of the way in which they are written, document the steps needed to deploy your application or infrastructure.

Formatting an Ansible Playbook

To help you understand the format of a playbook, review this ad hoc command from a previous chapter:

```
[student@workstation ~]$ ansible -m user -a "name=newbie uid=4000 state=present" \
> servera.lab.example.com
```

This can be rewritten as a single task play and saved in a playbook. The resulting playbook appears as follows:

```
---
- name: Configure important user consistently
  hosts: servera.lab.example.com
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        state: present
```

A playbook is a text file written in YAML format, and is normally saved with the extension `yml`. The playbook uses indentation with space characters to indicate the structure of its data. YAML does not place strict requirements on how many spaces are used for the indentation, but there are two basic rules.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation.
- Items that are children of another item must be indented more than their parents.

You can also add blank lines for readability.

Important

Only the space character can be used for indentation; tab characters are not allowed.

If you use the `vi` text editor, you can apply some settings which might make it easier to edit your playbooks. For example, you can add the following line to your `$HOME/.vimrc` file, and when `vi` detects that you are editing a YAML file, it performs a 2-space indentation when you press the `Tab` key and autoindents subsequent lines.

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

A playbook begins with a line consisting of three dashes (`---`) as a start of document marker. It may end with three dots (`...`) as an end of document marker, although in practice this is often omitted.

In between those markers, the playbook is defined as a list of plays. An item in a YAML list starts with a single dash followed by a space. For example, a YAML list might appear as follows:

```
- apple
- orange
- grape
```

In the preceding playbook example, the line after `---` begins with a dash and starts the first (and only) play in the list of plays.

The play itself is a collection of key-value pairs. Keys in the same play should have the same indentation. The following example shows a YAML snippet with three keys. The first two keys have simple values. The third has a list of three items as a value.

```
  name: just an example
  hosts: webservers
  tasks:
    - first
    - second
    - third
```

The original example play has three keys, `name`, `hosts`, and `tasks`, because these keys all have the same indentation.

The first line of the example play starts with a dash and a space (indicating the play is the first item of a list), and then the first key, the `name` attribute. The `name` key associates an arbitrary string with the play as a label. This identifies what the play is for. The `name` key is optional, but is recommended because it helps to document your playbook. This is especially useful when a playbook contains multiple plays.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Configure important user consistently
```

The second key in the play is a `hosts` attribute, which specifies the hosts against which the play's tasks are run. Like the argument for the `ansible` command, the `hosts` attribute takes a host pattern as a value, such as the names of managed hosts or groups in the inventory.

```
  hosts: servera.lab.example.com
```

Finally, the last key in the play is the `tasks` attribute, whose value specifies a list of tasks to run for this play. This example has a single task, which runs the `user` module with specific arguments (to ensure user `newbie` exists and has UID 4000).

```
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        state: present
```

The `tasks` attribute is the part of the play that actually lists, in order, the tasks to be run on the managed hosts. Each task in the list is itself a collection of key-value pairs.

In this example, the only task in the play has two keys:

- `name` is an optional label documenting the purpose of the task. It is a good idea to name all your tasks to help document the purpose of each step of the automation process.
- `user` is the module to run for this task. Its arguments are passed as a collection of key-value pairs, which are children of the module (`name`, `uid`, and `state`).

The following is another example of a `tasks` attribute with multiple tasks, using the `service` module to ensure that several network services are enabled to start at boot:

```
  tasks:
    - name: web server is enabled
      service:
        name: httpd
        enabled: true

    - name: NTP server is enabled
      service:
        name: chronyd
        enabled: true

    - name: Postfix is enabled
      service:
        name: postfix
        enabled: true
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

The order in which the plays and tasks are listed in a playbook is important, because Ansible runs them in the same order.

The playbooks you have seen so far are basic examples, and you will see more sophisticated examples of what you can do with plays and tasks as this course continues.

Running Playbooks

The `ansible-playbook` command is used to run playbooks. The command is executed on the control node and the name of the playbook to be run is passed as an argument:

```
[student@workstation ~]$ ansible-playbook site.yml
```

When you run the playbook, 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 contents of a simple playbook, and then the result of running it.

```
[student@workstation 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
...output omitted...
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

The value of the `name` key for each play and task is displayed when the playbook is run. (The `Gathering Facts` task is a special task that the `setup` module usually runs automatically at the start of a play. This is covered later in the course.) For playbooks with multiple plays and tasks, setting `name` attributes makes it easier to monitor the progress of a playbook's execution.

You should also see that the `latest httpd version installed` task is `changed` for `servera.lab.example.com`. This means that the task changed something on that host to ensure its specification was met. In this case, it means that the *httpd* package probably was not installed or was not the latest version.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

In general, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple times. If the targeted managed hosts are already in the correct state, no changes should be made. For example, assume that the playbook from the previous example is run again:

```
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ************************************************

TASK [Gathering Facts] *********************************************************
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
```

This time, all tasks passed with status `ok` and no changes were reported.

Increasing Output Verbosity

The default output provided by the `ansible-playbook` command does not provide detailed task execution information. The `ansible-playbook -v` command provides additional information, with up to four total levels.

Configuring the Output Verbosity of Playbook Execution

Option	Description
`-v`	The task results are displayed.
`-vv`	Both task results and task configuration are displayed
`-vvv`	Includes information about connections to managed hosts
`-vvvv`	Adds extra verbosity options to the connection plug-ins, including users being used in the managed hosts to execute scripts, and what scripts have been executed

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 that you can use to verify the syntax of a playbook. The following example shows the successful syntax verification of a playbook.

```
[student@workstation ~]$ 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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to have been in ...output omitted... 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

You can use the `-C` option to perform a *dry run* of the playbook execution. 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@workstation ~]$ ansible-playbook -C webserver.yml

PLAY [play to setup web server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

References

`ansible-playbook`(1) man page

Intro to Playbooks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html

Playbooks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks.html

Check Mode ("Dry Run") — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_checkmode.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Writing and Running Playbooks

In this exercise, you will write and run an Ansible Playbook.

Outcomes

You should be able to write a playbook using basic YAML syntax and Ansible Playbook structure, and successfully run it with the `ansible-playbook` command.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab playbook-basic start` command. This function ensures that the managed hosts, `serverc.lab.example.com` and `serverd.lab.example.com` are reachable on the network. It also ensures that the correct Ansible configuration file and inventory file are installed on the control node.

```
[student@workstation ~]$ lab playbook-basic start
```

Instructions

The `/home/student/playbook-basic` working directory has been created on `workstation` for this exercise. This directory has already been populated with an `ansible.cfg` configuration file, and also an `inventory` inventory file, which defines a `web` group that includes both managed hosts listed above as members.

In this directory, use a text editor to create a playbook named `site.yml`. This playbook contains one play, which should target members of the `web` host group. The playbook should use tasks to ensure that the following conditions are met on the managed hosts:

- The *httpd* package is present, using the `yum` module.
- The local `files/index.html` file is copied to `/var/www/html/index.html` on each managed host, using the `copy` module.
- The `httpd` service is started and enabled, using the `service` module.

You can use the `ansible-doc` command to help you understand the keywords needed for each of the modules.

After the playbook is written, verify its syntax and then use `ansible-playbook` to run the playbook to implement the configuration.

1. Change to the `/home/student/playbook-basic` directory.

```
[student@workstation ~]$ cd ~/playbook-basic
[student@workstation playbook-basic]$
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2. Use a text editor to create a new playbook called `/home/student/playbook-basic/site.yml`. Start writing a play that targets the hosts in the `web` host group.

2.1. Create and open `~/playbook-basic/site.yml`. The first line of the file should be three dashes to indicate the start of the playbook.

```
---
```

2.2. The next line starts the play. It needs to start with a dash and a space before the first keyword in the play. Name the play with an arbitrary string documenting the play's purpose, using the `name` keyword.

```
---
- name: Install and start Apache HTTPD
```

2.3. Add a `hosts` keyword-value pair to specify that the play run on hosts in the inventory's `web` host group. Make sure that the `hosts` keyword is indented two spaces so it aligns with the `name` keyword in the preceding line.

The complete `site.yml` file should now appear as follows:

```
---
- name: Install and start Apache HTTPD
  hosts: web
```

3. Continue to edit the `/home/student/playbook-basic/site.yml` file, and add a `tasks` keyword and the three tasks for your play that were specified in the instructions.

3.1. Add a `tasks` keyword indented by two spaces (aligned with the `hosts` keyword) to start the list of tasks. Your file should now appear as follows:

```
---
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
```

3.2. Add the first task. Indent by four spaces, and start the task with a dash and a space, and then give the task a name, such as `httpd package is present`. Use the `yum` module for this task. Indent the module keywords two more spaces; set the package name to `httpd` and the package state to `present`. The task should appear as follows:

```
    - name: httpd package is present
      yum:
        name: httpd
        state: present
```

3.3. Add the second task. Match the format of the previous task, and give the task a name, such as `correct index.html is present`. Use the `copy` module. The module keywords should set the `src` key to `files/index.html` and the `dest` key to `/var/www/html/index.html`. The task should appear as follows:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: correct index.html is present
      copy:
        src: files/index.html
        dest: /var/www/html/index.html
```

3.4. Add the third task to start and enable the `httpd` service. Match the format of the previous two tasks, and give the new task a name, such as `httpd is started`. Use the `service` module for this task. Set the `name` key of the service to `httpd`, the `state` key to `started`, and the `enabled` key to `true`. The task should appear as follows:

```
    - name: httpd is started
      service:
        name: httpd
        state: started
        enabled: true
```

3.5. Your entire `site.yml` Ansible Playbook should match the following example. Make sure that the indentation of your play's keywords, the list of tasks, and each task's keywords are all correct.

```
---
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
    - name: httpd package is present
      yum:
        name: httpd
        state: present

    - name: correct index.html is present
      copy:
        src: files/index.html
        dest: /var/www/html/index.html

    - name: httpd is started
      service:
        name: httpd
        state: started
        enabled: true
```

Save the file and exit your text editor.

4. Before running your playbook, run the `ansible-playbook --syntax-check site.yml` command to verify that its syntax is correct. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation playbook-basic]$ ansible-playbook --syntax-check site.yml

playbook: site.yml
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. Run your playbook. Read through the output generated to ensure that all tasks completed successfully.

```
[student@workstation playbook-basic]$ ansible-playbook site.yml

PLAY [Install and start Apache HTTPD] ******************************************

TASK [Gathering Facts] *********************************************************
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [httpd package is present] ************************************************
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [correct index.html is present] *******************************************
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [httpd is started] ********************************************************
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

PLAY RECAP *********************************************************************
serverc.lab.example.com    : ok=4    changed=3    unreachable=0    failed=0
serverd.lab.example.com    : ok=4    changed=3    unreachable=0    failed=0
```

6. If all went well, you should be able to run the playbook a second time and see all tasks complete with no changes to the managed hosts.

```
[student@workstation playbook-basic]$ ansible-playbook site.yml

PLAY [Install and start Apache HTTPD] ******************************************

TASK [Gathering Facts] *********************************************************
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [httpd package is present] ************************************************
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [correct index.html is present] *******************************************
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [httpd is started] ********************************************************
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

PLAY RECAP *********************************************************************
serverc.lab.example.com    : ok=4    changed=0    unreachable=0    failed=0
serverd.lab.example.com    : ok=4    changed=0    unreachable=0    failed=0
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

7. Use the `curl` command to verify that both `serverc` and `serverd` are configured as an HTTPD server.

```
[student@workstation playbook-basic]$ curl serverc.lab.example.com
This is a test page.
[student@workstation playbook-basic]$ curl serverd.lab.example.com
This is a test page.
```

Finish

On `workstation`, run the `lab playbook-basic finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab playbook-basic finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Implementing Multiple Plays

Objectives

After completing this section, you should be able to write a playbook that uses multiple plays and per-play privilege escalation, and effectively use `ansible-doc` to learn how to use new modules to implement tasks for a play

Writing Multiple Plays

A playbook is a YAML file containing a list of one or more plays. Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory. Therefore, if a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts.

This can be very useful when orchestrating a complex deployment which may involve different tasks on different hosts. You can write a playbook that runs one play against one set of hosts, and when that finishes runs another play against another set of hosts.

Writing a playbook that contains multiple plays is very straightforward. Each play in the playbook is written as a top-level list item in the playbook. Each play is a list item containing the usual play keywords.

The following example shows a simple playbook with two plays. The first play runs against `web.example.com`, and the second play runs against `database.example.com`.

```
---
# This is a simple playbook with two plays

- name: first play
  hosts: web.example.com
  tasks:
    - name: first task
      yum:
        name: httpd
        status: present

    - name: second task
      service:
        name: httpd
        enabled: true

- name: second play
  hosts: database.example.com
  tasks:
    - name: first task
      service:
        name: mariadb
        enabled: true
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Remote Users and Privilege Escalation in Plays

Plays can use different remote users or privilege escalation settings for a play than what is specified by the defaults in the configuration file. These are set in the play itself at the same level as the `hosts` or `tasks` keywords.

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 task execution depends on various keywords in the Ansible configuration file, `/etc/ansible/ansible.cfg`. The user that runs the tasks can be defined by the `remote_user` keyword. However, if privilege escalation is enabled, other keywords such as `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 overridden by using the `remote_user` keyword within a play.

```
remote_user: remoteuser
```

Privilege Escalation Attributes

Additional keywords are also available to define privilege escalation parameters from within a playbook. The `become` boolean keyword can be used to enable or disable privilege escalation regardless of how it is defined in the Ansible configuration file. It can take `yes` or `true` to enable privilege escalation, or `no` or `false` to disable it.

```
become: true
```

If privilege escalation is enabled, the `become_method` keyword can be used to define the privilege escalation method to use during a specific play. The example below specifies that `sudo` be used for privilege escalation.

```
become_method: sudo
```

Additionally, with privilege escalation enabled, the `become_user` keyword can define the user account to use for privilege escalation within the context of a specific play.

```
become_user: privileged_user
```

The following example demonstrates the use of these keywords in a play:

```
- name: /etc/hosts is up to date
  hosts: datacenter-west
  remote_user: automation
  become: yes

  tasks:
    - name: server.example.com in /etc/hosts
      lineinfile:
        path: /etc/hosts
        line: '192.0.2.42 server.example.com server'
        state: present
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finding Modules for Tasks

Module Documentation

The large number of modules packaged with Ansible provides administrators with many tools for common administrative tasks. Earlier in this course, we discussed the Ansible documentation website at http://docs.ansible.com. The *Module Index* on the website is an easy way to browse the list of modules shipped with Ansible. For example, modules for user and service management can be found under *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 that show you how to use each module and how to set their keywords in a task.

You have already worked with the `ansible-doc` command to look up information about modules installed on the local system. As a review, 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 functions.

```
[student@workstation modules]$ ansible-doc -l
a10_server              Manage A10 Networks ... devices' server object.
a10_server_axapi3       Manage A10 Networks ... devices
a10_service_group       Manage A10 Networks ... devices' service groups.
a10_virtual_server      Manage A10 Networks ... devices' virtual servers.
...output omitted...
zfs_facts               Gather facts about ZFS datasets.
znode                   Create, ... and update znodes using ZooKeeper
zpool_facts             Gather facts about ZFS pools.
zypper                  Manage packages on SUSE and openSUSE
zypper_repository       Add and remove Zypper repositories
```

Use the `ansible-doc` *`[module name]`* command to display detailed documentation for a module. 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    (/usr/lib/python3.6/site-packages/ansible/modules/packaging/os/yum.py)

        Installs, upgrade, downgrades, removes, and lists packages and groups with
 the `yum' package manager. This module only works on Python 2. If you require
 Python
        3 support see the [dnf] module.

  * This module is maintained by The Ansible Core Team
  * note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):

- allow_downgrade
        Specify if the named package and version is allowed to downgrade a maybe
 already installed higher version of that package. Note that setting
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        allow_downgrade=True can make this module behave in a non-idempotent way.
 The task could end up with a set of packages that does not match the complete
 list of
        specified packages to install (because dependencies between the downgraded
 package and others can cause changes to the packages which were in the earlier
        transaction).
        [Default: no]
        type: bool
        version_added: 2.4

- autoremove
        If `yes', removes all "leaf" packages from the system that were originally
 installed as dependencies of user-installed packages but which are no longer
 required
        by any such package. Should be used alone or when state is `absent'
        NOTE: This feature requires yum >= 3.4.3 (RHEL/CentOS 7+)
        [Default: no]
        type: bool
        version_added: 2.7

- bugfix
        If set to `yes', and `state=latest' then only installs updates that have
 been marked bugfix related.
        [Default: no]
        version_added: 2.6

- conf_file
        The remote yum configuration file to use for the transaction.
        [Default: (null)]
        version_added: 0.6

- disable_excludes
        Disable the excludes defined in YUM config files.
        If set to `all', disables all excludes.
        If set to `main', disable excludes defined in [main] in yum.conf.
        If set to `repoid', disable excludes defined for given repo id.
        [Default: (null)]
        version_added: 2.7

- 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'.
        [Default: no]
        type: bool
        version_added: 1.2

- disable_plugin
        `Plugin' name to disable for the install/update operation. The disabled
 plugins will not persist beyond the transaction.
        [Default: (null)]
        version_added: 2.5

- disablerepo
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        `Repoid' of repositories to disable for the install/update operation.
  These repos will not persist beyond the transaction. When specifying multiple
  repos,
        separate them with a `","'.
        As of Ansible 2.7, this can alternatively be a list instead of `","'
  separated string
        [Default: (null)]
```

The `ansible-doc` command also offers the `-s` 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 this output for the `yum` module.

```
[student@workstation ~]$ ansible-doc -s yum
- name: Manages packages with the `yum' package manager
  yum:
      allow_downgrade:       # Specify if the named package ...
      autoremove:            # If `yes', removes all "leaf" packages ...
      bugfix:                # If set to `yes', ...
      conf_file:             # The remote yum configuration file ...
      disable_excludes:      # Disable the excludes ...
      disable_gpg_check:     # Whether to disable the GPG ...
      disable_plugin:        # `Plugin' name to disable ...
      disablerepo:           # `Repoid' of repositories ...
      download_only:         # Only download the packages, ...
      enable_plugin:         # `Plugin' name to enable ...
      enablerepo:            # `Repoid' of repositories to enable ...
      exclude:               # Package name(s) to exclude ...
      installroot:           # Specifies an alternative installroot, ...
      list:                  # Package name to run ...
      name:                  # A package name or package specifier ...
      releasever:            # Specifies an alternative release ...
      security:              # If set to `yes', ...
      skip_broken:           # Skip packages with ...
      state:                 # Whether to install ... or remove ... a package.
      update_cache:          # Force yum to check if cache ...
      update_only:           # When using latest, only update ...
      use_backend:           # This module supports `yum' ...
      validate_certs:        # This only applies if using a https url ...
```

Module Maintenance

Ansible ships with a large number of modules that can be used for many tasks. The upstream community is very active, and these modules may be in different stages of development. The `ansible-doc` documentation for the module is expected to specify who maintains that module in the upstream Ansible community, and what its development status is. This is indicated in the `METADATA` section at the end of the output of `ansible-doc` for that module.

The `status` field records the development status of the module:

- `stableinterface`: The module's keywords are stable, and every effort will be made not to remove keywords or change their meaning.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- `preview`: The module is in technology preview, and might be unstable, its keywords might change, or it might require libraries or web services that are themselves subject to incompatible changes.
- `deprecated`: The module is deprecated, and will no longer be available in some future release.
- `removed`: The module has been removed from the release, but a stub exists for documentation purposes to help former users migrate to new modules.

Note

The `stableinterface` status only indicates that a module's interface is stable, it does not rate the module's code quality.

The `supported_by` field records who maintains the module in the upstream Ansible community. Possible values are:

- `core`: Maintained by the "core" Ansible developers upstream, and always included with Ansible.
- `curated`: Modules submitted and maintained by partners or companies in the community. Maintainers of these modules must watch for any issues reported or pull requests raised against the module. Upstream "core" developers review proposed changes to curated modules after the community maintainers have approved the changes. Core committers also ensure that any issues with these modules due to changes in the Ansible engine are remediated. These modules are currently included with Ansible, but might be packaged separately at some point in the future.
- `community`: Modules not supported by the core upstream developers, partners, or companies, but maintained entirely by the general open source community. Modules in this category are still fully usable, but the response rate to issues is purely up to the community. These modules are also currently included with Ansible, but will be packaged separately at some point in the future.

The upstream Ansible community has an issue tracker for Ansible and its integrated modules at https://github.com/ansible/ansible/issues.

Sometimes, a module does not exist for something you want to do. As an end user, you can also write your own private modules, or get modules from a third party. Ansible searches for custom modules in the location specified by the `ANSIBLE_LIBRARY` environment variable, or if that is not set, by a `library` keyword in the current Ansible configuration file. Ansible also searches for custom modules in the `./library` directory relative to the playbook currently being run.

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

Information on writing modules is beyond the scope of this course. Documentation on how to do this is available at https://docs.ansible.com/ansible/2.9/dev_guide/developing_modules.html.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

Use the `ansible-doc` command to find and learn how to use modules for your tasks.

When possible, try to avoid the `command`, `shell`, and `raw` modules in playbooks, even though they might seem simple to use. Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.

For example, the following task using the `shell` module is not idempotent. Every time the play is run, it rewrites `/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
```

There are several ways to write tasks using the `shell` module idempotently, and sometimes making those changes and using `shell` is the best approach. 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 does not rewrite the `/etc/resolv.conf` file if it already consists of the correct content:

```
- name: Idempotent approach with copy module
  copy:
    dest: /etc/resolv.conf
    content: "nameserver 192.0.2.1\n"
```

The `copy` module tests to see if the state has already been met, and if so, it makes no changes. The `shell` module allows a lot of flexibility, but also requires more attention to ensure that it runs idempotently.

Idempotent playbooks can be run repeatedly to ensure systems are in a particular state without disrupting those systems if they already are.

Playbook Syntax Variations

The last part of this chapter investigates some variations of YAML or Ansible Playbook syntax that you might encounter.

YAML Comments

Comments can also be used to aid readability. In YAML, everything to the right of the number or hash symbol (#) is a comment. If there is content to the left of the comment, precede the number symbol with a space.

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

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

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

YAML Strings

Strings in YAML do not normally need to be put in quotation marks even if there are spaces contained in the string. You can enclose strings 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 multiline strings. You can use 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
```

You can also write multiline strings using 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: >
        This is an example
        of a long string,
        that will become
        a single sentence once folded.
```

YAML Dictionaries

You have seen collections of key-value pairs written as an indented block, as follows:

```
  name: svcrole
  svcservice: httpd
  svcport: 80
```

Dictionaries can also be written in an inline block format enclosed in curly braces, as follows:

```
  {name: svcrole, svcservice: httpd, svcport: 80}
```

In most cases the inline block format should be avoided because it is harder to read. However, there is at least one situation in which it is more commonly used. The use of *roles* is discussed later in this course. When a playbook includes a list of roles, it is more common to use this syntax to make it easier to distinguish roles included in a play from the variables being passed to a role.

YAML Lists

You have also seen lists written with the normal single-dash syntax:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  hosts:
    - servera
    - serverb
    - serverc
```

Lists also have an inline format enclosed in square braces, as follows:

```
hosts: [servera, serverb, serverc]
```

You should avoid this syntax because it is usually harder to read.

Obsolete key=value Playbook Shorthand

Some playbooks might use an older shorthand method to define tasks by putting the key-value pairs for the module on the same line as the module name. For example, you might see this syntax:

```
  tasks:
    - name: shorthand form
      service: name=httpd enabled=true state=started
```

Normally you would write the same task as follows:

```
  tasks:
    - name: normal form
      service:
        name: httpd
        enabled: true
        state: started
```

You should generally avoid the shorthand form and use the normal form.

The normal form has more lines, but it is easier to work with. The task's keywords are stacked vertically and easier to differentiate. Your eyes can run straight down the play with less left-to-right motion. Also, the normal syntax is native YAML; the shorthand form is not. Syntax highlighting tools in modern text editors can help you more effectively if you use the normal format than if you use the shorthand format.

You might see this syntax in documentation and older playbooks from other people, and the syntax does still function.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

References

`ansible-playbook`(1) and `ansible-doc`(1) man pages

Intro to Playbooks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html

Playbooks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks.html

Developing Modules — Ansible Documentation
https://docs.ansible.com/ansible/2.9/dev_guide/developing_modules.html

Module Support — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/modules_support.html

YAML Syntax — Ansible Documentation
https://docs.ansible.com/ansible/2.9/reference_appendices/YAMLSyntax.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Implementing Multiple Plays

In this exercise, you will create a playbook containing multiple plays, then use it to perform configuration tasks on managed hosts.

Outcomes

You should be able to construct and execute a playbook to manage configuration and perform administration of a managed host.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab playbook-multi start` command. This function ensures that the managed host, `servera.lab.example.com`, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory file are installed on the control node.

```
[student@workstation ~]$ lab playbook-multi start
```

Instructions

1. A working directory, `/home/student/playbook-multi`, has been created on `workstation` for the Ansible project. The directory has already been populated with an `ansible.cfg` configuration file and an inventory file, `inventory`. The managed host, `servera.lab.example.com`, is already defined in this inventory file. Create a new playbook, `/home/student/playbook-multi/intranet.yml`, and add the lines needed to start the first play. It should target the managed host `servera.lab.example.com` and enable privilege escalation.

    1.1. Change directory into the `/home/student/playbook-multi` working directory.

```
[student@workstation ~]$ cd ~/playbook-multi
[student@workstation playbook-multi]$
```

    1.2. Create and open a new playbook, `/home/student/playbook-multi/intranet.yml`, and add a line consisting of three dashes to the beginning of the file to indicate the start of the YAML file.

```
---
```

    1.3. Add the following line to the `/home/student/playbook-multi/intranet.yml` file to denote the start of a play with a name of `Enable intranet services`.

```
- name: Enable intranet services
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1.4. Add the following line to indicate that the play applies to the `servera.lab.example.com` managed host. Be sure to indent the line with two spaces (aligning with the `name` keyword above it) to indicate that it is part of the first play.

```
hosts: servera.lab.example.com
```

1.5. Add the following line to enable privilege escalation. Be sure to indent the line with two spaces (aligning with the keywords above it) to indicate it is part of the first play.

```
become: yes
```

1.6. Add the following line to define the beginning of the `tasks` list. Indent the line with two spaces (aligning with the keywords above it) to indicate that it is part of the first play.

```
tasks:
```

2. As the first task in the first play, define a task that ensures that the *httpd* and *firewalld* packages are up to date.

Be sure to indent the first line of the task with four spaces. Under the `tasks` keyword in the first play, add the following lines.

```
  - name: latest version of httpd and firewalld installed
    yum:
      name:
        - httpd
        - firewalld
      state: latest
```

The first line provides a descriptive name for the task. The second line is indented with six spaces and calls the `yum` module. The next line is indented eight spaces and is a `name` keyword. It specifies which packages the `yum` module should ensure are up-to-date. The `yum` module's `name` keyword (which is different from the task name) can take a list of packages, which is indented ten spaces on the two following lines. After the list, the 8-space indented `state` keyword specifies that the `yum` module should ensure that the latest version of the packages is installed.

3. Add a task to the first play's list that ensures that the correct content is in `/var/www/html/index.html`.

Add the following lines to define the content for `/var/www/html/index.html`. Be sure to indent the first line with four spaces.

```
  - name: test html page is installed
    copy:
      content: "Welcome to the example.com intranet!\n"
      dest: /var/www/html/index.html
```

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 ensure that the correct content is in the web page.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4. Define two more tasks in the play to ensure that the `firewalld` service is running and will start on boot, and will allow connections to the `httpd` service.

4.1. Add the following lines to ensure that the `firewalld` service is enabled and running. Be sure to indent the first line with four spaces.

```
- name: firewalld enabled and running
  service:
    name: firewalld
    enabled: true
    state: started
```

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.

4.2. Add the following lines to ensure that `firewalld` allows HTTP connections from remote systems. Be sure to indent the first line with four spaces.

```
- name: firewalld permits access to httpd service
  firewalld:
    service: http
    permanent: true
    state: enabled
    immediate: yes
```

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 remote HTTP connections are permanently allowed.

5. Add a final task to the first play that ensures that the `httpd` service is running and will start at boot.

Add the following lines to ensure that the `httpd` service is enabled and running. Be sure to indent the first line with four spaces.

```
- name: httpd enabled and running
  service:
    name: httpd
    enabled: true
    state: started
```

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.

6. In `/home/student/playbook-multi/intranet.yml`, define a second play targeted at `localhost` which will test the intranet web server. It does not need privilege escalation.

6.1. Add the following line to define the start of a second play. Note that there is no indentation.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Test intranet web server
```

6.2. Add the following line 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 second play.

```
  hosts: localhost
```

6.3. Add the following line to disable privilege escalation. Be sure to align the indentation with the `hosts` keyword above it.

```
  become: no
```

6.4. Add the following line to the `/home/student/playbook-multi/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 second play.

```
  tasks:
```

7. Add a single task to the second play, and use the `uri` module to request content from `http://servera.lab.example.com`. The task should verify a return HTTP status code of `200`. Configure the task to place the returned content in the task results variable.

Add the following lines to create the task for verifying the web service from the control node. Be sure to indent the first line with four spaces.

```
    - name: connect to intranet web server
      uri:
        url: http://servera.lab.example.com
        return_content: yes
        status_code: 200
```

The first line provides a descriptive name for the task. The second line is indented with six spaces and calls the `uri` module. The remaining lines are indented with eight 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. The `return_content` keyword ensures that the server's response is added to the task results.

8. Verify that the final `/home/student/playbook-multi/intranet.yml` playbook reflects the following structured content, then save and close the file.

```
---
- name: Enable intranet services
  hosts: servera.lab.example.com
  become: yes
  tasks:
    - name: latest version of httpd and firewalld installed
      yum:
        name:
          - httpd
          - firewalld
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        state: latest

    - name: test html page is installed
      copy:
        content: "Welcome to the example.com intranet!\n"
        dest: /var/www/html/index.html

    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits access to httpd service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes

    - name: httpd enabled and running
      service:
        name: httpd
        enabled: true
        state: started

- name: Test intranet web server
  hosts: localhost
  become: no
  tasks:
    - name: connect to intranet web server
      uri:
        url: http://servera.lab.example.com
        return_content: yes
        status_code: 200
```

9. Run the `ansible-playbook --syntax-check` command to verify the syntax of the `/home/student/playbook-multi/intranet.yml` playbook.

```
[student@workstation playbook-multi]$ ansible-playbook --syntax-check intranet.yml

playbook: intranet.yml
```

10. Execute the playbook using the `-v` option to output detailed results for each task. Read through the output generated to ensure that all tasks completed successfully. Verify that an HTTP GET request to `http://servera.lab.example.com` provides the correct content.

```
[student@workstation playbook-multi]$ ansible-playbook -v intranet.yml
...output omitted...

PLAY [Enable intranet services] ************************************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [latest version of httpd and firewalld installed] *************************
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [test html page is installed] *********************************************
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [firewalld enabled and running] *******************************************
ok: [servera.lab.example.com] => {"changed": false, ...output omitted...

TASK [firewalld permits http service] ******************************************
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [httpd enabled and running] ***********************************************
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

PLAY [Test intranet web server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [connect to intranet web server] ******************************************
ok: [localhost] => {"accept_ranges": "bytes", "changed": false, "connection": "cl
ose", "content": "Welcome to the example.com intranet!\n", "content_length":(1)
 "37", "content_type": "text/html; charset=UTF-8", "cookies": {}, "cookies_string
": "", "date": "...output omitted...", "etag": "\"25-5790ddbcc5a48\"",
 "last_modified": "...output omitted...", "msg": "OK (37 bytes)", "redir
ected": false, "server": "Apache/2.4.6 (Red Hat Enterprise Linux)",
"status": 200, "url": "http://servera.lab.example.com"}(2)

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com    : ok=6    changed=4    unreachable=0    failed=0
```

(1) The server responded with the desired content, `Welcome to the example.com intranet!\n`.

(2) The server responded with an HTTP status code of `200`.

Finish

On `workstation`, run the `lab playbook-multi finish` command to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab playbook-multi finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Implementing Playbooks

Performance Checklist

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.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab playbook-review start` command. This function ensures that the managed host, `serverb.lab.example.com`, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory file are installed on the control node.

```
[student@workstation ~]$ lab playbook-review start
```

A working directory, `/home/student/playbook-review`, has been created on `workstation` for the Ansible project. The directory has already been populated with an `ansible.cfg` configuration file and an `inventory` file. The managed host, `serverb.lab.example.com`, is already defined in this inventory file.

Instructions

Note

The playbook used by this lab is very similar to the one you wrote in the preceding guided exercise in this chapter. If you do not want to create this lab's playbook from scratch, you can use that exercise's playbook as a starting point for this lab.

If you do, be careful to target the correct hosts and change the tasks to match the instructions for this exercise.

1. Create a new playbook, `/home/student/playbook-review/internet.yml`, and add the necessary entries to start a first play named `Enable internet services` and specify its intended managed host, `serverb.lab.example.com`. Add an entry to enable privilege escalation, and one to start a task list.
2. Add the required entries to the `/home/student/playbook-review/internet.yml` file to define a task that installs the latest versions of *firewalld*, *httpd*, *mariadb-server*, *php*, and *php-mysqlnd* packages.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. Add the necessary entries to the `/home/student/playbook-review/internet.yml` file to define the firewall configuration tasks. They should ensure that the `firewalld` service is `enabled` and `running`, and that access is allowed to the `http` service.
4. Add the necessary tasks to ensure the `httpd` and `mariadb` services are `enabled` and `running`.
5. Add the necessary entries to define the final task for generating web content for testing. Use the `get_url` module to copy `http://materials.example.com/labs/playbook-review/index.php` to `/var/www/html/` on the managed host.
6. In `/home/student/playbook-review/internet.yml`, define another play for the task to be performed on the control node. This play will test access to the web server that should be running on the `serverb` managed host. This play does not require privilege escalation, and will run on the `localhost` managed host.
7. Add a task that tests the web service running on `serverb` from the control node using the `uri` module. Check for a return status code of `200`.
8. Verify the syntax of the `internet.yml` playbook.
9. Use the `ansible-playbook` command to run the playbook. Read through the output generated to ensure that all tasks completed successfully.

Evaluation

Grade your work by running the `lab playbook-review grade` command from your `workstation` machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab playbook-review grade
```

Finish

On `workstation`, run the `lab playbook-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab playbook-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Implementing Playbooks

Performance Checklist

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.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab playbook-review start` command. This function ensures that the managed host, `serverb.lab.example.com`, is reachable on the network. It also ensures that the correct Ansible configuration file and inventory file are installed on the control node.

```
[student@workstation ~]$ lab playbook-review start
```

A working directory, `/home/student/playbook-review`, has been created on `workstation` for the Ansible project. The directory has already been populated with an `ansible.cfg` configuration file and an `inventory` file. The managed host, `serverb.lab.example.com`, is already defined in this inventory file.

Instructions

Note

The playbook used by this lab is very similar to the one you wrote in the preceding guided exercise in this chapter. If you do not want to create this lab's playbook from scratch, you can use that exercise's playbook as a starting point for this lab.

If you do, be careful to target the correct hosts and change the tasks to match the instructions for this exercise.

1. Create a new playbook, `/home/student/playbook-review/internet.yml`, and add the necessary entries to start a first play named `Enable internet services` and specify its intended managed host, `serverb.lab.example.com`. Add an entry to enable privilege escalation, and one to start a task list.

   1.1. Add the following entry to the beginning of `/home/student/playbook-review/internet.yml` to begin the YAML format.

```
---
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1.2. Add the following entry to denote the start of a play with a name of `Enable internet services`.

```
- name: Enable internet services
```

1.3. Add the following entry to indicate that the play applies to the `serverb` managed host.

```
  hosts: serverb.lab.example.com
```

1.4. Add the following entry to enable privilege escalation.

```
  become: yes
```

1.5. Add the following entry to define the beginning of the `tasks` list.

```
  tasks:
```

2. Add the required entries to the `/home/student/playbook-review/internet.yml` file to define a task that installs the latest versions of *firewalld*, *httpd*, *mariadb-server*, *php*, and *php-mysqlnd* packages.

```
    - name: latest version of all required packages installed
      yum:
        name:
          - firewalld
          - httpd
          - mariadb-server
          - php
          - php-mysqlnd
        state: latest
```

3. Add the necessary entries to the `/home/student/playbook-review/internet.yml` file to define the firewall configuration tasks. They should ensure that the `firewalld` service is `enabled` and `running`, and that access is allowed to the `http` service.

```
    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits http service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes
```

4. Add the necessary tasks to ensure the `httpd` and `mariadb` services are `enabled` and `running`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  - name: httpd enabled and running
    service:
      name: httpd
      enabled: true
      state: started

  - name: mariadb enabled and running
    service:
      name: mariadb
      enabled: true
      state: started
```

5. Add the necessary entries to define the final task for generating web content for testing. Use the `get_url` module to copy `http://materials.example.com/labs/playbook-review/index.php` to `/var/www/html/` on the managed host.

```
  - name: test php page is installed
    get_url:
      url: "http://materials.example.com/labs/playbook-review/index.php"
      dest: /var/www/html/index.php
      mode: 0644
```

6. In `/home/student/playbook-review/internet.yml`, define another play for the task to be performed on the control node. This play will test access to the web server that should be running on the `serverb` managed host. This play does not require privilege escalation, and will run on the `localhost` managed host.

   6.1. Add the following entry to denote the start of a second play with a name of `Test internet web server`.

```
- name: Test internet web server
```

   6.2. Add the following entry to indicate that the play applies to the `localhost` managed host.

```
  hosts: localhost
```

   6.3. Add the following line after the `hosts` keyword to disable privilege escalation for the second play.

```
  become: no
```

   6.4. Add an entry to the `/home/student/playbook-review/internet.yml` file to define the beginning of the `tasks` list.

```
  tasks:
```

7. Add a task that tests the web service running on `serverb` from the control node using the `uri` module. Check for a return status code of `200`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: connect to internet web server
      uri:
        url: http://serverb.lab.example.com
        status_code: 200
```

8. Verify the syntax of the `internet.yml` playbook.

```
[student@workstation playbook-review]$ ansible-playbook --syntax-check \
> internet.yml

playbook: internet.yml
```

9. Use the `ansible-playbook` command to run the playbook. Read through the output generated to ensure that all tasks completed successfully.

```
[student@workstation playbook-review]$ ansible-playbook internet.yml
PLAY [Enable internet services] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [latest version of all required packages installed] ***********************
changed: [serverb.lab.example.com]

TASK [firewalld enabled and running] *******************************************
ok: [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 [test php page installed] *************************************************
changed: [serverb.lab.example.com]

PLAY [Test internet web server] ************************************************

TASK [Gathering Facts] *********************************************************
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=7    changed=5    unreachable=0    failed=0
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Evaluation

Grade your work by running the `lab playbook-review grade` command from your `workstation` machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab playbook-review grade
```

Finish

On `workstation`, run the `lab playbook-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab playbook-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- A *play* is an ordered list of tasks, which runs against hosts selected from the inventory.
- A *playbook* is a text file that contains a list of one or more plays to run in order.
- Ansible Playbooks are written in YAML format.
- YAML files are structured using space indentation to represent the 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 run playbooks.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 3

Managing Variables and Facts

Goal

Write playbooks that use variables to simplify management of the playbook and facts to reference information about managed hosts.

Objectives

- Create and reference variables that affect particular hosts or host groups, the play, or the global environment, and describe how variable precedence works.
- Encrypt sensitive variables using Ansible Vault, and run playbooks that reference Vault-encrypted variable files.
- Reference data about managed hosts using Ansible facts, and configure custom facts on managed hosts.

Sections

- Managing Variables (and Guided Exercise)
- Managing Secrets (and Guided Exercise)
- Managing Facts (and Guided Exercise)

Lab

- Managing Variables and Facts

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Variables

Objectives

After completing this section, you should be able to create and reference variables that affect particular hosts or host groups, the play, or the global environment, and describe how variable precedence works.

Introduction to Ansible Variables

Ansible supports variables that can be used to store values that can then be reused throughout files in an Ansible project. This can simplify the creation and maintenance of a project and reduce the number of errors.

Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project. 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

Variable names must start with a letter, and they can only contain letters, numbers, and underscores.

The following table illustrates the difference between invalid and valid variable names.

Examples of Invalid and Valid Ansible Variable Names

Invalid variable names	Valid variable names
`web server`	`web_server`
`remote.file`	`remote_file`
`1st file`	`file_1` `file1`
`remoteserver$1`	`remote_server_1` `remote_server1`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Defining Variables

Variables can be defined in a variety of places in an Ansible project. If a variable is set using the same name in two places, and those settings have different values, *precedence* determines which value is used.

You can set a variable that affects a group of hosts or only individual hosts. Some variables are *facts* that can be set by Ansible based on the configuration of a system. Other variables can be set inside the playbook, and affect one play in that playbook, or only one task in that play. You can also set *extra variables* on the `ansible-playbook` command line by using the `--extra-vars` or `-e` option and specifying those variables, and they override all other values for that variable name.

Here is a simplified list of ways to define a variable, ordered from lowest precedence to highest:

- Group variables defined in the inventory.
- Group variables defined in files in a `group_vars` subdirectory in the same directory as the inventory or the playbook.
- Host variables defined in the inventory.
- Host variables defined in files in a `host_vars` subdirectory in the same directory as the inventory or the playbook.
- Host facts, discovered at runtime.
- Play variables in the playbook (`vars` and `vars_files`).
- Task variables.
- Extra variables defined on the command line.

For example, a variable that is set to affect the `all` host group will be overridden by a variable that has the same name and is set to affect a single host.

One recommended practice is to choose globally unique variable names so you do not have to consider precedence rules. However, sometimes you might want to use precedence to cause different hosts or host groups to get different settings than your defaults.

If the same variable name is defined at more than one level, the level with the highest precedence wins. A narrow scope, such as a host variable or a task variable, takes precedence over a wider scope, such as a group variable or a play variable. Variables defined by the inventory are overridden by variables defined by the playbook. Extra variables defined on the command line with the `--extra-vars`, or `-e`, option have the highest precedence.

A detailed and more precise discussion of variable precedence is available in the Ansible documentation at Variable Precedence: Where Should I Put A Variable? [https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable].

Variables in Playbooks

Variables play an important role in Ansible Playbooks because they ease the management of variable data in a playbook.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Defining Variables in Playbooks

When writing playbooks, you can define your own variables and then invoke those values in a task. For example, a variable named `web_package` can be defined with a value of `httpd`. A task can then call the variable using the `yum` module to install the *httpd* package.

Playbook variables can be defined in multiple ways. One common method is to place a variable 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 a `vars` block in the playbook, the `vars_files` directive may be used, followed by a list of names for external variable files relative to the location of the playbook:

```
- 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

After 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 La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

When a variable is used as the first element to start a value, quotes are mandatory. This prevents Ansible from interpreting the variable reference 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* apply to a specific host, and *group variables* 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 the easiest to work with, but you might still see it used because it does put all the inventory information and variable settings for hosts and host groups in one file.

- Defining the `ansible_user` host variable for `demo.example.com`:

```
[servers]
demo.example.com  ansible_user=joe
```

- Defining the `user` group variable for the `servers` host group.

```
[servers]
demo1.example.com
demo2.example.com

[servers:vars]
user=joe
```

- Defining the `user` group variable for the `servers` group, which consists of two host groups each with two servers.

```
[servers1]
demo1.example.com
demo2.example.com
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[servers2]
demo3.example.com
demo4.example.com

[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

Some disadvantages of this approach are that it makes the inventory file more difficult to work with, it mixes information about hosts and variables in the same file, and uses an obsolete syntax.

Using Directories to Populate Host and Group Variables

The preferred approach to defining variables for hosts and host groups is to create two directories, `group_vars` and `host_vars`, in the same working directory as the inventory file or playbook. 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 files.

To define group variables for the `servers` group, you would create a YAML file named `group_vars/servers`, and then the contents of that file would set variables to values using the same syntax as in a playbook:

```
user: joe
```

Likewise, to define host variables for a particular host, create a file with a name matching the host in the `host_vars` directory to contain the host variables.

The following examples illustrate this approach in more detail. Consider a scenario where there are two data centers to manage and the data center hosts are defined in the `~/project/inventory` inventory file:

```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- If you need to define a general value for all servers in both data centers, set a group variable for the `datacenters` host group:

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

- If the value to define varies for each data center, set a group variable for each data center host group:

```
[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 data center, then define the variables in separate host variable files:

```
[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 the example project, `project`, if it contained all the example files above, would appear as follows:

```
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
```

Note

Ansible looks for `host_vars` and `group_vars` subdirectories relative to both the inventory and the playbook. If your inventory and your playbook happen to be in the same directory, this is simple and Ansible looks in that directory for those subdirectories. If your inventory and your playbook are in separate directories, then Ansible will look in both places for `host_vars` and `group_vars` subdirectories. The playbook subdirectories have higher precedence.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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. Variables set on the command line are called *extra variables*.

Extra variables can be useful when you need to override the defined value for a variable for a one-off run of a playbook. For example:

```
[user@demo ~]$ ansible-playbook main.yml -e "package=apache"
```

Using Arrays as Variables

Instead of assigning configuration data that relates to the same element (a list of packages, a list of services, a list of users, and so on), to multiple variables, administrators can use arrays. One consequence of this is that an array can be browsed.

For example, consider the following snippet:

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_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
```

You can then use the following variables to access user data:

```
# 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.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
# 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 conflicts and errors.

Both syntaxes are valid, but to make troubleshooting easier, Red Hat recommends that you use one syntax consistently in all files throughout any given Ansible project.

Capturing Command Output with Registered Variables

You can use the `register` statement to capture the output of a command. The output is saved into a temporary variable that can be used later in the playbook for either debugging purposes or 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
  tasks:
    - name: Install the package
      yum:
        name: httpd
        state: installed
      register: install_result

    - debug:
        var: install_result
```

When you run the playbook, 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": {
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        "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
```

References

Inventory — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html

Variables — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html

Variable Precedence: Where Should I Put A Variable?
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

YAML Syntax — Ansible Documentation
https://docs.ansible.com/ansible/2.9/reference_appendices/YAMLSyntax.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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 tasks that use defined variables.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab data-variables start` command. This function creates the `data-variables` working directory, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab data-variables start
```

Instructions

1. On `workstation`, as the `student` user, change into the `/home/student/data-variables` directory.

```
[student@workstation ~]$ cd ~/data-variables
[student@workstation data-variables]$
```

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.

   Create the `playbook.yml` playbook and define the following variables in the `vars` section:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Variables

Variable	Description
`web_pkg`	Web server package to install.
`firewall_pkg`	Firewall package to install.
`web_service`	Web service to manage.
`firewall_service`	Firewall service to manage.
`python_pkg`	Required package for the `uri` module.
`rule`	The service name to open.

```
---
- name: Deploy and start Apache HTTPD service
  hosts: webserver
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python3-PyMySQL
    rule: http
```

3. Create the `tasks` block and create the first task, which should use the `yum` module to make sure the latest versions of the required packages are installed.

```
  tasks:
    - name: Required packages are installed and up to date
      yum:
        name:
          - "{{ web_pkg }}"
          - "{{ firewall_pkg }}"
          - "{{ python_pkg }}"
        state: latest
```

Note

You can use `ansible-doc yum` to review the syntax for the `yum` module. The syntax shows that its `name` directive can take a list of packages that the module should work with, so that you do not need separate tasks to makes sure each package is up-to-date.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4. Create two tasks to make sure that the `httpd` and `firewalld` services are started and enabled.

```
- name: The {{ firewall_service }} service is started and enabled
  service:
    name: "{{ firewall_service }}"
    enabled: true
    state: started

- name: The {{ web_service }} service is started and enabled
  service:
    name: "{{ web_service }}"
    enabled: true
    state: started
```

Note

The `service` module works differently from the `yum` module, as documented by `ansible-doc service`. Its `name` directive takes the name of exactly one service to work with.

You can write a single task that ensures both services are started and enabled, using the `loop` keyword covered later in this course.

5. Add a task that ensures specific content exists in the `/var/www/html/index.html` file.

```
- name: Web content is in place
  copy:
    content: "Example web content"
    dest: /var/www/html/index.html
```

6. Add a task that uses the `firewalld` module to ensure the firewall ports are open for the `firewalld` service named in the `rule` variable.

```
- name: The firewall port for {{ rule }} is open
  firewalld:
    service: "{{ rule }}"
    permanent: true
    immediate: true
    state: enabled
```

7. Create a new play that queries the web service to ensure everything has been correctly configured. It should run on `localhost`. Because of that Ansible fact, Ansible does not have to change identity, so set the `become` module to `false`. You can use the `uri` module

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

to check a URL. For this task, check for a status code of 200 to confirm the web server on `servera.lab.example.com` is running and correctly configured.

```
- name: Verify the Apache service
  hosts: localhost
  become: false
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://servera.lab.example.com
        status_code: 200
```

8. When completed, the playbook should appear as follows. Review the playbook and confirm that both plays are correct.

```
---
- name: Deploy and start Apache HTTPD service
  hosts: webserver
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python3-PyMySQL
    rule: http

  tasks:
    - name: Required packages are installed and up to date
      yum:
        name:
          - "{{ web_pkg  }}"
          - "{{ firewall_pkg }}"
          - "{{ python_pkg }}"
        state: latest

    - name: The {{ firewall_service }} service is started and enabled
      service:
        name: "{{ firewall_service }}"
        enabled: true
        state: started

    - name: The {{ web_service }} service is started and enabled
      service:
        name: "{{ web_service }}"
        enabled: true
        state: started

    - name: Web content is in place
      copy:
        content: "Example web content"
        dest: /var/www/html/index.html

    - name: The firewall port for {{ rule }} is open
      firewalld:
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        service: "{{ rule }}"
        permanent: true
        immediate: true
        state: enabled

- name: Verify the Apache service
  hosts: localhost
  become: false
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://servera.lab.example.com
        status_code: 200
```

9. Before you run the playbook, use the `ansible-playbook --syntax-check` command to verify its syntax. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation data-variables]$ ansible-playbook --syntax-check playbook.yml

playbook: playbook.yml
```

10. Use the `ansible-playbook` command to run the playbook. Watch the output as Ansible installs the packages, starts and enables the services, and ensures the web server is reachable.

```
[student@workstation data-variables]$ ansible-playbook playbook.yml

PLAY [Deploy and start Apache HTTPD service] ***********************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Required packages are installed and up to date] **************************
changed: [servera.lab.example.com]

TASK [The firewalld service is started and enabled] ****************************
ok: [servera.lab.example.com]

TASK [The httpd service is started and enabled] ********************************
changed: [servera.lab.example.com]

TASK [Web content is in place] *************************************************
changed: [servera.lab.example.com]

TASK [The firewall port for http is open] **************************************
changed: [servera.lab.example.com]

PLAY [Verify the Apache service] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
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=4    unreachable=0    failed=0
```

Finish

On `workstation`, run the `lab data-variables finish` script to clean up this exercise.

```
[student@workstation ~]$ lab data-variables finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Secrets

Objectives

After completing this section, you should be able to encrypt sensitive variables using Ansible Vault, and run playbooks that reference Vault-encrypted variable files.

Introducing Ansible Vault

Ansible may need access to sensitive data such as passwords or API keys in order to configure managed hosts. Normally, this information might be stored as plain text in inventory variables or other Ansible files. In that case, however, 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.

Ansible Vault, which is included with Ansible, can be used to encrypt and decrypt any structured data file used by Ansible. To use Ansible Vault, a command-line tool named `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 arguments when executing the playbook, or variables defined in Ansible roles.

Important

Ansible Vault does not implement its own cryptographic functions but rather 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.

Creating an Encrypted File

To create a new encrypted file, use the `ansible-vault create` *`filename`* command. The command prompts for the new vault password and then opens a file using the default editor, `vi`. You can set and export the `EDITOR` environment variable to specify a different default editor by setting and exporting. For example, to set the default editor to `nano`, `export EDITOR=nano`.

```
[student@demo ~]$ ansible-vault create secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

Instead of entering the vault password through standard input, you can use a vault password file to store the vault password. You need to carefully protect this file using 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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Viewing an Encrypted File

You can use the `ansible-vault view` *`filename`* command to view an Ansible Vault-encrypted file without opening it for editing.

```
[student@demo ~]$ ansible-vault view secret1.yml
Vault password: secret
my_secret: "yJJvPqhsiusmmPPZdnjndkdnYNDjdj782meUZcw"
```

Editing an Existing Encrypted File

To edit an existing encrypted file, Ansible Vault provides the `ansible-vault edit` *`filename`* command. This command decrypts the file to a temporary file and allows you to edit it. 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 you should only use it when making changes. This can have implications when the file is kept under version control. You should always use the `view` subcommand to view the file's contents without making changes.

Encrypting an Existing File

To encrypt a file that already exists, use the `ansible-vault encrypt` *`filename`* command. 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
```

Use the `--output=OUTPUT_FILE` option to save the encrypted file with a new name. You can only use one input file with the `--output` option.

Decrypting an Existing File

An existing encrypted file can be permanently decrypted by using the `ansible-vault decrypt` *`filename`* command. When decrypting a single file, you can use the `--output` option 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
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Changing the Password of an Encrypted File

You can use the `ansible-vault rekey` *`filename`* command to change the password of an encrypted file. This command can rekey multiple data files at once. It prompts for the original password and then 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 using a 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
```

Playbooks and Ansible Vault

To run a playbook that accesses files encrypted with Ansible Vault, you need to provide the encryption password to the `ansible-playbook` command. If you do not provide the password, the playbook returns 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 to the playbook, use the `--vault-id` option. For example, to provide the vault password interactively, use `--vault-id @prompt` as illustrated in the following example:

```
[student@demo ~]$ ansible-playbook --vault-id @prompt site.yml
Vault password (default): redhat
```

Important

If you are using a release of Ansible earlier than version 2.4, you need to use the `--ask-vault-pass` option to interactively provide the vault password. You can still use this option if all vault-encrypted files used by the playbook were encrypted with the same password.

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml
Vault password: redhat
```

Alternatively, you can use the `--vault-password-file` option to specify a file that stores the encryption password in plain text. The password should be a string stored as a single line in the file. Because 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
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

You can also use the `ANSIBLE_VAULT_PASSWORD_FILE` environment variable to specify the default location of the password file.

Important

Starting with Ansible 2.4, you can use multiple Ansible Vault passwords with `ansible-playbook`. To use multiple passwords, pass multiple `--vault-id` or `--vault-password-file` options to the `ansible-playbook` command.

```
[student@demo ~]$ ansible-playbook \
> --vault-id one@prompt --vault-id two@prompt site.yml
Vault password (one):
Vault password (two):
...output omitted...
```

The vault IDs `one` and `two` preceding `@prompt` can be anything and you can even omit them entirely. If you use the `--vault-id` *`id`* option when you encrypt a file with `ansible-vault` command, however, when you run `ansible-playbook` then the password for the matching ID is tried before any others. If it does not match, the other passwords you provided will be tried next. The vault ID `@prompt` with no ID is actually shorthand for `default@prompt`, which means to prompt for the password for vault ID `default`.

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 files containing 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 host names of managed hosts to which they apply.

However, instead of using files in `group_vars` or `host_vars`, you also 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 directory for `playbook.yml`, members of the `webservers` host group uses variables in the `group_vars/webservers/vars` file, and `demo.example.com` uses 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
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

In this scenario, the advantage is that most variables for `demo.example.com` can be placed in the `vars` file, but sensitive variables can be kept secret by placing them separately in the `vault` file. Then the administrator can use `ansible-vault` to encrypt the `vault` file, while leaving the `vars` file 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 in the playbook through a `vars_files` directive. This can be useful, because playbook variables take precedence over inventory variables.

If you are using multiple vault passwords with your playbook, make sure that each encrypted file is assigned a vault ID, and that you enter the matching password with that vault ID when running the playbook. This ensures that the correct password is selected first when decrypting the vault-encrypted file, which is faster than forcing Ansible to try all the vault passwords you provided until it finds the right one.

References

`ansible-playbook`(1) and `ansible-vault`(1) man pages

Encrypting content with Ansible Vault — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/vault.html

Keep vaulted variables safely visible — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#keep-vaulted-variables-safely-visible

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Secrets

In this exercise, you will encrypt sensitive variables with Ansible Vault to protect them, and then run a playbook that uses those variables.

Outcomes

You should be able to:

- Execute a playbook using variables defined in an encrypted file.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab data-secret start` command. This script ensures that Ansible is installed on `workstation` and creates a working directory for this exercise. This directory 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 data-secret start
```

Instructions

1. On `workstation`, as the `student` user, change to the `/home/student/data-secret` working directory.

```
[student@workstation ~]$ cd ~/data-secret
[student@workstation data-secret]$
```

2. Edit the contents of the provided encrypted file, `secret.yml`. The file can be decrypted using `redhat` as the password. Uncomment the `username` and `pwhash` variable entries.

    2.1. Edit the encrypted file `/home/student/data-secret/secret.yml`. Provide a password of `redhat` for the vault when prompted. The encrypted file opens in the default editor, `vim`.

```
[student@workstation data-secret]$ ansible-vault edit secret.yml
Vault password: redhat
```

    2.2. Uncomment the two variable entries, then save the file and exit the editor. They should appear as follows:

```
username: ansibleuser1
pwhash: $6$jf...uxhP1
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. Create a playbook named `/home/student/data-secret/create_users.yml` that uses the variables defined in the `/home/student/data-secret/secret.yml` encrypted file.

   Configure the playbook to use the `devservers` host group. Run this playbook as the `devops` user on the remote managed host. Configure the playbook to create the `ansibleuser1` user defined by the `username` variable. Set the user's password using the password hash stored in the `pwhash` variable.

```
---
- name: create user accounts for all our servers
  hosts: devservers
  become: True
  remote_user: devops
  vars_files:
    - secret.yml
  tasks:
    - name: Creating user from secret.yml
      user:
        name: "{{ username }}"
        password: "{{ pwhash }}"
```

4. Use the `ansible-playbook --syntax-check` command to verify the syntax of the `create_users.yml` playbook. Use the `--ask-vault-pass` option to prompt for the vault password, which decrypts `secret.yml`. Resolve any syntax errors before you continue.

```
[student@workstation data-secret]$ ansible-playbook --syntax-check \
> --ask-vault-pass create_users.yml
Vault password (default): redhat

playbook: create_users.yml
```

Note

Instead of using `--ask-vault-pass`, you can use the newer `--vault-id @prompt` option to do the same thing.

5. Create a password file named `vault-pass` to use for the playbook execution instead of asking for a password. The file must contain the plain text `redhat` as the vault password. Change the permissions of the file to `0600`.

```
[student@workstation data-secret]$ echo 'redhat' > vault-pass
[student@workstation data-secret]$ chmod 0600 vault-pass
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

6. Execute the Ansible Playbook using the `vault-pass` file, to create the `ansibleuser1` user on a remote system using the passwords stored as variables in the `secret.yml` Ansible Vault encrypted file.

```
[student@workstation data-secret]$ ansible-playbook \
> --vault-password-file=vault-pass create_users.yml

PLAY [create user accounts for all our servers] ******************************

TASK [Gathering Facts] *******************************************************
ok: [servera.lab.example.com]

TASK [Creating users from secret.yml] ****************************************
changed: [servera.lab.example.com]

PLAY RECAP *******************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

7. Verify that the playbook ran correctly. The user `ansibleuser1` should exist and have the correct password on `servera.lab.example.com`. Test this by using `ssh` to log in as that user on `servera.lab.example.com`. The password for `ansibleuser1` is `redhat`. To make sure that SSH only tries to authenticate by password and not by an SSH key, use the `-o PreferredAuthentications=password` option when you log in.

   Log off from `servera` when you have successfully logged in.

```
[student@workstation data-secret]$ ssh -o PreferredAuthentications=password \
> ansibleuser1@servera.lab.example.com
ansibleuser1@servera.lab.example.com's password: redhat
Activate the web console with: systemctl enable --now cockpit.socket

[ansibleuser1@servera ~]$ exit
logout
Connection to servera.lab.example.com closed.
```

Finish

On `workstation`, run the `lab data-secret finish` script to clean up this exercise.

```
[student@workstation ~]$ lab data-secret finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Facts

Objectives

After completing this section, you should be able to reference data about managed hosts using Ansible facts, and configure custom facts on managed hosts.

Describing Ansible Facts

Ansible *facts* are variables that are automatically discovered by Ansible on a managed host. Facts contain host-specific information that can be used just like regular variables in plays, conditionals, loops, or any other statement that depends on a value collected from a managed host.

Some of the facts gathered for a managed host might include:

- 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

Facts are a convenient way to retrieve the state of a managed host and to determine what action to take based on that state. For example:

- A server can be restarted by a conditional task which is run based on a fact containing the managed host's current kernel version.
- The MySQL configuration file can be customized depending on the available memory reported by a fact.
- The IPv4 address used in a configuration file can be set based on the value of a fact.

Normally, every play runs the `setup` module automatically before the first task in order to gather facts. This is reported as the `Gathering Facts` task in Ansible 2.3 and later, or simply as `setup` in older versions of Ansible. By default, you do not need to have a task to run `setup` in your play. It is normally run automatically for you.

One way to see what facts are gathered for your managed hosts is to run a short playbook that gathers facts and uses the `debug` module to print the value of the `ansible_facts` variable.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Fact dump
  hosts: all
  tasks:
    - name: Print all facts
      debug:
        var: ansible_facts
```

When you run the playbook, the facts are displayed in the job output:

```
[user@demo ~]$ ansible-playbook facts.yml

PLAY [Fact dump] ************************************************************

TASK [Gathering Facts] ******************************************************
ok: [demo1.example.com]

TASK [Print all facts] ******************************************************
ok: [demo1.example.com] => {
    "ansible_facts": {
        "all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "all_ipv6_addresses": [
           "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_local": {},
        "apparmor": {
            "status": "disabled"
        },
        "architecture": "x86_64",
        "bios_date": "01/01/2011",
        "bios_version": "0.5.1",
        "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 playbook displays the content of the `ansible_facts` variable in JSON format as a hash/dictionary of variables. You can browse the output to see what facts are gathered, to find facts that you might want to use in your plays.

The following table shows some facts which might be gathered from a managed node and may be useful in a playbook:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Examples of Ansible Facts

Fact	Variable
Short host name	`ansible_facts['hostname']`
Fully qualified domain name	`ansible_facts['fqdn']`
Main IPv4 address (based on routing)	`ansible_facts['default_ipv4']['address']`
List of the names of all network interfaces	`ansible_facts['interfaces']`
Size of the `/dev/vda1` disk partition	`ansible_facts['devices']['vda']['partitions']['vda1']['size']`
List of DNS servers	`ansible_facts['dns']['nameservers']`
Version of the currently running kernel	`ansible_facts['kernel']`

Note

Remember that when a variable's value is a hash/dictionary, there are two syntaxes that can be used to retrieve the value. To take two examples from the preceding table:

- `ansible_facts['default_ipv4']['address']` can also be written `ansible_facts.default_ipv4.address`
- `ansible_facts['dns']['nameservers']` can also be written `ansible_facts.dns.nameservers`

When a fact is used in a playbook, Ansible dynamically substitutes the variable name for the fact with the corresponding value:

```
---
- hosts: all
  tasks:
  - name: Prints various Ansible facts
    debug:
      msg: >
        The default IPv4 address of {{ ansible_facts.fqdn }}
        is {{ ansible_facts.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. Facts can also be used to create dynamic groups of hosts that match particular criteria.

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ***************************************************************

TASK [Gathering Facts] *********************************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
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"
}

PLAY RECAP **********************************************************
demo1.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

Ansible Facts Injected as Variables

Before Ansible 2.5, facts were injected as individual variables prefixed with the string `ansible_` instead of being part of the `ansible_facts` variable. For example, the `ansible_facts['distribution']` fact would have been called `ansible_distribution`.

Many older playbooks still use facts injected as variables instead of the new syntax that is namespaced under the `ansible_facts` variable. You can use an ad hoc command to run the `setup` module to print the value of all facts in this form. In the following example, an ad hoc command is used to run the `setup` module on the managed host `demo1.example.com`:

```
[user@demo ~]$ 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_apparmor": {
            "status": "disabled"
        },
        "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"
        }
...output omitted...
```

The following table compares the old and new fact names.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Comparison of Selected Ansible Fact Names

ansible_facts form	Old fact variable form
`ansible_facts['hostname']`	`ansible_hostname`
`ansible_facts['fqdn']`	`ansible_fqdn`
`ansible_facts['default_ipv4']['address']`	`ansible_default_ipv4['address']`
`ansible_facts['interfaces']`	`ansible_interfaces`
`ansible_facts['devices']['vda']['partitions']['vda1']['size']`	`ansible_devices['vda']['partitions']['vda1']['size']`
`ansible_facts['dns']['nameservers']`	`ansible_dns['nameservers']`
`ansible_facts['kernel']`	`ansible_kernel`

Important

Currently, Ansible recognizes both the new fact naming system (using `ansible_facts`) and the old pre-2.5 "facts injected as separate variables" naming system.

You can turn off the old naming system by setting the `inject_facts_as_vars` parameter in the `[default]` section of the Ansible configuration file to `false`. The default setting is currently `true`.

The default value of `inject_facts_as_vars` will probably change to `false` in a future version of Ansible. If it is set to `false`, you can only reference Ansible facts using the new `ansible_facts.*` naming system. In that case, attempts to reference facts through the old namespace results in the following error:

```
...output omitted...
TASK [Show me the facts] *******************************************
fatal: [demo.example.com]: FAILED! => {"msg": "The task includes an option
 with an undefined variable. The error was: 'ansible_distribution' is
 undefined\n\nThe error appears to have been in
 '/home/student/demo/playbook.yml': line 5, column 7, but may\nbe elsewhere in
 the file depending on the exact syntax problem.\n\nThe offending line appears
 to be:\n\n  tasks:\n    - name: Show me the facts\n      ^ here\n"}
...output omitted...
```

Turning Off Fact Gathering

Sometimes, you do not want to gather facts for your play. There are a couple of reasons why this might be the case. It might be that you are not using any facts and want to speed up the play or reduce load caused by the play on the managed hosts. It might be that the managed hosts cannot run the `setup` module for some reason, or need to install some prerequisite software before gathering facts.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

To disable fact gathering for a play, set the `gather_facts` keyword to `no`:

```
---
- name: This play gathers no facts automatically
  hosts: large_farm
  gather_facts: no
```

Even if `gather_facts: no` is set for a play, you can manually gather facts at any time by running a task that uses the `setup` module:

```
  tasks:
    - name: Manually gather facts
      setup:
...output omitted...
```

Creating Custom Facts

Administrators can create *custom facts* which are stored locally on each managed host. These facts are integrated into the list of standard facts gathered by the `setup` module when it runs on the managed host. These allow the managed host to provide arbitrary variables to Ansible which can be used to adjust the behavior of plays.

Custom facts can be defined in a static file, formatted as an INI file or using JSON. They can also be executable scripts which generate JSON output.

Custom facts allow administrators to define certain values for managed hosts, which plays might use to populate configuration files or conditionally run tasks. Dynamic custom facts allow the values for these facts, or even which facts are provided, to be determined programmatically when the play is run.

By default, the `setup` module loads custom facts from files and scripts in each managed host's `/etc/ansible/facts.d` directory. The name of each file or script must end in `.fact` in order to be used. Dynamic custom fact scripts must output JSON-formatted facts and must be executable.

This is an example of a static custom facts file written in INI format. An INI-formatted custom facts file contains a top level defined by a section, followed by the key-value pairs of the facts to define:

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

The same facts could be provided in JSON format. The following JSON facts are equivalent to the facts specified by the INI format in the preceding example. The JSON data could be stored in a static text file or printed to standard output by an executable script:

```
{
  "packages": {
    "web_package": "httpd",
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```

Note

Custom fact files cannot be in YAML format like a playbook. JSON format is the closest equivalent.

Custom facts are stored by the `setup` module in the `ansible_facts['ansible_local']` variable. Facts are organized based on the name of the file that defined them. For example, assume that the preceding custom facts are produced by a file saved as `/etc/ansible/facts.d/custom.fact` on the managed host. In that case, the value of `ansible_facts['ansible_local']['custom']['users']['user1']` is `joe`.

You can inspect the structure of your custom facts by running the `setup` module on the managed hosts with an ad hoc command.

```
[user@demo ~]$ ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "web_package": "httpd"
                },
                "users": {
                    "user1": "joe",
                    "user2": "jane"
                }
            }
        },
...output omitted...
    },
    "changed": false
}
```

Custom facts can be used the same way as default facts in playbooks:

```
[user@demo ~]$ cat playbook.yml
---
- hosts: all
  tasks:
  - name: Prints various Ansible facts
    debug:
      msg: >
           The package to install on {{ ansible_facts['fqdn'] }}
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
           is {{ ansible_facts['ansible_local']['custom']['packages']
['web_package'] }}

[user@demo ~]$ ansible-playbook playbook.yml
PLAY ***************************************************************

TASK [Gathering Facts] *********************************************
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
```

Using Magic Variables

Some variables are not facts or configured through the `setup` module, but are also automatically set by Ansible. These *magic variables* can also be useful to get information specific to a particular managed host.

Four of the most useful are:

`hostvars`
: Contains the variables for managed hosts, and can be used to get the values for another managed host's variables. It does not include the managed host's facts if they have not yet been gathered for that host.

`group_names`
: Lists all groups the current managed host is in.

`groups`
: Lists all groups and hosts in the inventory.

`inventory_hostname`
: Contains the host name for the current managed host as configured in the inventory. This may be different from the host name reported by facts for various reasons.

There are a number of other "magic variables" as well. For more information, see https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable. One way to get insight into their values is to use the `debug` module to report on the contents of the `hostvars` variable for a particular host:

```
[user@demo ~]$ ansible localhost -m debug -a 'var=hostvars["localhost"]'
localhost | SUCCESS => {
    "hostvars[\"localhost\"]": {
        "ansible_check_mode": false,
        "ansible_connection": "local",
        "ansible_diff_mode": false,
        "ansible_facts": {},
        "ansible_forks": 5,
        "ansible_inventory_sources": [
            "/home/student/demo/inventory"
        ],
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        "ansible_playbook_python": "/usr/bin/python2",
        "ansible_python_interpreter": "/usr/bin/python2",
        "ansible_verbosity": 0,
        "ansible_version": {
            "full": "2.7.0",
            "major": 2,
            "minor": 7,
            "revision": 0,
            "string": "2.7.0"
        },
        "group_names": [],
        "groups": {
            "all": [
                "serverb.lab.example.com"
            ],
            "ungrouped": [],
            "webservers": [
                "serverb.lab.example.com"
            ]
        },
        "inventory_hostname": "localhost",
        "inventory_hostname_short": "localhost",
        "omit": "__omit_place_holder__18d132963728b2cbf7143dd49dc4bf5745fe5ec3",
        "playbook_dir": "/home/student/demo"
    }
}
```


References

setup - Gathers facts about remote hosts — Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/setup_module.html

Variables discovered from systems: Facts — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variables-discovered-from-systems-facts

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Facts

In this exercise, you will gather Ansible facts from a managed host and use them in plays.

Outcomes

You should be able to:

- Gather facts from a host.
- Create tasks that use the gathered facts.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab data-facts start` command. This script creates the working directory, `data-facts`, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab data-facts start
```

Instructions

1. On `workstation`, as the `student` user, change into the `/home/student/data-facts` directory.

```
[student@workstation ~]$ cd ~/data-facts
[student@workstation data-facts]$
```

2. The Ansible `setup` module retrieves facts from systems. Run an ad hoc command to retrieve the facts for all servers in the `webserver` group. The output displays all the facts gathered for `servera.lab.example.com` in JSON format. Review some of the variables displayed.

```
[student@workstation data-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::2937:3aa3:ea8d:d3b1"
        ],
...output omitted...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. On `workstation`, create a fact file named `/home/student/data-facts/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
enabled = true
```

4. 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 }}"
```

5. Run an ad hoc command with the `setup` module. Search for the `ansible_local` section in the output. There should not be any custom facts at this point.

```
[student@workstation data-facts]$ ansible webserver -m setup
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
        "ansible_local": {}
...output omitted...
    },
    "changed": false
}
```

6. Before running the playbook, verify its syntax is correct by running `ansible-playbook --syntax-check`. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation data-facts]$ ansible-playbook --syntax-check setup_facts.yml

playbook: setup_facts.yml
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

7. Run the `setup_facts.yml` playbook.

```
[student@workstation data-facts]$ ansible-playbook setup_facts.yml

PLAY [Install remote facts] ****************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Create the remote directory] *********************************************
changed: [servera.lab.example.com]

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. 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. Create the playbook `playbook.yml` with the following:

```
---
- name: Install Apache and starts the service
  hosts: webserver
```

9. 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_facts['ansible_local']['custom']['general']
['package'] }}"
        state: latest
```

10. Create another task that uses the custom fact to start the `httpd` service.

```
    - name: Start the service
      service:
        name: "{{ ansible_facts['ansible_local']['custom']['general']
['service'] }}"
        state: "{{ ansible_facts['ansible_local']['custom']['general']
['state'] }}"
        enabled: "{{ ansible_facts['ansible_local']['custom']['general']
['enabled'] }}"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

11. When completed with all the tasks, the full playbook should look like the following. Review the playbook and ensure all the tasks are defined.

```
---
- name: Install Apache and starts the service
  hosts: webserver

  tasks:
    - name: Install the required package
      yum:
        name: "{{ ansible_facts['ansible_local']['custom']['general']
['package'] }}"
        state: latest

    - name: Start the service
      service:
        name: "{{ ansible_facts['ansible_local']['custom']['general']
['service'] }}"
        state: "{{ ansible_facts['ansible_local']['custom']['general']
['state'] }}"
        enabled: "{{ ansible_facts['ansible_local']['custom']['general']
['enabled'] }}"
```

12. Before running the playbook, use an ad hoc command to verify the `httpd` service is not currently running on `servera`.

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | FAILED | rc=4 >>
Unit httpd.service could not be found.non-zero return code
```

13. Verify the syntax of the playbook by running `ansible-playbook --syntax-check`. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation data-facts]$ ansible-playbook --syntax-check playbook.yml

playbook: playbook.yml
```

14. Run the playbook using the `ansible-playbook` command. Watch the output as Ansible installs the package and then enables the service.

```
[student@workstation data-facts]$ ansible-playbook playbook.yml

PLAY [Install Apache and start the service] ***********************************

TASK [Gathering Facts] ********************************************************
ok: [servera.lab.example.com]

TASK [Install the required package] *******************************************
changed: [servera.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
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 determine whether the `httpd` service is now running on `servera`.

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
 disabled)
   Active: active (running) since Mon 2019-05-27 07:50:55 EDT; 50s ago
     Docs: man:httpd.service(8)
 Main PID: 11603 (httpd)
   Status: "Running, listening on: port 80"
    Tasks: 213 (limit: 4956)
   Memory: 24.1M
   CGroup: /system.slice/httpd.service
...output omitted...
```

Finish

On `workstation`, run the `lab data-facts finish` script to clean up this exercise.

```
[student@workstation ~]$ lab data-facts finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Managing Variables and Facts

Performance Checklist

In this lab, you will write and run an Ansible Playbook that uses variables, secrets, and facts.

Outcomes

You should be able to define variables and use facts in a playbook, as well as use variables defined in an encrypted file.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab data-review start` command. The script creates the `/home/student/data-review` working directory and populates it with an Ansible configuration file and host inventory. The managed host `serverb.lab.example.com` is defined in this inventory as a member of the `webserver` host group. A developer has asked you to write an Ansible Playbook to automate the setup of a web server environment on `serverb.lab.example.com`, which controls user access to its website using basic authentication.

The `files` subdirectory contains:

- A `httpd.conf` configuration file for the Apache web service for basic authentication
- A `.htaccess` file, used to control access to the web server's document root directory
- A `htpasswd` file containing credentials for permitted users

```
[student@workstation ~]$ lab data-review start
```

Instructions

1. In the working directory, create the `playbook.yml` playbook and add the `webserver` host group as the managed host. Define the following play variables:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Variables

Variable	Values
`firewall_pkg`	`firewalld`
`firewall_svc`	`firewalld`
`web_pkg`	`httpd`
`web_svc`	`httpd`
`ssl_pkg`	`mod_ssl`
`httpdconf_src`	`files/httpd.conf`
`httpdconf_dest`	`/etc/httpd/conf/httpd.conf`
`htaccess_src`	`files/.htaccess`
`secrets_dir`	`/etc/httpd/secrets`
`secrets_src`	`files/htpasswd`
`secrets_dest`	`"{{ secrets_dir }}/htpasswd"`
`web_root`	`/var/www/html`

2. Add a `tasks` section to the play. Write a task that ensures the latest version of the necessary packages are installed. These packages are defined by the `firewall_pkg`, `web_pkg`, and `ssl_pkg` variables.
3. Add a second task to the playbook that ensures that the file specified by the `httpdconf_src` variable has been copied (with the `copy` module) to the location specified by the `httpdconf_dest` variable on the managed host. The file should be owned by the `root` user and the `root` group. Also set 0644 as the file permissions.
4. Add a third task that uses the `file` module to create the directory specified by the `secrets_dir` variable on the managed host. This directory holds the password files used for the basic authentication of web services. The file should be owned by the `apache` user and the `apache` group. Set 0500 as the file permissions.
5. Add a fourth task that uses the `copy` module to place a htpasswd file, used for basic authentication of web users. The source should be defined by the `secrets_src` variable. The destination should be defined by the `secrets_dest` variable. The file should be owned by the `apache` user and group. Set `0400` as the file permissions.
6. Add a fifth task that uses the `copy` module to create a `.htaccess` file in the document root directory of the web server. Copy the file specified by the `htaccess_src` variable to `{{ web_root }}/.htaccess`. The file should be owned by the `apache` user and the `apache` group. Set 0400 as the file permissions.
7. Add a sixth task that uses the `copy` module to create the web content file `index.html` in the directory specified by the `web_root` variable. The file should contain the message "*HOSTNAME* (*IPADDRESS*) has been customized by Ansible.", where `HOSTNAME` is the fully-qualified host name of the managed host and `IPADDRESS` is its IPv4 IP address. Use the

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

`content` option to the `copy` module to specify the content of the file, and Ansible facts to specify the host name and IP address.

8. Add a seventh task that uses the `service` module to enable and start the firewall service on the managed host.

9. Add an eighth task that uses the `firewalld` module to allow the `https` service needed for users to access web services on the managed host. This firewall change should be permanent and should take place immediately.

10. Add a final task that uses the `service` module to enable and start the web service on the managed host for all configuration changes to take effect. The name of the web service is defined by the `web_svc` variable.

11. Define a second play targeted at `localhost` which will test authentication to the web server. It does not need privilege escalation. Define a variable named `web_user` with the value `guest`.

12. Add a directive to the play that adds additional variables from a variable file named `vars/secret.yml`. This file contains a variable named `web_pass` that specifies the password for the web user. You will create this file later in the lab.

    Define the start of the task list.

13. Add two tasks to the second play.

    The first uses the `uri` module to request content from `https://serverb.lab.example.com` using basic authentication. Use the `web_user` and `web_pass` variables to authenticate to the web server. Note that the certificate presented by `serverb` will not be trusted, so you will need to avoid certificate validation. The task should verify a return HTTP status code of `200`. Configure the task to place the returned content in the task results variable. Register the task result in a variable.

    The second task uses the `debug` module to print the content returned from the web server.

14. Create a file encrypted with Ansible Vault, named `vars/secret.yml`. Use the password `redhat` to encrypt it. It should set the `web_pass` variable to `redhat`, which will be the web user's password.

15. Run the `playbook.yml` playbook. Verify that content is successfully returned from the web server, and that it matches what was configured in an earlier task.

Evaluation

Run the `lab data-review grade` command on *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab data-review grade
```

Finish

On `workstation`, run the `lab data-review finish` command to clean up this exercise.

```
[student@workstation ~]$ lab data-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Managing Variables and Facts

Performance Checklist

In this lab, you will write and run an Ansible Playbook that uses variables, secrets, and facts.

Outcomes

You should be able to define variables and use facts in a playbook, as well as use variables defined in an encrypted file.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab data-review start` command. The script creates the `/home/student/data-review` working directory and populates it with an Ansible configuration file and host inventory. The managed host `serverb.lab.example.com` is defined in this inventory as a member of the `webserver` host group. A developer has asked you to write an Ansible Playbook to automate the setup of a web server environment on `serverb.lab.example.com`, which controls user access to its website using basic authentication.

The `files` subdirectory contains:

- A `httpd.conf` configuration file for the Apache web service for basic authentication
- A `.htaccess` file, used to control access to the web server's document root directory
- A `htpasswd` file containing credentials for permitted users

```
[student@workstation ~]$ lab data-review start
```

Instructions

1. In the working directory, create the `playbook.yml` playbook and add the `webserver` host group as the managed host. Define the following play variables:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Variables

Variable	Values
`firewall_pkg`	`firewalld`
`firewall_svc`	`firewalld`
`web_pkg`	`httpd`
`web_svc`	`httpd`
`ssl_pkg`	`mod_ssl`
`httpdconf_src`	`files/httpd.conf`
`httpdconf_dest`	`/etc/httpd/conf/httpd.conf`
`htaccess_src`	`files/.htaccess`
`secrets_dir`	`/etc/httpd/secrets`
`secrets_src`	`files/htpasswd`
`secrets_dest`	`"{{ secrets_dir }}/htpasswd"`
`web_root`	`/var/www/html`

1.1. Change to the `/home/student/data-review` working directory.

```
[student@workstation ~]$ cd ~/data-review
[student@workstation data-review]$
```

1.2. Create the `playbook.yml` playbook file and edit it in a text editor. The beginning of the file should appear as follows:

```
---
- name: install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    httpdconf_src: files/httpd.conf
    httpdconf_dest: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_src: files/htpasswd
    secrets_dest: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2. Add a `tasks` section to the play. Write a task that ensures the latest version of the necessary packages are installed. These packages are defined by the `firewall_pkg`, `web_pkg`, and `ssl_pkg` variables.

   2.1. Define the beginning of the `tasks` section by adding the following line to the playbook:

```
tasks:
```

   2.2. Add the following lines to the playbook to define a task that uses the `yum` module to install the required packages.

```
  - name: latest version of necessary packages installed
    yum:
      name:
        - "{{ firewall_pkg }}"
        - "{{ web_pkg }}"
        - "{{ ssl_pkg }}"
      state: latest
```

3. Add a second task to the playbook that ensures that the file specified by the `httpdconf_src` variable has been copied (with the `copy` module) to the location specified by the `httpdconf_dest` variable on the managed host. The file should be owned by the `root` user and the `root` group. Also set 0644 as the file permissions.

   Add the following lines to the playbook to define a task that uses the `copy` module to copy the contents of the file defined by the `httpdconf_src` variable to the location specified by the `httpdconf_dest` variable.

```
  - name: configure web service
    copy:
      src: "{{ httpdconf_src }}"
      dest: "{{ httpdconf_dest }}"
      owner: root
      group: root
      mode: 0644
```

4. Add a third task that uses the `file` module to create the directory specified by the `secrets_dir` variable on the managed host. This directory holds the password files used for the basic authentication of web services. The file should be owned by the `apache` user and the `apache` group. Set 0500 as the file permissions.

   Add the following lines to the playbook to define a task that uses the `file` module to create the directory defined by the `secrets_dir` variable.

```
  - name: secrets directory exists
    file:
      path: "{{ secrets_dir }}"
      state: directory
      owner: apache
      group: apache
      mode: 0500
```

5. Add a fourth task that uses the `copy` module to place a htpasswd file, used for basic authentication of web users. The source should be defined by the `secrets_src` variable.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The destination should be defined by the `secrets_dest` variable. The file should be owned by the `apache` user and group. Set `0400` as the file permissions.

```
  - name: htpasswd file exists
    copy:
      src: "{{ secrets_src }}"
      dest: "{{ secrets_dest }}"
      owner: apache
      group: apache
      mode: 0400
```

6. Add a fifth task that uses the `copy` module to create a `.htaccess` file in the document root directory of the web server. Copy the file specified by the `htaccess_src` variable to `{{ web_root }}/.htaccess`. The file should be owned by the `apache` user and the `apache` group. Set 0400 as the file permissions.

   Add the following lines to the playbook to define a task which uses the `copy` module to create the `.htaccess` file using the file defined by the `htaccess_src` variable.

```
  - name: .htaccess file installed in docroot
    copy:
      src: "{{ htaccess_src }}"
      dest: "{{ web_root }}/.htaccess"
      owner: apache
      group: apache
      mode: 0400
```

7. Add a sixth task that uses the `copy` module to create the web content file `index.html` in the directory specified by the `web_root` variable. The file should contain the message "*HOSTNAME* (*IPADDRESS*) has been customized by Ansible.", where `HOSTNAME` is the fully-qualified host name of the managed host and `IPADDRESS` is its IPv4 IP address. Use the `content` option to the `copy` module to specify the content of the file, and Ansible facts to specify the host name and IP address.

   Add the following lines to the playbook to define a task that uses the `copy` module to create the `index.html` file in the directory defined by the `web_root` variable. Populate the file with the content specified using the `ansible_facts['fqdn']` and `ansible_facts['default_ipv4']['address']` Ansible facts retrieved from the managed host.

```
  - name: create index.html
    copy:
      content: "{{ ansible_facts['fqdn'] }} ({{ ansible_facts['default_ipv4']
['address'] }}) has been customized by Ansible.\n"
      dest: "{{ web_root }}/index.html"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

8. Add a seventh task that uses the `service` module to enable and start the firewall service on the managed host.

   Add the following lines to the playbook to define a task that uses the `service` module to enable and start the firewall service.

```
  - name: firewall service enabled and started
    service:
      name: "{{ firewall_svc }}"
      state: started
      enabled: true
```

9. Add an eighth task that uses the `firewalld` module to allow the `https` service needed for users to access web services on the managed host. This firewall change should be permanent and should take place immediately.

   Add the following lines to the playbook to define a task that uses the `firewalld` module to open the HTTPS port for the web service.

```
  - name: open the port for the web server
    firewalld:
      service: https
      state: enabled
      immediate: true
      permanent: true
```

10. Add a final task that uses the `service` module to enable and start the web service on the managed host for all configuration changes to take effect. The name of the web service is defined by the `web_svc` variable.

```
  - name: web service enabled and started
    service:
      name: "{{ web_svc }}"
      state: started
      enabled: true
```

11. Define a second play targeted at `localhost` which will test authentication to the web server. It does not need privilege escalation. Define a variable named `web_user` with the value `guest`.

    11.1. Add the following line to define the start of a second play. Note that there is no indentation.

```
- name: test web server with basic auth
```

    11.2. Add the following line to indicate that the play applies to the `localhost` managed host.

```
  hosts: localhost
```

    11.3. Add the following line to disable privilege escalation.

```
  become: no
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

11.4. Add the following lines to define a variables list and the `web_user` variable.

```
vars:
  web_user: guest
```

12. Add a directive to the play that adds additional variables from a variable file named `vars/secret.yml`. This file contains a variable named `web_pass` that specifies the password for the web user. You will create this file later in the lab.

Define the start of the task list.

12.1. Using the `vars_files` keyword, add the following lines to the playbook to instruct Ansible to use variables found in the `vars/secret.yml` variable file.

```
vars_files:
  - vars/secret.yml
```

12.2. Add the following line to define the beginning of the `tasks` list.

```
tasks:
```

13. Add two tasks to the second play.

The first uses the `uri` module to request content from `https://serverb.lab.example.com` using basic authentication. Use the `web_user` and `web_pass` variables to authenticate to the web server. Note that the certificate presented by `serverb` will not be trusted, so you will need to avoid certificate validation. The task should verify a return HTTP status code of `200`. Configure the task to place the returned content in the task results variable. Register the task result in a variable.

The second task uses the `debug` module to print the content returned from the web server.

13.1. Add the following lines to create the task for verifying the web service from the control node. Be sure to indent the first line with four spaces.

```
  - name: connect to web server with basic auth
    uri:
      url: https://serverb.lab.example.com
      validate_certs: no
      force_basic_auth: yes
      user: "{{ web_user }}"
      password: "{{ web_pass }}"
      return_content: yes
      status_code: 200
    register: auth_test
```

13.2. Create the second task using the debug module. The content returned from the web server is added to the registered variable as the key `content`.

```
  - debug:
      var: auth_test.content
```

13.3. The completed playbook should appear as follows:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    httpdconf_src: files/httpd.conf
    httpdconf_dest: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_src: files/htpasswd
    secrets_dest: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
  tasks:
    - name: latest version of necessary packages installed
      yum:
        name:
          - "{{ firewall_pkg }}"
          - "{{ web_pkg }}"
          - "{{ ssl_pkg }}"
        state: latest

    - name: configure web service
      copy:
        src: "{{ httpdconf_src }}"
        dest: "{{ httpdconf_dest }}"
        owner: root
        group: root
        mode: 0644

    - name: secrets directory exists
      file:
        path: "{{ secrets_dir }}"
        state: directory
        owner: apache
        group: apache
        mode: 0500

    - name: htpasswd file exists
      copy:
        src: "{{ secrets_src }}"
        dest: "{{ secrets_dest }}"
        owner: apache
        group: apache
        mode: 0400

    - name: .htaccess file installed in docroot
      copy:
        src: "{{ htaccess_src }}"
        dest: "{{ web_root }}/.htaccess"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        owner: apache
        group: apache
        mode: 0400

    - name: create index.html
      copy:
        content: "{{ ansible_facts['fqdn'] }} ({{ ansible_facts['default_ipv4']
['address'] }}) has been customized by Ansible.\n"
        dest: "{{ web_root }}/index.html"

    - name: firewall service enable and started
      service:
        name: "{{ firewall_svc }}"
        state: started
        enabled: true

    - name: open the port for the web server
      firewalld:
        service: https
        state: enabled
        immediate: true
        permanent: true

    - name: web service enabled and started
      service:
        name: "{{ web_svc }}"
        state: started
        enabled: true

- name: test web server with basic auth
  hosts: localhost
  become: no
  vars:
    - web_user: guest
  vars_files:
    - vars/secret.yml
  tasks:
    - name: connect to web server with basic auth
      uri:
        url: https://serverb.lab.example.com
        validate_certs: no
        force_basic_auth: yes
        user: "{{ web_user }}"
        password: "{{ web_pass }}"
        return_content: yes
        status_code: 200
      register: auth_test

    - debug:
        var: auth_test.content
```

13.4. Save and close the `playbook.yml` file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

14. Create a file encrypted with Ansible Vault, named `vars/secret.yml`. Use the password `redhat` to encrypt it. It should set the `web_pass` variable to `redhat`, which will be the web user's password.

14.1. Create a subdirectory named `vars` in the working directory.

```
[student@workstation data-review]$ mkdir vars
```

14.2. Create the encrypted variable file, `vars/secret.yml`, using Ansible Vault. Set the password for the encrypted file to `redhat`.

```
[student@workstation data-review]$ ansible-vault create vars/secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

14.3. Add the following variable definition to the file.

```
web_pass: redhat
```

14.4. Save and close the file.

15. Run the `playbook.yml` playbook. Verify that content is successfully returned from the web server, and that it matches what was configured in an earlier task.

15.1. Before running the playbook, verify that its syntax is correct by running `ansible-playbook --syntax-check`. Use the `--ask-vault-pass` to be prompted for the vault password. Enter `redhat` when prompted for the password. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation data-review]$ ansible-playbook --syntax-check \
> --ask-vault-pass playbook.yml
Vault password: redhat

playbook: playbook.yml
```

15.2. Using the `ansible-playbook` command, run the playbook with the `--ask-vault-pass` option. Enter `redhat` when prompted for the password.

```
[student@workstation data-review]$ ansible-playbook playbook.yml --ask-vault-pass
Vault password: redhat
PLAY [Install and configure webserver with basic auth] ********************

...output omitted...

TASK [connect to web server with basic auth] **********************************
ok: [localhost]

TASK [debug] ******************************************************************
ok: [localhost] => {
    "auth_test.content": "serverb.lab.example.com (172.25.250.11) has been
 customized by Ansible.\n"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
}

PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0
serverb.lab.example.com    : ok=10   changed=8    unreachable=0    failed=0
```

Evaluation

Run the `lab data-review grade` command on *workstation* to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab data-review grade
```

Finish

On `workstation`, run the `lab data-review finish` command to clean up this exercise.

```
[student@workstation ~]$ lab data-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- Ansible *variables* allow administrators to reuse values across files in an entire Ansible project.
- Variables can be defined for hosts and host groups in the inventory file.
- Variables can be defined for playbooks by using facts and external files. They can also be defined on the command line.
- The `register` keyword can be used to capture the output of a command in a variable.
- Ansible Vault is one way to protect sensitive data such as password hashes and private keys for deployment using Ansible Playbooks.
- Ansible *facts* are variables that are automatically discovered by Ansible from a managed host.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 4

Implementing Task Control

Goal	Manage task control, handlers, and task errors in Ansible Playbooks.
Objectives	• Use loops to write efficient tasks and use conditions to control when to run tasks. • Implement a task that runs only when another task changes the managed host. • Control what happens when a task fails, and what conditions cause a task to fail.
Sections	• Writing Loops and Conditional Tasks (and Guided Exercise) • Implementing Handlers (and Guided Exercise) • Handling Task Failure (and Guided Exercise)
Lab	• Implementing Task Control

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Writing Loops and Conditional Tasks

Objectives

After completing this section, you should be able to use loops to write efficient tasks and use conditions to control when to run tasks.

Task Iteration with Loops

Using loops saves administrators from the need to write multiple tasks that use the same module. For example, instead of writing five tasks to ensure five users exist, you can write one task that iterates over a list of five users to ensure they all exist.

Ansible supports iterating a task over a set of items using the `loop` keyword. You can configure loops to repeat a task using each item in a list, the contents of each of the files in a list, a generated sequence of numbers, or using more complicated structures. This section covers simple loops that iterate over a list of items. Consult the documentation for more advanced looping scenarios.

Simple Loops

A simple loop iterates a task over a list of items. The `loop` keyword is added to the task, and takes as a value the list of items over which the task should be iterated. The loop variable `item` holds the value used during each iteration.

Consider the following snippet that uses the `service` module twice in order to ensure two network services are running:

```
- name: Postfix is running
  service:
    name: postfix
    state: started

- name: Dovecot is running
  service:
    name: dovecot
    state: started
```

These two tasks can be rewritten to use a simple loop so that only one task is needed to ensure both services are running:

```
- name: Postfix and Dovecot are running
  service:
    name: "{{ item }}"
    state: started
  loop:
    - postfix
    - dovecot
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The list used by `loop` can be provided by a variable. In the following example, the variable `mail_services` contains the list of services that need to be running.

```
vars:
  mail_services:
    - postfix
    - dovecot

tasks:
  - name: Postfix and Dovecot are running
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ mail_services }}"
```

Loops over a List of Hashes or Dictionaries

The `loop` list does not need to be a list of simple values. In the following example, each item in the list is actually a hash or a dictionary. Each hash or dictionary in the example has two keys, `name` and `groups`, and the value of each key in the current `item` loop variable can be retrieved with the `item.name` and `item.groups` variables, respectively.

```
- name: Users exist and are in the correct groups
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: jane
      groups: wheel
    - name: joe
      groups: root
```

The outcome of the preceding task is that the user `jane` is present and a member of the group `wheel`, and that the user `joe` is present and a member of the group `root`.

Earlier-Style Loop Keywords

Before Ansible 2.5, most playbooks used a different syntax for loops. Multiple loop keywords were provided, which were prefixed with `with_`, followed by the name of an Ansible look-up plug-in (an advanced feature not covered in detail in this course). This syntax for looping is very common in existing playbooks, but will probably be deprecated at some point in the future.

A few examples are listed in the table below:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Earlier-Style Ansible Loops

Loop keyword	Description
`with_items`	Behaves the same as the `loop` keyword for simple lists, such as a list of strings or a list of hashes/dictionaries. Unlike `loop`, if lists of lists are provided to `with_items`, they are flattened into a single-level list. The loop variable `item` holds the list item used during each iteration.
`with_file`	This keyword requires a list of control node file names. The loop variable `item` holds the content of a corresponding file from the file list during each iteration.
`with_sequence`	Instead of requiring a list, this keyword requires parameters to generate a list of values based on a numeric sequence. The loop variable `item` holds the value of one of the generated items in the generated sequence during each iteration.

An example of `with_items` in a playbook is shown below:

```
vars:
  data:
    - user0
    - user1
    - user2
tasks:
  - name: "with_items"
    debug:
      msg: "{{ item }}"
    with_items: "{{ data }}"
```

Important

Since Ansible 2.5, the recommended way to write loops is to use the `loop` keyword.

However, you should still understand the old syntax, especially `with_items`, because it is widely used in existing playbooks. You are likely to encounter playbooks and roles that continue to use `with_*` keywords for looping.

Any task using the old syntax can be converted to use `loop` in conjunction with Ansible filters. You do not need to know how to use Ansible filters to do this. There is a good reference on how to convert the old loops to the new syntax, as well as examples of how to loop over items that are not simple lists, in the Ansible documentation in the section Migrating from with_X to loop [https://docs.ansible.com/ansible/2.9/user_guide/playbooks_loops.html#migrating-from-with-x-to-loop] of the *Ansible User Guide*.

You will likely encounter tasks from older playbooks that contain `with_*` keywords.

Advanced looping techniques are beyond the scope of this course. All iteration tasks in this course can be implemented with either the `with_items` or the `loop` keyword.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Using Register Variables with Loops

The `register` keyword can also capture the output of a task that loops. The following snippet shows the structure of the register variable from a task that loops:

```
[student@workstation loopdemo]$ cat loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results (1)

    - name: Show echo_results variable
      debug:
        var: echo_results (2)
```

(1) The `echo_results` variable is registered.

(2) The contents of the `echo_results` variable are displayed to the screen.

Running the above playbook yields the following output:

```
[student@workstation loopdemo]$ ansible-playbook loop_register.yml
PLAY [Loop Register Test] ******************************************************

TASK [Looping Echo Task] *******************************************************
...output omitted...
TASK [Show echo_results variable] **********************************************
ok: [localhost] => {
    "echo_results": { (1)
        "changed": true,
        "msg": "All items completed",
        "results": [ (2)
            { (3)
                "_ansible_ignore_errors": null,
                ...output omitted...
                "changed": true,
                "cmd": "echo This is my item: one",
                "delta": "0:00:00.011865",
                "end": "2018-11-01 16:32:56.080433",
                "failed": false,
                ...output omitted...
                "item": "one",
                "rc": 0,
                "start": "2018-11-01 16:32:56.068568",
                "stderr": "",
                "stderr_lines": [],
                "stdout": "This is my item: one",
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
                "stdout_lines": [
                    "This is my item: one"
                ]
            },
            {(4)
                "_ansible_ignore_errors": null,
                ...output omitted...
                "changed": true,
                "cmd": "echo This is my item: two",
                "delta": "0:00:00.011142",
                "end": "2018-11-01 16:32:56.828196",
                "failed": false,
                ...output omitted...
                "item": "two",
                "rc": 0,
                "start": "2018-11-01 16:32:56.817054",
                "stderr": "",
                "stderr_lines": [],
                "stdout": "This is my item: two",
                "stdout_lines": [
                    "This is my item: two"
                ]
            }
        ](5)
    }
}
...output omitted...
```

(1) The { character indicates that the start of the `echo_results` variable is composed of key-value pairs.

(2) The `results` key contains the results from the previous task. The [character indicates the start of a list.

(3) The start of task metadata for the first item (indicated by the `item` key). The output of the `echo` command is found in the `stdout` key.

(4) The start of task result metadata for the second item.

(5) The] character indicates the end of the `results` list.

In the above, the `results` key contains a list. Below, the playbook is modified such that the second task iterates over this list:

```
[student@workstation loopdemo]$ cat new_loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
      register: echo_results

    - name: Show stdout from the previous task.
      debug:
        msg: "STDOUT from previous task: {{ item.stdout }}"
      loop: "{{ echo_results['results'] }}"
```

After executing the above playbook, the output is:

```
PLAY [Loop Register Test] ******************************************************

TASK [Looping Echo Task] *******************************************************
...output omitted...

TASK [Show stdout from the previous task.] *************************************
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: one"
}
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: two"
}
...output omitted...
```

Running Tasks Conditionally

Ansible can use *conditionals* to execute tasks or plays when certain conditions are met. For example, a conditional can be used to determine 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 to compare strings, numeric data, and Boolean values are available.

The following scenarios illustrate the use of conditionals in Ansible:

- A hard limit can be defined in a variable (for example, `min_memory`) and compared 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 is skipped.
- Use Ansible facts 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.
- Compare a registered variable with a predefined variable to determine if a service changed. For example, test the MD5 checksum of a service configuration file to see if the service is changed.

Conditional Task Syntax

The `when` statement is used to run a task conditionally. It takes as a value the condition to test. If the condition is met, the task runs. If the condition is not met, the task is skipped.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

One of the simplest conditions that can be tested is whether a Boolean variable is true or false. The `when` statement in the following example causes the task to run only if `run_my_task` is true:

```
---
- name: Simple Boolean Task Demo
  hosts: all
  vars:
    run_my_task: true

  tasks:
    - name: httpd package is installed
      yum:
        name: httpd
      when: run_my_task
```

The next example is a bit more sophisticated, and tests whether the `my_service` variable has a value. If it does, the value of `my_service` is used as the name of the package to install. If the `my_service` variable is not defined, then the task is skipped without an error.

```
---
- name: Test Variable is Defined Demo
  hosts: all
  vars:
    my_service: httpd

  tasks:
    - name: "{{ my_service }} package is installed"
      yum:
        name: "{{ my_service }}"
      when: my_service is defined
```

The following table shows some of the operations that administrators can use when working with conditionals:

Example Conditionals

Operation	Example
Equal (value is a string)	`ansible_machine == "x86_64"`
Equal (value is numeric)	`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`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Operation	Example
Variable does not exist	`min_memory is not defined`
Boolean variable is `true`. The values of `1`, `True`, or `yes` evaluate to `true`.	`memory_available`
Boolean variable is `false`. The values of `0`, `False`, or `no` evaluate to `false`.	`not memory_available`
First variable's value is present as a value in second variable's list	`ansible_distribution in supported_distros`

The last entry in the preceding table might be confusing at first. The following example illustrates how it works.

In the example, the `ansible_distribution` variable is a fact determined during the `Gathering Facts` task, and identifies the managed host's operating system distribution. The variable `supported_distros` was created by the playbook author, and contains a list of operating system distributions that the playbook supports. If the value of `ansible_distribution` is in the `supported_distros` list, the conditional passes and the task runs.

```
---
- name: Demonstrate the "in" keyword
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
      - RedHat
      - Fedora
  tasks:
    - name: Install httpd using yum, where supported
      yum:
        name: http
        state: present
      when: ansible_distribution in supported_distros
```

Important

Notice the indentation 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.

A task is a YAML hash/dictionary, and the `when` statement is simply one more key in the task like the task's name and the module it uses. A common convention places any `when` keyword that might be present after the task's name and the module (and module arguments).

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Testing Multiple Conditions

One `when` statement can be used to evaluate multiple conditionals. To do so, conditionals can be combined with either the `and` or `or` keywords, and grouped with parentheses.

The following snippets show some examples of how to express multiple conditions.

- If a conditional statement should be met when either condition is true, then you should use the `or` statement. For example, the following condition is met if the machine is running either Red Hat Enterprise Linux or Fedora:

```
when: ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

- With the `and` operation, both conditions have to be true for the entire conditional statement to be met. For example, the following condition is met if the remote host is a Red Hat Enterprise Linux 7.5 host, and the installed kernel is the specified version:

```
when: ansible_distribution_version == "7.5" and ansible_kernel ==
 "3.10.0-327.el7.x86_64"
```

  The `when` keyword also supports using a list to describe a list of conditions. When a list is provided to the `when` keyword, all of the conditionals are combined using the `and` operation. The example below demonstrates another way to combine multiple conditional statements using the `and` operator:

```
when:
  - ansible_distribution_version == "7.5"
  - ansible_kernel == "3.10.0-327.el7.x86_64"
```

  This format improves readability, a key goal of well-written Ansible Playbooks.

- More complex conditional statements can be expressed by grouping conditions with parentheses. This ensures 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 28. This example uses the greater-than character (`>`) so that the long conditional can be split over multiple lines in the playbook, to make it easier to read.

```
when: >
    ( ansible_distribution == "RedHat" and
      ansible_distribution_major_version == "7" )
    or
    ( ansible_distribution == "Fedora" and
    ansible_distribution_major_version == "28" )
```

Combining Loops and Conditional Tasks

You can combine loops and conditionals.

In the following example, the *mariadb-server* package is 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.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: latest
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 300000000
```

Important

When you use `when` with `loop` for a task, the `when` statement is checked for each item.

Here is another example that combines conditionals and register variables. The following annotated playbook restarts the `httpd` service only if the `postfix` service is running:

```
---
- name: Restart HTTPD if Postfix is Running
  hosts: all
  tasks:
    - name: Get Postfix server status
      command: /usr/bin/systemctl is-active postfix (1)
      ignore_errors: yes (2)
      register: result (3)

    - name: Restart Apache HTTPD based on Postfix status
      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`.

(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 restarts the `httpd` service.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

References

Loops – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_loops.html

Tests – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_tests.html

Conditionals – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html

What Makes A Valid Variable Name – Variables – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#what-makes-a-valid-variable-name

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Writing Loops and Conditional Tasks

In this exercise, you will write a playbook containing tasks that have conditionals and loops.

Outcomes

You should be able to:

- Implement Ansible conditionals using the `when` keyword.
- Implement task iteration using the `loop` keyword in conjunction with conditionals.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab control-flow start` command. This script creates the working directory, `/home/student/control-flow`.

```
[student@workstation ~]$ lab control-flow start
```

Instructions

1. On `workstation.lab.example.com`, change to the `/home/student/control-flow` project directory.

```
[student@workstation ~]$ cd ~/control-flow
[student@workstation control-flow]$
```

2. The lab script created an Ansible configuration file as well as an inventory file. This inventory file contains the server `servera.lab.example.com` in the `database_dev` host group, and the `serverb.lab.example.com` in the `database_prod` host group. Review the file before proceeding.

```
[student@workstation control-flow]$ cat inventory
[database_dev]
servera.lab.example.com

[database_prod]
serverb.lab.example.com
```

3. Create the `playbook.yml` playbook, which contains a play with two tasks. Use the `database_dev` host group. The first task installs the MariaDB required packages, and the second task ensures that the MariaDB service is running.

   3.1. Open the playbook in a text editor. Define the variable `mariadb_packages` with two values: `mariadb-server`, and `python3-PyMySQL`. The playbook uses the variable to install the required packages. The file should read as follows:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: MariaDB server is running
  hosts: database_dev
  vars:
    mariadb_packages:
      - mariadb-server
      - python3-PyMySQL
```

3.2. Define a task that uses the `yum` module and the variable `mariadb_packages`. The task installs the required packages. The task should read as follows:

```
  tasks:
    - name: MariaDB packages are installed
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ mariadb_packages }}"
```

3.3. Define a second task to start the `mariadb` service. The task should read as follows:

```
    - name: Start MariaDB service
      service:
        name: mariadb
        state: started
        enabled: true
```

4. Run the playbook and watch the output of the play.

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [MariaDB packages are installed] ******************************************
changed: [servera.lab.example.com] => (item=mariadb-server)
changed: [servera.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

5. Update the first task to execute only if the managed host uses Red Hat Enterprise Linux as its operating system. Update the play to use the `database_prod` host group. The task should read as follows:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: MariaDB server is running
  hosts: database_prod
  vars:
...output omitted...
  tasks:
    - name: MariaDB packages are installed
      yum:
        name: "{{ item }}"
        state: present
      loop: "{{ mariadb_packages }}"
      when: ansible_distribution == "RedHat"
```

6. Verify that the managed hosts in the `database_prod` host group use Red Hat Enterprise Linux as its operating system.

```
[student@workstation control-flow]$ ansible database_prod -m command \
> -a 'cat /etc/redhat-release' -u devops --become
serverb.lab.example.com | CHANGED | rc=0 >>
Red Hat Enterprise Linux release 8.4 (Ootpa)
```

7. Run the playbook again and watch the output of the play.

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [MariaDB packages are installed] ******************************************
changed: [serverb.lab.example.com] => (item=mariadb-server)
changed: [serverb.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ***************************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

Ansible executes the task because `serverb.lab.example.com` uses Red Hat Enterprise Linux.

Finish

On `workstation`, run the `lab control-flow finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab control-flow finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Implementing Handlers

Objectives

After completing this section, you should be able to implement a task that runs only when another task changes the managed host.

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 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. Tasks only notify their handlers when the task changes something on a managed host. Each handler is triggered by its name after the play's block of tasks. If no task notifies the handler by name then the handler will not run. If one or more tasks notify the handler, the handler 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 use for any other task. Normally, handlers are used to reboot hosts and restart services.

Note

Use unique names for your handlers. When multiple handlers are defined with the same name, only the last handler defined with the shared name will run.

Handlers can be considered 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` handler when a configuration file is updated and notifies it:

```
tasks:
  - name: copy demo.example.conf configuration template(1)
    template:
      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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

❷ The `notify` statement indicates the task needs to trigger a handler.

❸ The name of the handler to run.

❹ The `handlers` keyword indicates the start of the list of handler tasks.

❺ The name of the handler invoked by tasks.

❻ The module to use for the handler.

In the previous example, the `restart apache` handler triggers when notified by the `template` task that a change happened. A task may call more than one handler in its `notify` section. Ansible treats the `notify` statement as an array and iterates over the handler names:

```
tasks:
  - name: copy demo.example.conf configuration template
    template:
      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
```

Describing the Benefits of Using Handlers

As discussed in the Ansible documentation, there are some important things to remember about using handlers:

- Handlers always run in the order specified by the `handlers` section of the play. They do not run in the order in which they are listed by `notify` statements in a task, or in the order in which tasks notify them.
- Handlers normally 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* tasks under `tasks` have been processed. (There are some minor exceptions to this.)
- Handler names exist in a per-play namespace. If two handlers are incorrectly given the same name, only one will run.
- Even if more than one task notifies a handler, the handler only runs once. If no tasks notify it, a handler will not run.
- If a task that includes a `notify` statement does not report a `changed` result (for example, a package is already installed and the task reports `ok`), the handler is not notified. The handler is skipped unless another task notifies it. Ansible notifies handlers only if the task reports the `changed` status.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

Handlers are meant to perform an extra action when a task makes a change to a managed host. They should not be used as a replacement for normal tasks.

References

Intro to Playbooks — Ansible Documentation

https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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 control-handlers start` on `workstation` to configure the environment for the exercise. This script creates the `/home/student/control-handlers` project directory and downloads the Ansible configuration file and the host inventory file needed for the exercise. The project directory also contains a partially complete playbook, `configure_db.yml`.

```
[student@workstation ~]$ lab control-handlers start
```

Instructions

1. On `workstation.lab.example.com`, open a new terminal and change to the `/home/student/control-handlers` project directory.

```
[student@workstation ~]$ cd ~/control-handlers
[student@workstation control-handlers]$
```

2. In that directory, use a text editor to edit the `configure_db.yml` playbook file. This playbook installs and configures a database server. When the database server configuration changes, the playbook triggers a restart of the database service and configures the database administrative password.

   2.1. Using a text editor, review the `configure_db.yml` playbook. It begins with the initialization of some variables:

```
---
- name: MariaDB server is installed
  hosts: databases
  vars:
    db_packages: (1)
      - mariadb-server
      - python3-PyMySQL
    db_service: mariadb (2)
    resources_url: http://materials.example.com/labs/control-handlers (3)
    config_file_url: "{{ resources_url }}/my.cnf.standard" (4)
    config_file_dst: /etc/my.cnf (5)
  tasks:
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1. `db_packages` defines the name of the packages to install for the database service.
2. `db_service` defines the name of the database service.
3. `resources_url` represents the URL for the resource directory where remote configuration files are located.
4. `config_file_url` represents the URL of the database configuration file to install.
5. `config_file_dst`: Location of the installed configuration file on the managed hosts.

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. If the task changes the system, the database was not installed, and you need to notify the `set db password` handler to set your initial database user and password. Remember that the handler task, if it is notified, will not run until every task in the `tasks` section has run.

The task should read as follows:

```
tasks:
  - name: "{{ db_packages }} packages are installed"
    yum:
      name: "{{ db_packages }}"
      state: present
    notify:
      - set db password
```

2.3. 2.3)

Add a task to start and enable the database service. The task should read as follows:

```
  - name: Make sure the database service is running
    service:
      name: "{{ db_service }}"
      state: started
      enabled: true
```

2.4. Add a task to download `my.cnf.standard` to `/etc/my.cnf` on the managed host, using the `get_url` module. Add a condition that notifies the `restart db service` handler to restart the database service after a configuration file change. The task should read:

```
  - name: The {{ config_file_dst }} file has been installed
    get_url:
      url: "{{ config_file_url }}"
      dest: "{{ config_file_dst }}"
      owner: mysql
      group: mysql
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        force: yes
      notify:
        - restart db service
```

2.5. Add the `handlers` keyword to define the start of the handler tasks. Define the first handler, `restart db service`, which restarts the `mariadb` service. It should read as follows:

```
  handlers:
    - name: restart db service
      service:
        name: "{{ db_service }}"
        state: restarted
```

2.6. Define the second handler, `set db password`, which sets the administrative password for the database service. The handler uses the `mysql_user` module to perform the command. The handler should read as follows:

```
    - name: set db password
      mysql_user:
        name: root
        password: redhat
```

When completed, the playbook should appear as follows:

```
---
- name: MariaDB server is installed
  hosts: databases
  vars:
    db_packages:
      - mariadb-server
      - python3-PyMySQL
    db_service: mariadb
    resources_url: http://materials.example.com/labs/control-handlers
    config_file_url: "{{ resources_url }}/my.cnf.standard"
    config_file_dst: /etc/my.cnf
  tasks:
    - name: "{{ db_packages }} packages are installed"
      yum:
        name: "{{ db_packages }}"
        state: present
      notify:
        - set db password

    - name: Make sure the database service is running
      service:
        name: "{{ db_service }}"
        state: started
        enabled: true

    - name: The {{ config_file_dst }} file has been installed
      get_url:
        url: "{{ config_file_url }}"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        dest: "{{ config_file_dst }}"
        owner: mysql
        group: mysql
        force: yes
      notify:
        - restart db service

  handlers:
    - name: restart db service
      service:
        name: "{{ db_service }}"
        state: restarted

    - name: set db password
      mysql_user:
        name: root
        password: redhat
```

3. Before running the playbook, verify that its syntax is correct by running `ansible-playbook` with the `--syntax-check` option. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml \
> --syntax-check

playbook: configure_db.yml
```

4. Run the `configure_db.yml` playbook. The output shows that the handlers are being executed.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [['mariadb-server', 'python3-PyMySQL'] packages are installed] **********
changed: [servera.lab.example.com]

TASK [Make sure the database service is running] ****************************
changed: [servera.lab.example.com]

TASK [The /etc/my.cnf file has been installed] *****************************
changed: [servera.lab.example.com]

RUNNING HANDLER [restart db service] ***************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [set db password] ******************************************
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=6    changed=5    unreachable=0    failed=0
```

5. Run the playbook again.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [['mariadb-server', 'python3-PyMySQL'] packages are installed] **********
ok: [servera.lab.example.com]

TASK [Make sure the database service is running] ****************************
ok: [servera.lab.example.com]

TASK [The /etc/my.cnf file has been installed] *****************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=0    unreachable=0    failed=0
```

This time the handlers are skipped. In the event that the remote configuration file is changed in the future, executing the playbook would trigger the `restart db service` handler but not the `set db password` handler.

Finish

On `workstation`, run the `lab control-handlers finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab control-handlers finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Handling Task Failure

Objectives

After completing this section, you should be able to control what happens when a task fails, and what conditions cause a task to fail.

Managing Task Errors in Plays

Ansible evaluates the return code of each task to determine whether the task succeeded or failed. Normally, when a task fails Ansible immediately aborts the rest of the play on that host, skipping all subsequent tasks.

However, sometimes you might want to have play execution continue even if a task fails. For example, you might expect that a particular task could fail, and you might want to recover by running some other task conditionally. There are a number of Ansible features that can be used to manage task errors.

Ignoring Task Failure

By default, if a task fails, the play is aborted. However, this behavior can be overridden by ignoring failed tasks. You can use the `ignore_errors` keyword in a task to accomplish this.

The following snippet shows how to use `ignore_errors` in a task to continue playbook execution on the host even if the task fails. For example, if the *notapkg* package does not exist then the yum module fails, but having `ignore_errors` set to `yes` allows execution to continue.

```
- name: Latest version of notapkg is installed
  yum:
    name: notapkg
    state: latest
  ignore_errors: yes
```

Forcing Execution of Handlers after Task Failure

Normally when a task fails and the play aborts on that host, any handlers that had been notified by earlier tasks in the play will not run. If you set the `force_handlers: yes` keyword on the play, then notified handlers are called even if the play aborted because a later task failed.

The following snippet shows hows to use the `force_handlers` keyword in a play to force execution of the handler even if a task fails:

```
---
- hosts: all
  force_handlers: yes
  tasks:
    - name: a task which always notifies its handler
      command: /bin/true
      notify: restart the database
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: a task which fails because the package doesn't exist
      yum:
        name: notapkg
        state: latest

  handlers:
    - name: restart the database
      service:
        name: mariadb
        state: restarted
```

Note

Remember that handlers are notified when a task reports a `changed` result but are not notified when it reports an `ok` or `failed` result.

Specifying Task Failure Conditions

You can use the `failed_when` keyword on a task to specify which conditions indicate that the task has failed. This is often used with command modules that may successfully execute a command, but the command's output indicates a failure.

For example, you can run a script that outputs an error 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:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    failed_when: "'Password missing' in command_result.stdout"
```

The `fail` module can also be used to force a task failure. The above scenario can alternatively be written as two tasks:

```
tasks:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    ignore_errors: yes

  - name: Report script failure
    fail:
      msg: "The password is missing in the output"
    when: "'Password missing' in command_result.stdout"
```

You can use the `fail` module to provide a clear failure message for the task. This approach also enables delayed failure, allowing you to run intermediate tasks to complete or roll back other changes.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Specifying When a Task Reports "Changed" Results

When a task makes a change to a managed host, it reports the `changed` state and notifies handlers. When a task does not need to make a change, it reports `ok` and does not notify handlers.

The `changed_when` keyword can be used to control when a task reports that it has changed. For example, the `shell` module in the next example is being used to get a Kerberos credential which will be used by subsequent tasks. It normally would always report `changed` when it runs. To suppress that change, `changed_when: false` is set so that it only reports `ok` or `failed`.

```
  - name: get Kerberos credentials as "admin"
    shell: echo "{{ krb_admin_pass }}" | kinit -f admin
    changed_when: false
```

The following example uses the `shell` module to report `changed` based on the output of the module that is collected by a registered variable:

```
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 logically group tasks, and can be used to control how tasks are executed. For example, a task block can have a `when` keyword to apply a conditional to multiple tasks:

```
- name: block example
  hosts: all
  tasks:
    - name: installing and configuring Yum versionlock plugin
      block:
      - name: package needed by yum
        yum:
          name: yum-plugin-versionlock
          state: present
      - name: lock version of tzdata
        lineinfile:
          dest: /etc/yum/pluginconf.d/versionlock.list
          line: tzdata-2016j-1
          state: present
      when: ansible_distribution == "RedHat"
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Blocks also allow for error handling in combination with the `rescue` and `always` statements. If any task in a block fails, tasks in its `rescue` block are executed in order to recover. After the tasks in the block clause run, as well as the tasks in the rescue clause if there was a failure, then tasks in the `always` clause run. To summarize:

- `block`: Defines the main tasks to run.
- `rescue`: Defines the tasks to run if the tasks defined in the `block` clause fail.
- `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` clauses are executed.

```
tasks:
  - name: Upgrade DB
    block:
      - name: upgrade the database
        shell:
          cmd: /usr/local/lib/upgrade-database
    rescue:
      - name: revert the database upgrade
        shell:
          cmd: /usr/local/lib/revert-database
    always:
      - name: always restart the database
        service:
          name: mariadb
          state: restarted
```

The `when` condition on a `block` clause also applies to its `rescue` and `always` clauses if present.

References

Error Handling in Playbooks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_error_handling.html

Error Handling — Blocks — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_blocks.html#blocks-error-handling

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Handling Task Failure

In this exercise, you will explore different ways to handle task failure in an Ansible Playbook.

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 `block`, `rescue`, and `always` in playbooks.

Before You Begin

On `workstation`, run the lab start script to confirm the environment is ready for the lab to begin. This script creates the working directory, `/home/student/control-errors`.

```
[student@workstation ~]$ lab control-errors start
```

Instructions

1. On `workstation.lab.example.com`, change to the `/home/student/control-errors` project directory.

```
[student@workstation ~]$ cd ~/control-errors
[student@workstation control-errors]$
```

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 `databases` group. Review the file before proceeding.

3. Create the `playbook.yml` playbook, which contains a play with two tasks. Write the first task with a deliberate error to cause failure.

   3.1. Open the playbook in a text editor. Define 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`. These 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:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Task Failure Exercise
  hosts: databases
  vars:
    web_package: http
    db_package: mariadb-server
    db_service: mariadb
```

3.2. Define two tasks that use the `yum` module and the two variables, `web_package` and `db_package`. The tasks will install the required packages. The tasks should read as follows:

```
  tasks:
    - name: Install {{ web_package }} package
      yum:
        name: "{{ web_package }}"
        state: present

    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: present
```

4. Run the playbook and watch the output of the play.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures":
 ["No package http available."], "msg": "Failed to install some of the specified
 packages", "rc": 1, "results": []}

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=1    changed=0    unreachable=0    failed=1
```

The task failed because there is no existing package called `http`. Because the first task failed, the second task was not run.

5. 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: present
      ignore_errors: yes
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: Install {{ db_package }} package
      yum:
        name: "{{ db_package }}"
        state: present
```

6. Run the playbook again and watch the output of the play.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures":
 ["No package http available."], "msg": "Failed to install some of the specified
 packages", "rc": 1, "results": []}
...ignoring

TASK [Install mariadb-server package] ******************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=0
```

Despite the fact that the first task failed, Ansible executed the second one.

7. In this step, you will set up a `block` keyword so you can experiment with how they work.

7.1. Update the playbook by nesting the first task in a `block` clause. Remove the line that sets `ignore_errors: yes`. The block should read as follows:

```
    - name: Attempt to set up a webserver
      block:
        - name: Install {{ web_package }} package
          yum:
            name: "{{ web_package }}"
            state: present
```

7.2. Nest the task that installs the *mariadb-server* package in a `rescue` clause. The task will execute if the task listed in the `block` clause fails. The block clause should read as follows:

```
      rescue:
        - name: Install {{ db_package }} package
          yum:
            name: "{{ db_package }}"
            state: present
```

7.3. Finally, add an `always` clause to start the database server upon installation using the `service` module. The clause should read as follows:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    always:
      - name: Start {{ db_service }} service
        service:
          name: "{{ db_service }}"
          state: started
```

7.4. The completed task section should read as follows:

```
tasks:
  - name: Attempt to set up a webserver
    block:
      - name: Install {{ web_package }} package
        yum:
          name: "{{ web_package }}"
          state: present
    rescue:
      - name: Install {{ db_package }} package
        yum:
          name: "{{ db_package }}"
          state: present
    always:
      - name: Start {{ db_service }} service
        service:
          name: "{{ db_service }}"
          state: started
```

8. Now run the playbook again and observe the output.

8.1. Run the playbook. The task in the block that makes sure `web_package` is installed fails, which causes the task in the `rescue` block to run. Then the task in the `always` block runs.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install http package] ****************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures":
 ["No package http available."], "msg": "Failed to install some of the specified
 packages", "rc": 1, "results": []}

TASK [Install mariadb-server package] ******************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=1
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

8.2. Edit the playbook, correcting the value of the `web_package` variable to read `httpd`. That will cause the task in the block to succeed the next time you run the playbook.

```
  vars:
    web_package: httpd
    db_package: mariadb-server
    db_service: mariadb
```

8.3. Run the playbook again. This time, the task in the block does not fail. This causes the task in the `rescue` section to be ignored. The task in the `always` will still run.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Install httpd package] ***************************************************
changed: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=1    unreachable=0    failed=0
```

9. This step explores how to control the condition that causes a task to be reported as "changed" to the managed host.

9.1. Edit the playbook to add two tasks to the start of the play, preceding the `block`. The first task uses the `command` module to run the `date` command and register the result in the `command_result` variable. The second task uses the `debug` module to print the standard output of the first task's command.

```
  tasks:
    - name: Check local time
      command: date
      register: command_result

    - name: Print local time
      debug:
        var: command_result.stdout
```

9.2. Run the playbook. You should see that the first task, which runs the `command` module, reports `changed`, even though it did not change the remote system; it only collected information about the time. That is because the `command` module cannot tell the difference between a command that collects data and a command that changes state.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Gathering Facts] ***********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] **********************************************************
changed: [servera.lab.example.com]

TASK [Print local time] **********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:07:08 EDT 2019"
}

TASK [Install httpd package] *****************************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] *****************************************************
ok: [servera.lab.example.com]

PLAY RECAP ***********************************************************************
servera.lab.example.com    : ok=5    changed=1    unreachable=0    failed=0
```

If you run the playbook again, the `Check local time` task returns `changed` again.

9.3. That `command` task should not report `changed` every time it runs because it is not changing the managed host. Because you know that the task will never change a managed host, add the line `changed_when: false` to the task to suppress the change.

```
  tasks:
    - name: Check local time
      command: date
      register: command_result
      changed_when: false

    - name: Print local time
      debug:
        var: command_result.stdout
```

9.4. Run the playbook again and notice that the task now reports `ok`, but the task is still being run and is still saving the time in the variable.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] *****************************************************

TASK [Gathering Facts] ***********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] **********************************************************
ok: [servera.lab.example.com]

TASK [Print local time] **********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:08:36 EDT 2019"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
}

TASK [Install httpd package] ***************************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=0    unreachable=0    failed=0
```

10. As a final exercise, edit the playbook to explore how the `failed_when` keyword interacts with tasks.

10.1. Edit the `Install {{ web_package }} package` task so that it reports as having failed when `web_package` has the value `httpd`. Because this is the case, the task will report failure when you run the play.

Be careful with your indentation to make sure the keyword is correctly set on the task.

```
    - block:
        - name: Install {{ web_package }} package
          yum:
            name: "{{ web_package }}"
            state: present
          failed_when: web_package == "httpd"
```

10.2. Run the playbook.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Check local time] ********************************************************
ok: [servera.lab.example.com]

TASK [Print local time] ********************************************************
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:09:35 EDT 2019"
}

TASK [Install httpd package] ***************************************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false,
 "failed_when_result": true, "msg": "Nothing to do", "rc": 0, "results":
 ["Installed: httpd"]}

TASK [Install mariadb-server package] ******************************************
ok: [servera.lab.example.com]

TASK [Start mariadb service] ***************************************************
ok: [servera.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=0    unreachable=0    failed=1
```

Look carefully at the output. The `Install httpd package` task *reports* that it failed, but it actually ran and made sure the package is installed first. The `failed_when` keyword changes the status the task reports *after* the task runs; it does not change the behavior of the task itself.

However, the reported failure might change the behavior of the rest of the play. Because that task was in a block and reported that it failed, the `Install mariadb-server package` task in the block's `rescue` section was run.

Finish

On `workstation`, run the `lab control-errors finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab control-errors finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Implementing Task Control

Performance Checklist

In this lab, you will install the Apache web server and secure it using `mod_ssl`. You will use conditions, handlers, and task failure handling in your playbook 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, and handle task errors.

Before You Begin

Log in as the `student` user on `workstation` and run `lab control-review start`. This 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 control-review start
```

Instructions

1. On `workstation`, change to the `/home/student/control-review` project directory.
2. The project directory contains a partially completed playbook, `playbook.yml`. Using a text editor, add a task that uses the `fail` module under the `#Fail Fast Message` comment. Be sure to provide an appropriate name for the task. This task should only be executed when the remote system does not meet the minimum requirements.

   The minimum requirements for the remote host are listed below:

   - Has at least the amount of RAM specified by the `min_ram_mb` variable. The `min_ram_mb` variable is defined in the `vars.yml` file and has a value of `256`.
   - Is running Red Hat Enterprise Linux.
3. Add a single task to the playbook under the `#Install all Packages` comment to install the latest version of any missing packages. Required packages are specified by the `packages` variable, which is defined in the `vars.yml` file.

   The task name should be `Ensure required packages are present`.
4. Add a single task to the playbook under the `#Enable and start services` comment to start all services. All services specified by the `services` variable, which is defined in the `vars.yml` file, should be started and enabled. Be sure to provide an appropriate name for the task.
5. Add a task block to the playbook under the `#Block of config tasks` comment. This block contains two tasks:

   - A task to ensure the directory specified by the `ssl_cert_dir` variable exists on the remote host. This directory stores the web server's certificates.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- A task to copy all files specified by the `web_config_files` variable to the remote host. Examine the structure of the `web_config_files` variable in the `vars.yml` file. Configure the task to copy each file to the correct destination on the remote host.

  This task should trigger the `restart web service` handler if any of these files are changed on the remote server.

Additionally, a debug task is executed if either of the two tasks above fail. In this case, the task prints the message: `One or more of the configuration changes failed, but the web service is still active.`

Be sure to provide an appropriate name for all tasks.

6. The playbook configures the remote host to listen for standard HTTPS requests. Add a single task to the playbook under the `#Configure the firewall` comment to configure `firewalld`.

   This task should ensure that the remote host allows standard HTTP and HTTPS connections. These configuration changes should be effective immediately and persist after a system reboot. Be sure to provide an appropriate name for the task.

7. Define the `restart web service` handler.

   When triggered, this task should restart the web service defined by the `web_service` variable, defined in the `vars.yml` file.

8. From the project directory, `~/control-review`, run the `playbook.yml` playbook. The playbook should execute without errors, and trigger the execution of the handler task.

9. Verify that the web server now responds to HTTPS requests, using the self-signed custom certificate to encrypt the connection. The web server response should match the string `Configured for both HTTP and HTTPS.`

Evaluation

Run the `lab control-review grade` command on `workstation` to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab control-review grade
```

Finish

Run the `lab control-review finish` command to clean up after the lab.

```
[student@workstation ~]$ lab control-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Implementing Task Control

Performance Checklist

In this lab, you will install the Apache web server and secure it using `mod_ssl`. You will use conditions, handlers, and task failure handling in your playbook 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, and handle task errors.

Before You Begin

Log in as the `student` user on `workstation` and run `lab control-review start`. This 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 control-review start
```

Instructions

1. On `workstation`, change to the `/home/student/control-review` project directory.

```
[student@workstation ~]$ cd ~/control-review
[student@workstation control-review]$
```

2. The project directory contains a partially completed playbook, `playbook.yml`. Using a text editor, add a task that uses the `fail` module under the `#Fail Fast Message` comment. Be sure to provide an appropriate name for the task. This task should only be executed when the remote system does not meet the minimum requirements.

   The minimum requirements for the remote host are listed below:

   - Has at least the amount of RAM specified by the `min_ram_mb` variable. The `min_ram_mb` variable is defined in the `vars.yml` file and has a value of `256`.
   - Is running Red Hat Enterprise Linux.

   The completed task matches:

```
tasks:
  #Fail Fast Message
  - name: Show Failed System Requirements Message
    fail:
      msg: "The {{ inventory_hostname }} did not meet minimum reqs."
    when: >
      ansible_memtotal_mb < min_ram_mb or
      ansible_distribution != "RedHat"
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. Add a single task to the playbook under the `#Install all Packages` comment to install the latest version of any missing packages. Required packages are specified by the `packages` variable, which is defined in the `vars.yml` file.

   The task name should be `Ensure required packages are present`.

   The completed task matches:

```
#Install all Packages
- name: Ensure required packages are present
  yum:
    name: "{{ packages }}"
    state: latest
```

4. Add a single task to the playbook under the `#Enable and start services` comment to start all services. All services specified by the `services` variable, which is defined in the `vars.yml` file, should be started and enabled. Be sure to provide an appropriate name for the task.

   The completed task matches:

```
#Enable and start services
- name: Ensure services are started and enabled
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ services }}"
```

5. Add a task block to the playbook under the `#Block of config tasks` comment. This block contains two tasks:

   - A task to ensure the directory specified by the `ssl_cert_dir` variable exists on the remote host. This directory stores the web server's certificates.
   - A task to copy all files specified by the `web_config_files` variable to the remote host. Examine the structure of the `web_config_files` variable in the `vars.yml` file. Configure the task to copy each file to the correct destination on the remote host.

     This task should trigger the `restart web service` handler if any of these files are changed on the remote server.

   Additionally, a debug task is executed if either of the two tasks above fail. In this case, the task prints the message: `One or more of the configuration changes failed, but the web service is still active.`

   Be sure to provide an appropriate name for all tasks.

   The completed task block matches below:

```
#Block of config tasks
- name: Setting up the SSL cert directory and config files
  block:
    - name: Create SSL cert directory
      file:
        path: "{{ ssl_cert_dir }}"
        state: directory
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        - name: Copy Config Files
          copy:
            src: "{{ item.src }}"
            dest: "{{ item.dest }}"
          loop: "{{ web_config_files }}"
          notify: restart web service

      rescue:
        - name: Configuration Error Message
          debug:
            msg: >
              One or more of the configuration
              changes failed, but the web service
              is still active.
```

6. The playbook configures the remote host to listen for standard HTTPS requests. Add a single task to the playbook under the `#Configure the firewall` comment to configure `firewalld`.

   This task should ensure that the remote host allows standard HTTP and HTTPS connections. These configuration changes should be effective immediately and persist after a system reboot. Be sure to provide an appropriate name for the task.

   The completed task matches:

```
  #Configure the firewall
  - name: ensure web server ports are open
    firewalld:
      service: "{{ item }}"
      immediate: true
      permanent: true
      state: enabled
    loop:
      - http
      - https
```

7. Define the `restart web service` handler.

   When triggered, this task should restart the web service defined by the `web_service` variable, defined in the `vars.yml` file.

   A `handlers` section is added to the end of the playbook:

```
handlers:
  - name: restart web service
    service:
      name: "{{ web_service }}"
      state: restarted
```

   The completed playbook contains:

```
---
- name: Playbook Control Lab
  hosts: webservers
  vars_files: vars.yml
  tasks:
    #Fail Fast Message
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  - name: Show Failed System Requirements Message
    fail:
      msg: "The {{ inventory_hostname }} did not meet minimum reqs."
    when: >
      ansible_memtotal_mb < min_ram_mb or
      ansible_distribution != "RedHat"

  #Install all Packages
  - name: Ensure required packages are present
    yum:
      name: "{{ packages }}"
      state: latest

  #Enable and start services
  - name: Ensure services are started and enabled
    service:
      name: "{{ item }}"
      state: started
      enabled: yes
    loop: "{{ services }}"

  #Block of config tasks
  - name: Setting up the SSL cert directory and config files
    block:
      - name: Create SSL cert directory
        file:
          path: "{{ ssl_cert_dir }}"
          state: directory

      - name: Copy Config Files
        copy:
          src: "{{ item.src }}"
          dest: "{{ item.dest }}"
        loop: "{{ web_config_files }}"
        notify: restart web service

    rescue:
      - name: Configuration Error Message
        debug:
          msg: >
            One or more of the configuration
            changes failed, but the web service
            is still active.

  #Configure the firewall
  - name: ensure web server ports are open
    firewalld:
      service: "{{ item }}"
      immediate: true
      permanent: true
      state: enabled
    loop:
      - http
      - https
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  #Add handlers
  handlers:
    - name: restart web service
      service:
        name: "{{ web_service }}"
        state: restarted
```

8. From the project directory, `~/control-review`, run the `playbook.yml` playbook. The playbook should execute without errors, and trigger the execution of the handler task.

```
[student@workstation control-review]$ ansible-playbook playbook.yml

PLAY [Playbook Control Lab] ****************************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [Show Failed System Requirements Message] *********************************
skipping: [serverb.lab.example.com]

TASK [Ensure required packages are present] *************************************
changed: [serverb.lab.example.com]

TASK [Ensure services are started and enabled] **********************************
changed: [serverb.lab.example.com] => (item=httpd)
ok: [serverb.lab.example.com] => (item=firewalld)

TASK [Create SSL cert directory] ************************************************
changed: [serverb.lab.example.com]

TASK [Copy Config Files] ********************************************************
changed: [serverb.lab.example.com] => (item={'src': 'server.key', 'dest': '/etc/
httpd/conf.d/ssl'})
changed: [serverb.lab.example.com] => (item={'src': 'server.crt', 'dest': '/etc/
httpd/conf.d/ssl'})
changed: [serverb.lab.example.com] => (item={'src': 'ssl.conf', 'dest': '/etc/
httpd/conf.d'})
changed: [serverb.lab.example.com] => (item={'src': 'index.html', 'dest': '/var/
www/html'})

TASK [ensure web server ports are open] *****************************************
changed: [serverb.lab.example.com] => (item=http)
changed: [serverb.lab.example.com] => (item=https)

RUNNING HANDLER [restart web service] *******************************************
changed: [serverb.lab.example.com]

PLAY RECAP **********************************************************************
serverb.lab.example.com    : ok=7    changed=6    unreachable=0    failed=0
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

9. Verify that the web server now responds to HTTPS requests, using the self-signed custom certificate to encrypt the connection. The web server response should match the string `Configured for both HTTP and HTTPS.`

```
[student@workstation control-review]$ curl -k -vvv https://serverb.lab.example.com
* About to connect() to serverb.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to serverb.lab.example.com (172.25.250.11) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
...output omitted...
*  start date: Nov 13 15:52:18 2018 GMT
*  expire date: Aug 09 15:52:18 2021 GMT
*  common name: serverb.lab.example.com
...output omitted...
< Accept-Ranges: bytes
< Content-Length: 36
< Content-Type: text/html; charset=UTF-8
<
Configured for both HTTP and HTTPS.
* Connection #0 to host serverb.lab.example.com left intact
```

Evaluation

Run the `lab control-review grade` command on `workstation` to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab control-review grade
```

Finish

Run the `lab control-review finish` command to clean up after the lab.

```
[student@workstation ~]$ lab control-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- Loops are used to iterate over a set of values, for example, a simple list of strings, or a list of either hashes or dictionaries.
- Conditionals are used to execute tasks or plays only when certain conditions have been met.
- Handlers are special tasks that execute at the end of the play if notified by other tasks.
- Handlers are only notified when a task reports that it changed something on a managed host.
- Tasks are 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 are used to group tasks as a unit and to execute other tasks depending upon whether or not all the tasks in the block succeed.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 5

Deploying Files to Managed Hosts

Goal

Deploy, manage, and adjust files on hosts managed by Ansible.

Objectives

- Create, install, edit, and remove files on managed hosts, and manage permissions, ownership, SELinux context, and other characteristics of those files.
- Deploy files to managed hosts that are customized by using Jinja2 templates.

Sections

- Modifying and Copying Files to Hosts (and Guided Exercise)
- Deploying Custom Files with Jinja2 Templates (and Guided Exercise)

Lab

- Deploying Files to Managed Hosts

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Modifying and Copying Files to Hosts

Objectives

After completing this section, you should be able to create, install, edit, and remove files on managed hosts, and manage permissions, ownership, SELinux context, and other characteristics of those files.

Describing Files Modules

Red Hat Ansible Automation Platform ships with a large collection of modules (the "module library") that are developed as part of the upstream Ansible project. To make it easier to organize, document, and manage them, they are organized into groups based on function in the documentation and when installed on a system.

The `Files` modules library includes modules allowing you to accomplish most tasks related to Linux file management, such as creating, copying, editing, and modifying permissions and other attributes of files. The following table provides a list of frequently used file management modules:

Commonly Used Files Modules

Module name	Module description
`blockinfile`	Insert, update, or remove a block of multiline text surrounded by customizable marker lines.
`copy`	Copy a file from the local or remote machine to a location on a managed host. Similar to the `file` module, the `copy` module can also set file attributes, including SELinux context.
`fetch`	This module works like the `copy` module, but in reverse. This module is used for fetching files from remote machines to the control node and storing them in a file tree, organized by host name.
`file`	Set attributes such as permissions, ownership, SELinux contexts, and time stamps of regular files, symlinks, hard links, and directories. This module can also create or remove regular files, symlinks, hard links, and directories. A number of other file-related modules support the same options to set attributes as the `file` module, including the `copy` module.
`lineinfile`	Ensure that a particular line is in a file, or replace an existing line using a back-reference regular expression. This module is primarily useful when you want to change a single line in a file.
`stat`	Retrieve status information for a file, similar to the Linux `stat` command.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Module name	Module description
`synchronize`	A wrapper around the `rsync` command to make common tasks quick and easy. The `synchronize` module is not intended to provide access to the full power of the `rsync` command, but does make the most common invocations easier to implement. You may still need to call the `rsync` command directly via the `run command` module depending on your use case.

Automation Examples with Files Modules

Creating, copying, editing, and removing files on managed hosts are common tasks that you can implement using modules from the `Files` modules library. The following examples show ways that you can use these modules to automate common file management tasks.

Ensuring a File Exists on Managed Hosts

Use the `file` module to touch a file on managed hosts. This works like the `touch` command, creating an empty file if it does not exist, and updating its modification time if it does exist. In this example, in addition to touching the file, Ansible ensures that the owning user, group, and permissions of the file are set to specific values.

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

Example outcome:

```
[user@host ~]$ ls -l file
-rw-r-----.  user1 group1 0 Nov 25 08:00 file
```

Modifying File Attributes

You can use the `file` module to ensure that a new or existing file has the correct permissions or SELinux type as well.

For example, the following file has retained the default SELinux context relative to a user's home directory, which is not the desired context.

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:user_home_t:s0 samba_file
```

The following task ensures that the SELinux context type attribute of the `samba_file` file is the desired `samba_share_t` type. This behavior is similar to the Linux `chcon` command.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: SELinux type is set to samba_share_t
  file:
    path: /path/to/samba_file
    setype: samba_share_t
```

Example outcome:

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

Note

File attribute parameters are available in multiple file management modules. Run the `ansible-doc file` and `ansible-doc copy` commands for additional information.

Making SELinux File Context Changes Persistent

The `file` module acts like `chcon` when setting file contexts. Changes made with that module could be unexpectedly undone by running `restorecon`. After using `file` to set the context, you can use `sefcontext` from the collection of `System` modules to update the SELinux policy like `semanage fcontext`.

```
- name: SELinux type is persistently set to samba_share_t
  sefcontext:
    target: /path/to/samba_file
    setype: samba_share_t
    state: present
```

Example outcome:

```
[user@host ~]$ ls -Z samba_file
-rw-r--r--.  owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

Important

The `sefcontext` module updates the default context for the target in the SELinux policy, but does not change the context on existing files.

Copying and Editing Files on Managed Hosts

In this example, the `copy` module is used to copy a file located in the Ansible working directory on the control node to selected managed hosts.

By default this module assumes that `force: yes` is set. That forces the module to overwrite the remote file if it exists but contains different contents from the file being copied. If `force: no` is set, then it only copies the file to the managed host if it does not already exist.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

To retrieve files from managed hosts use the `fetch` module. This could be used to retrieve a file such as an SSH public key from a reference system before distributing it to other managed hosts.

```
- name: Retrieve SSH key from reference host
  fetch:
    src: "/home/{{ user }}/.ssh/id_rsa.pub
    dest: "files/keys/{{ user }}.pub"
```

To ensure a specific single line of text exists in an existing file, use the `lineinfile` module:

```
- name: Add a line of text to a file
  lineinfile:
    path: /path/to/file
    line: 'Add this line to the file'
    state: present
```

To add a block of text to an existing file, use the `blockinfile` module:

```
- name: Add additional lines to a file
  blockinfile:
    path: /path/to/file
    block: |
      First line in the additional block of text
      Second line in the additional block of text
    state: present
```

Note

When using the `blockinfile` module, commented block markers are inserted at the beginning and end of the block to ensure idempotency.

```
# BEGIN ANSIBLE MANAGED BLOCK
First line in the additional block of text
Second line in the additional block of text
# END ANSIBLE MANAGED BLOCK
```

You can use the `marker` parameter to the module to help ensure that the right comment character or text is being used for the file in question.

Removing a File from Managed Hosts

A basic example to remove a file from managed hosts is to use the `file` module with the `state: absent` parameter. The `state` parameter is optional to many modules. You should always make your intentions clear whether you want `state: present` or `state: absent` for several reasons. Some modules support other options as well. It is possible that the default could change

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

at some point, but perhaps most importantly, it makes it easier to understand the state the system should be in based on your task.

```
- name: Make sure a file does not exist on managed hosts
  file:
    dest: /path/to/file
    state: absent
```

Retrieving the Status of a File on Managed Hosts

The `stat` module retrieves facts for a file, similar to the Linux `stat` command. Parameters provide the functionality to retrieve file attributes, determine the checksum of a file, and more.

The `stat` module returns a hash dictionary of values containing the file status data, which allows you to refer to individual pieces of information using separate variables.

The following example registers the results of a `stat` module and then prints the MD5 checksum of the file that it checked. (The more modern SHA256 algorithm is also available; MD5 is being used here for easier legibility.)

```
- name: Verify the checksum of a file
  stat:
    path: /path/to/file
    checksum_algorithm: md5
  register: result

- debug
    msg: "The checksum of the file is {{ result.stat.checksum }}"
```

The outcome should be similar to the following:

```
TASK [Get md5 checksum of a file] ************************************
ok: [hostname]

TASK [debug] *********************************************************
ok: [hostname] => {
    "msg": "The checksum of the file is 5f76590425303022e933c43a7f2092a3"
}
```

Information about the values returned by the `stat` module are documented by `ansible-doc`, or you can register a variable and display its contents to see what is available:

```
- name: Examine all stat output of /etc/passwd
  hosts: localhost

  tasks:
    - name: stat /etc/passwd
      stat:
        path: /etc/passwd
      register: results
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: Display stat results
      debug:
        var: results
```

Synchronizing Files Between the Control Node and Managed Hosts

The `synchronize` module is a wrapper around the `rsync` tool, which simplifies common file management tasks in your playbooks. The `rsync` tool must be installed on both the local and remote host. By default, when using the `synchronize` module, the "local host" is the host that the synchronize task originates on (usually the control node), and the "destination host" is the host that `synchronize` connects to.

The following example synchronizes a file located in the Ansible working directory to the managed hosts:

```
- name: synchronize local file to remote files
  synchronize:
    src: file
    dest: /path/to/file
```

There are many ways to use the `synchronize` module and its many parameters, including synchronizing directories. Run the `ansible-doc synchronize` command for additional parameters and playbook examples.

References

`ansible-doc`(1), `chmod`(1), `chown`(1), `rsync`(1), `stat`(1) and `touch`(1) man pages

Files modules

https://docs.ansible.com/ansible/2.9/modules/list_of_files_modules.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Modifying and Copying Files to Hosts

In this exercise, you will use standard Ansible modules to create, install, edit, and remove files on managed hosts and manage the permissions, ownership, and SELinux contexts of those files.

Outcomes

You should be able to:

- Retrieve files from managed hosts, by host name, and store them locally.
- Create playbooks that use common file management modules such as `copy`, `file`, `lineinfile`, and `blockinfile`.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab file-manage start` command. The script creates the `file-manage` project directory, and downloads the Ansible configuration file and the host inventory file needed for the exercise.

```
[student@workstation ~]$ lab file-manage start
```

Instructions

1. As the `student` user on `workstation`, change to the `/home/student/file-manage` working directory. Create a playbook called `secure_log_backups.yml` in the current working directory. Configure the playbook to use the `fetch` module to retrieve the `/var/log/secure` log file from each of the managed hosts and store them on the control node. The playbook should create the `secure-backups` directory with subdirectories named after the host name of each managed host. Store the backup files in their respective subdirectories.

    1.1. Navigate to the `/home/student/file-manage` working directory.

```
[student@workstation ~]$ cd ~/file-manage
[student@workstation file-manage]$
```

    1.2. Create the `secure_log_backups.yml` playbook with initial content:

```
---
- name: Use the fetch module to retrieve secure log files
  hosts: all
  remote_user: root
```

    1.3. Add a task to the `secure_log_backups.yml` playbook that retrieves the `/var/log/secure` log file from the managed hosts and stores it in the `~/file-manage/`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

`secure-backups` directory. The `fetch` module creates the `~/file-manage/secure-backups` directory if it does not exist. Use the `flat: no` parameter to ensure the default behavior of appending the host name, path, and file name to the destination:

```
  tasks:
    - name: Fetch the /var/log/secure log file from managed hosts
      fetch:
        src: /var/log/secure
        dest: secure-backups
        flat: no
```

1.4. Before running the playbook, run the `ansible-playbook --syntax-check secure_log_backups.yml` command to verify its syntax. Correct any errors before moving to the next step.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> secure_log_backups.yml

playbook: secure_log_backups.yml
```

1.5. Run `ansible-playbook secure_log_backups.yml` to execute the playbook:

```
[student@workstation file-manage]$ ansible-playbook secure_log_backups.yml
PLAY [Use the fetch module to retrieve secure log files] *****************

TASK [Gathering Facts] ***************************************************
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]

TASK [Fetch the /var/log/secure file from managed hosts] *****************
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ***************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

1.6. Verify the playbook results:

```
[student@workstation file-manage]$ tree -F secure-backups
secure-backups
├── servera.lab.example.com/
│   └── var/
│       └── log/
│           └── secure
└── serverb.lab.example.com/
    └── var/
        └── log/
            └── secure
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2. Create the `copy_file.yml` playbook in the current working directory. Configure the playbook to copy the `/home/student/file-manage/files/users.txt` file to all managed hosts as the root user.

2.1. Add the following initial content to the `copy_file.yml` playbook:

```
---
- name: Using the copy module
  hosts: all
  remote_user: root
```

2.2. Add a task to use the `copy` module to copy the `/home/student/file-manage/files/users.txt` file to all managed hosts. Use the `copy` module to set the following parameters for the `users.txt` file:

Parameter	Values
src	files/users.txt
dest	/home/devops/users.txt
owner	devops
group	devops
mode	u+rw,g-wx,o-rwx
setype	samba_share_t

```
  tasks:
    - name: Copy a file to managed hosts and set attributes
      copy:
        src: files/users.txt
        dest: /home/devops/users.txt
        owner: devops
        group: devops
        mode: u+rw,g-wx,o-rwx
        setype: samba_share_t
```

2.3. Use the `ansible-playbook --syntax-check copy_file.yml` command to verify the syntax of the `copy_file.yml` playbook.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check copy_file.yml

playbook: copy_file.yml
```

2.4. Run the playbook:

```
[student@workstation file-manage]$ ansible-playbook copy_file.yml
PLAY [Using the copy module] ***************************************

TASK [Gathering Facts] *********************************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Copy a file to managed hosts and set attributes] ****************
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP ************************************************************
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

2.5. Use an ad hoc command to execute the `ls -Z` command as user `devops` to verify the attributes of the `users.txt` file on the managed hosts.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:samba_share_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:samba_share_t:s0 users.txt
```

3. In a previous step, the `samba_share_t` SELinux type field was set for the `users.txt` file. However, it is now determined that default values should be set for the SELinux file context.

Create a playbook called `selinux_defaults.yml` in the current working directory. Configure the playbook to use the `file` module to ensure the default SELinux context for `user`, `role`, `type`, and `level` fields.

Note

In the real world you would also edit `copy_file.yml` and remove the `setype` keyword.

3.1. Create the `selinux_defaults.yml` playbook:

```
---
- name: Using the file module to ensure SELinux file context
  hosts: all
  remote_user: root
  tasks:
    - name:  SELinux file context is set to defaults
      file:
        path: /home/devops/users.txt
        seuser: _default
        serole: _default
        setype: _default
        selevel: _default
```

3.2. Use the `ansible-playbook --syntax-check selinux_defaults.yml` command to verify the syntax of the `selinux_defaults.yml` playbook.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> selinux_defaults.yml

playbook: selinux_defaults.yml
```

3.3. Run the playbook:

```
[student@workstation file-manage]$ ansible-playbook selinux_defaults.yml
PLAY [Using the file module to ensure SELinux file context] ************

TASK [Gathering Facts] *************************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [SELinux file context is set to defaults] *************************
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP *************************************************************
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

3.4. Use an ad hoc command to execute the `ls -Z` command as user `devops` to verify the default file attributes of `unconfined_u:object_r:user_home_t:s0`.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt
```

4. Create a playbook called `add_line.yml` in the current working directory. Configure the playbook to use the `lineinfile` module to append the line `This line was added by the lineinfile module.` to the `/home/devops/users.txt` file on all managed hosts.

4.1. Create the `add_line.yml` playbook:

```
---
- name: Add text to an existing file
  hosts: all
  remote_user: devops
  tasks:
    - name: Add a single line of text to a file
      lineinfile:
        path: /home/devops/users.txt
        line: This line was added by the lineinfile module.
        state: present
```

4.2. Use `ansible-playbook --syntax-check add_line.yml` command to verify the syntax of the `add_line.yml` playbook.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_line.yml

playbook: add_line.yml
```

4.3. Run the playbook:

```
[student@workstation file-manage]$ ansible-playbook add_line.yml
PLAY [Add text to an existing file] **********************************

TASK [Gathering Facts] ***********************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Add a single line of text to a file] ***************************
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP ***********************************************************
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

4.4. Use the `command` module with the `cat` option, as the `devops` user, to verify the content of the `users.txt` file on the managed hosts.

```
[student@workstation file-manage]$ ansible all -m command \
> -a 'cat users.txt' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.

servera.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
```

5. Create a playbook called `add_block.yml` in the current working directory. Configure the playbook to use the `blockinfile` module to append the following block of text to the `/home/devops/users.txt` file on all managed hosts.

```
This block of text consists of two lines.
They have been added by the blockinfile module.
```

5.1. Create the `add_block.yml` playbook:

```
---
- name: Add block of text to a file
  hosts: all
  remote_user: devops
  tasks:
    - name: Add a block of text to an existing file
      blockinfile:
        path: /home/devops/users.txt
        block: |
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
          This block of text consists of two lines.
          They have been added by the blockinfile module.
        state: present
```

5.2. Use the `ansible-playbook --syntax-check add_block.yml` command to verify the syntax of the `add_block.yml` playbook.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_block.yml

playbook: add_block.yml
```

5.3. Run the playbook:

```
[student@workstation file-manage]$ ansible-playbook add_block.yml

PLAY [Add block of text to a file] *************************************

TASK [Gathering Facts] *************************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Add a block of text to an existing file] *************************
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP *************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

5.4. Use the `command` module with the `cat` command to verify the correct content of the `/home/devops/users.txt` file on the managed host.

```
[student@workstation file-manage]$ ansible all -m command \
> -a 'cat users.txt' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
# BEGIN ANSIBLE MANAGED BLOCK
This block of text consists of two lines.
They have been added by the blockinfile module.
# END ANSIBLE MANAGED BLOCK

servera.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
# BEGIN ANSIBLE MANAGED BLOCK
This block of text consists of two lines.
They have been added by the blockinfile module.
# END ANSIBLE MANAGED BLOCK
```

6. Create a playbook called `remove_file.yml` in the current working directory. Configure the playbook to use the `file` module to remove the `/home/devops/users.txt` file from all managed hosts.

6.1. Create the `remove_file.yml` playbook:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Use the file module to remove a file
  hosts: all
  remote_user: devops
  tasks:
    - name: Remove a file from managed hosts
      file:
        path: /home/devops/users.txt
        state: absent
```

6.2. Use the `ansible-playbook --syntax-check remove_file.yml` command to verify the syntax of the `remove_file.yml` playbook.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check remove_file.yml

playbook: remove_file.yml
```

6.3. Run the playbook:

```
[student@workstation file-manage]$ ansible-playbook remove_file.yml
PLAY [Use the file module to remove a file] ******************************

TASK [Gathering Facts] ***************************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Remove a file from managed hosts] **********************************
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ***************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

6.4. Use an ad hoc command to execute the `ls -l` command to confirm that the `users.txt` file no longer exists on the managed hosts.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -l' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
total 0

servera.lab.example.com | CHANGED | rc=0 >>
total 0
```

Finish

On `workstation`, run the `lab file-manage finish` script to clean up this exercise.

```
[student@workstation ~]$ lab file-manage finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Deploying Custom Files with Jinja2 Templates

Objectives

After completing this section, you should be able to deploy files to managed hosts that are customized by using Jinja2 templates.

Templating Files

Red Hat Ansible Automation Platform has a number of modules that can be used to modify existing files. These include `lineinfile` and `blockinfile`, among others. However, they are not always easy to use effectively and correctly.

A much more powerful way to manage files is to *template* them. With this method, you can write a template configuration file that is automatically customized for the managed host when the file is deployed, using Ansible variables and facts. This can be easier to control and is less error-prone.

Introduction to Jinja2

Ansible uses the Jinja2 templating system for template files. Ansible also uses Jinja2 syntax to reference variables in playbooks, so you already know a little bit about how to use it.

Using Delimiters

Variables and 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 that should not appear in the final file.

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_facts['default_ipv4']['address'] }}    {{ ansible_facts['hostname'] }}
```

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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The following example shows how to create a template for `/etc/ssh/sshd_config` with variables and facts retrieved by Ansible from managed hosts. When the associated playbook is executed, any facts are replaced by their values in the managed host being configured.

Note

A file containing a Jinja2 template does not need to have any specific file extension (for example, `.j2`). However, providing such a file extension may make it easier for you to remember that it is a template file.

```
# {{ ansible_managed }}
# DO NOT MAKE LOCAL MODIFICATIONS TO THIS FILE AS THEY WILL BE LOST

Port {{ ssh_port }}
ListenAddress {{ ansible_facts['default_ipv4']['address'] }}

HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

SyslogFacility AUTHPRIV

PermitRootLogin {{ root_allowed }}
AllowGroups {{ groups_allowed }}

AuthorizedKeysFile /etc/.rht_authorized_keys .ssh/authorized_keys

PasswordAuthentication {{ passwords_allowed }}

ChallengeResponseAuthentication no

GSSAPIAuthentication yes
GSSAPICleanupCredentials no

UsePAM yes

X11Forwarding yes
UsePrivilegeSeparation sandbox

AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS

Subsystem sftp /usr/libexec/openssh/sftp-server
```

Deploying Jinja2 Templates

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 on the control node to the managed hosts.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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 with 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 also allows you to specify the owner (the user that owns the file), group, permissions, and SELinux context of the deployed file, just like the `file` module. It can also take a `validate` option to run an arbitrary command (such as `visudo -c`) to check the syntax of a file for correctness before copying it into place.

For more details, see `ansible-doc template`.

Managing Templated Files

To avoid having system administrators modify files deployed by Ansible, it is a good practice to include a comment at the top of the template to indicate that the file should not be manually edited.

One way to do this is to use the "Ansible managed" string set in the `ansible_managed` directive. This is not a normal variable but can be used as one in a template. The `ansible_managed` directive is set in the `ansible.cfg` file:

```
ansible_managed = Ansible managed
```

To include the `ansible_managed` string inside a Jinja2 template, use the following syntax:

```
{{ ansible_managed }}
```

Control Structures

You can use Jinja2 control structures in template files to reduce repetitive typing, to enter entries for each host in a play dynamically, or conditionally insert text into a file.

Using 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 the `users` variable, one value per line.

```
{% for user in users %}
      {{ user }}
{% endfor %}
```

The following example template uses a `for` statement to run through all the values in the `users` variable, replacing `myuser` with each value, except when the value is `root`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
{# for statement #}
{% for myuser in users if not myuser == "root" %}
User number {{ 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.

As another example, this template also uses a `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 in the file.

```
{% for myhost in groups['myhosts'] %}
{{ myhost }}
{% endfor %}
```

For a more practical example, you can use this to generate an `/etc/hosts` file from host facts dynamically. Assume that you have the following playbook:

```
- name: /etc/hosts is up to date
  hosts: all
  gather_facts: yes
  tasks:
    - name: Deploy /etc/hosts
      template:
        src: templates/hosts.j2
        dest: /etc/hosts
```

The following three-line `templates/hosts.j2` template constructs the file from all hosts in the group `all`. (The middle line is extremely long in the template due to the length of the variable names.) It iterates over each host in the group to get three facts for the `/etc/hosts` file.

```
{% for host in groups['all'] %}
{{ hostvars[host]['ansible_facts']['default_ipv4']['address'] }} {{ hostvars[host]
['ansible_facts']['fqdn'] }} {{ hostvars[host]['ansible_facts']['hostname'] }}
{% endfor %}
```

Using Conditionals

Jinja2 uses the `if` statement to provide conditional control. This allows you to put a line in a deployed file if certain conditions are met.

In the following example, the value of the `result` variable is placed in the deployed file only if the value of the `finished` variable is `True`.

```
{% if finished %}
{{ result }}
{% endif %}
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

You can use Jinja2 loops and conditionals in Ansible templates, but not in Ansible Playbooks.

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 and JSON. The `to_json` filter formats the expression output using JSON, and the `to_yaml` filter formats the expression output using YAML.

```
{{ 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 strings in either JSON or YAML format, respectively, to parse them.

```
{{ output | from_json }}
{{ output | from_yaml }}
```

Variable Tests

The expressions used with `when` clauses in Ansible Playbooks are Jinja2 expressions. Built-in Ansible tests used to test return values include `failed`, `changed`, `succeeded`, and `skipped`. The following task shows how tests can be used inside of conditional expressions.

```
tasks:
...output omitted...
  - debug: msg="the execution was aborted"
    when: returnvalue is failed
```

References

template - Templates a file out to a remote server – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/template_module.html

Variables – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html

Filters – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_filters.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Deploying Custom Files with Jinja2 Templates

In this exercise, you will create a simple template file that your playbook will use to install a customized Message of the Day file on each managed host.

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 the password.

On `workstation`, run the `lab file-template start` command. This script ensures that Ansible is installed on `workstation`, creates the `/home/student/file-template` directory, and downloads the `ansible.cfg` file into that directory.

```
[student@workstation ~]$ lab file-template start
```

Note

All the files used during this exercise are available for reference on `workstation` in the `/home/student/file-template/files` directory.

Instructions

1. On `workstation`, navigate to the `/home/student/file-template` working directory. Review the `inventory` file in the current working directory. This file configures two groups: `webservers` and `workstations`. The `servera.lab.example.com` system is in the `webservers` group, and the `workstation.lab.example.com` system is in the `workstations` group.

    1.1. Navigate to the `/home/student/file-template` working directory.

```
[student@workstation ~]$ cd ~/file-template
[student@workstation file-template]$
```

    1.2. Display the content of the `inventory` file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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

[workstations]
workstation.lab.example.com
```

2. Create a template for the Message of the Day and include it in the `motd.j2` file in the current working directory. Include the following variables and facts in the template:

   - `ansible_facts['fqdn']`, to insert the FQDN of the managed host.
   - `ansible_facts['distribution']` and `ansible_facts['distribution_version']`, to provide distribution information.
   - `system_owner`, for the system owner's email. This variable needs to be defined with an appropriate value in the `vars` section of the playbook template.

```
This is the system {{ ansible_facts['fqdn'] }}.
This is a {{ ansible_facts['distribution'] }} version
 {{ ansible_facts['distribution_version'] }} system.
Only use this system with permission.
Please report issues to: {{ system_owner }}.
```

3. Create a playbook file named `motd.yml` in the current working directory. Define the `system_owner` variable in the `vars` section, and include a task for the `template` module that maps the `motd.j2` Jinja2 template to the remote file `/etc/motd` on the managed hosts. Set the owner and group to `root`, and the mode to 0644.

```
---
- name: configure SOE
  hosts: all
  remote_user: devops
  become: true
  vars:
    - system_owner: clyde@example.com
  tasks:
    - name: configure /etc/motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```

4. Before running the playbook, use the `ansible-playbook --syntax-check` command to verify the syntax. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation file-template]$ ansible-playbook --syntax-check motd.yml

playbook: motd.yml
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. Run the `motd.yml` playbook.

```
[student@workstation file-template]$ ansible-playbook motd.yml
PLAY [all] ********************************************************************

TASK [Gathering Facts] ********************************************************
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` as the `devops` user to verify that the MOTD is correctly displayed when logged in. Log off when you have finished.

```
[student@workstation file-template]$ ssh devops@servera.lab.example.com
This is the system servera.lab.example.com.
This is a RedHat version 8.4 system.
Only use this system with permission.
Please report issues to: clyde@example.com.
...output omitted...
[devops@servera ~]# exit
Connection to servera.lab.example.com closed.
```

Finish

Run the `lab file-template finish` command to clean up after the exercise.

```
[student@workstation ~]$ lab file-template finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Deploying Files to Managed Hosts

Performance Checklist

In this lab, you will run a playbook that creates a customized file on your managed hosts by using a Jinja2 template.

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 the password.

On `workstation`, run the `lab file-review start` command. This ensures that Ansible is installed on `workstation`, creates the `/home/student/file-review` directory, and downloads the `ansible.cfg` file into that directory. It also downloads the `motd.yml`, `motd.j2`, `issue`, and `inventory` files into the `/home/student/file-review/files` directory.

```
[student@workstation ~]$ lab file-review start
```

Note

All files used in this exercise are available on `workstation` in the `/home/student/file-review/files` directory.

Instructions

1. Review the `inventory` file in the `/home/student/file-review` directory. This inventory file defines the `servers` group, which has the `serverb.lab.example.com` managed host associated with it.
2. Identify the facts on `serverb.lab.example.com` that show the total amount of system memory, and the number of processors.
3. Create a template for the Message of the Day, named `motd.j2`, in the current directory. When the `devops` user logs in to `serverb.lab.example.com`, a message should display that shows the system's total memory and processor count. Use the `ansible_facts['memtotal_mb']` and `ansible_facts['processor_count']` facts to provide the system resource information for the message.
4. Create a new playbook file called `motd.yml` in the current directory. Using the `template` module, configure the `motd.j2` Jinja2 template file previously created to map to the file `/etc/motd` on the managed hosts. This file has the `root` user as owner and group, and its

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

permissions are 0644. Using the `stat` and `debug` modules, create tasks to verify that `/etc/motd` exists on the managed hosts and displays the file information for `/etc/motd`. Use the `copy` module to place `files/issue` into the `/etc/` directory on the managed host, use the same ownership and permissions as `/etc/motd`. Use the `file` module to ensure that `/etc/issue.net` is a symbolic link to `/etc/issue` on the managed host. Configure the playbook so that it uses the `devops` user, and sets the `become` parameter to `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

On `workstation`, run the `lab file-review grade` script to confirm success on this exercise.

```
[student@workstation ~]$ lab file-review grade
```

Finish

On `workstation`, run the `lab file-review finish` script to clean up after the lab.

```
[student@workstation ~]$ lab file-review finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Deploying Files to Managed Hosts

Performance Checklist

In this lab, you will run a playbook that creates a customized file on your managed hosts by using a Jinja2 template.

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 the password.

On `workstation`, run the `lab file-review start` command. This ensures that Ansible is installed on `workstation`, creates the `/home/student/file-review` directory, and downloads the `ansible.cfg` file into that directory. It also downloads the `motd.yml`, `motd.j2`, `issue`, and `inventory` files into the `/home/student/file-review/files` directory.

```
[student@workstation ~]$ lab file-review start
```

Note

All files used in this exercise are available on `workstation` in the `/home/student/file-review/files` directory.

Instructions

1. Review the `inventory` file in the `/home/student/file-review` directory. This inventory file defines the `servers` group, which has the `serverb.lab.example.com` managed host associated with it.

   1.1. On `workstation`, change to the `/home/student/file-review` directory.

```
[student@workstation ~]$ cd ~/file-review/
```

   1.2. Display the content of the `inventory` file.

```
[servers]
serverb.lab.example.com
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2. Identify the facts on `serverb.lab.example.com` that show the total amount of system memory, and the number of processors.

   Use the `setup` module to get a list of all the facts for the `serverb.lab.example.com` managed host. The `ansible_processor_count` and `ansible_memtotal_mb` facts provide information about the resource limits of the managed host.

```
[student@workstation file-review]$ ansible serverb.lab.example.com -m setup
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
 "ansible_processor_count": 1,
...output omitted...
 "ansible_memtotal_mb": 821,
...output omitted...
    },
    "changed": false
}
```

3. Create a template for the Message of the Day, named `motd.j2`, in the current directory. When the `devops` user logs in to `serverb.lab.example.com`, a message should display that shows the system's total memory and processor count. Use the `ansible_facts['memtotal_mb']` and `ansible_facts['processor_count']` facts to provide the system resource information for the message.

```
System total memory: {{ ansible_facts['memtotal_mb'] }} MiB.
System processor count: {{ ansible_facts['processor_count'] }}
```

4. Create a new playbook file called `motd.yml` in the current directory. Using the `template` module, configure the `motd.j2` Jinja2 template file previously created to map to the file `/etc/motd` on the managed hosts. This file has the `root` user as owner and group, and its permissions are 0644. Using the `stat` and `debug` modules, create tasks to verify that `/etc/motd` exists on the managed hosts and displays the file information for `/etc/motd`. Use the `copy` module to place `files/issue` into the `/etc/` directory on the managed host, use the same ownership and permissions as `/etc/motd`. Use the `file` module to ensure that `/etc/issue.net` is a symbolic link to `/etc/issue` on the managed host. Configure the playbook so that it uses the `devops` user, and sets the `become` parameter to `true`.

```
---
- name: Configure system
  hosts: all
  remote_user: devops
  become: true
  tasks:
    - name: Configure a custom /etc/motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644

    - name: Check file exists
      stat:
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        path: /etc/motd
      register: motd

    - name: Display stat results
      debug:
        var: motd

    - name: Copy custom /etc/issue file
      copy:
        src: files/issue
        dest: /etc/issue
        owner: root
        group: root
        mode: 0644

    - name: Ensure /etc/issue.net is a symlink to /etc/issue
      file:
        src: /etc/issue
        dest: /etc/issue.net
        state: link
        owner: root
        group: root
        force: yes
```

5. Run the playbook included in the `motd.yml` file.

5.1. Before you run the playbook, use the `ansible-playbook --syntax-check` command to verify its syntax. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation file-review]$ ansible-playbook --syntax-check motd.yml

playbook: motd.yml
```

5.2. Run the playbook included in the `motd.yml` file.

```
[student@workstation file-review]$ ansible-playbook motd.yml

PLAY [Configure system] ********************************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [Configure a custom /etc/motd] ********************************************
changed: [serverb.lab.example.com]

TASK [Check file exists] *******************************************************
ok: [serverb.lab.example.com]

TASK [Display stat results] ****************************************************
ok: [serverb.lab.example.com] => {
    "motd": {
        "changed": false,
        "failed": false,
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
...output omitted...

TASK [Copy custom /etc/issue file] ***************************************
changed: [serverb.lab.example.com]

TASK [Ensure /etc/issue.net is a symlink to /etc/issue] ******************
changed: [serverb.lab.example.com]

PLAY RECAP ***************************************************************
serverb.lab.example.com    : ok=6    changed=3    unreachable=0    failed=0
```

6. Check that the playbook included in the `motd.yml` file has been executed correctly.

   Log in to `serverb.lab.example.com` as the `devops` user, and verify that the `/etc/motd` and `/etc/issue` contents are displayed when logging in. Log off when you have finished.

```
[student@workstation file-review]$ ssh devops@serverb.lab.example.com
*------------------------------ PRIVATE SYSTEM -----------------------------*
*   Access to this computer system is restricted to authorised users only.   *
*                                                                            *
*      Customer information is confidential and must not be disclosed.       *
*----------------------------------------------------------------------------*
System total memory: 821 MiB.
System processor count: 1
Activate the web console with: systemctl enable --now cockpit.socket

This system is not registered to Red Hat Insights. See https://cloud.redhat.com/
To register this system, run: insights-client --register

Last login: Thu Apr 25 22:09:33 2019 from 172.25.250.9
[devops@serverb ~]$ logout
```

Evaluation

On `workstation`, run the `lab file-review grade` script to confirm success on this exercise.

```
[student@workstation ~]$ lab file-review grade
```

Finish

On `workstation`, run the `lab file-review finish` script to clean up after the lab.

```
[student@workstation ~]$ lab file-review finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- The `Files` modules library includes modules that allow you to accomplish most tasks related to file management, such as creating, copying, editing, and modifying permissions and other attributes of files.
- You can use Jinja2 templates to dynamically construct files for deployment.
- A Jinja2 template is usually composed of two elements: variables and expressions. Those variables and expressions are replaced with values when the Jinja2 template is rendered.
- Jinja2 filters transform template expressions from one kind or format of data into another.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 6

Managing Complex Plays and Playbooks

Goal

Write playbooks for larger, more complex plays and playbooks.

Objectives

- Write sophisticated host patterns to efficiently select hosts for a play or ad hoc command.
- Manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.

Sections

- Selecting Hosts with Host Patterns (and Guided Exercise)
- Including and Importing Files (and Guided Exercise)

Lab

- Managing Complex Plays and Playbooks

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Selecting Hosts with Host Patterns

Objectives

After completing this section, you will be able to write sophisticated host patterns to efficiently select hosts for a play or ad hoc command.

Referencing Inventory Hosts

Host patterns are used to specify the hosts to target by a play or ad hoc command. In its simplest form, the name of a managed host or a host group in the inventory is a host pattern that specifies that host or host group.

You have already used host patterns in this course. In a play, the `hosts` directive specifies the managed hosts to run the play against. For an ad hoc command, provide the host pattern as a command line argument to the `ansible` command.

It is usually easier to control what hosts a play targets by carefully using host patterns and having appropriate inventory groups, instead of setting complex conditionals on the play's tasks. Therefore, it is important to have a robust understanding of host patterns.

The following example inventory is used throughout this section to illustrate host patterns.

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

[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
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

To demonstrate how host patterns are resolved, you will execute an Ansible Playbook, `playbook.yml`, using different host patterns to target different subsets of managed hosts from this example inventory.

Managed Hosts

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

When the playbook runs, the first `Gathering Facts` task should run on all managed hosts that match the host pattern. A failure during this task can cause the managed host to be removed from the play.

If an IP address is listed explicitly in the inventory, instead of a host name, then it can be used as a host pattern. If the IP address is not listed in the inventory, then you cannot use it to specify the host even if the IP address resolves to that host name in the DNS.

The following example shows how a host pattern can be used to reference an IP address contained in an inventory.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 192.168.2.1
...output omitted...

[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [192.168.2.1]
...output omitted...
```

Note

One problem with referring to managed hosts by IP address in the inventory is that it can be hard to remember which IP address matches which host for your plays and ad hoc commands. However, you may have to specify the host by IP address for connection purposes if the host does not have a resolvable host name.

It is possible to point an alias at a particular IP address in your inventory by setting the `ansible_host` host variable. For example, you could have a host in your inventory named `dummy.example`, and then direct connections using that name to the IP address 192.168.2.1 by creating a `host_vars/dummy.example` file containing the following host variable:

```
ansible_host: 192.168.2.1
```

Specifying Hosts Using a Group

You have already used inventory host groups as host patterns. When a group name is used as a host pattern, it specifies that Ansible will act on the hosts that are members of the group.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [labhost2.example.com]
...output omitted...
```

Remember that there is a special group named `all` that matches all managed hosts in the inventory.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: all
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

There is also a special group named `ungrouped`, which includes all managed hosts in the inventory that are not members of any other group:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: ungrouped
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [web.example.com]
ok: [data.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Matching Multiple Hosts with Wildcards

Another method of accomplishing the same thing as the `all` host pattern is to use the asterisk (*) wildcard character, which matches any string. If the host pattern is just a quoted asterisk, then all hosts in the inventory will match.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: '*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

Important

Some characters that are used in host patterns also have meaning for the shell. This can be a problem when using host patterns to run ad hoc commands from the command line with `ansible`. It is a recommended practice to enclose host patterns used on the command line in single quotes to protect them from unwanted shell expansion.

Likewise, if you are using any special wildcards or list characters in an Ansible Playbook, then you must put your host pattern in single quotes to ensure it is parsed correctly.

```
---
  hosts: '!test1.example.com,development'
```

The asterisk character can also be used to match any managed hosts or groups that contain a particular substring.

For example, the following wildcard host pattern matches all inventory names that end in `.example.com`:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: '*.example.com'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Gathering Facts] ***************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
```

The following example uses a wildcard host pattern to match the names of hosts or host groups that start with `192.168.2.`:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: '192.168.2.*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ************************************************

TASK [Gathering Facts] ***************************************************
ok: [192.168.2.1]
ok: [192.168.2.2]
```

The next example uses a wildcard host pattern to match the names of hosts or host groups that begin with `datacenter`.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 'datacenter*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ************************************************

TASK [Gathering Facts] ***************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

The wildcard host patterns match all inventory names, hosts, and host groups. They do not distinguish between names that are DNS names, IP addresses, or groups, which can lead to some unexpected matches.

For example, compare the results of specifying the `datacenter*` host pattern from the preceding example with the results of the `data*` host pattern based on the example inventory:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 'data*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [data.example.com]
```

Lists

Multiple entries in an inventory can be referenced using logical lists. A comma-separated list of host patterns matches all hosts that match any of those host patterns.

If you provide a comma-separated list of managed hosts, then all those managed hosts will be targeted:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: labhost1.example.com,test2.example.com,192.168.2.2
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [test2.example.com]
ok: [192.168.2.2]
```

If you provide a comma-separated list of groups, then all hosts in any of those groups will be targeted:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,datacenter1
...output omitted...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [labhost2.example.com]
ok: [test1.example.com]
```

You can also mix managed hosts, host groups, and wildcards, as shown below:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,data*,192.168.2.2
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [labhost2.example.com]
ok: [test1.example.com]
ok: [test2.example.com]
ok: [data.example.com]
ok: [192.168.2.2]
```

Note

The colon character (:) can be used instead of a comma. However, the comma is the preferred separator, especially when working with IPv6 addresses as managed host names. You may see the colon syntax in older examples.

If an item in a list starts with an ampersand character (&), then hosts must match that item in order to match the host pattern. It operates similarly to a logical AND.

For example, based on our example inventory, the following host pattern matches machines in the `lab` group only if they are also in the `datacenter1` group:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,&datacenter1
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
```

You could also specify that machines in the `datacenter1` group match only if they are in the `lab` group with the host patterns `&lab,datacenter1` or `datacenter1,&lab`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

You can exclude hosts that match a pattern from a list by using the exclamation point or "bang" character (!) in front of the host pattern. This operates like a logical NOT.

This example matches all hosts defined in the `datacenter` group, except `test2.example.com` based on the example inventory:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: datacenter,!test2.example.com
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
```

The pattern `'!test2.example.com,datacenter'` could have been used in the preceding example to achieve the same result.

The final example shows the use of a host pattern that matches all hosts in the test inventory, except the managed hosts in the `datacenter1` group.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: all,!datacenter1
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [192.168.2.1]
ok: [192.168.2.2]
```

References

Working with Patterns — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/intro_patterns.html

Working with Inventory — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Selecting Hosts with Host Patterns

In this exercise, you will explore how to use host patterns to specify hosts from the inventory for plays or ad hoc commands. You will be provided with several example inventories to explore host patterns.

Outcomes

You will be able to use different host patterns to access various hosts in an inventory.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab projects-host start` command. The script creates the `projects-host` project directory, and then downloads the Ansible configuration file and the host inventory file needed for this exercise.

```
[student@workstation ~]$ lab projects-host start
```

Instructions

1. On `workstation`, change to the working directory for the exercise, `/home/student/projects-host`, and review the contents of the directory.

```
[student@workstation ~]$ cd ~/projects-host
[student@workstation projects-host]$
```

1.1. List the contents of the directory.

```
[student@workstation projects-host]$ ls
ansible.cfg inventory1 inventory2 playbook.yml
```

1.2. Inspect the example inventory file, `inventory1`. Notice how the inventory is organized. Explore which hosts and groups are in the inventory, and which domains are used.

```
srv1.example.com
srv2.example.com
s1.lab.example.com
s2.lab.example.com

[web]
jupiter.lab.example.com
saturn.example.com

[db]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
db1.example.com
db2.example.com
db3.example.com

[lb]
lb1.lab.example.com
lb2.lab.example.com

[boston]
db1.example.com
jupiter.lab.example.com
lb2.lab.example.com

[london]
db2.example.com
db3.example.com
file1.lab.example.com
lb1.lab.example.com

[dev]
web1.lab.example.com
db3.example.com

[stage]
file2.example.com
db2.example.com

[prod]
lb2.lab.example.com
db1.example.com
jupiter.lab.example.com

[function:children]
web
db
lb
city

[city:children]
boston
london
environments

[environments:children]
dev
stage
prod
new

[new]
172.25.252.23
172.25.252.44
172.25.252.32
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1.3. Inspect the example inventory file, `inventory2`. Notice how the inventory is organized. Explore which hosts and groups are in the inventory, and which domains are used.

```
workstation.lab.example.com

[london]
servera.lab.example.com

[berlin]
serverb.lab.example.com

[tokyo]
serverc.lab.example.com

[atlanta]
serverd.lab.example.com

[europe:children]
london
berlin
```

1.4. Lastly, inspect the contents of the playbook, `playbook.yml`. Notice how the playbook uses the `debug` module to display the name of each managed host.

```
---
- name: Resolve host patterns
  hosts:
  tasks:
    - name: Display managed host name
      debug:
        msg: "{{ inventory_hostname }}"
```

2. Using an ad hoc command, determine if the `db1.example.com` server is present in the `inventory1` inventory file.

```
[student@workstation projects-host]$ ansible db1.example.com -i inventory1 \
> --list-hosts
  hosts (1):
    db1.example.com
```

3. Using an ad hoc command, reference an IP address contained in the `inventory1` inventory with a host pattern.

```
[student@workstation projects-host]$ ansible 172.25.252.44 -i inventory1 \
> --list-hosts
 hosts (1):
    172.25.252.44
```

4. With an ad hoc command, use the `all` group to list all managed hosts in the `inventory1` inventory file.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation projects-host]$ ansible all -i inventory1 --list-hosts
  hosts (17):
    srv1.example.com
    srv2.example.com
    s1.lab.example.com
    s2.lab.example.com
    jupiter.lab.example.com
    saturn.example.com
    db1.example.com
    db2.example.com
    db3.example.com
    lb1.lab.example.com
    lb2.lab.example.com
    file1.lab.example.com
    web1.lab.example.com
    file2.example.com
    172.25.252.23
    172.25.252.44
    172.25.252.32
```

5. With an ad hoc command, use the asterisk (*) character to list all hosts that end in `.example.com` in the `inventory1` inventory file.

```
[student@workstation projects-host]$ ansible '*.example.com' -i inventory1 \
> --list-hosts
  hosts (14):
    jupiter.lab.example.com
    saturn.example.com
    db1.example.com
    db2.example.com
    db3.example.com
    lb1.lab.example.com
    lb2.lab.example.com
    file1.lab.example.com
    web1.lab.example.com
    file2.example.com
    srv1.example.com
    srv2.example.com
    s1.lab.example.com
    s2.lab.example.com
```

6. As you can see in the output of the preceeding command, there are 14 hosts in the `*.example.com` domain. Modify the host pattern in the previous ad hoc command so that hosts in the `*.lab.example.com` domain are ignored.

```
[student@workstation projects-host]$ ansible '*.example.com, !*.lab.example.com' \
> -i inventory1 --list-hosts
  hosts (7):
    saturn.example.com
    db1.example.com
    db2.example.com
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    db3.example.com
    file2.example.com
    srv1.example.com
    srv2.example.com
```

7. Without accessing the groups in the `inventory1` inventory file, use an ad hoc command to list these three hosts: `lb1.lab.example.com`, `s1.lab.example.com`, and `db1.example.com`.

```
[student@workstation projects-host]$ ansible \
> lb1.lab.example.com,s1.lab.example.com,db1.example.com -i inventory1 \
> --list-hosts
  hosts (3):
    lb1.lab.example.com
    s1.lab.example.com
    db1.example.com
```

8. Use a wildcard host pattern in an ad hoc command to list hosts that start with a `172.25.` IP address in the `inventory1` inventory file.

```
[student@workstation projects-host]$ ansible '172.25.*' -i inventory1 --list-hosts
  hosts (3):
    172.25.252.23
    172.25.252.44
    172.25.252.32
```

9. Use a host pattern in an ad hoc command to list all hosts in the `inventory1` inventory file that start with the letter "s."

```
[student@workstation projects-host]$ ansible 's*' -i inventory1 --list-hosts
  hosts (7):
    saturn.example.com
    srv1.example.com
    srv2.example.com
    s1.lab.example.com
    s2.lab.example.com
    file2.example.com
    db2.example.com
```

Notice the `file2.example.com` and `db2.example.com` hosts in the output of the preceding command. They appear in the list because they are both members of a group called `stage`, which also begins with the letter "s."

10. Using a list and wildcard host patterns in an ad hoc command, list all hosts in the `inventory1` inventory in the `prod` group, those hosts with an IP address beginning with `172`, and hosts that contain `lab` in their name.

```
[student@workstation projects-host]$ ansible 'prod,172*,*lab*' -i inventory1 \
> --list-hosts
  hosts (11):
    lb2.lab.example.com
    db1.example.com
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    jupiter.lab.example.com
    172.25.252.23
    172.25.252.44
    172.25.252.32
    lb1.lab.example.com
    file1.lab.example.com
    web1.lab.example.com
    s1.lab.example.com
    s2.lab.example.com
```

11. Use an ad hoc command to list all hosts that belong to both the `db` and `london` groups.

```
[student@workstation projects-host]$ ansible 'db,&london' -i inventory1 \
> --list-hosts
  hosts (2):
    db2.example.com
    db3.example.com
```

12. Modify the `hosts` value in the `playbook.yml` file so that all servers in the `london` group are targeted. Execute the playbook using the `inventory2` inventory file.

```
...output omitted...
  hosts: london
...output omitted...
```

```
[student@workstation projects-host]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] *********************************************
ok: [servera.lab.example.com]
...output omitted...
```

13. Modify the `hosts` value in the `playbook.yml` file so that all servers in the `europe` nested group are targeted. Execute the playbook using the `inventory2` inventory file.

```
...output omitted...
  hosts: europe
...output omitted...
```

```
[student@workstation projects-host]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] *********************************************
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]
...output omitted...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

14. Modify the `hosts` value in the `playbook.yml` file so that all servers that do not belong to any group are targeted. Execute the playbook using the `inventory2` inventory file.

```
...output omitted...
  hosts: ungrouped
...output omitted...
```

```
[student@workstation projects-hosts]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] *********************************************
ok: [workstation.lab.example.com]
...output omitted...
```

Finish

On `workstation`, run the `lab projects-host finish` script to clean up this exercise.

```
[student@workstation ~]$ lab projects-host finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Including and Importing Files

Objectives

After completing this section, you will be able to manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.

Managing Large Playbooks

When a playbook gets long or complex, you can divide it up into smaller files to make it easier to manage. You can combine multiple playbooks into a main playbook modularly, or insert lists of tasks from a file into a play. This can make it easier to reuse plays or sequences of tasks in different projects.

Including or Importing Files

There are two operations that Ansible can use to bring content into a playbook. You can *include* content, or you can *import* content.

When you include content, it is a *dynamic* operation. Ansible processes included content during the run of the playbook, as content is reached.

When you import content, it is a *static* operation. Ansible preprocesses imported content when the playbook is initially parsed, before the run starts.

Importing Playbooks

The `import_playbook` directive allows you to import external files containing lists of plays into a playbook. In other words, you can have a master playbook that imports one or more additional playbooks.

Because the content being imported is a complete playbook, the `import_playbook` feature can only be used at the top level of a playbook and cannot be used inside a play. If you import multiple playbooks, then they will be imported and run in order.

A simple example of a master playbook that imports two additional playbooks is shown below:

```
- name: Prepare the web server
  import_playbook: web.yml

- name: Prepare the database server
  import_playbook: db.yml
```

You can also interleave plays in your master playbook with imported playbooks.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Play 1
  hosts: localhost
  tasks:
    - debug:
        msg: Play 1

- name: Import Playbook
  import_playbook: play2.yml
```

In the preceding example, the `Play 1` runs first, followed by the plays imported from the `play2.yml` playbook.

Importing and Including Tasks

You can import or include a list of tasks from a task file into a play. A task file is a file that contains a flat list of tasks:

```
[admin@node ~]$ cat webserver_tasks.yml
- name: Installs the httpd package
  yum:
    name: httpd
    state: latest

- name: Starts the httpd service
  service:
    name: httpd
    state: started
```

Importing Task Files

You can statically import a task file into a play inside a playbook by using the `import_tasks` feature. When you import a task file, the tasks in that file are directly inserted when the playbook is parsed. The location of `import_tasks` in the playbook controls where the tasks are inserted and the order in which multiple imports are run.

```
---
- name: Install web server
  hosts: webservers
  tasks:
  - import_tasks: webserver_tasks.yml
```

When you import a task file, the tasks in that file are directly inserted when the playbook is parsed. Because `import_tasks` statically imports the tasks when the playbook is parsed, there are some effects on how it works.

- When using the `import_tasks` feature, conditional statements set on the import, such as `when`, are applied to each of the tasks that are imported.
- You cannot use loops with the `import_tasks` feature.
- If you use a variable to specify the name of the file to import, then you cannot use a host or group inventory variable.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Including Task Files

You can also dynamically include a task file into a play inside a playbook by using the `include_tasks` feature.

```
---
- name: Install web server
  hosts: webservers
  tasks:
  - include_tasks: webserver_tasks.yml
```

The `include_tasks` feature does not process content in the playbook until the play is running and that part of the play is reached. The order in which playbook content is processed impacts how the include tasks feature works.

- When using the `include_tasks` feature, conditional statements such as `when` set on the include determine whether or not the tasks are included in the play at all.
- If you run `ansible-playbook --list-tasks` to list the tasks in the playbook, then tasks in the included task files are not displayed. The tasks that include the task files are displayed. By comparison, the `import_tasks` feature would not list tasks that import task files, but instead would list the individual tasks from the imported task files.
- You cannot use `ansible-playbook --start-at-task` to start playbook execution from a task that is in an included task file.
- You cannot use a `notify` statement to trigger a handler name that is in an included task file. You can trigger a handler in the main playbook that includes an entire task file, in which case all tasks in the included file will run.

Note

You can find a more detailed discussion of the differences in behavior between `import_tasks` and `include_tasks` when conditionals are used at "Conditionals" [https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html#applying-when-to-roles-imports-and-includes] in the *Ansible User Guide*.

Use Cases for Task Files

Consider the following examples where it might be useful to manage sets of tasks as external files separate from the playbook:

- If new servers require complete configuration, then 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, then every organization can write its own task file which can then be reviewed and integrated by the system manager.
- If a server requires a particular configuration, then it can be integrated as a set of tasks that are executed based on a conditional. In other words, including the tasks only if specific criteria are met.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- If a group of servers need to run a particular task or set of tasks, then the tasks might only be run on a server if it is part of a specific host group.

Managing Task Files

You can create a dedicated directory for task files, and save all task files in that directory. Then your playbook can simply include or import task files from that directory. This allows construction of a complex playbook while making it easy to manage its structure and components.

Defining Variables for External Plays and Tasks

The incorporation of plays or tasks from external files into playbooks using Ansible's import and include features greatly enhance the ability to reuse tasks and playbooks across an Ansible environment. To maximize the possibility of reuse, these task and play files should be as generic as possible. Variables can be used to parameterize play and task elements to expand the application of tasks and plays.

For example, the following task file installs the package needed for a web service, and then enables and starts the necessary service.

```
---
  - name: Install the httpd package
    yum:
      name: httpd
      state: latest
  - name: Start the httpd service
    service:
      name: httpd
      enabled: true
      state: started
```

If you parameterize the package and service elements as shown in the following example, then the task file can also be used for the installation and administration of other software and their services, rather than being useful for web service only.

```
---
  - name: Install the {{ package }} package
    yum:
      name: "{{ package }}"
      state: latest
  - name: Start the {{ service }} service
    service:
      name: "{{ service }}"
      enabled: true
      state: started
```

Subsequently, when incorporating the task file into a playbook, define the variables to use for the task execution as follows:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
...output omitted...
  tasks:
    - name: Import task file and set variables
      import_tasks: task.yml
      vars:
        package: httpd
        service: httpd
```

Ansible makes the passed variables available to the tasks imported from the external file.

You can use the same technique to make play files more reusable. When incorporating a play file into a playbook, pass the variables to use for the play execution as follows:

```
...output omitted...
- name: Import play file and set the variable
  import_playbook: play.yml
  vars:
    package: mariadb
```

Important

Earlier versions of Ansible used an `include` feature to include both playbooks and task files, depending on context. This functionality is being deprecated for a number of reasons.

Prior to Ansible 2.0, `include` operated like a static import. In Ansible 2.0 it was changed to operate dynamically, but this created some limitations. In Ansible 2.1 it became possible for `include` to be dynamic or static depending on task settings, which was confusing and error-prone. There were also issues with ensuring that `include` worked correctly in all contexts.

Thus, `include` was replaced in Ansible 2.4 with new directives such as `include_tasks`, `import_tasks`, and `import_playbook`. You might find examples of `include` in older playbooks, but you should avoid using it in new ones.

References

Including and Importing – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_includes.html

Creating Reusable Playbooks – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse.html

Conditionals – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Including and Importing Files

In this exercise, you will include and import playbooks and tasks in a top-level Ansible Playbook.

Outcomes

You will be able to include task and playbook files in playbooks.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab projects-file start` command. The script creates the working directory, `/home/student/projects-file`, and associated project files.

```
[student@workstation ~]$ lab projects-file start
```

Instructions

1. On `workstation`, as the `student` user, change to the `/home/student/projects-file` directory.

```
[student@workstation ~]$ cd ~/projects-file
[student@workstation projects-file]$
```

2. Review the contents of the three files in the `tasks` subdirectory.

   2.1. Review the contents of the `tasks/environment.yml` file. The file contains tasks for package installation and service administration.

```
---
  - name: Install the {{ package }} package
    yum:
      name: "{{ package }}"
      state: latest
  - name: Start the {{ service }} service
    service:
      name: "{{ service }}"
      enabled: true
      state: started
```

   2.2. Review the contents of the `tasks/firewall.yml` file. The file contains tasks for installation, administration, and configuration of firewall software.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
  - name: Install the firewall
    yum:
      name: "{{ firewall_pkg }}"
      state: latest

  - name: Start the firewall
    service:
      name: "{{ firewall_svc }}"
      enabled: true
      state: started

  - name: Open the port for {{ rule }}
    firewalld:
      service: "{{ item }}"
      immediate: true
      permanent: true
      state: enabled
    loop: "{{ rule }}"
```

2.3. Review the contents of the `tasks/placeholder.yml` file. This file contains a task for populating a placeholder web content file.

```
---
  - name: Create placeholder file
    copy:
      content: "{{ ansible_facts['fqdn'] }} has been customized using Ansible.\n"
      dest: "{{ file }}"
```

3. Review the contents of the `test.yml` file in the `plays` subdirectory. This file contains a play which tests connections to a web service.

```
---
- name: Test web service
  hosts: localhost
  become: no
  tasks:
    - name: connect to internet web server
      uri:
        url: "{{ url }}"
        status_code: 200
```

4. Create a playbook named `playbook.yml`. Define the first play with the name `Configure web server`. The play should execute against the `servera.lab.example.com` managed host defined in the `inventory` file. The beginning of the file should look like the following:

```
---
- name: Configure web server
  hosts: servera.lab.example.com
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. In the `playbook.yml` playbook, define the tasks section with three sets of tasks. Include the first set of tasks from the `tasks/environment.yml` tasks file. Define the necessary variables to install the *httpd* package and to enable and start the `httpd` service. Import the second set of tasks from the `tasks/firewall.yml` tasks file. Define the necessary variables to install the *firewalld* package to enable and start the `firewalld` service, and to allow `http` connections. Import the third task set from the `tasks/placeholder.yml` task file.

5.1. Create the tasks section in the first play by adding the following entry to the `playbook.yml` playbook.

```
tasks:
```

5.2. Include the first set of tasks from `tasks/environment.yml` using the `include_tasks` feature. Set the `package` and `service` variables to `httpd`.

```
  - name: Include the environment task file and set the variables
    include_tasks: tasks/environment.yml
    vars:
      package: httpd
      service: httpd
```

5.3. Import the second set of tasks from `tasks/firewall.yml` using the `import_tasks` feature. Set the `firewall_pkg` and `firewall_svc` variables to `firewalld`. Set the `rule` variable to `http`.

```
  - name: Import the firewall task file and set the variables
    import_tasks: tasks/firewall.yml
    vars:
      firewall_pkg: firewalld
      firewall_svc: firewalld
      rule:
        - http
        - https
```

5.4. Import the last task set from `tasks/placeholder.yml` using the `import_tasks` feature. Set the `file` variable to `/var/www/html/index.html`.

```
  - name: Import the placeholder task file and set the variable
    import_tasks: tasks/placeholder.yml
    vars:
      file: /var/www/html/index.html
```

6. Add a second and final play to the `playbook.yml` playbook using the contents of the `plays/test.yml` playbook.

6.1. Add a second play to the `playbook.yml` playbook to validate the web server installation. Import the play from `plays/test.yml`. Set the `url` variable to `http://servera.lab.example.com`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

6.2. Your playbook should look like the following after the changes are complete:

```
---
- name: Configure web server
  hosts: servera.lab.example.com

  tasks:
    - name: Include the environment task file and set the variables
      include_tasks: tasks/environment.yml
      vars:
        package: httpd
        service: httpd

    - name: Import the firewall task file and set the variables
      import_tasks: tasks/firewall.yml
      vars:
        firewall_pkg: firewalld
        firewall_svc: firewalld
        rule:
          - http
          - https

    - name: Import the placeholder task file and set the variable
      import_tasks: tasks/placeholder.yml
      vars:
        file: /var/www/html/index.html

- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

6.3. Save the changes to the `playbook.yml` playbook.

7. Before running the playbook, verify its syntax is correct by running `ansible-playbook --syntax-check`. If errors are reported, correct them before moving to the next step.

```
[student@workstation projects-file]$ ansible-playbook playbook.yml --syntax-check

playbook: playbook.yml
```

8. Execute the `playbook.yml` playbook. The output of the playbook shows the import of the task and play files.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation projects-file]$ ansible-playbook playbook.yml

PLAY [Configure web server] ***************************************************

TASK [Gathering Facts] ********************************************************
ok: [servera.lab.example.com]

TASK [Install the httpd package] **********************************************
changed: [servera.lab.example.com]

TASK [Start the httpd service] ************************************************
changed: [servera.lab.example.com]

TASK [Install the firewall] ***************************************************
ok: [servera.lab.example.com]

TASK [Start the firewall] *****************************************************
ok: [servera.lab.example.com]

TASK [Open the port for ['http', 'https']] ************************************
changed: [servera.lab.example.com] => (item=http)
changed: [servera.lab.example.com] => (item=https)

TASK [Create placeholder file] ************************************************
changed: [servera.lab.example.com]

PLAY [Test web service] *******************************************************

TASK [Gathering Facts] ********************************************************
ok: [localhost]

TASK [connect to internet web server] *****************************************
ok: [localhost]

PLAY RECAP ********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
servera.lab.example.com    : ok=8    changed=4    unreachable=0    failed=0
```

Finish

On `workstation`, run the `lab projects-file finish` script to clean up this exercise.

```
[student@workstation ~]$ lab projects-file finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Managing Complex Plays and Playbooks

Performance Checklist

In this lab, you will modify a complex playbook to be easier to manage by using host patterns, includes, and imports.

Outcomes

You should be able to:

- Simplify host references in a playbook by specifying host patterns.
- Restructure a playbook so that tasks are imported from external task files.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab projects-review start` command. This setup script ensures that the managed hosts are reachable on the network. It also ensures that the correct Ansible configuration file, inventory file, and playbook are installed on the control node in the `/home/student/projects-review` directory.

```
[student@workstation ~]$ lab projects-review start
```

Instructions

You have inherited a playbook from the previous administrator. The playbook is used to configure a web service on `servera.lab.example.com`, `serverb.lab.example.com`, `serverc.lab.example.com`, and `serverd.lab.example.com`. The playbook also configures the firewall on the four managed hosts so that web traffic is allowed.

Make the following changes to the `playbook.yml` playbook file so that it is easier to manage.

1. Simplify the list of managed hosts in the `/home/student/projects-review/playbook.yml` playbook by using a wildcard host pattern.
2. Restructure the playbook so that the first three tasks in the playbook are kept in an external task file located at `tasks/web_tasks.yml`. Use the `import_tasks` feature to incorporate this task file into the playbook.
3. Restructure the playbook so that the fourth, fifth, and sixth tasks in the playbook are kept in an external task file located at `tasks/firewall_tasks.yml`. Use the `import_tasks` feature to incorporate this task file into the playbook.
4. There is some duplication of tasks between the `tasks/web_tasks.yml` and `tasks/firewall_tasks.yml` files. Move the tasks that install packages and enable services into a new file named `tasks/install_and_enable.yml` and update them to use variables. Replace the original tasks with `import_tasks` statements, passing in appropriate variable values.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. Verify the changes to the `playbook.yml` playbook were correctly made and then execute the playbook.

Evaluation

Run the `lab projects-review grade` command from `workstation` to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab projects-review grade
```

Finish

On `workstation`, run the `lab projects-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab projects-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Managing Complex Plays and Playbooks

Performance Checklist

In this lab, you will modify a complex playbook to be easier to manage by using host patterns, includes, and imports.

Outcomes

You should be able to:

- Simplify host references in a playbook by specifying host patterns.
- Restructure a playbook so that tasks are imported from external task files.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab projects-review start` command. This setup script ensures that the managed hosts are reachable on the network. It also ensures that the correct Ansible configuration file, inventory file, and playbook are installed on the control node in the `/home/student/projects-review` directory.

```
[student@workstation ~]$ lab projects-review start
```

Instructions

You have inherited a playbook from the previous administrator. The playbook is used to configure a web service on `servera.lab.example.com`, `serverb.lab.example.com`, `serverc.lab.example.com`, and `serverd.lab.example.com`. The playbook also configures the firewall on the four managed hosts so that web traffic is allowed.

Make the following changes to the `playbook.yml` playbook file so that it is easier to manage.

1. Simplify the list of managed hosts in the `/home/student/projects-review/playbook.yml` playbook by using a wildcard host pattern.

    1.1. Change directory to the `/home/student/projects-review` working directory. Review the `hosts` parameter in the `playbook.yml` file.

```
[student@workstation ~]$ cd ~/projects-review
[student@workstation projects-review]$ cat playbook.yml
---
- name: Install and configure web service
  hosts:
    - servera.lab.example.com
    - serverb.lab.example.com
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - serverc.lab.example.com
    - serverd.lab.example.com
...output omitted...
```

1.2. Verify that the host pattern `server*.lab.example.com` correctly identifies the four managed hosts that are targeted by the `playbook.yml` playbook.

```
[student@workstation projects-review]$ ansible server*.lab.example.com \
> --list-hosts
  hosts (4):
    servera.lab.example.com
    serverb.lab.example.com
    serverc.lab.example.com
    serverd.lab.example.com
```

1.3. Replace the host list in the `playbook.yml` playbook with the `server*.lab.example.com` host pattern.

```
---
- name: Install and configure web service
  hosts: server*.lab.example.com
...output omitted...
```

2. Restructure the playbook so that the first three tasks in the playbook are kept in an external task file located at `tasks/web_tasks.yml`. Use the `import_tasks` feature to incorporate this task file into the playbook.

2.1. Create the `tasks` subdirectory.

```
[student@workstation projects-review]$ mkdir tasks
```

2.2. Place the contents of the first three tasks in the `playbook.yml` playbook into the `tasks/web_tasks.yml` file. The task file should contain the following content:

```
---
- name: Install httpd
  yum:
    name: httpd
    state: latest

- name: Enable and start httpd
  service:
    name: httpd
    enabled: true
    state: started

- name: Tuning configuration installed
  copy:
    src: files/tune.conf
    dest: /etc/httpd/conf.d/tune.conf
    owner: root
    group: root
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    mode: 0644
  notify:
    - restart httpd
```

2.3. Remove the first three tasks from the `playbook.yml` playbook and put the following lines in their place to import the `tasks/web_tasks.yml` task file.

```
    - name: Import the web_tasks.yml task file
      import_tasks: tasks/web_tasks.yml
```

3. Restructure the playbook so that the fourth, fifth, and sixth tasks in the playbook are kept in an external task file located at `tasks/firewall_tasks.yml`. Use the `import_tasks` feature to incorporate this task file into the playbook.

3.1. Place the contents of the three remaining tasks in the `playbook.yml` playbook into the `tasks/firewall_tasks.yml` file. The task file should contain the following content.

```
---
- name: Install firewalld
  yum:
    name: firewalld
    state: latest

- name: Enable and start the firewall
  service:
    name: firewalld
    enabled: true
    state: started

- name: Open the port for http
  firewalld:
    service: http
    immediate: true
    permanent: true
    state: enabled
```

3.2. Remove the remaining three tasks from the `playbook.yml` playbook and put the following lines in their place to import the `tasks/firewall_tasks.yml` task file.

```
    - name: Import the firewall_tasks.yml task file
      import_tasks: tasks/firewall_tasks.yml
```

4. There is some duplication of tasks between the `tasks/web_tasks.yml` and `tasks/firewall_tasks.yml` files. Move the tasks that install packages and enable services into a new file named `tasks/install_and_enable.yml` and update them to use variables. Replace the original tasks with `import_tasks` statements, passing in appropriate variable values.

4.1. Copy the yum and service tasks from `tasks/web_tasks.yml` into a new file named `tasks/install_and_enable.yml`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Install httpd
  yum:
    name: httpd
    state: latest

- name: Enable and start httpd
  service:
    name: httpd
    enabled: true
    state: started
```

4.2. Replace the package and service names in `tasks/install_and_enable.yml` with the variables `package` and `service`.

```
---
- name: Install {{ package }}
  yum:
    name: "{{ package }}"
    state: latest

- name: Enable and start {{ service }}
  service:
    name: "{{ service }}"
    enabled: true
    state: started
```

4.3. Replace the yum and service tasks in `tasks/web_tasks.yml` and `tasks/firewall_tasks.yml` with `import_tasks` statements.

```
---
- name: Install and start httpd
  import_tasks: install_and_enable.yml
  vars:
    package: httpd
    service: httpd
```

```
---
- name: Install and start firewalld
  import_tasks: install_and_enable.yml
  vars:
    package: firewalld
    service: firewalld
```

5. Verify the changes to the `playbook.yml` playbook were correctly made and then execute the playbook.

5.1. Verify that the `playbook.yml` playbook contains the following contents.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Install and configure web service
  hosts: server*.lab.example.com

  tasks:
    - name: Import the web_tasks.yml task file
      import_tasks: tasks/web_tasks.yml

    - name: Import the firewall_tasks.yml task file
      import_tasks: tasks/firewall_tasks.yml

  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted
```

5.2. Execute the playbook with `ansible-playbook --syntax-check` to verify the playbook contains no syntax errors. If errors are present, correct them before preceding.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml \
> --syntax-check

playbook: playbook.yml
```

5.3. Execute the playbook.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml

PLAY [Install and configure web service] ***********************************

TASK [Gathering Facts] *****************************************************
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Install httpd] *******************************************************
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [Enable and start httpd] **********************************************
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [Tuning configuration installed] **************************************
changed: [serverd.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
changed: [serverc.lab.example.com]
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

TASK [Install firewalld] ***************************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [Enable and start firewalld] ******************************************
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [Open the port for http] **********************************************
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
changed: [serverc.lab.example.com]

RUNNING HANDLER [restart httpd] ********************************************
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverc.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP *****************************************************************
servera.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverb.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverc.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverd.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
```

Evaluation

Run the `lab projects-review grade` command from `workstation` to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab projects-review grade
```

Finish

On `workstation`, run the `lab projects-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab projects-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- Host patterns are used to specify the managed hosts to be targeted by plays or ad hoc commands.
- Dynamic inventory scripts can be used to generate dynamic lists of managed hosts from directory services or other sources external to Ansible.
- The `forks` parameter in the Ansible configuration file sets the maximum number of parallel connections to managed hosts.
- The `serial` parameter can be used to implement rolling updates across managed hosts by defining the number of managed hosts in each rolling update batch.
- You can use the `import_playbook` feature to incorporate external play files into playbooks.
- You can use the `include_tasks` or `import_tasks` features to incorporate external task files into playbooks.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 7

Simplifying Playbooks with Roles

Goal

Use Ansible roles to develop playbooks more quickly and to reuse Ansible code.

Objectives

- Describe what a role is, how it is structured, and how you can use it in a playbook.
- Write playbooks that take advantage of Red Hat Enterprise Linux System Roles to perform standard operations.
- Create a role in a playbook's project directory and run it as part of one of the plays in the playbook.
- Select and retrieve roles from Ansible Galaxy or other sources such as a Git repository, and use them in your playbooks.
- Obtain a set of related roles, supplementary modules, and other content from content collections, and use them in a playbook.

Sections

- Describing Role Structure (and Quiz)
- Reusing Content with System Roles (and Guided Exercise)
- Creating Roles (and Guided Exercise)
- Deploying Roles with Ansible Galaxy (and Guided Exercise)
- Getting Roles and Modules from Content Collections (and Guided Exercise)

Lab

- Simplifying Playbooks with Roles

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Describing Role Structure

Objectives

After completing this section, you should be able to describe what a role is, how it is structured, and how you can use it in a playbook.

Structuring Ansible Playbooks with Roles

As you develop more playbooks, you will probably discover that you have many opportunities to reuse code from playbooks that you have already written. Perhaps a play to configure a MySQL database for one application could be re-purposed, with different hostnames, passwords, and users, to configure a MySQL database for another application.

But in the real world, that play might be long and complex, with many included or imported files, and with tasks and handlers to manage various situations. Copying all that code into another playbook might be nontrivial work.

Ansible *roles* provide a way for you to make it easier to reuse Ansible code generically. You can package, in a standardized directory structure, all the tasks, variables, files, templates, and other resources needed to provision infrastructure or deploy applications. Copy that role from project to project simply by copying the directory. You can then simply call that role from a play to execute it.

A well-written role will allow you to pass variables to the role from the playbook that adjust its behavior, setting all the site-specific hostnames, IP addresses, user names, secrets, or other locally-specific details you need. For example, a role to deploy a database server might have been written to support variables which set the hostname, database admin user and password, and other parameters that need customization for your installation. The author of the role can also ensure that reasonable default values are set for those variables if you choose not to set them in the play.

Ansible roles have 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

In addition to writing, using, reusing, and sharing your own roles, you can get roles from other sources. Some roles are included as part of Red Hat Enterprise Linux, in the *rhel-system-roles* package. You can also get numerous community-supported roles from the Ansible Galaxy website. Later in this chapter, you will learn more about these roles.

Examining the Ansible Role Structure

An Ansible role is defined by a standardized structure of subdirectories and files. The top-level directory defines the name of the role itself. Files are organized into subdirectories that are named according to each file's purpose in the role, such as `tasks` and `handlers`. The `files` and `templates` subdirectories contain files referenced by tasks in other YAML files.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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
├── 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. These variables have low precedence and are intended to be changed and customized in plays.
`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. Often these variables are used for internal purposes within the role. These variables have high precedence, and are not intended to be changed when used in a playbook.

Not every role will have all of these directories.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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 precedence and can not be overridden by inventory variables. The intent of these variables is that they are used by the internal functioning of the role.

Default variables allow default values to be set for variables that can be used in a play to configure the role or customize its behavior. They are defined by creating a `defaults/main.yml` file with key: value pairs in the role directory hierarchy. Default variables have the lowest precedence of any variables available. They can be easily overridden by any other variable, including inventory variables. These variables are intended to provide the person writing a play that uses the role with a way to customize or control exactly what it is going to do. They can be used to provide information to the role that it needs to configure or deploy something properly.

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.

Important

Roles should not have site-specific data in them. They definitely should not contain any secrets like passwords or private keys.

This is because roles are supposed to be generic, reusable, and freely shareable. Site-specific details should not be hard coded into them.

Secrets should be provided to the role through other means. This is one reason you might want to set role variables when calling a role. Role variables set in the play could provide the secret, or point to an Ansible Vault-encrypted file containing the secret.

Using Ansible Roles in a Playbook

Using roles in a playbook is straightforward. The following example shows one way to call 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 imported into the playbook, in that order. Any `copy`, `script`, `template`, or `include_tasks/import_tasks` tasks in the role can reference the relevant files, templates, or task files in the role without absolute or relative path names. Ansible looks for them in the role's `files`, `templates`, or `tasks` subdirectories respectively.

When you use a `roles` section to import roles into a play, the roles will run first, before any tasks that you define for that play.

The following example sets values for two role variables of `role2`, `var1` and `var2`. Any `defaults` and `vars` variables are overridden when `role2` is used.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- hosts: remote.example.com
  roles:
    - role: role1
    - role: role2
      var1: val1
      var2: val2
```

Another equivalent YAML syntax which you might see in this case is:

```
---
- hosts: remote.example.com
  roles:
    - role: role1
    - { role: role2, var1: val1, var2: val2 }
```

There are situations in which this can be harder to read, even though it is more compact.

Important

Role variables set inline (role parameters), as in the preceding examples, have very high precedence. They will override most other variables.

Be very careful not to reuse the names of any role variables that you set inline anywhere else in your play, since the values of the role variables will override inventory variables and any play `vars`.

Controlling Order of Execution

For each play in a playbook, tasks execute as ordered in the tasks list. After all tasks execute, any notified handlers are executed.

When a role is added to a play, role tasks are added to the beginning of the tasks list. If a second role is included in a play, its tasks list is added after the first role.

Role handlers are added to plays in the same manner that role tasks are added to plays. Each play defines a handlers list. Role handlers are added to the handlers list first, followed by any handlers defined in the `handlers` section of the play.

In certain scenarios, it may be necessary to execute some play tasks before the roles. To support such scenarios, plays can be configured with a `pre_tasks` section. Any task listed in this section executes before any roles are executed. If any of these tasks notify a handler, those handler tasks execute before the roles or normal tasks.

Plays also support a `post_tasks` keyword. These tasks execute after the play's normal tasks, and any handlers they notify, are run.

The following play shows an example with `pre_tasks`, `roles`, `tasks`, `post_tasks` and `handlers`. It is unusual that a play would contain all of these sections.

```
- name: Play to illustrate order of execution
  hosts: remote.example.com
  pre_tasks:
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - debug:
        msg: 'pre-task'
      notify: my handler
  roles:
    - role1
  tasks:
    - debug:
        msg: 'first task'
      notify: my handler
  post_tasks:
    - debug:
        msg: 'post-task'
      notify: my handler
  handlers:
    - name: my handler
      debug:
        msg: Running my handler
```

In the above example, a `debug` task executes in each section to notify the `my handler` handler. The `my handler` task is executed three times:

- after all the `pre_tasks` tasks execute
- after all role tasks and tasks from the `tasks` section execute
- after all the `post_tasks` execute

Roles can be added to a play using an ordinary task, not just by including them in the `roles` section of a play. Use the `include_role` module to dynamically include a role, and use the `import_role` module to statically import a role.

The following playbook demonstrates how a role can be included using a task with the `include_role` module.

```
- name: Execute a role as a task
  hosts: remote.example.com
  tasks:
    - name: A normal task
      debug:
        msg: 'first task'
    - name: A task to include role2 here
      include_role: role2
```

Note

The `include_role` module was added in Ansible 2.3, and the `import_role` module in Ansible 2.4.

References

Roles – Ansible Documentation

https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Quiz

Describing Role Structure

Choose the correct answer to the following questions:

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

1. Which of the following statements best describes roles?

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 La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Describing Role Structure

Choose the correct answer to the following questions:

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

1. Which of the following statements best describes roles?

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 La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Reusing Content with System Roles

Objectives

After completing this section, you should be able to write playbooks that take advantage of Red Hat Enterprise Linux System Roles to perform standard operations.

Red Hat Enterprise Linux System Roles

Beginning with Red Hat Enterprise Linux 7.4, a number of Ansible roles have been provided with the operating system as part of the *rhel-system-roles* package. In Red Hat Enterprise Linux 8 the package is available in the AppStream channel. A brief description of each role:

RHEL System Roles

Name	State	Role Description
`rhel-system-roles.kdump`	Fully Supported	Configures the `kdump` crash recovery service.
`rhel-system-roles.network`	Fully Supported	Configures network interfaces.
`rhel-system-roles.selinux`	Fully Supported	Configures and manages SELinux customization, including SELinux mode, file and port contexts, Boolean settings, and SELinux users.
`rhel-system-roles.timesync`	Fully Supported	Configures time synchronization using Network Time Protocol or Precision Time Protocol.
`rhel-system-roles.postfix`	Technology Preview	Configures each host as a Mail Transfer Agent using the Postfix service.
`rhel-system-roles.firewall`	In Development	Configures a host's firewall.
`rhel-system-roles.tuned`	In Development	Configures the `tuned` service to tune system performance.

System roles aim to standardize the configuration of Red Hat Enterprise Linux subsystems across multiple versions. Use system roles to configure any Red Hat Enterprise Linux, version 6.10 and onward.

Simplified Configuration Management

As an example, the recommended time synchronization service for Red Hat Enterprise Linux 7 is the `chronyd` service. In Red Hat Enterprise Linux 6 however, the recommended service is the `ntpd` service. In an environment with a mixture of Red Hat Enterprise Linux 6 and 7 hosts, an administrator must manage the configuration files for both services.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

With RHEL System Roles, administrators no longer need to maintain configuration files for both services. Administrators can use `rhel-system-roles.timesync` role to configure time synchronization for both Red Hat Enterprise Linux 6 and 7 hosts. A simplified YAML file containing role variables defines the configuration of time synchronization for both types of hosts.

Support for RHEL System Roles

RHEL System Roles are derived from the open source Linux System Roles project, found on Ansible Galaxy. Unlike Linux System Roles, RHEL System Roles are supported by Red Hat as part of a standard Red Hat Enterprise Linux subscription. RHEL System Roles have the same life cycle support benefits that come with a Red Hat Enterprise Linux subscription.

Every system role is tested and stable. The `Fully Supported` system roles also have stable interfaces. For any `Fully Supported` system role, Red Hat will endeavour to ensure that role variables are unchanged in future versions. Playbook refactoring due to system role changes should be minimal.

The `Technology Preview` system roles may utilize different role variables in future versions. Integration testing is recommended for playbooks that incorporate any `Technology Preview` role. Playbooks may require refactoring if role variables change in a future version of the role.

Other roles are in development in the upstream Linux System Roles project, but are not yet available through a RHEL subscription. These roles are available through Ansible Galaxy.

Installing RHEL System Roles

The RHEL System Roles are provided by the *rhel-system-roles* package, which is available in the `AppStream` channel. Install this package on the Ansible control node.

Use the following procedure to install the *rhel-system-roles* package. The procedure assumes the control node is registered to a Red Hat Enterprise Linux subscription and that Ansible is installed. See the section on *Installing Ansible* for more information.

1. Install RHEL System Roles.

```
[root@host ~]# yum install rhel-system-roles
```

After installation, the RHEL System roles are located in the `/usr/share/ansible/roles` directory:

```
[root@host ~]# ls -l /usr/share/ansible/roles/
total 20
...output omitted... linux-system-roles.kdump -> rhel-system-roles.kdump
...output omitted... linux-system-roles.network -> rhel-system-roles.network
...output omitted... linux-system-roles.postfix -> rhel-system-roles.postfix
...output omitted... linux-system-roles.selinux -> rhel-system-roles.selinux
...output omitted... linux-system-roles.timesync -> rhel-system-roles.timesync
...output omitted... rhel-system-roles.kdump
...output omitted... rhel-system-roles.network
...output omitted... rhel-system-roles.postfix
...output omitted... rhel-system-roles.selinux
...output omitted... rhel-system-roles.timesync
```

The corresponding upstream name of each role is linked to the RHEL System Role. This allows a role to be referenced in a playbook by either name.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The default `roles_path` on Red Hat Enterprise Linux includes `/usr/share/ansible/roles` in the path, so Ansible should automatically find those roles when referenced by a playbook.

Note

Ansible might not find the system roles if `roles_path` has been overridden in the current Ansible configuration file, if the environment variable `ANSIBLE_ROLES_PATH` is set, or if there is another role of the same name in a directory listed earlier in `roles_path`.

Accessing Documentation for RHEL System Roles

After installation, documentation for the RHEL System Roles is found in the `/usr/share/doc/rhel-system-roles-<version>/` directory. Documentation is organized into subdirectories by subsystem:

```
[root@host ~]# ls -l /usr/share/doc/rhel-system-roles/
total 4
drwxr-xr-x. ...output omitted... kdump
drwxr-xr-x. ...output omitted... network
drwxr-xr-x. ...output omitted... postfix
drwxr-xr-x. ...output omitted... selinux
drwxr-xr-x. ...output omitted... timesync
```

Each role's documentation directory contains a `README.md` file. The `README.md` file contains a description of the role, along with role usage information.

The `README.md` file also describes role variables that affect the behavior of the role. Often the `README.md` file contains a playbook snippet that demonstrates variable settings for a common configuration scenario.

Some role documentation directories contain example playbooks. When using a role for the first time, review any additional example playbooks in the documentation directory.

Role documentation for RHEL System Roles matches the documentation for Linux System Roles. Use a web browser to access role documentation for the upstream roles at the Ansible Galaxy site, `https://galaxy.ansible.com`.

Time Synchronization Role Example

Suppose you need to configure NTP time synchronization on your servers. You could write automation yourself to perform each of the necessary tasks. But RHEL System Roles includes a role that can do this, `rhel-system-roles.timesync`.

The role is documented in its `README.md` in the `/usr/share/doc/rhel-system-roles/timesync` directory. The file describes all the variables that affect the role's behavior and contains three playbook snippets illustrating different time synchronization configurations.

To manually configure NTP servers, the role has a variable named `timesync_ntp_servers`. It takes a list of NTP servers to use. Each item in the list is made up of one or more attributes. The two key attributes are:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

timesync_ntp_servers attributes

Attribute	Purpose
`hostname`	The hostname of an NTP server with which to synchronize.
`iburst`	A Boolean that enables or disables fast initial synchronization. Defaults to `no` in the role, you should normally set this to `yes`.

Given this information, the following example is a play that uses the `rhel-system-roles.timesync` role to configure managed hosts to get time from three NTP servers using fast initial synchronization. In addition, a task has been added that uses the `timezone` module to set the hosts' time zone to UTC.

```
- name: Time Synchronization Play
  hosts: servers
  vars:
    timesync_ntp_servers:
      - hostname: 0.rhel.pool.ntp.org
        iburst: yes
      - hostname: 1.rhel.pool.ntp.org
        iburst: yes
      - hostname: 2.rhel.pool.ntp.org
        iburst: yes
    timezone: UTC

  roles:
    - rhel-system-roles.timesync

  tasks:
    - name: Set timezone
      timezone:
        name: "{{ timezone }}"
```

Note

If you want to set a different time zone, you can use the `tzselect` command to look up other valid values. You can also use the `timedatectl` command to check current clock settings.

This example sets the role variables in a `vars` section of the play, but a better practice might be to configure them as inventory variables for hosts or host groups.

Consider a playbook project with the following structure:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[root@host playbook-project]# tree
.
├── ansible.cfg
├── group_vars
│   └── servers
│       └── timesync.yml❶
├── inventory
└── timesync_playbook.yml❷
```

❶ Defines the time synchronization variables overriding the role defaults for hosts in group `servers` in the inventory. This file would look something like:

```
timesync_ntp_servers:
  - hostname: 0.rhel.pool.ntp.org
    iburst: yes
  - hostname: 1.rhel.pool.ntp.org
    iburst: yes
  - hostname: 2.rhel.pool.ntp.org
    iburst: yes
timezone: UTC
```

❷ The content of the playbook simplifies to:

```
- name: Time Synchronization Play
  hosts: servers
  roles:
    - rhel-system-roles.timesync
  tasks:
    - name: Set timezone
      timezone:
        name: "{{ timezone }}"
```

This structure cleanly separates the role, the playbook code, and configuration settings. The playbook code is simple, easy to read, and should not require complex refactoring. The role content is maintained and supported by Red Hat. All the settings are handled as inventory variables.

This structure also supports a dynamic, heterogeneous environment. Hosts with new time synchronization requirements may be placed in a new host group. Appropriate variables are defined in a YAML file, and placed in the appropriate `group_vars` (or `host_vars`) subdirectory.

SELinux Role Example

As another example, the `rhel-system-roles.selinux` role simplifies management of SELinux configuration settings. It is implemented using the SELinux-related Ansible modules. The advantage of using this role instead of writing your own tasks is that it relieves you from the responsibility of writing those tasks. Instead, you provide variables to the role to configure it, and the maintained code in the role will ensure your desired SELinux configuration is applied.

Among the tasks this role can perform:

- Set enforcing or permissive mode
- Run `restorecon` on parts of the file system hierarchy

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Set SELinux Boolean values
- Set SELinux file contexts persistently
- Set SELinux user mappings

Calling the SELinux Role

Sometimes, the SELinux role must ensure the managed hosts are rebooted in order to completely apply its changes. However, it does not ever reboot hosts itself. This is so that you can control how the reboot is handled. But it means that it is a little more complicated than usual to properly use this role in a play.

The way this works is that the role will set a Boolean variable, `selinux_reboot_required`, to `true` and fail if a reboot is needed. You can use a `block/rescue` structure to recover from the failure, by failing the play if that variable is not set to `true` or rebooting the managed host and rerunning the role if it is `true`. The block in your play should look something like this:

```
    - name: Apply SELinux role
      block:
        - include_role:
            name: rhel-system-roles.selinux
      rescue:
        - name: Check for failure for other reasons than required reboot
          fail:
          when: not selinux_reboot_required

        - name: Restart managed host
          reboot:

        - name: Reapply SELinux role to complete changes
          include_role:
            name: rhel-system-roles.selinux
```

Configuring the SELinux Role

The variables used to configure the `rhel-system-roles.selinux` role are documented in its `README.md` file. The following examples show some ways to use this role.

The `selinux_state` variable sets the mode SELinux runs in. It can be set to `enforcing`, `permissive`, or `disabled`. If it is not set, the mode is not changed.

```
selinux_state: enforcing
```

The `selinux_booleans` variable takes a list of SELinux Boolean values to adjust. Each item in the list is a hash/dictionary of variables: the `name` of the Boolean, the `state` (whether it should be `on` or `off`), and whether the setting should be `persistent` across reboots.

This example sets `httpd_enable_homedirs` to `on` persistently:

```
selinux_booleans:
  - name: 'httpd_enable_homedirs'
    state: 'on'
    persistent: 'yes'
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The `selinux_fcontext` variable takes a list of file contexts to persistently set (or remove). It works much like the `selinux fcontext` command.

The following example ensures the policy has a rule to set the default SELinux type for all files under `/srv/www` to `httpd_sys_content_t`.

```
selinux_fcontexts:
  - target: '/srv/www(/.*)?'
    setype: 'httpd_sys_content_t'
    state: 'present'
```

The `selinux_restore_dirs` variable specifies a list of directories on which to run `restorecon`:

```
selinux_restore_dirs:
  - /srv/www
```

The `selinux_ports` variable takes a list of ports that should have a specific SELinux type.

```
selinux_ports:
  - ports: '82'
    setype: 'http_port_t'
    proto: 'tcp'
    state: 'present'
```

There are other variables and options for this role. See its `README.md` file for more information.

References

Red Hat Enterprise Linux (RHEL) System Roles
https://access.redhat.com/articles/3050101

Linux System Roles
https://linux-system-roles.github.io/

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Reusing Content with System Roles

In this exercise, you will use one of the Red Hat Enterprise Linux System Roles in conjunction with a normal task to configure time synchronization and the time zone on your managed hosts.

Outcomes

You should be able to:

- Install the Red Hat Enterprise Linux System Roles.
- Find and use the RHEL System Roles documentation.
- Use the `rhel-system-roles.timesync` role in a playbook to configure time synchronization on remote hosts.

Scenario Overview

Your organization maintains two data centers: one in the United States (Chicago) and one in Finland (Helsinki). To aid log analysis of database servers across data centers, ensure the system clock on each host is synchronized using Network Time Protocol. To aid time-of-day activity analysis across data centers, ensure each database server has a time zone set that corresponds to the host's data center location.

Time synchronization has the following requirements:

- Use the NTP server located at `classroom.example.com`. Enable the `iburst` option to accelerate initial time synchronization.
- Use the *chrony* package for time synchronization.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-system start` command. This creates the working directory, `/home/student/role-system`, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab role-system start
```

Instructions

1. Change to the `/home/student/role-system` working directory.

```
[student@workstation ~]$ cd ~/role-system
[student@workstation role-system]$
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2. Install the Red Hat Enterprise Linux system roles on the control node, `workstation.lab.example.com`. Verify the installed location of the roles on the control node.

   2.1. Use the `ansible-galaxy` command to verify that no roles are initially available for use in the playbook project.

```
[student@workstation role-system]$ ansible-galaxy list
# /home/student/role-system/roles
# /usr/share/ansible/roles
# /etc/ansible/roles
```

   The `ansible-galaxy` command searches three directories for roles, as indicated by the `roles_path` entry in the `ansible.cfg` file:

   - `./roles`
   - `/usr/share/ansible/roles`
   - `/etc/ansible/roles`

   The above output indicates there are no roles in any of these directories.

   2.2. Install the *rhel-system-roles* package.

```
[student@workstation role-system]$ sudo yum install rhel-system-roles
```

   Enter `y` when prompted to install the package.

   2.3. Use the `ansible-galaxy` command to verify that the system roles are now available.

```
[student@workstation role-system]$ ansible-galaxy list
# /home/student/role-system/roles
# /usr/share/ansible/roles
...output omitted...
- rhel-system-roles.timesync, (unknown version)
- rhel-system-roles.tlog, (unknown version)
# /etc/ansible/roles
```

   The roles are located in the `/usr/share/ansible/roles` directory. Any role beginning with `linux-system-roles` is actually a symlink to the corresponding `rhel-system-roles` role.

3. Create a playbook, `configure_time.yml`, with one play that targets the `database_servers` host group. Include the `rhel-system-roles.timesync` role in the `roles` section of the play.

```
---
- name: Time Synchronization
  hosts: database_servers

  roles:
    - rhel-system-roles.timesync
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4. The role documentation contains a description of each role variable, including the default value for the variable. Determine the role variables to override to meet the requirements for time synchronization.

Place role variable values in a file named `timesync.yml`. Because these variable values apply to all hosts in the inventory, place the `timesync.yml` file in the `group_vars/all` subdirectory.

4.1. Review the *Role Variables* section of the `README.md` file for the `rhel-system-roles.timesync` role.

```
[student@workstation role-system]$ cat \
> /usr/share/doc/rhel-system-roles/timesync/README.md
...output omitted...
Role Variables
--------------
...output omitted...
# List of NTP servers
timesync_ntp_servers:
  - hostname: foo.example.com   # Hostname or address of the server
    minpoll: 4                  # Minimum polling interval (default 6)
    maxpoll: 8                  # Maximum polling interval (default 10)
    iburst: yes                 # Flag enabling fast initial synchronization
                                # (default no)
    pool: no                    # Flag indicating that each resolved address
                                # of the hostname is a separate NTP server
                                # (default no)
...output omitted...
# Name of the package which should be installed and configured for NTP.
# Possible values are "chrony" and "ntp". If not defined, the currently active
# or enabled service will be configured. If no service is active or enabled, a
# package specific to the system and its version will be selected.
timesync_ntp_provider: chrony
...output omitted...
```

4.2. Create the `group_vars/all` subdirectory.

```
[student@workstation role-system]$ mkdir -pv group_vars/all
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/all'
```

4.3. Create a new file `group_vars/all/timesync.yml` using a text editor. Add variable definitions to satisfy the time synchronization requirements. The file now contains:

```
---
#rhel-system-roles.timesync variables for all hosts

timesync_ntp_provider: chrony

timesync_ntp_servers:
  - hostname: classroom.example.com
    iburst: yes
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. Add a task to `configure_time.yml`, to set the time zone for each host. Ensure the task uses the `timezone` module and executes after the `rhel-system-roles.timesync` role.

Because hosts do not belong to the same time zone, use a variable (`host_timezone`) for the time zone name.

5.1. Review the *Examples* section of the `timezone` module documentation.

```
[student@workstation role-system]$ ansible-doc timezone | grep -A 4 "EXAMPLES"
EXAMPLES:

- name: set timezone to Asia/Tokyo
  timezone:
    name: Asia/Tokyo
```

5.2. Add a task to the `post_tasks` section of the play in the `configure_time.yml` playbook. Model the task after the example from the documentation, but use the `host_timezone` variable for the time zone name.

The documentation in `ansible-doc timezone` recommends a restart of the Cron service if the module changes the timezone, to make sure Cron jobs run at the right times. Since system logging and other services use the system time zone, reboot each host when the time zone is modified. Add a `notify` keyword to the task, with an associated value of `reboot host`. The `post_tasks` section of the play should read:

```
  post_tasks:
    - name: Set timezone
      timezone:
        name: "{{ host_timezone }}"
      notify: reboot host
```

5.3. Add the `reboot host` handler to the `Time Synchronization` play. The complete playbook now contains:

```
---
- name: Time Synchronization
  hosts: database_servers

  roles:
    - rhel-system-roles.timesync

  post_tasks:
    - name: Set timezone
      timezone:
        name: "{{ host_timezone }}"
      notify: reboot host

  handlers:
    - name: reboot host
      reboot:
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

6. For each data center, create a file named `timezone.yml` that contains an appropriate value for the `host_timezone` variable. Use the `timedatectl list-timezones` command to find the valid time zone string for each data center.

6.1. Create the `group_vars` subdirectories for the `na_datacenter` and `europe_datacenter` host groups.

```
[student@workstation role-system]$ mkdir -pv \
> group_vars/{na_datacenter,europe_datacenter}
mkdir: created directory 'group_vars/na_datacenter'
mkdir: created directory 'group_vars/europe_datacenter'
```

6.2. Use the `timedatectl list-timezones` command to determine the time zone for both the US and European data centers:

```
[student@workstation role-system]$ timedatectl list-timezones | grep Chicago
America/Chicago
[student@workstation role-system]$ timedatectl list-timezones | grep Helsinki
Europe/Helsinki
```

6.3. Create the `timezone.yml` for both data centers:

```
[student@workstation role-system]$ echo "host_timezone: America/Chicago" > \
> group_vars/na_datacenter/timezone.yml
[student@workstation role-system]$ echo "host_timezone: Europe/Helsinki" > \
> group_vars/europe_datacenter/timezone.yml
```

7. Run the playbook.

```
[student@workstation role-system]$ ansible-playbook configure_time.yml

PLAY [Time Synchronization] ***************************************************

TASK [Gathering Facts] ********************************************************
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [rhel-system-roles.timesync : Check if only NTP is needed] ***************
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]

...output omitted...

TASK [rhel-system-roles.timesync : Enable timemaster] *************************
skipping: [servera.lab.example.com]
skipping: [serverb.lab.example.com]

RUNNING HANDLER [rhel-system-roles.timesync : restart chronyd] ****************
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

TASK [Set timezone] ***********************************************************
changed: [serverb.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
changed: [servera.lab.example.com]

RUNNING HANDLER [reboot host] ************************************************
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

servera.lab.example.com    : ok=17   changed=6    unreachable=0    failed=0
  skipped=20   rescued=0    ignored=6
serverb.lab.example.com    : ok=17   changed=6    unreachable=0    failed=0
  skipped=20   rescued=0    ignored=6
```

8. Verify the time zone settings of each server. Use an Ansible ad hoc command to see the output of the `date` command on all the database servers.

Note

The actual timezones listed will vary depending on the time of year, and whether daylight savings is active.

```
[student@workstation role-system]$ ansible database_servers -m shell -a date
servera.lab.example.com | CHANGED | rc=0 >>
Fri Jul 16 17:38:40 CDT 2021
serverb.lab.example.com | CHANGED | rc=0 >>
Sat Jul 17 01:38:40 EEST 2021
```

Each server has a time zone setting based on its geographic location.

Finish

Run the `lab role-system finish` command to cleanup the managed host.

```
[student@workstation ~]$ lab role-system finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Creating Roles

Objectives

After completing this section, you should be able to create a role in a playbook's project directory and run it as part of one of the plays in the playbook.

The Role Creation Process

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

By default, Ansible looks for roles in a subdirectory called `roles` in the directory containing your Ansible Playbook. This allows you to store roles with the playbook and other supporting files.

If Ansible cannot find the role there, it looks at the directories specified by the Ansible configuration setting `roles_path`, in order. This variable contains a colon-separated list of directories to search. The default value of this variable is:

```
~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles
```

This allows you to install roles on your system that are shared by multiple projects. For example, you could have your own roles installed your home directory in the `~/.ansible/roles` subdirectory, and the system can have roles installed for all users in the `/usr/share/ansible/roles` directory.

Each role has its own directory with a standardized directory structure. For example, the following directory structure contains the files that define the `motd` role.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
|   └── main.yml
└── templates
    └── motd.j2
```

The `README.md` provides a basic human-readable description of the role, documentation and examples of how to use it, and any non-Ansible requirements it might have in order to work. The `meta` subdirectory contains a `main.yml` file that specifies information about the author, license, compatibility, and dependencies for the module. 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 this example, it is ignored. If a role does not use a feature, the subdirectory can be omitted altogether. For example, the `vars` subdirectory has been omitted from this example.

Creating a Role Skeleton

You can create all the subdirectories and files needed for a new role using standard Linux commands. Alternatively, command line utilities exist to automate the process of new role creation.

The `ansible-galaxy` command line tool (covered in more detail later in this course) is used to manage Ansible roles, including the creation of new roles. You can run `ansible-galaxy init` to create the directory structure for a new role. Specify the name of the role as an argument to the command, which creates a subdirectory for the new role in the current working directory.

```
[user@host playbook-project]$ cd roles
[user@host roles]$ ansible-galaxy init my_new_role
- my_new_role was created successfully
[user@host roles]$ ls my_new_role/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

Defining the Role Content

Once you have created the directory structure, you must write the content of the role. A good place to start is the *ROLENAME*`/tasks/main.yml` task file, the main list of tasks run by the role.

The following `tasks/main.yml` file manages the `/etc/motd` file on managed hosts. It uses the `template` module to deploy the template named `motd.j2` to the managed host. Because the `template` module is configured within a role task, instead of a playbook task, the `motd.j2` template is retrieved from the role's `templates` subdirectory.

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file
  template:
    src: motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The following command displays the contents of the `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_facts['hostname'] }}.

Today's date is: {{ ansible_facts['date_time']['date'] }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

The role defines a default value for the `system_owner` variable. The `defaults/main.yml` file in the role's directory structure is where this value is 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
```

Recommended Practices for Role Content Development

Roles allow playbooks to be written modularly. To maximize the effectiveness of newly developed roles, consider implementing the following recommended practices into your role development:

- Maintain each role in its own version control repository. Ansible works well with `git`-based repositories.
- Sensitive information, such as passwords or SSH keys, should not be stored in the role repository. Sensitive values should be parameterized as variables with default values that are not sensitive. Playbooks that use the role are responsible for defining sensitive variables through Ansible Vault variable files, environment variables, or other `ansible-playbook` options.
- Use `ansible-galaxy init` to start your role, and then remove any directories and files that you do not need.
- Create and maintain `README.md` and `meta/main.yml` files to document what your role is for, who wrote it, and how to use it.
- Keep your role focused on a specific purpose or function. Instead of making one role do many things, you might write more than one role.
- Reuse and refactor roles often. Resist creating new roles for edge configurations. If an existing role accomplishes a majority of the required configuration, refactor the existing role to integrate the new configuration scenario. Use integration and regression testing techniques to ensure that the role provides the required new functionality and also does not cause problems for existing playbooks.

Defining Role Dependencies

Role dependencies allow a role to include other roles as dependencies. 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.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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.

Important

Limit your role's dependencies on other roles. Dependencies make it harder to maintain your role, especially if it has many complex dependencies.

Using the Role in a Playbook

To access a role, reference it in the `roles:` section of a play. The following playbook refers to the `motd` role. Because no variables are specified, the role is applied with its default variable values.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_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 :` prefix in the task name:

```
[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
```

The above scenario assumes that the `motd` role is located in the `roles` directory. Later in the course you will see how to use a role that is remotely located in a version control repository.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Changing a Role's Behavior with Variables

A well-written role uses default variables to alter the role's behavior to match a related configuration scenario. This helps make the role more generic and reusable in a variety of contexts.

The value of any variable defined in a role's `defaults` directory will be overwritten if that same variable is defined:

- in an inventory file, either as a host variable or a group variable.
- in a YAML file under the `group_vars` or `host_vars` directories of a playbook project
- as a variable nested in the `vars` keyword of a play
- as a variable when including the role in `roles` keyword of a play

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
  remote_user: devops
  become: true
  vars:
    system_owner: someone@host.example.com
  roles:
    - role: motd
```

When defined in this way, the `system_owner` variable replaces the value of the default variable of the same name. Any variable definitions nested within the `vars` keyword will not replace the value of the same variable if defined in a role's `vars` directory.

The following example also 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 regardless of being defined in the role's `vars` or `defaults` directory.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  roles:
    - role: motd
      system_owner: someone@host.example.com
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Important

Variable precedence can be confusing when working with role variables in a play.

- Almost any other variable will override a role's default variables: inventory variables, play `vars`, inline *role parameters*, and so on.
- Fewer variables can override variables defined in a role's `vars` directory. Facts, variables loaded with `include_vars`, registered variables, and role parameters are some variables that can do that. Inventory variables and play `vars` cannot. This is important because it helps keep your play from accidentally changing the internal functioning of the role.
- However, variables declared inline as role parameters, like the last of the preceding examples, have very high precedence. They can override variables defined in a role's `vars` directory. If a role parameter has the same name as a variable set in play `vars`, a role's `vars`, or an inventory or playbook variable, the role parameter overrides the other variable.

References

Using Roles — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html#using-roles

Using Variables — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Creating Roles

In this exercise, you will create an Ansible role that uses variables, files, templates, tasks, and handlers to deploy a network service.

Outcomes

You should be able to create a role that uses variables and parameters.

The `myvhost` role installs and configures the Apache service on a host. A template named `vhost.conf.j2` is provided that will be used to generate `/etc/httpd/conf.d/vhost.conf`.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-create start` command. This creates the working directory, `/home/student/role-create`, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab role-create start
```

Instructions

1. Change to the `/home/student/role-create` working directory.

```
[student@workstation ~]$ cd ~/role-create
[student@workstation role-create]$
```

2. Create the directory structure for a role called `myvhost`. The role includes fixed files, templates, tasks, and handlers.

```
[student@workstation role-create]$ mkdir -v roles; cd roles
mkdir: created directory 'roles'
[student@workstation roles]$ ansible-galaxy init myvhost
- myvhost was created successfully
[student@workstation roles]$ rm -rvf myvhost/{defaults,vars,tests}
removed 'myvhost/defaults/main.yml'
removed directory: 'myvhost/defaults'
removed 'myvhost/vars/main.yml'
removed directory: 'myvhost/vars'
removed 'myvhost/tests/inventory'
removed 'myvhost/tests/test.yml'
removed directory: 'myvhost/tests'
[student@workstation roles]$ cd ..
[student@workstation role-create]$
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. Edit the `main.yml` file in the `tasks` subdirectory of the role. The role should perform the following tasks:

- The *httpd* package is installed
- The `httpd` service is started and enabled
- The web server configuration file is installed, using a template provided by the role

3.1. Edit the `roles/myvhost/tasks/main.yml` file. Include code to use the `yum` module to install the *httpd* package. The file contents should look like the following:

```
---
# tasks file for myvhost

- name: Ensure httpd is installed
  yum:
    name: httpd
    state: latest
```

3.2. Add additional code to the `roles/myvhost/tasks/main.yml` file to use the `service` module to start and enable the `httpd` service.

```
- name: Ensure httpd is started and enabled
  service:
    name: httpd
    state: started
    enabled: true
```

3.3. 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: vhost file is installed
  template:
    src: vhost.conf.j2
    dest: /etc/httpd/conf.d/vhost.conf
    owner: root
    group: root
    mode: 0644
  notify:
    - restart httpd
```

3.4. Save your changes and exit the `roles/myvhost/tasks/main.yml` file.

4. Create the handler for restarting the `httpd` service. Edit the `roles/myvhost/handlers/main.yml` file and include code to use the `service` module, then save and exit. The file contents should look like the following:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
# handlers file for myvhost

- name: restart httpd
  service:
    name: httpd
    state: restarted
```

5. Move the `vhost.conf.j2` template from the project directory to the role's `templates` subdirectory.

```
[student@workstation role-create]$ mv -v vhost.conf.j2 roles/myvhost/templates/
renamed 'vhost.conf.j2' -> 'roles/myvhost/templates/vhost.conf.j2'
```

6. Create the HTML content to be served by the web server.

   6.1. Create the `files/html/` directory to store the content in.

```
[student@workstation role-create]$ mkdir -pv files/html
mkdir: created directory 'files/html'
```

   6.2. Create an `index.html` file below that directory with the contents: `simple index`.

```
[student@workstation role-create]$ echo \
> 'simple index' > files/html/index.html
```

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`. Include a task to copy the HTML content from `files/html/`. Use the `copy` module and include a trailing slash after the source directory name. It should have the following content:

```
---
- name: Use myvhost role playbook
  hosts: webservers
  pre_tasks:
    - name: pre_tasks message
      debug:
        msg: 'Ensure web server configuration.'

  roles:
    - myvhost

  post_tasks:
    - name: HTML content is installed
      copy:
        src: files/html/
        dest: "/var/www/vhosts/{{ ansible_hostname }}"
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: post_tasks message
      debug:
        msg: 'Web server is configured.'
```

Note

The trailing slash causes the source directory and all of its contents to be copied to the managed host.

7.2. Before running the playbook, verify that its syntax is correct by running `ansible-playbook` with the `--syntax-check`. If it reports any errors, correct them before moving to the next step. You should see output similar to the following:

```
[student@workstation role-create]$ ansible-playbook use-vhost-role.yml \
> --syntax-check

playbook: use-vhost-role.yml
```

7.3. Run the playbook. Review the output to confirm that Ansible performed the actions on the web server, `servera`.

```
[student@workstation role-create]$ ansible-playbook use-vhost-role.yml

PLAY [Use myvhost role playbook] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [pre_tasks message] *******************************************************
ok: [servera.lab.example.com] => {
    "msg": "Ensure web server configuration."
}

TASK [myvhost : Ensure httpd is installed] *************************************
changed: [servera.lab.example.com]

TASK [myvhost : Ensure httpd is started and enabled] ***************************
changed: [servera.lab.example.com]

TASK [myvhost : vhost file is installed] ***************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [myvhost : restart httpd] **************************************
changed: [servera.lab.example.com]

TASK [HTML content is installed] ***********************************
changed: [servera.lab.example.com]

TASK [post_tasks message] ******************************************************
ok: [servera.lab.example.com] => {
    "msg": "Web server is configured."
}
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
```

7.4. 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 role-create]$ ansible webservers -a \
> 'systemctl is-active httpd'
servera.lab.example.com | CHANGED | rc=0 >>
active

[student@workstation role-create]$ ansible webservers -a \
> 'systemctl is-enabled httpd'
servera.lab.example.com | CHANGED | rc=0 >>
enabled
```

7.5. The Apache configuration should be installed with template variables expanded.

```
[student@workstation role-create]$ ansible webservers -a \
> 'cat /etc/httpd/conf.d/vhost.conf'
servera.lab.example.com | CHANGED | rc=0 >>
# Ansible managed:

<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
        Allow from all
    </Directory>
</VirtualHost>
```

7.6. 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 role-create]$ ansible webservers -a \
> 'cat /var/www/vhosts/servera/index.html'
servera.lab.example.com | CHANGED | rc=0 >>
simple index
```

7.7. Use the `uri` module in an ad hoc command to check that the web content is available locally. Set the `return_content` parameter to `true` to have the content of the server's response added to the output. The server content should be the string `simple index\n`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation role-create]$ ansible webservers -m uri \
> -a 'url=http://localhost return_content=true'
servera.lab.example.com | SUCCESS => {
    "accept_ranges": "bytes",
    "changed": false,
    "connection": "close",
    "content": "simple index\n",
...output omitted...
    "status": 200,
    "url": "http://localhost"
}
```

7.8. Confirm that the web server content is available to remote clients.

```
[student@workstation role-create]$ curl http://servera.lab.example.com
simple index
```

Finish

Run the `lab role-create finish` command to clean up the managed host.

```
[student@workstation ~]$ lab role-create finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Deploying Roles with Ansible Galaxy

Objectives

After completing this section, you should be able to select and retrieve roles from Ansible Galaxy or other sources such as a Git repository, and use them in your playbooks.

Introducing Ansible Galaxy

Ansible Galaxy [https://galaxy.ansible.com] is a public library of Ansible content written by a variety of Ansible administrators and users. It 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.

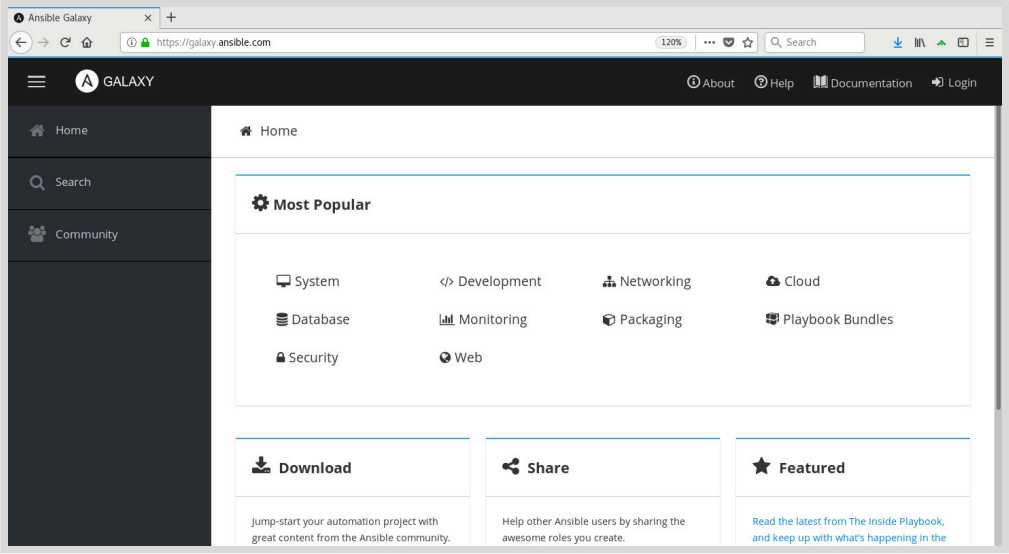


Figure 7.1: Ansible Galaxy home page

In addition, the `ansible-galaxy` command that you use to get and manage roles from Ansible Galaxy can also be used to get and manage roles your projects need from your own Git repositories.

Getting Help with Ansible Galaxy

The **Documentation** 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.

Browsing Ansible Galaxy for Roles

The **Search** tab on the left side of the Ansible Galaxy website home page gives users access to information about the roles published on Ansible Galaxy. You can search for an Ansible role by its name, using tags, or by other role attributes. Results are presented in descending order of the

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

`Best Match` score, which is a computed score based on role quality, role popularity, and search criteria.

> **Note**
> Content Scoring [https://galaxy.ansible.com/docs/contributing/content_scoring.html] in the documentation has more information on how roles are scored by Ansible Galaxy.

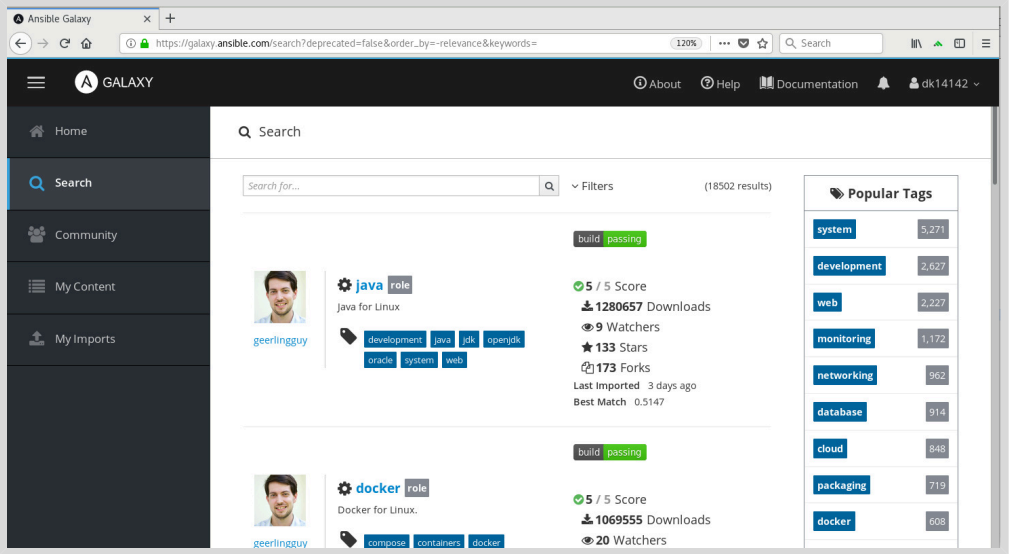


Figure 7.2: Ansible Galaxy search screen

Ansible Galaxy reports the number of times each role has been downloaded from Ansible Galaxy. In addition, Ansible Galaxy also reports the number of watchers, forks, and stars the role's GitHub repository has. Users can use this information to help determine how active development is for a role and how popular it is in the community.

The following figure shows the search results that Ansible Galaxy displayed after a keyword search for `redis` was performed. Notice the first result has a `Best Match` score of `0.9009`.

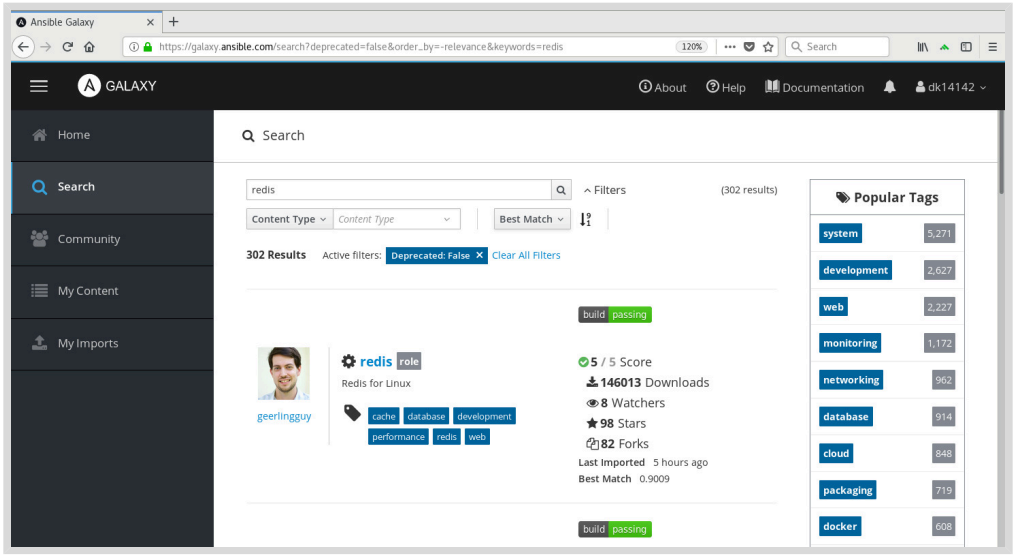


Figure 7.3: Ansible Galaxy search results example

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The **Filters** pulldown menu to the right of the search box allow searches to be performed on keywords, author IDs, platform, and tags. Possible platform values include `EL` for Red Hat Enterprise Linux (and closely related distributions such as CentOS) and `Fedora`, among others.

Tags are arbitrary single-word strings set by the role author that describe and categorize the role. Users can use tags to find relevant roles. Possible tag values include `system`, `development`, `web`, `monitoring`, and others. A role can have up to 20 tags in Ansible Galaxy.

Important

In the Ansible Galaxy search interface, keyword searches match words or phrases in the README file, content name, or content description. Tag searches, by contrast, specifically match tag values set by the author for the role.

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.

Searching for Roles from the Command Line

The `ansible-galaxy search` subcommand searches Ansible Galaxy for roles. If you specify a string as an argument, it is used to search Ansible Galaxy for roles by keyword. You can use the `--author`, `--platforms`, and `--galaxy-tags` options to narrow the search results. You can also use those options as the main search key. For example, the command `ansible-galaxy search --author geerlingguy` will display all roles submitted by the user `geerlingguy`.

Results are displayed in alphabetical order, not by descending `Best Match` score. The following example displays the names of roles that include `redis`, and are available for the Enterprise Linux (`EL`) platform.

```
[user@host ~]$ ansible-galaxy search 'redis' --platforms EL

Found 124 roles matching your search:

 Name                                Description
 ----                                -----------
 1it.sudo                            Ansible role for managing sudoers
 AerisCloud.librato                  Install and configure the Librato Agent
 AerisCloud.redis                    Installs redis on a server
 AlbanAndrieu.java                   Manage Java installation
 andrewrothstein.redis               builds Redis from src and installs
...output omitted...
 geerlingguy.php-redis               PhpRedis support for Linux
 geerlingguy.redis                   Redis for Linux
 gikoluo.filebeat                    Filebeat for Linux.
...output omitted...
```

The `ansible-galaxy info` subcommand displays more detailed information about a role. Ansible Galaxy gets this information from a number of places including the role's `meta/main.yml` file and its GitHub repository. The following command displays information about the `geerlingguy.redis` role, available from Ansible Galaxy.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@host ~]$ ansible-galaxy info geerlingguy.redis

Role: geerlingguy.redis
        description: Redis for Linux
        active: True
...output omitted...
        download_count: 146209
        forks_count: 82
        github_branch: master
        github_repo: ansible-role-redis
        github_user: geerlingguy
...output omitted...
        license: license (BSD, MIT)
        min_ansible_version: 2.4
        modified: 2018-11-19T14:53:29.722718Z
        open_issues_count: 11
        path: [u'/etc/ansible/roles', u'/usr/share/ansible/roles']
        role_type: ANS
        stargazers_count: 98
...output omitted...
```

Installing Roles from Ansible Galaxy

The `ansible-galaxy install` subcommand downloads a role from Ansible Galaxy, then installs it locally on the control node.

By default, roles are installed into the first directory that is writable in the user's `roles_path`. Based on the default `roles_path` set for Ansible, normally the role will be installed into the user's `~/.ansible/roles` directory. The default `roles_path` might be overridden by your current Ansible configuration file or by the environment variable `ANSIBLE_ROLES_PATH`, which affects the behavior of `ansible-galaxy`.

You can also specify a specific directory to install the role into by using the `-p` *`DIRECTORY`* option.

In the following example, `ansible-galaxy` installs the `geerlingguy.redis` role into a playbook project's `roles` directory. The command's current working directory is `/opt/project`.

```
[user@host project]$ ansible-galaxy install geerlingguy.redis -p roles/
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/...output omitted...
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
[user@host project]$ ls roles/
geerlingguy.redis
```

Installing Roles Using a Requirements File

You can also use `ansible-galaxy` to install a list of roles based on definitions in a text file. For example, if you have a playbook that needs to have specific roles installed, you can create a `roles/requirements.yml` file in the project directory that specifies which roles are needed. This file acts as a dependency manifest for the playbook project which enables playbooks to be developed and tested separately from any supporting roles.

For example, a simple `requirements.yml` to install `geerlingguy.redis` might read like this:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- src: geerlingguy.redis
  version: "1.5.0"
```

The `src` attribute specifies the source of the role, in this case the `geerlingguy.redis` role from Ansible Galaxy. The `version` attribute is optional, and specifies the version of the role to install, in this case `1.5.0`.

Important

You should specify the version of the role in your `requirements.yml` file, especially for playbooks in production.

If you do not specify a version, you will get the latest version of the role. If the upstream author makes changes to the role that are incompatible with your playbook, it may cause an automation failure or other problems.

To install the roles using a role file, use the `-r` *`REQUIREMENTS-FILE`* option:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
> -p roles
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
```

You can use `ansible-galaxy` to install roles that are not in Ansible Galaxy. You can host your own proprietary or internal roles in a private Git repository or on a web server. The following example shows how to configure a requirements file using a variety of remote sources.

```
[user@host project]$ cat roles/requirements.yml
# from Ansible Galaxy, using the latest version
- src: geerlingguy.redis

# from Ansible Galaxy, overriding the name and using a specific version
- src: geerlingguy.redis
  version: "1.5.0"
  name: redis_prod

# from any Git-based repository, using HTTPS
- src: https://gitlab.com/guardianproject-ops/ansible-nginx-acme.git
  scm: git
  version: 56e00a54
  name: nginx-acme

# from any Git-based repository, using SSH
- src: git@gitlab.com:guardianproject-ops/ansible-nginx-acme.git
  scm: git
  version: master
  name: nginx-acme-ssh

# from a role tar ball, given a URL;
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
#   supports 'http', 'https', or 'file' protocols
- src: file:///opt/local/roles/myrole.tar
  name: myrole
```

The `src` keyword specifies the Ansible Galaxy role name. If the role is not hosted on Ansible Galaxy, the `src` keyword indicates the role's URL.

If the role is hosted in a source control repository, the `scm` attribute is required. The `ansible-galaxy` command is capable of downloading and installing roles from either a Git-based or mercurial-based software repository. A Git-based repository requires an `scm` value of `git`, while a role hosted on a mercurial repository requires a value of `hg`. If the role is hosted on Ansible Galaxy or as a tar archive on a web server, the `scm` keyword is omitted.

The `name` keyword is used to override the local name of the role. The `version` keyword is used to specify a role's version. The `version` keyword can be any value that corresponds to a branch, tag, or commit hash from the role's software repository.

To install the roles associated with a playbook project, execute the `ansible-galaxy install` command:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
> -p roles
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.5.0.tar.gz
- extracting redis_prod to /opt/project/roles/redis_prod
- redis_prod (1.5.0) was installed successfully
- extracting nginx-acme to /opt/project/roles/nginx-acme
- nginx-acme (56e00a54) was installed successfully
- extracting nginx-acme-ssh to /opt/project/roles/nginx-acme-ssh
- nginx-acme-ssh (master) was installed successfully
- downloading role from file:///opt/local/roles/myrole.tar
- extracting myrole to /opt/project/roles/myrole
- myrole was installed successfully
```

Managing Downloaded Roles

The `ansible-galaxy` command can also manage local roles, such as those roles found in the `roles` directory of a playbook project. The `ansible-galaxy list` subcommand lists the roles that are found locally.

```
[user@host project]$ ansible-galaxy list
- geerlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- nginx-acme-ssh, master
- redis_prod, 1.5.0
```

A role can be removed locally with the `ansible-galaxy remove` subcommand.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@host ~]$ ansible-galaxy remove nginx-acme-ssh
- successfully removed nginx-acme-ssh
[user@host ~]$ ansible-galaxy list
- geerlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- redis_prod, 1.5.0
```

Use downloaded and installed roles in playbooks like any other role. They may be referenced in the `roles` section using their downloaded role name. If a role is not in the project's `roles` directory, the `roles_path` will be checked to see if the role is installed in one of those directories, first match being used. The following `use-role.yml` playbook references the `redis_prod` and `geerlingguy.redis` roles:

```
[user@host project]$ cat use-role.yml
---
- name: use redis_prod for Prod machines
  hosts: redis_prod_servers
  remote_user: devops
  become: true
  roles:
    - redis_prod

- name: use geerlingguy.redis for Dev machines
  hosts: redis_dev_servers
  remote_user: devops
  become: true
  roles:
    - geerlingguy.redis
```

This playbook causes different versions of the `geerlingguy.redis` role to be applied to the production and development servers. In this manner, changes to the role can be systematically tested and integrated before deployment to the production servers. If a recent change to a role causes problems, using version control to develop the role allows you to roll back to a previous, stable version of the role.

References

Ansible Galaxy – Ansible Documentation

https://docs.ansible.com/ansible/2.9/cli/ansible-galaxy.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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.

Outcomes

You should be able to:

- create a roles file to specify role dependencies for a playbook
- install roles specified in a roles file
- list roles using the `ansible-galaxy` command

Scenario Overview

Your organization places custom files in the `/etc/skel` directory on all hosts. As a result, new user accounts are configured with a standardized organization-specific Bash environment.

You will test the development version of the Ansible role responsible for deploying Bash environment skeleton files.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-galaxy start` command. This creates the working directory, `/home/student/role-galaxy`, and populates it with an Ansible configuration file and host inventory.

```
[student@workstation ~]$ lab role-galaxy start
```

Instructions

1. Change to the `role-galaxy` working directory.

```
[student@workstation ~]$ cd ~/role-galaxy
[student@workstation role-galaxy]$
```

2. To test the Ansible role that configures skeleton files, add the role specification to a roles file.

   Launch your favorite text editor and create a file called `requirements.yml` in the `roles` subdirectory. The URL of the role's Git repository is: `git@workstation.lab.example.com:student/bash_env`. To see how the role affects the behavior of production hosts, use the `master` branch of the repository. Set the local name of the role to `student.bash_env`.

   The `roles/requirements.yml` now contains the following content:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
# requirements.yml

- src: git@workstation.lab.example.com:student/bash_env
  scm: git
  version: master
  name: student.bash_env
```

3. Use the `ansible-galaxy` command to process the roles file you just created and install the `student.bash_env` role.

   3.1. For comparison, display the contents of the `roles` subdirectory before the role is installed.

```
[student@workstation role-galaxy]$ ls roles/
requirements.yml
```

   3.2. Use Ansible Galaxy to download and install the roles listed in the `roles/requirements.yml` file. Be sure that any downloaded roles are stored in the `roles` subdirectory.

```
[student@workstation role-galaxy]$ ansible-galaxy install -r \
> roles/requirements.yml -p roles
- extracting student.bash_env to /home/student/role-galaxy/roles/student.bash_env
- student.bash_env (master) was installed successfully
```

   3.3. Display the `roles` subdirectory after the role has been installed. Confirm that it has a new subdirectory called `student.bash_env`, matching the `name` value specified in the YAML file.

```
[student@workstation role-galaxy]$ ls roles/
requirements.yml  student.bash_env
```

   3.4. Try using the `ansible-galaxy` command, without any options, to list the project roles:

```
[student@workstation role-galaxy]$ ansible-galaxy list
# /usr/share/ansible/roles
...output omitted...
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
 [WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

   Because you used the `-p` option with the `ansible-galaxy install` command, the `student.bash_env` role was not installed in the default location. Use the `-p` option with the `ansible-galaxy list` command to list the downloaded roles:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation role-galaxy]$ ansible-galaxy list -p roles
# /home/student/role-galaxy/roles
- student.bash_env, master
...output omitted...
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

Note

The `/home/student/.ansible/roles` directory is in your default `roles_path`, but since you have not attempted to install a role without using the `-p` option, `ansible-galaxy` has not yet created the directory.

4. Create a playbook, called `use-bash_env-role.yml`, that uses the `student.bash_env` role. The contents of the playbook should match the following:

```
---
- name: use student.bash_env role playbook
  hosts: devservers
  vars:
    default_prompt: '[\u on \h in \W dir]\$ '
  pre_tasks:
    - name: Ensure test user does not exist
      user:
        name: student2
        state: absent
        force: yes
        remove: yes

  roles:
    - student.bash_env

  post_tasks:
    - name: Create the test user
      user:
        name: student2
        state: present
        password: "{{ 'redhat' | password_hash('sha512', 'mysecretsalt') }}"
```

To see the effects of the configuration change, a new user account must be created. The `pre_tasks` and `post_tasks` section of the playbook ensure that the `student2` user account is created each time the playbook is executed. After playbook execution, the `student2` account is accessed with a password of `redhat`.

Note

The `user2` password is generated using a filter. Filters take data and modify it; here, the string `redhat` is modified by passing it to the `password_hash` module. Filters are an advanced topic not covered in this course.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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 role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] **********************************

TASK [Gathering Facts] *****************************************************
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] *************************************
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]

TASK [Create the test user] ************************************************
changed: [servera.lab.example.com]

PLAY RECAP *****************************************************************
servera.lab.example.com    : ok=6    changed=4    unreachable=0    failed=0
```

6. Connect to `servera` as the `student2` user using SSH. Observe the custom prompt for the `student2` user, and then disconnect from `servera`.

```
[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

7. Execute the playbook using the development version of the `student.bash_env` role.

   The development version of the role is located in the `dev` branch of the Git repository. The development version of the role uses a new variable, `prompt_color`. Before executing the playbook, add the `prompt_color` variable to the `vars` section of the playbook and set its value to `blue`.

   7.1. Update the `roles/requirements.yml` file, and set the `version` value to `dev`. The `roles/requirements.yml` file now contains:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
# requirements.yml

- src: git@workstation.lab.example.com:student/bash_env
  scm: git
  version: dev
  name: student.bash_env
```

7.2. Use the `ansible-galaxy install` command to install the role using the updated roles file. Use the `--force` option to overwrite the existing `master` version of the role with the `dev` version of the role.

```
[student@workstation role-galaxy]$ ansible-galaxy install \
> -r roles/requirements.yml --force -p roles
- changing role student.bash_env from master to dev
- extracting student.bash_env to /home/student/role-galaxy/roles/student.bash_env
- student.bash_env (dev) was installed successfully
```

7.3. Edit the `use-bash_env-role.yml` file. Add the `prompt_color` variable with a value of `blue` to the `vars` section of the playbook. The file now contains:

```
---
- name: use student.bash_env role playbook
  hosts: devservers
  vars:
    prompt_color: blue
    default_prompt: '[\u on \h in \W dir]\$ '
  pre_tasks:
...output omitted...
```

7.4. Execute the `use-bash_env-role.yml` playbook.

```
[student@workstation role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] ***********************************

TASK [Gathering Facts] ******************************************************
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] **************************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bashrc] **********************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bash_profile] ****************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ***********************************
okay: [servera.lab.example.com]

TASK [Create the test user] *************************************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=6    changed=4    unreachable=0    failed=0
```

8. Connect again to `servera` as the `student2` using SSH. Observe the error for the `student2` user, and then disconnect from `servera`.

```
[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

-bash: [: missing `]'
[student2@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

A Bash error occurred while parsing the `student2` user's `.bash_profile` file.

9. Correct the error in the development version of the `student.bash_env` role, and re-execute the playbook.

9.1. Edit the `roles/student.bash_env/templates/_bash_profile.j2` file. Add the missing `]` character to line 4 and save the file. The top of the file is now:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
```

Save the file.

9.2. Execute the `use-bash_env-role.yml` playbook.

```
[student@workstation role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] **************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] *****************************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bashrc] *************************************
ok: [servera.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [student.bash_env : put away .bash_profile] ****************************
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ***********************************
ok: [servera.lab.example.com]

TASK [Create the test user] *************************************************
changed: [servera.lab.example.com]

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=6    changed=3    unreachable=0    failed=0
```

9.3. Connect again to `servera` as the `student2` using SSH.

```
[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

The error message is no longer present. The custom prompt for the `student2` user now displays with blue characters.

10. The steps above demonstrate that the development version of the `student.bash_env` role is defective. Based on testing results, developers will commit necessary fixes back to the development branch of the role. When the development branch passes required quality checks, developers merge features from the development branch into the `master` branch.

Committing role changes to a Git repository is beyond the scope of this course.

Important

When tracking the latest version of a role in a project, periodically reinstall the role to update it. This ensures that the local copy stays current with bug fixes, patches, and other features.

However, if using a third-party role in production, specify the version to use in order to avoid breakage due to unexpected changes. Periodically update to the latest role version in your test environment so as to adopt improvements and changes in a controlled manner.

Finish

Run the `lab role-galaxy finish` command to clean up the managed host.

```
[student@workstation ~]$ lab role-galaxy finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Getting Roles and Modules from Content Collections

Objectives

After completing this section, you should be able to obtain a set of related roles, supplementary modules, and other content from content collections, and use them in a playbook.

Discussing Content Collections

Ansible content collections are a distribution format for Ansible content. A collection provides a set of related modules, roles, and plug-ins that you can download to your control node and then use in your playbooks.

For example:

- The `redhat.insights` collection groups modules and roles that you can use to register a system with Red Hat Insights for Red Hat Enterprise Linux.
- The `cisco.ios` collection groups modules and plug-ins that manage Cisco IOS network appliances. The Cisco company supports and maintains that collection.
- The `community.crypto` collection provides modules that create SSL/TLS certificates.

Content collections allow updates to the core Ansible code to be separated from updates to modules and plug-ins. This allows vendors and developers to maintain and distribute their collections at their own pace, independently of Ansible releases. You can develop your own collections to provide custom roles and modules to your teams.

Content collections also give you more flexibility. By using collections, you can install only the content you need instead of installing all supported modules. You can also select a specific version of a collection (possibly an earlier or later one) or choose between a version of a collection supported by Red Hat or vendors or one provided by the community.

Ansible 2.9 and later support collections. Later versions of Ansible and Red Hat Ansible Automation Platform will provide further enhancements to collection support and will use them extensively. Understanding how collections work will be important.

The `ansible` RPM package provided with Red Hat Ansible Automation Platform 1.2 and Ansible 2.9 automatically installs all the modules that earlier versions of Ansible did. Future versions of Ansible and Red Hat Ansible Automation Platform will remove most modules from the main RPM package and place them in collections that might be included or that you might have to download.

Organizing Collections in Namespaces

To make it easier to specify collections and their contents by name, collection names are organized into *namespaces*. Vendors, partners, developers, and content creators can use namespaces to assign unique names to their collections without conflicting with other developers.

The namespace is the first part of a collection name. For example, all the collections that the Ansible community maintains are in the `community` namespace, and have names like `community.crypto`, `community.postgresql`, and `community.rabbitmq`. Collections

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

that Red Hat maintains and supports might use the `redhat` namespace, and have names like `redhat.rhv`, `redhat.satellite`, and `redhat.insights`.

Selecting Collection Sources

Ansible provides two official sources to download and install collections: Ansible automation hub and Ansible Galaxy.

Ansible automation hub

Ansible automation hub hosts Ansible content collections that Red Hat and its partners support for their customers. Red Hat reviews, maintains, updates, and fully supports those collections. For example, the `redhat.rhv`, `redhat.satellite`, `redhat.insights`, and `cisco.ios` collections are available on that platform.

You need a valid Red Hat Ansible Automation Platform subscription to access Ansible automation hub. Use the Ansible automation hub web UI at https://cloud.redhat.com/ansible/automation-hub/ to list and access the collections.

Ansible Galaxy

Ansible Galaxy hosts collections that have been submitted by a variety of Ansible developers and users. Ansible Galaxy is a public library with no formal support guarantees, but which allows public access. For example, the `community.crypto`, `community.postgresql`, and `community.rabbitmq` collections are all available from that platform.

Use the Ansible Galaxy web UI at https://galaxy.ansible.com/ to search it for collections.

Installing Content Collections

Before your playbooks can use content from a collection, you must install that collection on your control node. Use the `ansible-galaxy` command to download collections from a number of possible sources, including Ansible Galaxy.

The following example uses the `ansible-galaxy` command with the `collection` argument to download and then install the `community.crypto` collection on the local system.

```
[user@controlnode ~]$ ansible-galaxy collection install community.crypto
```

The command can also install a collection from a local or a remote tar archive.

```
[user@controlnode ~]$ ansible-galaxy collection install \
> /tmp/community-dns-1.2.0.tar.gz
[user@controlnode ~]$ ansible-galaxy collection install \
> http://www.example.com/redhat-insights-1.0.5.tar.gz
```

The Ansible configuration directive `collections_paths` specifies a colon-separated list of paths on the system where Ansible looks for installed collections. You can set this directive in the `ansible.cfg` configuration file. By default, the `ansible-galaxy` command installs the collections in the first directory that the `collections_paths` directive defines.

The default value for `collections_paths` is `~/.ansible/collections:/usr/share/ansible/collections`. Therefore, the `ansible-galaxy` command installs collections in the `~/.ansible/collections` directory by default.

If you want to install a collection in a different directory, use the `--collections-path` (or `-p`) option.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[root@controlnode ~]# ansible-galaxy collection install \
> -p /usr/share/ansible/collections community.postgresql
```

Important

When using the `--collections-path` (or `-p`) option, ensure you select a directory listed in the `collections_paths` directive. The `ansible-playbook` command also uses that directive to locate collections. If you do not use a path defined in the `collections_paths` directive, then your playbooks will not find the collections that you installed.

Installing Collections with a Requirements File

You can create a `requirements.yml` file to list all the collections that you need to install. By adding a `collections/requirements.yml` file in your Ansible project, your team members can immediately identify the required collections. Also, automation controller detects that file and automatically installs the collections before running your playbooks.

The following `requirements.yml` file lists several collections to install. Notice that you can target a specific collection version and also provide local or remote tar archives.

```
---
collections:
  - name: community.crypto

  - name: ansible.posix
    version: 1.2.0

  - name: /tmp/community-dns-1.2.0.tar.gz

  - name: http://www.example.com/redhat-insights-1.0.5.tar.gz
```

The `ansible-galaxy` command can then use that file to install all those collections. Use the `--requirements-file` (or `-r`) option to provide the `requirements.yml` file to the command.

```
[root@controlnode ~]# ansible-galaxy collection install -r requirements.yml
```

Configuring Collection Sources

By default, the `ansible-galaxy` command uses Ansible Galaxy at https://galaxy.ansible.com/ to download collections.

For the command to also use Ansible automation hub, add the following directives to the `ansible.cfg` file.

```
...output omitted...
[galaxy]
server_list = automation_hub, galaxy   (1)

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/   (2)
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token ❸
token=eyJh...Jf0o ❹

[galaxy_server.galaxy]
url=https://galaxy.ansible.com/
```

❶ List all the repositories that the `ansible-galaxy` command must use in order. For each name you define, add a `[galaxy_server.`*`name`*`]` section to provide the connection parameters. Because Ansible automation hub might not provide all the collections that your playbooks need, you can add Ansible Galaxy in the last position as a fallback. This way, if the collection is not available in Ansible automation hub, then the `ansible-galaxy` command uses Ansible Galaxy to retrieve it.

❷ Provide the URL to access the repository.

❸ Provide the URL for authentication.

❹ To access Ansible automation hub, you need an authentication token associated with your account. Use the Ansible automation hub web UI to generate that token. For more details on that process, see the links in the References section.

Instead of a token, you can use the `username` and `password` parameters to provide your customer portal user name and password.

```
...output omitted...
[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
username=operator1
password=Sup3r53cR3t
...output omitted...
```

You might not want to expose your credentials in the `ansible.cfg` file because the file could potentially get committed using version control. In that case, remove the authentication parameters from the `ansible.cfg` file and define them as environment variables. You define the environment variables as follows:

```
ANSIBLE_GALAXY_SERVER_<server_id>_<key>=value
```

`server_id`
: Server identifier in uppercase. The server identifier is the name you use in the `server_list` parameter and in the name of the `[galaxy_server.`*`server_id`*`]` section.

`key`
: Name of the parameter in uppercase.

The following example provides the `token` parameter as an environment variable:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@controlnode ~]$ cat ansible.cfg
...output omitted...
[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token
[user@controlnode ~]$ export \
> ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN='eyJh...Jf0o'
[user@controlnode ~]$ ansible-galaxy collection install ansible.posix
```

Using Collections

After you install a collection, you can use it with ad hoc commands and playbooks. Access the collection documentation from Ansible automation hub or the Ansible Galaxy web UI to retrieve information about the roles and modules it provides. Alternatively, you can inspect the collection directory structure on your system. The collection stores the modules in the `plugins/modules/` directory and the roles in the `roles/` directory.

```
[user@controlnode ~]$ tree \
> ~/.ansible/collections/ansible_collections/redhat/insights/
/home/user/.ansible/collections/ansible_collections/redhat/insights/
...output omitted...
├── plugins
│   ├── action
│   │   └── insights_config.py
│   ├── inventory
│   │   └── insights.py
│   └── modules
│       ├── insights_config.py
│       └── insights_register.py
...output omitted...
├── roles
│   ├── compliance
│   │   ├── meta
│   │   │   └── main.yml
│   │   ├── README.md
│   │   ├── tasks
│   │   │   ├── install.yml
│   │   │   ├── main.yml
│   │   │   └── run.yml
│   │   └── tests
│   │       ├── compliance.yml
│   │       ├── install-only.yml
│   │       └── run-only.yml
│   └── insights_client
...output omitted...
```

To use a module or a role, refer to it with its *fully qualified collection name (FQCN)*. Based on the preceding output, you refer to the `insights_client` role as `redhat.insights.insights_client`.

The following ad hoc command calls the `mail` module from the `community.general` collection.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[user@controlnode ~]$ ansible localhost -m community.general.mail \
> -a 'subject="Hello World" to=root'
```

The following playbook invokes the `mysql_user` module from the `community.mysql` collection.

```
---
- name: Create the operator1 user in the test database
  hosts: db.example.com

  tasks:
    - name: Ensure the operator1 database user is defined
      community.mysql.mysql_user:
        name: operator1
        password: Secret0451
        priv: '.:ALL'
        state: present
```

The following playbook uses the `organizations` role from the `redhat.satellite` collection.

```
---
- name: Add the test organizations to Red Hat Satellite
  hosts: localhost

  tasks:
    - name: Ensure the organizations exist
      include_role:
        name: redhat.satellite.organizations
      vars:
        satellite_server_url: https://sat.example.com
        satellite_username: admin
        satellite_password: Sup3r53cr3t
        satellite_organizations:
          - name: test1
            label: tst1
            state: present
            description: Test organization 1
          - name: test2
            label: tst2
            state: present
            description: Test organization 2
```

Using Ansible Built-in Collections after Ansible 2.9

In future versions of Ansible, the core installation will always include a special collection named `ansible.builtin`. This collection will include a set of common modules, such as `copy`, `template`, `file`, `yum`, `command`, and `service`.

You will always be able to use the short names of these modules in your playbooks. For example, you will still be able to use `file` instead of `ansible.builtin.file`. This will allow many Ansible 2.9 playbooks to work without modification, although you might need to install additional collections for modules that are not included in `ansible.builtin`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

However, Red Hat recommends that you use the FQCN notation to prevent future conflicts with collections that might use the same module names.

The following playbook uses the FQCN notation for the `yum`, `copy`, and `service` modules.

```
---
- name: Install and start Apache HTTPD
  hosts: web

  tasks:
    - name: Ensure the httpd package is present
      ansible.builtin.yum:
        name: httpd
        state: present

    - name: Ensure the index.html file is present
      ansible.builtin.copy:
        src: files/index.html
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: 0644
        setype: httpd_sys_content_t

    - name: Ensure the httpd service is started
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: true
```

References

Using collections – Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/collections_using.html

Galaxy User Guide – Ansible Documentation
https://docs.ansible.com/ansible/2.9/galaxy/user_guide.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Getting Roles and Modules from Content Collections

In this exercise, you will install a content collection and use a role or module from that content collection in a playbook.

Outcomes

You should be able to:

- Use the `ansible-galaxy` command to install a content collection.
- Use a `requirements.yml` file to install multiple collections.
- Invoke content collections roles and modules from playbooks.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-collections start` command. This command creates the working directory, `/home/student/role-collections`, and populates it with an Ansible project.

```
[student@workstation ~]$ lab role-collections start
```

Instructions

1. Install and then inspect the `gls.utils` collection.

1.1. Install the `gls.utils` collection from the tar file at `http://materials.example.com/labs/role-collections/gls-utils-0.0.1.tar.gz`. You can copy and paste that URL from the `/home/student/role-collections/url.txt` file.

```
[student@workstation ~]$ ansible-galaxy collection install \
> http://materials.example.com/labs/role-collections/gls-utils-0.0.1.tar.gz
Process install dependency map
Starting collection install process
Installing 'gls.utils:0.0.1' to '/home/student/.ansible/collections/
ansible_collections/gls/utils'
```

Notice that the preceding command installs the collection in the `/home/student/.ansible/collections/ansible_collections/gls/utils` directory.

1.2. List the roles that the collection provides.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ ls \
> ~/.ansible/collections/ansible_collections/gls/utils/roles
backup  restore
```

From the preceding output, notice that the collection provides two roles: `backup` and `restore`.

1.3. Each role provides a `README.md` file. Consult the `README.md` file for the `backup` role.

```
[student@workstation ~]$ cat \
> ~/.ansible/collections/ansible_collections/gls/utils/roles/backup/README.md
...output omitted...
```

1.4. List the modules that the collection provides.

```
[student@workstation ~]$ ls \
> ~/.ansible/collections/ansible_collections/gls/utils/plugins/modules
newping.py
```

The collection provides the `newping` module.

1.5. Use the `ansible-doc` command to consult the `newping` module documentation.

```
[student@workstation ~]$ ansible-doc gls.utils.newping
...output omitted...
```

2. Complete and then run the `/home/student/role-collections/bck.yml` playbook. That playbook uses the `gls.utils.newping` module and the `gls.utils.backup` role.

2.1. Change to the `role-collections` working directory.

```
[student@workstation ~]$ cd ~/role-collections
[student@workstation role-collections]$
```

2.2. Edit the `bck.yml` playbook. In the first task, invoke the `gls.utils.newping` module.

```
...output omitted...
  tasks:
    - name: Ensure the machine is up
      gls.utils.newping:
        data: pong
...output omitted...
```

Do not close the file yet.

2.3. In the second task, invoke the `gls.utils.backup` role. When done, save and close the file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
...output omitted...
    - name: Ensure configuration files are saved
      include_role:
        name: gls.utils.backup
      vars:
        backup_id: backup_etc
        backup_files:
          - /etc/sysconfig
          - /etc/yum.repos.d
...output omitted...
```

The resulting file should display as follows:

```
---
- name: Backup the system configuration
  hosts: servera.lab.example.com
  become: true
  gather_facts: false

  tasks:
    - name: Ensure the machine is up
      gls.utils.newping:
        data: pong

    - name: Ensure configuration files are saved
      include_role:
        name: gls.utils.backup
      vars:
        backup_id: backup_etc
        backup_files:
          - /etc/sysconfig
          - /etc/yum.repos.d
```

2.4. Verify the syntax of the `bck.yml` playbook.

```
[student@workstation role-collections]$ ansible-playbook --syntax-check bck.yml

playbook: bck.yml
```

2.5. Run the playbook.

```
[student@workstation role-collections]$ ansible-playbook bck.yml
...output omitted...
```

3. In the second part of this exercise, install content collections specified by a `requirements.yml` file.

To test your work when done, run the `new_system.yml` playbook. That playbook uses the `redhat.insights.insights_client` and `redhat.rhel_system_roles.selinux` roles to configure Red Hat Insights and SELinux on the `servera` machine.

3.1. Review the `requirements.yml` file. The file lists two collections to install from tar files hosted on the `materials.example.com` web server.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
collections:
  - name: http://materials.example.com/labs/role-collections/redhat-
insights-1.0.5.tar.gz
  - name: http://materials.example.com/labs/role-collections/redhat-
rhel_system_roles-1.0.1.tar.gz
```

3.2. Use the `ansible-galaxy` command with the `requirements.yml` file to install the collections.

```
[student@workstation role-collections]$ ansible-galaxy collection install \
> -r requirements.yml
Process install dependency map
Starting collection install process
Installing 'redhat.insights:1.0.5' to '/home/student/.ansible/collections/
ansible_collections/redhat/insights'
Installing 'redhat.rhel_system_roles:1.0.1' to '/home/student/.ansible/
collections/ansible_collections/redhat/rhel_system_roles'
```

3.3. Review the `new_system.yml` playbook.

```
---
- name: Configure the system
  hosts: servera.lab.example.com
  become: true
  gather_facts: true

  tasks:
    - name: Ensure the system is registered with Insights
      include_role:
        name: redhat.insights.insights_client
      vars:
        auto_config: false
        insights_proxy: http://proxy.example.com:8080

    - name: Ensure SELinux mode is Enforcing
      include_role:
        name: redhat.rhel_system_roles.selinux
      vars:
        selinux_state: enforcing
```

3.4. Run the `new_system.yml` playbook in check mode to confirm that you correctly installed the required collections. Because the classroom systems are not registered with Red Hat and might not have internet access, the `new_system.yml` playbook cannot complete successfully. However, to confirm that you correctly installed the required collections, you can still run the playbook in check mode.

```
[student@workstation role-collections]$ ansible-playbook --check new_system.yml
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finish

Run the `lab role-collections finish` command to clean up the managed host.

```
[student@workstation ~]$ lab role-collections finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Simplifying Playbooks with Roles

Performance Checklist

In this lab, you will create Ansible roles that use variables, files, templates, tasks, and handlers.

Outcomes

You should be able to:

- Create Ansible roles that use variables, files, templates, tasks, and handlers to configure a development web server.
- Use a role that is hosted in a remote repository in a playbook.
- Use a Red Hat Enterprise Linux system role in a playbook.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-review start` command. The script creates the working directory, `/home/student/role-review`, and populates it with an Ansible configuration file, host inventory, and other lab files.

```
[student@workstation ~]$ lab role-review start
```

Instructions

Your organization must provide a single web server to host development code for all web developers. You are tasked with writing a playbook to configure this development web server.

The development web server must satisfy several requirements:

- The development server configuration matches the production server configuration. The production server is configured using an Ansible role, developed by the organization's infrastructure team.
- Each developer is given a directory on the development server to host code and content. Each developer's content is accessed using an assigned, nonstandard port.
- SELinux is set to enforcing and targeted.

Your playbook will:

- Use a role to configure directories and ports for each developer on the web server. You must write this role.

  This role has a dependency on a role written by the organization to configure Apache. You should define the dependency using version `v1.4` of the organizational role. The URL of the dependency's repository is: `git@workstation.lab.example.com:infra/apache`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Use the `rhel-system-roles.selinux` role to configure SELinux for the nonstandard HTTP ports used by your web server. You will be provided with a `selinux.yml` variable file that can be installed as a `group_vars` file to pass the correct settings to the role.

1. Change to the `/home/student/role-review` working directory.
2. Create a playbook named `web_dev_server.yml` with a single play named `Configure Dev Web Server`. Configure the play to target the host group `dev_webserver`. Do not add any roles or tasks to the play yet.

   Ensure that the play forces handlers to execute, because you may encounter an error while developing the playbook.
3. Check the syntax of the playbook. Run the playbook. The syntax check should pass and the playbook should run successfully.
4. Make sure that playbook's role dependencies are installed.

   The `apache.developer_configs` role that you will create depends on the `infra.apache` role. Create a `roles/requirements.yml` file. It should install the role from the Git repository at `git@workstation.lab.example.com:infra/apache`, use version `v1.4`, and name it `infra.apache` locally. You can assume that your SSH keys are configured to allow you to get roles from that repository automatically. Install the role with the `ansible-galaxy` command.

   In addition, install the *rhel-system-roles* package if not present.
5. Initialize a new role named `apache.developer_configs` in the `roles` subdirectory.

   Add the `infra.apache` role as a dependency for the new role, using the same information for name, source, version, and version control system as the `roles/requirements.yml` file.

   The `developer_tasks.yml` file in the project directory contains tasks for the role. Move this file to the correct location to be the tasks file for this role, and replace the existing file in that location.

   The `developer.conf.j2` file in the project directory is a Jinja2 template used by the tasks file. Move it to the correct location for template files used by this role.
6. The `apache.developer_configs` role will process a list of users defined in a variable named `web_developers`. The `web_developers.yml` file in the project directory defines the `web_developers` user list variable. Review this file and use it to define the `web_developers` variable for the development web server host group.
7. Add the role `apache.developer_configs` to the play in the `web_dev_server.yml` playbook.
8. Check the syntax of the playbook. Run the playbook. The syntax check should pass, but the playbook should fail when the `infra.apache` role attempts to restart Apache HTTPD.
9. Apache HTTPD failed to restart in the preceding step because the network ports it uses for your developers are labeled with the wrong SELinux contexts. You have been provided with a variable file, `selinux.yml`, which can be used with the `rhel-system-roles.selinux` role to fix the issue.

   Create a `pre_tasks` section for your play in the `web_dev_server.yml` playbook. In that section, use a task to include the `rhel-system-roles.selinux` role in a `block/rescue` structure so that it is properly applied. Review the lecture or the documentation for this role to see how to do this.

   Inspect the `selinux.yml` file. Move it to the correct location so that its variables are set for the `dev_webserver` host group.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

10. Verify the final `web_dev_server.yml` playbook and run a syntax check. The syntax check should pass.

    Validate that the `web_dev_server.yml` playbook passes a syntax check.

11. Run the playbook. It should succeed.

12. Test the configuration of the development web server. Verify that all endpoints are accessible and serving each developer's content.

Evaluation

Grade your work by running the `lab role-review grade` command from your `workstation` machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab role-review grade
```

Finish

On `workstation`, run the `lab role-review finish` script to clean up this exercise.

```
[student@workstation ~]$ lab role-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Simplifying Playbooks with Roles

Performance Checklist

In this lab, you will create Ansible roles that use variables, files, templates, tasks, and handlers.

Outcomes

You should be able to:

- Create Ansible roles that use variables, files, templates, tasks, and handlers to configure a development web server.
- Use a role that is hosted in a remote repository in a playbook.
- Use a Red Hat Enterprise Linux system role in a playbook.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab role-review start` command. The script creates the working directory, `/home/student/role-review`, and populates it with an Ansible configuration file, host inventory, and other lab files.

```
[student@workstation ~]$ lab role-review start
```

Instructions

Your organization must provide a single web server to host development code for all web developers. You are tasked with writing a playbook to configure this development web server.

The development web server must satisfy several requirements:

- The development server configuration matches the production server configuration. The production server is configured using an Ansible role, developed by the organization's infrastructure team.
- Each developer is given a directory on the development server to host code and content. Each developer's content is accessed using an assigned, nonstandard port.
- SELinux is set to enforcing and targeted.

Your playbook will:

- Use a role to configure directories and ports for each developer on the web server. You must write this role.

  This role has a dependency on a role written by the organization to configure Apache. You should define the dependency using version `v1.4` of the organizational role. The URL of the dependency's repository is: `git@workstation.lab.example.com:infra/apache`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Use the `rhel-system-roles.selinux` role to configure SELinux for the nonstandard HTTP ports used by your web server. You will be provided with a `selinux.yml` variable file that can be installed as a `group_vars` file to pass the correct settings to the role.

1. Change to the `/home/student/role-review` working directory.

```
[student@workstation ~]$ cd ~/role-review
[student@workstation role-review]$
```

2. Create a playbook named `web_dev_server.yml` with a single play named `Configure Dev Web Server`. Configure the play to target the host group `dev_webserver`. Do not add any roles or tasks to the play yet.

   Ensure that the play forces handlers to execute, because you may encounter an error while developing the playbook.

   Once complete, the `/home/student/role-review/web_dev_server.yml` playbook contains:

```
---
- name: Configure Dev Web Server
  hosts: dev_webserver
  force_handlers: yes
```

3. Check the syntax of the playbook. Run the playbook. The syntax check should pass and the playbook should run successfully.

```
[student@workstation role-review]$ ansible-playbook \
> --syntax-check web_dev_server.yml

playbook: web_dev_server.yml
[student@workstation role-review]$ ansible-playbook web_dev_server.yml
PLAY [Configure Dev Web Server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=1    changed=0    unreachable=0    failed=0
```

4. Make sure that playbook's role dependencies are installed.

   The `apache.developer_configs` role that you will create depends on the `infra.apache` role. Create a `roles/requirements.yml` file. It should install the role from the Git repository at `git@workstation.lab.example.com:infra/apache`, use version `v1.4`, and name it `infra.apache` locally. You can assume that your SSH keys are configured to allow you to get roles from that repository automatically. Install the role with the `ansible-galaxy` command.

   In addition, install the *rhel-system-roles* package if not present.

   4.1. Create a `roles` subdirectory for the playbook project.

```
[student@workstation role-review]$ mkdir -v roles
mkdir: created directory 'roles'
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4.2. Create a `roles/requirements.yml` file and add an entry for the `infra.apache` role. Use version `v1.4` from the role's git repository.

Once complete, the `roles/requirements.yml` file contains:

```
- name: infra.apache
  src: git@workstation.lab.example.com:infra/apache
  scm: git
  version: v1.4
```

4.3. Install the project dependencies.

```
[student@workstation role-review]$ ansible-galaxy install \
> -r roles/requirements.yml -p roles
- extracting infra.apache to /home/student/role-review/roles/infra.apache
- infra.apache (v1.4) was installed successfully
```

4.4. Install the RHEL System Roles package if not present. This was installed during an earlier exercise.

```
[student@workstation role-review]$ sudo yum install rhel-system-roles
```

5. Initialize a new role named `apache.developer_configs` in the `roles` subdirectory.

Add the `infra.apache` role as a dependency for the new role, using the same information for name, source, version, and version control system as the `roles/requirements.yml` file.

The `developer_tasks.yml` file in the project directory contains tasks for the role. Move this file to the correct location to be the tasks file for this role, and replace the existing file in that location.

The `developer.conf.j2` file in the project directory is a Jinja2 template used by the tasks file. Move it to the correct location for template files used by this role.

5.1. Use the `ansible-galaxy init` to create a role skeleton for the `apache.developer_configs` role.

```
[student@workstation role-review]$ cd roles
[student@workstation roles]$ ansible-galaxy init apache.developer_configs
- apache.developer_configs was created successfully
[student@workstation roles]$ cd ..
[student@workstation role-review]$
```

5.2. Update the `roles/apache.developer_configs/meta/main.yml` file of the `apache.developer_configs` role to reflect a dependency on the `infra.apache` role.

After editing, the `dependencies` variable is defined as follows:

```
dependencies:
  - name: infra.apache
    src: git@workstation.lab.example.com:infra/apache
    scm: git
    version: v1.4
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5.3. Overwrite the role's `tasks/main.yml` file with the `developer_tasks.yml` file.

```
[student@workstation role-review]$ mv -v developer_tasks.yml \
> roles/apache.developer_configs/tasks/main.yml
renamed 'developer_tasks.yml' -> 'roles/apache.developer_configs/tasks/main.yml'
```

5.4. Place the `developer.conf.j2` file in the role's `templates` directory.

```
[student@workstation role-review]$ mv -v developer.conf.j2 \
> roles/apache.developer_configs/templates/
renamed 'developer.conf.j2' -> 'roles/apache.developer_configs/templates/
developer.conf.j2'
```

6. The `apache.developer_configs` role will process a list of users defined in a variable named `web_developers`. The `web_developers.yml` file in the project directory defines the `web_developers` user list variable. Review this file and use it to define the `web_developers` variable for the development web server host group.

6.1. Review the `web_developers.yml` file.

```
---
web_developers:
  - username: jdoe
    name: John Doe
    user_port: 9081
  - username: jdoe2
    name: Jane Doe
    user_port: 9082
```

A `name`, `username`, `user_port` is defined for each web developer.

6.2. Place the `web_developers.yml` in the `group_vars/dev_webserver` subdirectory.

```
[student@workstation role-review]$ mkdir -pv group_vars/dev_webserver
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/dev_webserver'
[student@workstation role-review]$ mv -v web_developers.yml \
> group_vars/dev_webserver/
renamed 'web_developers.yml' -> 'group_vars/dev_webserver/web_developers.yml'
```

7. Add the role `apache.developer_configs` to the play in the `web_dev_server.yml` playbook.

The edited playbook:

```
---
- name: Configure Dev Web Server
  hosts: dev_webserver
  force_handlers: yes
  roles:
    - apache.developer_configs
```

8. Check the syntax of the playbook. Run the playbook. The syntax check should pass, but the playbook should fail when the `infra.apache` role attempts to restart Apache HTTPD.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation role-review]$ ansible-playbook \
> --syntax-check web_dev_server.yml

playbook: web_dev_server.yml
[student@workstation role-review]$ ansible-playbook web_dev_server.yml

PLAY [Configure Dev Web Server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

...output omitted...

TASK [infra.apache : Install a skeleton index.html] ****************************
skipping: [servera.lab.example.com]

TASK [apache.developer_configs : Create user accounts] *************************
changed: [servera.lab.example.com] => (item={u'username': u'jdoe', u'user_port':
 9081, u'name': u'John Doe'})
changed: [servera.lab.example.com] => (item={u'username': u'jdoe2', u'user_port':
 9082, u'name': u'Jane Doe'})

...output omitted...

RUNNING HANDLER [infra.apache : restart firewalld] *****************************
changed: [servera.lab.example.com]

RUNNING HANDLER [infra.apache : restart apache] ********************************
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "msg": "Unable to
 restart service httpd: Job for httpd.service failed because the control process
 exited with error code. See \"systemctl status httpd.service\" and \"journalctl -
xe\" for details.\n"}

NO MORE HOSTS LEFT *************************************************************
  to retry, use: --limit @/home/student/role-review/web_dev_server.retry

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=13   changed=11   unreachable=0    failed=1
skipped=1    rescued=0    ignored=0
```

An error occurs when the `httpd` service is restarted. The `httpd` service daemon cannot bind to the non-standard HTTP ports, due to the SELinux context on those ports.

9. Apache HTTPD failed to restart in the preceding step because the network ports it uses for your developers are labeled with the wrong SELinux contexts. You have been provided with a variable file, `selinux.yml`, which can be used with the `rhel-system-roles.selinux` role to fix the issue.

   Create a `pre_tasks` section for your play in the `web_dev_server.yml` playbook. In that section, use a task to include the `rhel-system-roles.selinux` role in a `block/rescue` structure so that it is properly applied. Review the lecture or the documentation for this role to see how to do this.

   Inspect the `selinux.yml` file. Move it to the correct location so that its variables are set for the `dev_webserver` host group.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

9.1. The `pre_tasks` section can be added to the end of the play in the `web_dev_server.yml` playbook.

You can look at the block in `/usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml` for a basic outline of how to apply the role. Replace the complex `shell` and `wait_for_connection` logic with the `reboot` module.

The `pre_tasks` section should contain:

```
  pre_tasks:
    - name: Check SELinux configuration
      block:
        - include_role:
            name: rhel-system-roles.selinux
      rescue:
        # Fail if failed for a different reason than selinux_reboot_required.
        - name: Check for general failure
          fail:
            msg: "SELinux role failed."
          when: not selinux_reboot_required

        - name: Restart managed host
          reboot:
            msg: "Ansible rebooting system for updates."

        - name: Reapply SELinux role to complete changes
          include_role:
            name: rhel-system-roles.selinux
```

9.2. The `selinux.yml` file contains variable definitions for the `rhel-system-roles.selinux` role. Use the file to define variables for the play's host group.

```
[student@workstation role-review]$ cat selinux.yml
---
# variables used by rhel-system-roles.selinux

selinux_policy: targeted
selinux_state: enforcing

selinux_ports:
  - ports:
      - "9081"
      - "9082"
    proto: 'tcp'
    setype: 'http_port_t'
    state: 'present'

[student@workstation role-review]$ mv -v selinux.yml \
> group_vars/dev_webserver/
renamed 'selinux.yml' -> 'group_vars/dev_webserver/selinux.yml'
```

10. Verify the final `web_dev_server.yml` playbook and run a syntax check. The syntax check should pass.

The final `web_dev_server.yml` playbook should read as follows:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Configure Dev Web Server
  hosts: dev_webserver
  force_handlers: yes
  roles:
    - apache.developer_configs
  pre_tasks:
    - name: Check SELinux configuration
      block:
        - include_role:
            name: rhel-system-roles.selinux
      rescue:
        # Fail if failed for a different reason than selinux_reboot_required.
        - name: Check for general failure
          fail:
            msg: "SELinux role failed."
          when: not selinux_reboot_required

        - name: Restart managed host
          reboot:
            msg: "Ansible rebooting system for updates."

        - name: Reapply SELinux role to complete changes
          include_role:
            name: rhel-system-roles.selinux
```

Note

Whether `pre_tasks` is at the end of the play or in the "correct" position in terms of execution order in the playbook file does not matter to `ansible-playbook`. It will still run the play's tasks in the correct order.

Validate that the `web_dev_server.yml` playbook passes a syntax check.

```
[student@workstation role-review]$ ansible-playbook \
> --syntax-check web_dev_server.yml

playbook: web_dev_server.yml
```

11. Run the playbook. It should succeed.

```
[student@workstation role-review]$ ansible-playbook web_dev_server.yml

PLAY [Configure Dev Web Server] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [include_role : rhel-system-roles.selinux] ********************************

TASK [rhel-system-roles.selinux : Install SELinux python3 tools] ***************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
ok: [servera.lab.example.com]

...output omitted...

TASK [infra.apache : Apache Service is started] ****************************
changed: [servera.lab.example.com]

...output omitted...

TASK [apache.developer_configs : Copy Per-Developer Config files] ************
ok: [servera.lab.example.com] => (item={'username': 'jdoe', 'name': 'John Doe',
 'user_port': 9081})
ok: [servera.lab.example.com] => (item={'username': 'jdoe2', 'name': 'Jane Doe',
 'user_port': 9082})

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=19   changed=3    unreachable=0    failed=0
skipped=14   rescued=0    ignored=0
```

12. Test the configuration of the development web server. Verify that all endpoints are accessible and serving each developer's content.

```
[student@workstation role-review]$ curl servera
This is the production server on servera.lab.example.com
[student@workstation role-review]$ curl servera:9081
This is index.html for user: John Doe (jdoe)
[student@workstation role-review]$ curl servera:9082
This is index.html for user: Jane Doe (jdoe2)
[student@workstation role-review]$
```

Evaluation

Grade your work by running the `lab role-review grade` command from your `workstation` machine. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab role-review grade
```

Finish

On `workstation`, run the `lab role-review finish` script to clean up this exercise.

```
[student@workstation ~]$ lab role-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

In this chapter, you learned:

- Roles organize Ansible code in a way that allows reuse and sharing.
- Red Hat Enterprise Linux System Roles are a collection of tested and supported roles intended to help you configure host subsystems across versions of Red Hat Enterprise Linux.
- 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.
- External roles needed by a playbook may be defined in the `roles/requirements.yml` file. The `ansible-galaxy install -r roles/requirements.yml` command uses this file to install the roles on the control node.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 8

Troubleshooting Ansible

Goal	Troubleshoot playbooks and managed hosts.
Objectives	• Troubleshoot generic issues with a new playbook and repair them. • Troubleshoot failures on managed hosts when running a playbook.
Sections	• Troubleshooting Playbooks (and Guided Exercise) • Troubleshooting Ansible Managed Hosts (and Guided Exercise)
Lab	• Troubleshooting Ansible

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Troubleshooting Playbooks

Objectives

After completing this section, you should be able to troubleshoot generic issues with a new playbook and repair them.

Log Files for 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 `$ANSIBLE_LOG_PATH` environment variable. If any or both are configured, Ansible stores output from both the `ansible` and `ansible-playbook` commands in the log file configured, either through the `ansible.cfg` configuration file or the `$ANSIBLE_LOG_PATH` environment variable.

If you configure Ansible to write log files to `/var/log`, then Red Hat recommends that you configure `logrotate` to manage the Ansible log files.

The Debug Module

The `debug` module provides insight into what is happening in the play. This module can display the value for a certain variable at a certain point in the play. This feature is key to debugging 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 use the `msg` and `var` settings inside of `debug` tasks. The first example displays the value at run time of the `ansible_facts['memfree_mb']` fact as part of a message printed to the output of `ansible-playbook`. The second example displays the value of the `output` variable.

```
- name: Display free memory
  debug:
    msg: "Free memory for this system is {{ ansible_facts['memfree_mb'] }}"
```

```
- name: Display the "output" variable
  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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Earlier in this course, you learned about the `--syntax-check` option, which checks the YAML syntax for the playbook. It is a good practice to run a syntax check on your playbook before using it or if you are having problems with it.

```
[student@demo ~]$ ansible-playbook play.yml --syntax-check
```

You can also use the `--step` option to step through a playbook one task at a time. The `ansible-playbook --step` command interactively prompts for confirmation that you want each task to run.

```
[student@demo ~]$ ansible-playbook play.yml --step
```

The `--start-at-task` option allows you to start execution of a playbook from a specific task. It takes as an argument the name of the task at which to start.

```
[student@demo ~]$ ansible-playbook play.yml --start-at-task="start httpd service"
```

Debugging

The output given by a playbook that was run with the `ansible-playbook` command is a good starting point for troubleshooting issues related to hosts managed by Ansible. Consider the following output from a playbook execution:

```
PLAY [Service Deployment] ******************************************************
...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, followed by one or more TASK headers. Each of these headers represents their associated *task* in the playbook, and it is executed in all the managed hosts belonging to the group included in the playbook in the *hosts* parameter.

As each managed host executes each play's tasks, the name of the managed host is displayed under the corresponding TASK header, along with the task state on that managed host. Task states can appear as `ok`, `fatal`, `changed`, or `skipping`.

At the bottom of the output for each play, the PLAY RECAP section displays the number of tasks executed for each managed host.

As discussed earlier in the course, you can increase the verbosity of the output from `ansible-playbook` by adding one or more `-v` options. The `ansible-playbook -v` command provides additional debugging information, with up to four total levels.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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`	Includes additional information such scripts that are executed on each remote host, and the user that is executing each script.

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 troubleshooting process. Some recommended practices for playbook development are listed below:

- Use a concise description of the play's or task's purpose to name plays and tasks. The play name or task name is displayed when the playbook is executed. This also helps document what each play or task is supposed to accomplish, and possibly why it is needed.
- Include comments to add additional inline documentation about tasks.
- Make effective use of vertical white space. In general, organize task attributes vertically to make them easier to read.
- Consistent horizontal indentation is critical. Use spaces, not tabs, to avoid indentation errors. Set up your text editor to insert spaces when you press the `Tab` key to make this easier.
- Try to keep the playbook as simple as possible. Only use the features that you need.

References

Configuring Ansible — Ansible Documentation
https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html

debug — Print statements during execution — Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/debug_module.html

Best Practices — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Troubleshooting Playbooks

In this exercise, you will troubleshoot a playbook that has been given to you that does not work properly.

Outcomes

You should be able to troubleshoot and resolve issues in playbooks.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab troubleshoot-playbook start` script. It verifies whether Ansible is installed on `workstation`. It also creates the `/home/student/troubleshoot-playbook/` directory, and downloads to this directory the `inventory`, `samba.yml`, and `samba.conf.j2` files from `http://materials.example.com/labs/troubleshoot-playbook/`.

```
[student@workstation ~]$ lab troubleshoot-playbook start
```

Instructions

1. On `workstation`, change to the `/home/student/troubleshoot-playbook/` directory.

```
[student@workstation ~]$ cd ~/troubleshoot-playbook/
[student@workstation troubleshoot-playbook]$
```

2. Create a file named `ansible.cfg` in the current directory. It should set the `log_path` parameter to write Ansible logs to the `/home/student/troubleshoot-playbook/ansible.log` file. It should set the `inventory` parameter to use the `/home/student/troubleshoot-playbook/inventory` file deployed by the lab script.

   When you are finished, `ansible.cfg` should have the following contents:

```
[defaults]
log_path = /home/student/troubleshoot-playbook/ansible.log
inventory = /home/student/troubleshoot-playbook/inventory
```

3. Run the playbook. It will fail with an error.

   This playbook would set up a Samba server if everything were correct. However, the run will fail due to missing double quotes on the `random_var` variable definition. Read the error message to see how `ansible-playbook` reports the problem. Notice the variable `random_var` is assigned a value that contains a colon and is not quoted.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to be in '/home/student/troubleshoot-playbook/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
```

4. Confirm that the error has been properly logged to the `/home/student/troubleshoot-playbook/ansible.log` file.

```
[student@workstation troubleshoot-playbook]$ tail ansible.log

The error appears to be in '/home/student/troubleshoot-playbook/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 `samba.yml` playbook and correct the error by adding quotes to the entire value being assigned to `random_var`. The corrected version of the playbook contains the following content:

```
...output omitted...
  vars:
    install_state: installed
    random_var: "This is colon: test"
...output omitted...
```

6. Check the playbook using the `--syntax-check` option. Another error is issued due to extra white space in the indentation on the last task, `deliver samba config`.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
Syntax Error while loading YAML.
  did not find expected key

The error appears to be in '/home/student/troubleshoot-playbook/samba.yml': line
 44, 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 should appear as follows:

```
...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
      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 troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  found unacceptable key (unhashable type: 'AnsibleMapping')

The error appears to be in '/home/student/troubleshoot-playbook/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
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
      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 troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml

playbook: samba.yml
```

11. Run the playbook. An error, related to SSH, will be issued.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
PLAY [Install a samba server] **************************************************

TASK [Gathering Facts] *********************************************************
fatal: [servera.lab.exammple.com]: UNREACHABLE! => {"changed": false,
 "msg": "Failed to connect to the host via ssh: ssh: connect to host
 servera.lab.exammple.com port 22: Connection timed out", "unreachable": true}

PLAY RECAP *********************************************************************
servera.lab.exammple.com   : ok=0    changed=0    unreachable=1    failed=0  ...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

12. Ensure the managed host `servera.lab.example.com` is running, using the `ping` command.

```
[student@workstation troubleshoot-playbook]$ 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 off again when you have finished.

```
[student@workstation troubleshoot-playbook]$ ssh devops@servera.lab.example.com
Activate the web console with: systemctl enable --now cockpit.socket
...output omitted...
[devops@servera ~]$ exit
logout
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 troubleshoot-playbook]$ ansible-playbook -vvvv samba.yml
...output omitted...

PLAYBOOK: samba.yml ************************************************
1 plays in samba.yml

PLAY [Install a samba server] **************************************

TASK [Gathering Facts] *********************************************
task path: /home/student/troubleshoot-playbook/samba.yml:2
<servera.lab.exammple.com> ESTABLISH SSH CONNECTION FOR USER: devops
...output omitted...
fatal: [servera.lab.exammple.com]: UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: OpenSSH_8.0p1, OpenSSL ...
 Control socket \"/home/student/.ansible/cp/d4775f48c9\" does not exist\r\ndebug2:
 resolving \"servera.lab.exammple.com\" port 22\r\ndebug2: ssh_connect_direct
\r\ndebug1: Connecting to servera.lab.exammple.com [3.223.115.185] port 22.\r
\ndebug2: fd 4 setting O_NONBLOCK\r\ndebug1: connect to address 3.223.115.185 port
 22: Connection timed out\r\nssh: connect to host servera.lab.exammple.com port
 22: Connection timed out",
    "unreachable": true
}
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
...output omitted...
PLAY RECAP *********************************************************************
servera.lab.exammple.com    : ok=0    changed=0    unreachable=1    failed=0
```

15. When using the highest level of verbosity with Ansible, examining the Ansible log file is a better option than checking console output. You might view the log file using the `less` command, or you might search for patterns in the log file using the `grep` command. Search for the word `fatal` in the `/home/student/troubleshoot-playbook/ansible.log` file.

```
[student@workstation troubleshoot-playbook]$ grep -i fatal ansible.log
2021-07-15 13:56:21,766 p=45752 u=student n=ansible | fatal:
 [servera.lab.exammple.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to
 connect to the host via ssh: ssh: connect to host servera.lab.exammple.com port
 22: Connection timed out", "unreachable": true}
2021-07-15 14:22:43,262 p=46055 u=student n=ansible | fatal:
 [servera.lab.exammple.com]: UNREACHABLE! => {
```

16. Investigate the `inventory` file for errors. Notice the `[samba_servers]` group has misspelled `servera.lab.example.com`. Correct this error as shown below:

```
[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, because no `samba.j2` file is available in the working directory, `/home/student/troubleshoot-playbook/`.

```
[student@workstation troubleshoot-playbook]$ 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, "msg": "Could not
 find or access 'samba.j2'\nSearched in:\n\t/home/student/troubleshoot-playbook/
templates/samba.j2\n\t/home/student/troubleshoot-playbook/samba.j2\n\t/home/
student/troubleshoot-playbook/templates/samba.j2\n\t/home/student/troubleshoot-
playbook/samba.j2 on the Ansible Controller.\nIf you are using a module and expect
 the file to exist on the remote, see the remote_src option"}
...output omitted...
PLAY RECAP *********************************************************************
servera.lab.example.com     : ok=7    changed=3    unreachable=0    failed=1   ...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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 troubleshoot-playbook]$ ansible-playbook samba.yml --step

PLAY [Install a samba server] ***************************************************
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: install samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: install firewalld (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: debug install_state variable (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: start samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: start firewalld (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: configure firewall for samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: deliver samba config (N)o/(y)es/(c)ontinue: y
...output omitted...
PLAY RECAP **********************************************************************
servera.lab.example.com    : ok=8    changed=1    unreachable=0    failed=0
```

Finish

On `workstation`, run the `lab troubleshoot-playbook finish` script to clean up this exercise.

```
[student@workstation ~]$ lab troubleshoot-playbook finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Troubleshooting Ansible Managed Hosts

Objectives

After completing this section, you should be able to troubleshoot failures on managed hosts when running a playbook.

Using 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 changes to the managed hosts' configuration. If a module used within the playbook supports *check mode* then the changes that would have been made to the managed hosts are displayed but not performed. If check mode is not supported by a module then the changes are not displayed but the module still takes no action.

```
[student@demo ~]$ ansible-playbook --check playbook.yml
```

Note

The `ansible-playbook --check` command might not work properly if your tasks use conditionals.

You can also control whether individual tasks run in check mode with the `check_mode` setting. If a task has `check_mode: yes` set, it always runs in check mode, whether or not you passed the `--check` option to `ansible-playbook`. Likewise, if a task has `check_mode: no` set, it always runs normally, even if you pass `--check` to `ansible-playbook`.

The following task is always run in check mode, and does not make changes.

```
tasks:
  - name: task always in check mode
    shell: uname -a
    check_mode: yes
```

The following task is always run normally, even when started with `ansible-playbook --check`.

```
tasks:
  - name: task always runs even in check mode
    shell: uname -a
    check_mode: no
```

This can be useful because you can run most of a playbook normally while testing individual tasks with `check_mode: yes`. Likewise, you can make test runs in check mode more likely to provide reasonable results by running selected tasks that gather facts or set variables for conditionals but do not change the managed hosts with `check_mode: no`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

A task can determine if the playbook is running in check mode by testing the value of the magic variable `ansible_check_mode`. This Boolean variable is set to `true` if the playbook is running in check mode.

Warning

Tasks that have `check_mode: no` set will run even when the playbook is run with `ansible-playbook --check`. Therefore, you cannot trust that the `--check` option will make no changes to managed hosts, without confirming this to be the case by inspecting the playbook and any roles or tasks associated with it.

Note

If you have older playbooks that use `always_run: yes` to force tasks to run normally even in check mode, you will have to replace that code with `check_mode: no` in Ansible 2.6 and later.

The `ansible-playbook` command also provides a `--diff` option. This option reports the changes made to the template files on managed hosts. If used with the `--check` option, those changes are displayed in the command's output but not actually made.

```
[student@demo ~]$ ansible-playbook --check --diff playbook.yml
```

Testing with Modules

Some modules can provide additional information about the status of a managed host. 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:
  - 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 executing a script on managed hosts, and fails if the return code for that script is nonzero. The script must exist on the control node and is transferred to and executed on the managed hosts.

```
tasks:
  - script: check_free_memory
```

- The `stat` module gathers facts for a file much like the `stat` command. You can use it to register a variable and then test to determine if the file exists or to get other information about

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

the file. If the file does not exist, the `stat` task will not fail, but its registered variable will report `false` for `*.stat.exists`.

In this example, an application is still running if `/var/run/app.lock` exists, in which case the play should abort.

```
tasks:
  - name: Check if /var/run/app.lock exists
    stat:
      path: /var/run/app.lock
    register: lock

  - name: Fail if the application is running
    fail:
    when: lock.stat.exists
```

- The `assert` module is an alternative to the `fail` module. The `assert` module supports a `that` option that takes a list of conditionals. If any of those conditionals are false, the task fails. You can use the `success_msg` and `fail_msg` options to customize the message it prints if it reports success or failure.

  The following example repeats the preceding one, but uses `assert` instead of `fail`.

```
tasks:
  - name: Check if /var/run/app.lock exists
    stat:
      path: /var/run/app.lock
    register: lock

  - name: Fail if the application is running
    assert:
      that:
        - not lock.stat.exists
```

Troubleshooting Connections

Many common problems when using Ansible to manage hosts are associated with connections to the host and with configuration problems around the remote user and privilege escalation.

If you are having problems authenticating to a managed host, make sure that you have `remote_user` set correctly in your configuration file or in your play. You should also confirm that you have the correct SSH keys set up or are providing the correct password for that user.

Make sure that `become` is set properly, and that you are using the correct `become_user` (this is `root` by default). You should confirm that you are entering the correct `sudo` password and that `sudo` on the managed host is configured correctly.

A more subtle problem has to do with inventory settings. For a complex server with multiple network addresses, you may need to use a particular address or DNS name when connecting to that system. You might not want to use that address as the machine's inventory name for better readability. You can set a host inventory variable, `ansible_host`, that will override the inventory name with a different name or IP address and be used by Ansible to connect to that host. This variable could be set in the `host_vars` file or directory for that host, or could be set in the inventory file itself.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For example, the following inventory entry configures Ansible to connect to `192.0.2.4` when processing the host `web4.phx.example.com`:

```
web4.phx.example.com ansible_host=192.0.2.4
```

This is a useful way to control how Ansible connects to managed hosts. However, it can also cause problems if the value of `ansible_host` is incorrect.

Testing Managed Hosts Using Ad Hoc Commands

The following examples illustrate some of the checks that can be made on a managed host through the use of ad hoc commands.

You have used the `ping` module to test whether you can connect to managed hosts. Depending on the options you pass, you can also use it to test whether privilege escalation and credentials are correctly configured.

```
[student@demo ~]$ ansible demohost -m ping
demohost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
[student@demo ~]$ ansible demohost -m ping --become
demohost | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "module_stderr": "sudo: a password is required\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error",
    "rc": 1
}
```

This example returns the currently available space on the disks configured in the `demohost` managed host. That can be useful to confirm that the file system on the managed host is not full.

```
[student@demo ~]$ ansible demohost -m command -a 'df'
```

This example returns the currently available free memory on the `demohost` managed host.

```
[student@demo ~]$ ansible demohost -m command -a 'free -m'
```

The Correct Level of Testing

Ansible is designed to ensure that the configuration included in playbooks and performed by its modules is correct. It monitors all modules for reported failures, and stops the playbook immediately if any failure is encountered. This helps ensure that any task performed before the failure has no errors.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Because of this, there is usually 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. But, you should be careful about adding too much complexity to your tasks and plays in an effort to double check the tests performed by the modules themselves.

References

Check Mode ("Dry Run") — Ansible Documentation
https://docs.ansible.com/ansible/2.9/user_guide/playbooks_checkmode.html

Testing Strategies — Ansible Documentation
https://docs.ansible.com/ansible/2.9/reference_appendices/test_strategies.html

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Troubleshooting Ansible Managed Hosts

In this exercise, you will troubleshoot task failures that are occurring on one of your managed hosts when running a playbook.

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-host start` script. It ensures that Ansible is installed on `workstation`. It also downloads the `inventory`, `mailrelay.yml`, and `postfix-relay-main.conf.j2` files from `http://materials.example.com/labs/troubleshoot-host/` to the `/home/student/troubleshoot-host/` directory.

```
[student@workstation ~]$ lab troubleshoot-host start
```

Instructions

1. On `workstation`, change to the `/home/student/troubleshoot-host/` directory.

```
[student@workstation ~]$ cd ~/troubleshoot-host/
[student@workstation troubleshoot-host]$
```

2. Run the `mailrelay.yml` playbook using check mode.

```
[student@workstation troubleshoot-host]$ ansible-playbook mailrelay.yml --check
PLAY [create mail relay servers] ***********************************************
...output omitted...
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"
}
...output omitted...
TASK [email notification of always_bcc config] *********************************
fatal: [servera.lab.example.com]: FAILED! => {"msg": "The conditional check
 'bcc_state.stdout != 'always_bcc =''' failed. The error was: error while
 evaluating conditional (bcc_state.stdout != 'always_bcc ='): 'dict object'
 has no attribute 'stdout'\n\nThe error appears to have been in '/home/student/
troubleshoot-host/mailrelay.yml': line 42, column 7, but may\nbe elsewhere in the
 file depending on the exact syntax problem.\n\nThe offending line appears to be:
\n\n\n    - name: email notification of always_bcc config\n      ^ here\n"}
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
...output omitted...
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=6    changed=3    unreachable=0    failed=1
```

The *verify main.cf file exists* task uses the `stat` module. It confirmed that `main.cf` exists on `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.

3. Using an ad hoc command, check the header for the `/etc/postfix/main.cf` file.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "head /etc/postfix/main.cf"
servera.lab.example.com | FAILED | rc=1 >>
head: cannot open '/etc/postfix/main.cf' for reading: No such file or
 directorynon-zero return code
```

The command failed because the playbook was executed using check mode. Postfix is not installed on `servera.lab.example.com`

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 troubleshoot-host]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ***********************************************
...output omitted...
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=5    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

5. Using an ad hoc command, display the top of the `/etc/postfix/main.cf` file.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "head /etc/postfix/main.cf"
servera.lab.example.com | SUCCESS | rc=0 >>
# Ansible managed
#
# 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 use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
# 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/BASIC_CONFIGURATION_README.html etc.
```

Now it starts with a line that contains the string, "Ansible managed". This file was updated and is now managed by Ansible.

6. Edit the `mailrelay.yml` playbook and add a task to enable the `smtp` service through the firewall. Add the task as the last task, before the handlers.

```
...output omitted...
    - name: postfix firewalld config
      firewalld:
        state: enabled
        permanent: true
        immediate: true
        service: smtp
...output omitted...
```

7. Run the playbook. The `postfix firewalld config` task should have been executed with no errors.

```
[student@workstation troubleshoot-host]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ***********************************************
...output omitted...
TASK [postfix firewalld config] ************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=8    changed=2    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

8. Using an ad hoc command, check that the `smtp` service is now configured on the firewall at `servera.lab.example.com`.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "firewall-cmd --list-services"
servera.lab.example.com | CHANGED | rc=0 >>
cockpit dhcpv6-client samba smtp ssh
```

9. Use `telnet` to test if the SMTP service is listening on port *TCP/25* on servera.lab.example.com. Disconnect when you are finished.

```
[student@workstation troubleshoot-host]$ 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
221 2.0.0 Bye
Connection closed by foreign host.
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finish

On `workstation`, run the `lab troubleshoot-host finish` script to clean up this exercise.

```
[student@workstation ~]$ lab troubleshoot-host finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Troubleshooting Ansible

Performance Checklist

In this lab, you will troubleshoot problems that occur when you try to run a playbook that has been provided to you.

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-review start` command.

```
[student@workstation ~]$ lab troubleshoot-review start
```

This script verifies that Ansible is installed on `workstation`, and creates the `~student/troubleshoot-review/html/` directory. It downloads the `ansible.cfg`, `inventory-lab`, `secure-web.yml`, and `vhosts.conf` files from `http://materials.example.com/labs/troubleshoot-review/` to the `/home/student/troubleshoot-review/` directory. It also downloads the `index.html` file to the `/home/student/troubleshoot-review/html/` directory.

Instructions

1. From the `~/troubleshoot-review` directory, check the syntax of the `secure-web.yml` playbook. This playbook contains one play that sets up Apache HTTPD with TLS/SSL for hosts in the group `webservers`. Fix the issue that is reported.
2. Check the syntax of the `secure-web.yml` playbook again. Fix the issue that is reported.
3. Check the syntax of the `secure-web.yml` playbook a third time. Fix the issue that is reported.
4. Check the syntax of the `secure-web.yml` playbook a fourth time. It should not show any syntax errors.
5. Run the `secure-web.yml` playbook. Ansible is not able to connect to `serverb.lab.example.com`. Fix this problem.
6. Run the `secure-web.yml` playbook again. Ansible should authenticate as the `devops` remote user on the managed host. Fix this issue.
7. Run the `secure-web.yml` playbook a third time. Fix the issue that is reported.
8. Run the `secure-web.yml` playbook one more time. It should complete successfully. Use an ad hoc command to verify that the `httpd` service is running.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Evaluation

On `workstation`, run the `lab troubleshoot-review grade` script to confirm success on this exercise.

```
[student@workstation ~]$ lab troubleshoot-review grade
```

Finish

On `workstation`, run the `lab troubleshoot-review finish` script to clean up this lab.

```
[student@workstation ~]$ lab troubleshoot-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Troubleshooting Ansible

Performance Checklist

In this lab, you will troubleshoot problems that occur when you try to run a playbook that has been provided to you.

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-review start` command.

```
[student@workstation ~]$ lab troubleshoot-review start
```

This script verifies that Ansible is installed on `workstation`, and creates the `~student/troubleshoot-review/html/` directory. It downloads the `ansible.cfg`, `inventory-lab`, `secure-web.yml`, and `vhosts.conf` files from `http://materials.example.com/labs/troubleshoot-review/` to the `/home/student/troubleshoot-review/` directory. It also downloads the `index.html` file to the `/home/student/troubleshoot-review/html/` directory.

Instructions

1. From the `~/troubleshoot-review` directory, check the syntax of the `secure-web.yml` playbook. This playbook contains one play that sets up Apache HTTPD with TLS/SSL for hosts in the group `webservers`. Fix the issue that is reported.

    1.1. On `workstation`, change to the `/home/student/troubleshoot-review` project directory.

```
[student@workstation ~]$ cd ~/troubleshoot-review/
```

    1.2. Check the syntax of the `secure-web.yml` playbook. This playbook sets up Apache HTTPD with TLS/SSL for hosts in the `webservers` group when everything is correct.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to be in '/home/student/troubleshoot-review/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 definition of the `random_var` variable by adding double quotes to the `This is colon: test` string. The resulting change should appear as follows:

```
...output omitted...
  vars:
    random_var: "This is colon: test"
...output omitted...
```

2. Check the syntax of the `secure-web.yml` playbook again. Fix the issue that is reported.

2.1. Check the syntax of `secure-web.yml` using `ansible-playbook --syntax-check` again.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  did not find expected '-' indicator

The error appears to be in '/home/student/troubleshoot-review/secure-web.yml':
 line 38, 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 appear as follows:

```
...output omitted...
          args:
            creates: /etc/pki/tls/certs/serverb.lab.example.com.crt
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        - name: start and enable web services
          service:
            name: httpd
            state: started
            enabled: yes

        - name: deliver content
          copy:
            dest: /var/www/vhosts/serverb-secure
            src: html/
...output omitted...
```

3. Check the syntax of the `secure-web.yml` playbook a third time. Fix the issue that is reported.

3.1. Check the syntax of the `secure-web.yml` playbook.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  found unacceptable key (unhashable type: 'AnsibleMapping')

The error appears to be in '/home/student/troubleshoot-review/secure-web.yml':
 line 13, column 20, but may
be elsewhere in the file depending on the exact syntax problem.

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 appear as follows:

```
...output omitted...
        - name: install web server packages
          yum:
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
            name: "{{ item }}"
            state: latest
          notify:
            - restart services
          loop:
            - httpd
            - mod_ssl
...output omitted...
```

4. Check the syntax of the `secure-web.yml` playbook a fourth time. It should not show any syntax errors.

4.1. Review the syntax of the `secure-web.yml` playbook. It should not show any syntax errors.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml

playbook: secure-web.yml
```

5. Run the `secure-web.yml` playbook. Ansible is not able to connect to `serverb.lab.example.com`. Fix this problem.

5.1. Run the `secure-web.yml` playbook. This will fail with an error.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml

PLAY [create secure web service] ***********************************************

TASK [Gathering Facts] *********************************************************
fatal: [serverb.lab.example.com]: UNREACHABLE! => {"changed": false, "msg":
 "Failed to connect to the host via ssh: students@serverc.lab.example.com:
 Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
 "unreachable": true}

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=0    changed=0    unreachable=1    failed=0  ...
```

5.2. Run the `secure-web.yml` playbook again, adding the `-vvv` parameter to increase the verbosity of the output.

Notice that Ansible appears to be connecting to `serverc.lab.example.com` instead of `serverb.lab.example.com`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
TASK [Gathering Facts] ************************************************************
task path: /home/student/troubleshoot-review/secure-web.yml:3
<serverc.lab.example.com> ESTABLISH SSH CONNECTION FOR USER: students
<serverc.lab.example.com> SSH: EXEC ssh -C -o ControlMaster=auto
 -o ControlPersist=60s -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/bc0c05136a serverc.lab.example.com '/bin/sh
 -c '"'"'echo ~students && sleep 0'"'"''
...output omitted...
```

5.3. Correct the line in the `inventory` file. Delete the `ansible_host` host variable so the file appears as shown below:

```
[webservers]
serverb.lab.example.com
```

6. Run the `secure-web.yml` playbook again. Ansible should authenticate as the `devops` remote user on the managed host. Fix this issue.

6.1. Run the `secure-web.yml` playbook.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
TASK [Gathering Facts] ************************************************************
task path: /home/student/troubleshoot-review/secure-web.yml:3
<serverb.lab.example.com> ESTABLISH SSH CONNECTION FOR USER: students
...output omitted...
fatal: [serverb.lab.example.com]: UNREACHABLE! => {
...output omitted...
```

6.2. Edit the `secure-web.yml` playbook to make sure `devops` is the `remote_user` for the play. The first lines of the playbook should appear as follows:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
...output omitted...
```

7. Run the `secure-web.yml` playbook a third time. Fix the issue that is reported.

7.1. Run the `secure-web.yml` playbook.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
failed: [serverb.lab.example.com] (item=mod_ssl) => {
    "ansible_loop_var": "item",
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "changed": false,
    "invocation": {
        "module_args": {
            "allow_downgrade": false,
            "autoremove": false,
...output omitted...
            "validate_certs": true
        }
    },
    "item": "mod_ssl",
    "msg": "This command has to be run under the root user.",
    "results": []
}
...output omitted...
```

7.2. Edit the play to make sure that it has `become: true` or `become: yes` set. The resulting change should appear as follows:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
  become: true
...output omitted...
```

8. Run the `secure-web.yml` playbook one more time. It should complete successfully. Use an ad hoc command to verify that the `httpd` service is running.

8.1. Run the `secure-web.yml` playbook.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml

PLAY [create secure web service] ***********************************************
...output omitted...
TASK [install web server packages] *********************************************
changed: [serverb.lab.example.com] => (item=httpd)
changed: [serverb.lab.example.com] => (item=mod_ssl)
...output omitted...
TASK [httpd_conf_syntax variable] **********************************************
ok: [serverb.lab.example.com] => {
    "msg": "The httpd_conf_syntax variable value is {'cmd': ['/sbin/httpd',
 '-t'], 'stdout': '', 'stderr': 'Syntax OK', 'rc': 0, 'start': '2021-07-16
 14:08:35.304347', 'end': '2021-07-16 14:08:35.342415', 'delta': '0:00:00.038068',
 'changed': True, 'stdout_lines': [], 'stderr_lines': ['Syntax OK'], 'failed':
 False, 'failed_when_result': False}"
}
...output omitted...
RUNNING HANDLER [restart services] *********************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=10   changed=7    unreachable=0    failed=0  ...
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

8.2. Use an ad hoc command to determine the state of the `httpd` service on `serverb.lab.example.com`. The `httpd` service should now be running on `serverb.lab.example.com`.

```
[student@workstation troubleshoot-review]$ ansible all -u devops -b \
> -m command -a 'systemctl status httpd'
serverb.lab.example.com | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
 disabled)
   Active: active (running) since Fri 2021-07-16 14:08:37 EDT; 3min 1s ago
...output omitted...
```

Evaluation

On `workstation`, run the `lab troubleshoot-review grade` script to confirm success on this exercise.

```
[student@workstation ~]$ lab troubleshoot-review grade
```

Finish

On `workstation`, run the `lab troubleshoot-review finish` script to clean up this lab.

```
[student@workstation ~]$ lab troubleshoot-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 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 the 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.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 9

Automating Linux Administration Tasks

Goal

Automate common Linux system administration tasks with Ansible.

Objectives

- Subscribe systems, configure software channels and repositories, enable module streams, and manage RPM packages on managed hosts.
- Manage Linux users and groups, configure SSH, and modify Sudo configuration on managed hosts.
- Manage service startup, schedule processes with at, cron, and systemd, reboot, and control the default boot target on managed hosts.
- Partition storage devices, configure LVM, format partitions or logical volumes, mount file systems, and add swap files or spaces.
- Configure network settings and name resolution on managed hosts, and collect network-related Ansible facts.

Sections

- Managing Software and Subscriptions (Guided Exercise)
- Managing Users and Authentication (Guided Exercise)
- Managing the Boot Process and Scheduled Processes (Guided Exercise)
- Managing Storage (Guided Exercise)
- Managing Network Configuration (Guided Exercise)

Lab

- Automating Linux Administration Tasks

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Software and Subscriptions

Objectives

After completing this section, you should be able to subscribe systems, configure software channels and repositories, enable module streams, and manage RPM packages on managed hosts.

Managing Packages with Ansible

The `yum` Ansible module uses the *Yum Package Manager* on the managed hosts to handle the package operations. The following example is a playbook that installs the *httpd* package on the `servera.lab.example.com` managed host.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the httpd packages
      yum:
        name: httpd      (1)
        state: present   (2)
```

(1) The `name` keyword gives the name of the package to install.

(2) The `state` keyword indicates the expected state of the package on the managed host:

`present`
: Ansible installs the package if it is not already there.

`absent`
: Ansible removes the package if it is installed.

`latest`
: Ansible updates the package if it is not already at the most recent available version. If the package is not installed, Ansible installs it.

The following table compares some usage of the `yum` Ansible module with the equivalent `yum` command.

Ansible task	Yum command

`- name: Install httpd`
`  yum:`
`  name: httpd`
`  state: present`
	`yum install httpd`

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Ansible task	Yum command

`- name: Install or update httpd`
`  yum:`
`  name: httpd`
`  state: latest`
	`yum update httpd` or `yum install httpd` if the package is not yet installed.

`- name: Update all packages`
`  yum:`
`  name: '*'`
`  state: latest`
	`yum update`

`- name: Remove httpd`
`  yum:`
`  name: httpd`
`  state: absent`
	`yum remove httpd`

`- name: Install Development Tools`
`  yum:`
`  name: '@Development Tools'` (1)
`  state: present`
 	
(1) With the `yum` Ansible module, you must prefix group names with `@`. Remember that you can retrieve the list of groups with the `yum group list` command.
	`yum group install "Development Tools"`

`- name: Remove Development Tools`
`  yum:`
`  name: '@Development Tools'`
`  state: absent`
	`yum group remove "Development Tools"`

`- name: Inst perl AppStream module`
`  yum:`
`  name: '@perl:5.26/minimal'` (1)
`  state: present`
 	
(1) To manage a Yum AppStream module, prefix its name with `@`. The syntax is the same as with the `yum` command. For example, you can omit the profile part to use the default profile: `@perl:5.26`. Remember that you can list the available Yum AppStream modules with the `yum module list` command.
	`yum module install perl:5.26/minimal`

Run the `ansible-doc yum` command for additional parameters and playbook examples.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Optimizing Multiple Package Installation

To operate on several packages, the `name` keyword accepts a list. The following example shows a playbook that installs three packages on `servera.lab.example.com`.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name:
          - httpd
          - mod_ssl
          - httpd-tools
        state: present
```

With this syntax, Ansible installs the packages in a single Yum transaction. This is equivalent to running the `yum install httpd mod_ssl httpd-tools` command.

A commonly seen but less efficient and slower version of this task is to use a loop.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name: "{{ item }}""
        state: present
      loop:
        - httpd
        - mod_ssl
        - httpd-tools
```

Avoid using this method as it requires the module to perform three individual transactions, one for each package.

Gathering Facts about Installed Packages

The `package_facts` Ansible module collects the installed package details on managed hosts. It sets the `ansible_facts.packages` variable with the package details.

The following playbook calls the `package_facts` module, the `debug` module to display the content of the `ansible_facts.packages` variable, and the `debug` module again to view the version of the installed *NetworkManager* package.

```
---
- name: Display installed packages
  hosts: servera.lab.example.com
  tasks:
    - name: Gather info on installed packages
      package_facts:
        manager: auto
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: List installed packages
      debug:
        var: ansible_facts.packages

    - name: Display NetworkManager version
      debug:
        msg: "Version {{ansible_facts.packages['NetworkManager'][0].version}}"
      when: "'NetworkManager' in ansible_facts.packages"
```

When run, the playbook displays the package list and the version of the *NetworkManager* package:

```
[user@controlnode ~]$ ansible-playbook lspackages.yml

PLAY [Display installed packages] ***************************************

TASK [Gathering Facts] **************************************************
ok: [servera.lab.example.com]

TASK [Gather info on installed packages] ********************************
ok: [servera.lab.example.com]

TASK [List installed packages] ******************************************
ok: [servera.lab.example.com] => {
    "ansible_facts.packages": {
        "NetworkManager": [
            {
                "arch": "x86_64",
                "epoch": 1,
                "name": "NetworkManager",
                "release": "14.el8",
                "source": "rpm",
                "version": "1.14.0"
            }
        ],
...output omitted...
        "zlib": [
            {
                "arch": "x86_64",
                "epoch": null,
                "name": "zlib",
                "release": "10.el8",
                "source": "rpm",
                "version": "1.2.11"
            }
        ]
    }
}

TASK [Display NetworkManager version] ***********************************
ok: [servera.lab.example.com] => {
    "msg": "Version 1.14.0"
}
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=4    changed=0    unreachable=0    failed=0
```

Reviewing Alternative Modules to Manage Packages

The `yum` Ansible module works on managed hosts that are using the Yum Package Manager. For other package managers, Ansible usually provides a dedicated module. For example, the `dnf` module manages packages on operating systems such as Fedora using the *DNF package manager*. The `apt` module uses the *APT package tool* available on Debian or Ubuntu. The `win_package` module can install software on Microsoft Windows systems.

The following playbook uses conditionals to select the appropriate module in an environment composed of Red Hat Enterprise Linux and Fedora systems.

```
---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd on RHEL
      yum:
        name: httpd
        state: present
      when: "ansible_distribution == 'RedHat'"

    - name: Install httpd on Fedora
      dnf:
        name: httpd
        state: present
      when: "ansible_distribution == 'Fedora'"
```

As an alternative, the generic `package` module automatically detects and uses the package manager available on the managed hosts. With the `package` module, you can rewrite the previous playbook as follows.

```
---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd
      package:
        name: httpd
        state: present
```

However, notice that the `package` module does not support all the features that the more specialized modules provide. Also, operating systems often have different names for the packages they provide. For example, the package that installs the Apache HTTP Server is *httpd* on Red Hat Enterprise Linux and *apache2* on Ubuntu. In that situation, you still need a conditional for selecting the correct package name depending on the operating system of the managed host.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Registering and Managing Systems with Red Hat Subscription Management

To entitle your new Red Hat Enterprise Linux systems to product subscriptions, Ansible provides the `redhat_subscription` and `rhsm_repository` modules. These modules interface with the *Red Hat Subscription Management* tool on the managed hosts.

Registering and Subscribing New systems

The first two tasks you usually perform with the Red Hat Subscription Management tool is to register the new system and attach an available subscription.

Without Ansible, you perform these tasks with the `subscription-manager` command:

```
[user@host ~]$ subscription-manager register --username=yourusername \
> --password=yourpassword
[user@host ~]$ subscription-manager attach --pool=poolID
```

Remember that you list the available pools in your account with the `subscription-manager list --available` command.

The `redhat_subscription` Ansible module performs the registration and the subscription in one task.

```
- name: Register and subscribe the system
  redhat_subscription:
    username: yourusername
    password: yourpassword
    pool_ids: poolID
    state: present
```

A `state` keyword set to `present` indicates to register and to subscribe the system. When it is set to `absent`, the module unregisters the system.

Enabling Red Hat Software Repositories

The next task after the subscription is to enable Red Hat software repositories on the new system.

Without Ansible, you usually execute the `subscription-manager` command for that purpose:

```
[user@host ~]$ subscription-manager repos \
> --enable "rhel-8-for-x86_64-baseos-rpms" \
> --enable "rhel-8-for-x86_64-baseos-debug-rpms"
```

Remember that you can list the available repositories with the `subscription-manager repos --list` command.

With Ansible, use the `rhsm_repository` module:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: Enable Red Hat repositories
  rhsm_repository:
    name:
      - rhel-8-for-x86_64-baseos-rpms
      - rhel-8-for-x86_64-baseos-debug-rpms
    state: present
```

Configuring a Yum Repository

To enable support for a third-party repository on a managed host, Ansible provides the `yum_repository` module.

Declaring a Yum Repository

When run, the following playbook declares a new repository on `servera.lab.example.com`.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Ensure Example Repo exists
      yum_repository:
        file: example    (1)
        name: example-internal
        description: Example Inc. Internal YUM repo
        baseurl: http://materials.example.com/yum/repository/
        enabled: yes
        gpgcheck: yes    (2)
        state: present   (3)
```

(1) The `file` keyword gives the name of the file to create under the `/etc/yum.repos.d/` directory. The module automatically adds the `.repo` extension to that name.

(2) Typically, software providers digitally sign RPM packages using GPG keys. By setting the `gpgcheck` keyword to `yes`, the RPM system verifies package integrity by confirming that the package was signed by the appropriate GPG key. It refuses to install a package if the GPG signature does not match. Use the `rpm_key` Ansible module, described later on, to install the required GPG public key.

(3) When you set the `state` keyword to `present`, Ansible creates or updates the `.repo` file. When `state` is set to `absent`, Ansible deletes the file.

The resulting `/etc/yum.repos.d/example.repo` file on `servera.lab.example.com` is as follows.

```
[example-internal]
baseurl = http://materials.example.com/yum/repository/
enabled = 1
gpgcheck = 1
name = Example Inc. Internal YUM repo
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The `yum_repository` module exposes most of the Yum repository configuration parameters as keywords. Run the `ansible-doc yum_repository` command for additional parameters and playbook examples.

Note

Some third-party repositories provide the configuration file and the GPG public key as part of an RPM package that can be downloaded and installed using the `yum install` command. For example, the *Extra Packages for Enterprise Linux (EPEL)* project provides the *https://dl.fedoraproject.org/pub/epel/epel-release-latest-VER.noarch.rpm* package that deploys the `/etc/yum.repos.d/epel.repo` configuration file. For this repository, use the `yum` Ansible module to install the EPEL package instead of the `yum_repository` module.

Importing an RPM GPG key

When the `gpgcheck` keyword is set to `yes` in the `yum_repository` module, you also need to install the GPG key on the managed host. The `rpm_key` module in the following example deploys on `servera.lab.example.com` the GPG public key hosted on a remote web server.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Deploy the GPG public key
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present

    - name: Ensure Example Repo exists
      yum_repository:
        file: example
        name: example-internal
        description: Example Inc. Internal YUM repo
        baseurl: http://materials.example.com/yum/repository/
        enabled: yes
        gpgcheck: yes
        state: present
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

References

`yum`(8), `yum.conf`(5), and `subscription-manager`(8) man pages

yum – Manages packages with the yum package manager – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/yum_module.html

package_facts – package information as facts – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/package_facts_module.html

redhat_subscription – Manage registration and subscriptions to RHSM using the subscription-manager command – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/redhat_subscription_module.html

rhsm_repository – Manage RHSM repositories using the subscription-manager command – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/rhsm_repository_module.html

yum_repository – Add or remove YUM repositories – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/yum_repository_module.html

rpm_key – Adds or removes a gpg key from the rpm db – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/rpm_key_module.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Software and Subscriptions

In this exercise you will configure a new Yum repository and install packages from it on your managed hosts.

Outcomes

You should be able to:

- Configure a yum repository using the `yum_repository` module.
- Manage RPM GPG keys using the `rpm_key` module.
- Obtain information about the installed packages on a host using the `package_facts` module.

Before You Begin

On `workstation`, run the lab start script to confirm that the environment is ready for the lab to begin. The script creates the working directory, called `system-software`, and populates it with an Ansible configuration file, a host inventory, and lab files.

```
[student@workstation ~]$ lab system-software start
```

Instructions

Your organization requires that all hosts have the *example-motd* package installed. This package is provided by an internal Yum repository maintained by your organization to host internally developed software packages.

You are tasked with writing a playbook to ensure that the *example-motd* package is installed on the remote host. The playbook must ensure the configuration of the internal Yum repository.

The repository is located at `http://materials.example.com/yum/repository`. All RPM packages are signed with an organizational GPG key pair. The GPG public key is available at `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

1. As the `student` user on `workstation`, change to the `/home/student/system-software` working directory.

```
[student@workstation ~]$ cd ~/system-software
[student@workstation system-software]$
```

2. Begin writing the `repo_playbook.yml` playbook. Define a single play in the playbook that targets all hosts. Add a `vars` clause that defines a single variable `custom_pkg` with a value of `example-motd`. Add the `tasks` clause to to the playbook.

   The playbook now contains:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
```

3. Add two tasks to the playbook.

Use the `package_facts` module in the first task to gather information about installed packages on the remote host. This task populates the `ansible_facts.packages` fact.

Use the `debug` module in the second task to print the installed version of the package referenced by the `custom_pkg` variable. Only execute this task if the custom package is found in the `ansible_facts.packages` fact.

Execute the `repo_playbook.yml` playbook.

3.1. Add the first task to the playbook. Configure the `manager` keyword of the `package_facts` module with a value of `auto`. The first task contains the following:

```
    - name: Gather Package Facts
      package_facts:
        manager: auto
```

3.2. Add a second task to the playbook that uses the `debug` module to display the value of the `ansible_facts.packages[custom_pkg]` variable. Add a `when` clause to the task to check if the value of the `custom_pkg` variable is contained in the `ansible_facts.packages` variable. The second task contains the following:

```
    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages
```

3.3. Execute the playbook:

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Gather Package Facts] ****************************************************
ok: [servera.lab.example.com]

TASK [Show Package Facts for the custom package] *******************************
skipping: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The debug task is skipped because the *example-motd* package is not installed on the remote host.

4. Add a third task that uses the `yum_repository` module to ensure the configuration of the internal yum repository on the remote host. Ensure that:

   - The repository's configuration is stored in the file `/etc/yum.repos.d/example.repo`
   - The repository ID is `example-internal`
   - The base URL is `http://materials.example.com/yum/repository`
   - The repository is configured to check RPM GPG signatures
   - The repository description is `Example Inc. Internal YUM repo`

   The third task contains the following:

```
- name: Ensure Example Repo exists
  yum_repository:
    name: example-internal
    description: Example Inc. Internal YUM repo
    file: example
    baseurl: http://materials.example.com/yum/repository/
    gpgcheck: yes
```

5. Add a fourth task to the play that uses the `rpm_key` module to ensure that the repository public key is present on the remote host. The repository public key URL is `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

   The fourth task appears as follows:

```
- name: Ensure Repo RPM Key is Installed
  rpm_key:
    key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
    state: present
```

6. Add a fifth task to ensure that the package referenced by the `custom_pkg` variable is installed on the remote host.

   The fifth task appears as follows:

```
- name: Install Example motd package
  yum:
    name: "{{ custom_pkg }}"
    state: present
```

7. The `ansible_facts.packages` fact is not updated when a new package is installed on a remote host.

   Copy the second task and add it as the sixth task in the play. Execute the playbook and verify that the `ansible_facts.packages` fact does not contain information about the *example-motd* installed on the remote host.

   7.1. The sixth task contains a copy of the second task:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages
```

The entire playbook now looks as follows:

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages

    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
        gpgcheck: yes

    - name: Ensure Repo RPM Key is Installed
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present

    - name: Install Example motd package
      yum:
        name: "{{ custom_pkg }}"
        state: present

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages
```

7.2. Execute the playbook.

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml
PLAY [Repository Configuration] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Gather Package Facts] ****************************************************
ok: [servera.lab.example.com]❶

TASK [Show Package Facts for the custom package] ***************************
skipping: [servera.lab.example.com]

TASK [Ensure Example Repo exists] ******************************************
changed: [servera.lab.example.com]

TASK [Ensure Repo RPM Key is Installed] ************************************
changed: [servera.lab.example.com]

TASK [Install Example motd package] ****************************************
changed: [servera.lab.example.com]

TASK [Show Package Facts for the custom package] ***************************
skipping: [servera.lab.example.com]❷

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=5    changed=3    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

❶ The `Gather Package Facts` task determines the data contained in the `ansible_facts.packages` fact.

❷ The task is skipped because the *example-motd* package is installed after the `Gather Package Facts` task.

8. Insert a task immediately after the `Install Example motd package` task using the `package_facts` module to update the package facts. Set the module's `manager` keyword with a value of `auto`.

   The complete playbook is shown below:

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages

    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        gpgcheck: yes

    - name: Ensure Repo RPM Key is Installed
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present

    - name: Install Example motd package
      yum:
        name: "{{ custom_pkg }}"
        state: present

    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages
```

9. Use an Ansible ad hoc command to remove the *example-motd* package installed during the previous execution of the playbook. Execute the playbook with the inserted `package_facts` task and use the output to verify that the installation of the *example-motd* package.

9.1. To remove the *example-motd* package from all hosts, use the `ansible all` command with the `-m yum` and `-a 'name=example-motd state=absent'` options.

```
[student@workstation system-software]$ ansible all -m yum \
> -a 'name=example-motd state=absent'
servera.lab.example.com | CHANGED => {
...output omitted...
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Removed: example-motd-1.0-1.el7.x86_64"
    ]
...output omitted...
```

9.2. Execute the playbook.

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Gather Package Facts] ****************************************************
ok: [servera.lab.example.com]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Show Package Facts for the custom package] ****************************
skipping: [servera.lab.example.com] (1)

...output omitted...

TASK [Install Example motd package] *****************************************
changed: [servera.lab.example.com] (2)

TASK [Gather Package Facts] *************************************************
ok: [servera.lab.example.com] (3)

TASK [Show Package Facts for example-motd] ***********************************
ok: [servera.lab.example.com] => {
    "ansible_facts.packages[custom_pkg]": [ (4)
        {
            "arch": "x86_64",
            "epoch": null,
            "name": "example-motd",
            "release": "1.el7",
            "source": "rpm",
            "version": "1.0"
        }
    ]
}

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=7    changed=1    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

(1) No package fact exists for the *example-motd* package because the package is not installed on the remote host.

(2) The *example-motd* package is installed as a result of this task, as indicated by the `changed` status.

(3) This task updates the package facts with information about the *example-motd* package.

(4) The *example-motd* package fact exists and indicates only one *example-motd* package is installed. The installed package is at version `1.0`.

Finish

On `workstation`, run the `lab system-software finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab system-software finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Users and Authentication

Objectives

After completing this section, you should be able to manage Linux users and groups, configure SSH, and modify Sudo configuration on managed hosts.

The User Module

The Ansible user module lets you manage user accounts on a remote host. You can manage a number of parameters including remove user, set home directory, set the UID for system accounts, manage passwords and associated groupings. To create a user that can log into the machine, you need to provide a hashed password for the password parameter. See the reference section for a link to "How do I generate encrypted passwords for the user module?"

Example of the User Module

```
- name: Add new user to the development machine and assign the appropriate groups.
  user:
    name: devops_user (1)
    shell: /bin/bash (2)
    groups: sys_admins, developers (3)
    append: yes
```

(1) The `name` parameter is the only requirement in the user module and is usually the service account or user account.

(2) The `shell` parameter optionally sets the user's shell. On other operating systems, the default shell is decided by the tool being used.

(3) The `groups` parameter along with the `append` parameter tells the machine that we want to append the groups sys_asmins and developers with this user. If you do not use the append parameter then the groups will overwrite in place.

When creating a user you can specify it to `generate_ssh_key`. This will `not` overwrite an existing SSH key.

Example of User Module Generating an ssh key

```
- name: Create a SSH key for user1
  user:
    name: user1
    generate_ssh_key: yes
    ssh_key_bits: 2048
    ssh_key_file: .ssh/id_my_rsa
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Note

The user module also offers some return values. Ansible modules can take a return value and register them into a variable. Find out more with ansible-doc and on the main doc site.

Some commonly used parameters

Parameter	Comments
comment	Optionally sets the description of a user account.
group	Optionally sets the user's primary group.
groups	List of multiple groups. When set to a null value, all groups except the primary group is removed.
home	Optionally sets the user's home directory.
create_home	Takes a boolean value of yes or no. A home directory will be created for the user if the value is set to yes.
system	When creating an account state=present, setting this to yes makes the user a system account. This setting cannot be changed on existing users.
uid	Sets the UID od user.

The Group Module

The `group` module allows you to manage (add, delete, modify) groups on the managed hosts. You need to have `groupadd`, `groupdel` or `groupmod`. For windows targets, use the `win_group` module.

Example of the group module

```
- name: Verify that auditors group exists
  group:
    name: auditors
    state: present
```

Parameters for the group module

Parameter	Comments
gid	Optional GID to set for the group.
local	Forces the use of "local" command alternatives on platforms that implement it.
name	Name of the group to manage.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Parameter	Comments
state	Whether the group should be present or not on the remote host.
system	If set to yes, indicates that the group created is a system group.

The Known Hosts Module

If you have a large number of host keys to manage you will want to use the `known_hosts` module. The `known_hosts` module lets you add or remove host keys from the known_hosts file on managed host.

Example of known_host Tasks

```
- name: copy host keys to remote servers
  known_hosts:
    path: /etc/ssh/ssh_known_hosts
    name: host1
    key: "{{ lookup('file', 'pubkeys/host1') }}"
```
(1)

(1) A `lookup` plugin allows Ansible to access data from outside sources.

The Authorized Key Module

The `authorized_key` module allows you to add or remove SSH authorized keys per user accounts. When adding and subtracting users to a large bank of servers, you need to be able to manage ssh keys.

Example of authorized_key Tasks

```
- name: Set authorized key
  authorized_key:
    user: user1
    state: present
    key: "{{ lookup('file', '/home/user1/.ssh/id_rsa.pub') }}
```
(1)

(1) A key can also be taken from a url: https://github.com/user1.keys.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

References

Users Module Ansible Documentation
http://docs.ansible.com/ansible/2.9/modules/user_module.html#user-module

How do I generate encrypted passwords for the user module
https://docs.ansible.com/ansible/2.9/reference_appendices/faq.html#how-do-i-generate-encrypted-passwords-for-the-user-module

Group Module Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/group_module.html#group-module

SSH Known Hosts Module Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/known_hosts_module.html#known-hosts-module

Authorized_key module Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/authorized_key_module.html#authorized-key-module

The Lookup Plugin Ansible Documentation
https://docs.ansible.com/ansible/2.9/plugins/lookup.html?highlight=lookup

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Users and Authentication

In this exercise, you will create multiple users on your managed hosts and populate the authorized SSH keys for them.

Outcomes

You should be able to:

- Create a new user group.
- Manage users with the `user` module.
- Populate SSH authorized keys using the `authorized_key` module.
- Modify both the `sudoers` and the `sshd_config` files using the `lineinfile` module.

Before You Begin

On `workstation`, run the lab start script to confirm the environment is ready for the lab to begin. The script creates the working directory, called `system-users`, and populates it with an Ansible configuration file, a host inventory, and some lab files.

```
[student@workstation ~]$ lab system-users start
```

Instructions

Your organization requires that all hosts have the same local users available. These users should belong to the `webadmin` user group, which has the ability to use the `sudo` command without specifying a password. Also, the users' SSH public keys should be distributed in the environment and the `root` user should not be allowed to log in using SSH directly.

You are tasked with writing a playbook to ensure that the users and user group are present on the remote host. The playbook must ensure the users can log in using the authorized SSH key, as well as use `sudo` without specifying a password, and that the `root` user can't log in directly using SSH.

1. As the `student` user on `workstation`, change to the `/home/student/system-users` working directory.

```
[student@workstation ~]$ cd ~/system-users
[student@workstation system-users]$
```

2. Take a look at the existing `vars/users_vars.yml` variable file.

```
[student@workstation system-users]$ cat vars/users_vars.yml
---
users:
  - username: user1
    groups: webadmin
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  - username: user2
    groups: webadmin
  - username: user3
    groups: webadmin
  - username: user4
    groups: webadmin
  - username: user5
    groups: webadmin
```

It uses the `username` variable name to set the correct username, and the `groups` variable to define additional groups that the user should belong to.

3. Start writing the `users.yml` playbook. Define a single play in the playbook that targets the `webservers` host group. Add a `vars_files` clause that defines the location of the `vars/users_vars.yml` filename, which has been created for you, and contains all the user names that are required for this exercise. Add the `tasks` clause to to the playbook.

   Use a text editor to create the `users.yml` playbook. The playbook should contain the following:

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  tasks:
```

4. Add two tasks to the playbook.

   Use the `group` module in the first task to create the `webadmin` user group on the remote host. This task creates the `webadmin` group.

   Use the `user` module in the second task to create the users from the `vars/users_vars.yml` file.

   Execute the `users.yml` playbook.

   4.1. Add the first task to the playbook. The first task contains the following:

```
  - name: Add webadmin group
    group:
      name: webadmin
      state: present
```

   4.2. Add a second task to the playbook that uses the `user` module to create the users. Add a `loop: "{{ users }}"` clause to the task to loop through the variable file for every username found in the `vars/users_vars.yml` file. As the `name:` for the users, use the `item.username` as the variable name. This allows the variable file to contain additional information that might be useful for creating the users, such as the groups that the users should belong to. The second task contains the following:

```
  - name: Create user accounts
    user:
      name: "{{ item.username }}"
      groups: "{{ item.groups }}"
    loop: "{{ users }}"
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4.3. Execute the playbook:

```
[student@workstation system-users]$ ansible-playbook users.yml

PLAY [Create multiple local users] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Add webadmin group] ******************************************************
changed: [servera.lab.example.com]

TASK [Create user accounts] ****************************************************
changed: [servera.lab.example.com] => (item={u'username': u'user1', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user2', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user3', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user4', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user5', u'groups':
 u'webadmin'})

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

5. Add a third task that uses the `authorized_key` module to ensure the SSH public keys have been properly distributed on the remote host. In the `files` directory, each of the users has a unique SSH public key file. The module loops through the list of users, finds the appropriate key by using the `username` variable, and pushes the key to the remote host.

   The third task contains the following:

```
  - name: Add authorized keys
    authorized_key:
      user: "{{ item.username }}"
      key: "{{ lookup('file', 'files/'+ item.username + '.key.pub') }}"
    loop: "{{ users }}"
```

6. Add a fourth task to the play that uses the `copy` module to modify the `sudo` config file and allow the `webadmin` group members to use `sudo` without a password on the remote host.

   The fourth task appears as follows:

```
  - name: Modify sudo config to allow webadmin users sudo without a password
    copy:
      content: "%webadmin ALL=(ALL) NOPASSWD: ALL"
      dest: /etc/sudoers.d/webadmin
      mode: 0440
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

7. Add a fifth task to ensure that the `root` user is not permitted to log in using SSH directly. Use `notify: "Restart sshd"` to trigger a handler to restart SSH.

   The fifth task appears as follows:

```
  - name: Disable root login via SSH
    lineinfile:
      dest: /etc/ssh/sshd_config
      regexp: "^PermitRootLogin"
      line: "PermitRootLogin no"
    notify: Restart sshd
```

8. In the first line after the location of the variable file, add a new handler definition. Give it a name of `Restart sshd`.

   8.1. The handler should be defined as follows:

```
...output omitted...
    - vars/users_vars.yml
  handlers:
  - name: Restart sshd
    service:
      name: sshd
      state: restarted
```

The entire playbook now looks as follows:

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  handlers:
  - name: Restart sshd
    service:
      name: sshd
      state: restarted

  tasks:

  - name: Add webadmin group
    group:
      name: webadmin
      state: present

  - name: Create user accounts
    user:
      name: "{{ item.username }}"
      groups: "{{ item.groups }}"
    loop: "{{ users }}"

  - name: Add authorized keys
    authorized_key:
      user: "{{ item.username }}"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
      key: "{{ lookup('file', 'files/'+ item.username + '.key.pub') }}"
    loop: "{{ users }}"

  - name: Modify sudo config to allow webadmin users sudo without a password
    copy:
      content: "%webadmin ALL=(ALL) NOPASSWD: ALL"
      dest: /etc/sudoers.d/webadmin
      mode: 0440

  - name: Disable root login via SSH
    lineinfile:
      dest: /etc/ssh/sshd_config
      regexp: "^PermitRootLogin"
      line: "PermitRootLogin no"
    notify: "Restart sshd"
```

8.2. Execute the playbook.

```
[student@workstation system-users]$ ansible-playbook users.yml

PLAY [Create multiple local users] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Add webadmin group] ******************************************************
ok: [servera.lab.example.com]

TASK [Create user accounts] ****************************************************
ok: [servera.lab.example.com] => (item={u'username': u'user1', u'groups':
 u'webadmin'})
ok: [servera.lab.example.com] => (item={u'username': u'user2', u'groups':
 u'webadmin'})
ok: [servera.lab.example.com] => (item={u'username': u'user3', u'groups':
 u'webadmin'})
ok: [servera.lab.example.com] => (item={u'username': u'user4', u'groups':
 u'webadmin'})
ok: [servera.lab.example.com] => (item={u'username': u'user5', u'groups':
 u'webadmin'})

TASK [Add authorized keys] *****************************************************
changed: [servera.lab.example.com] => (item={u'username': u'user1', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user2', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user3', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user4', u'groups':
 u'webadmin'})
changed: [servera.lab.example.com] => (item={u'username': u'user5', u'groups':
 u'webadmin'})

TASK [Modify sudo config to allow webadmin users sudo without a password] ***
changed: [servera.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Disable root login via SSH] ******************************************
changed: [servera.lab.example.com]

RUNNING HANDLER [Restart sshd] *********************************************
changed: [servera.lab.example.com]

PLAY RECAP *****************************************************************
servera.lab.example.com    : ok=7    changed=4    unreachable=0    failed=0
```

9. As the `user1` user, log in to `servera` server using SSH. Once logged in, use `sudo su -` command to switch identity to the `root` user.

9.1. Use SSH as the `user1` user and log in to `servera` server.

```
[student@workstation system-users]$ ssh user1@servera
Activate the web console with: systemctl enable --now cockpit.socket

[user1@servera ~]$
```

9.2. Switch identity to the `root` user.

```
[user1@servera ~]$ sudo -i
root@servera ~]#
```

9.3. Log out from `servera`.

```
[root@servera ~]$ exit
logout
[user1@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation system-users]$
```

10. Try to log in to `servera` as the `root` user directly. This step should fail because the SSH daemon configuration has been modified not to permit direct `root` user logins.

10.1. From `workstation` use SSH as `root` to log in to `servera` server.

```
[student@workstation system-users]$ ssh root@servera
root@servera's password: redhat
Permission denied, please try again.
root@servera's password:
```

This confirms that the SSH configuration denied direct access to the system for the `root` user.

Finish

On `workstation`, run the `lab system-users finish` script to clean up the resources created in this exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ lab system-users finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing the Boot Process and Scheduled Processes

Objectives

After completing this section, you should be able to manage service startup, schedule processes with at, cron, and systemd, reboot, and control the default boot target on managed hosts.

Scheduling with the at Module

Quick one-time scheduling is done with the `at` module. You create the job for a future time to run and it is held until that time comes to execute. There are six parameters that come with this module. They are: command, count, script_file, state, unique, and units.

The at Module Example:

```
- name: remove tempuser.
  at:
    command: userdel -r tempuser
    count: 20
    units: minutes
    unique: yes
```

Parameters

Parameter	Options	Comments
command	Null	A command that is scheduled to run.
count	Null	The count of units. (Must run with units)
script_file	Null	An existing script file to be executed in the future.
state	absent, present	The state adds or removes a command or script.
unique	yes, no	If a job is already running, it will not be executed again.
units	minutes/hours/days/weeks	The time denominations.

Appending Commands with the cron Module

When setting a jobs scheduled task the `cron` module is used. The cron module will append commands directly into the crontab of the user you designate.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The cron module example:

```
- cron:
    name: "Flush Bolt"
    user: "root"
    minute: 45
    hour: 11
    job: "php ./app/nut cache:clear"
```

This play uses a company's cache:clear command immediately flushes Bolt cache, removing cached files and directories.flushes cache of the CMS server every morning at 11:45.

Ansible will write the play to the crontab using the correct syntax as the user stated.

Checking the crontab will verify that it has been appended to.

Some commonly used parameters for the cron module are:

Parameters

Parameter	Options	Comments
special_time	reboot, yearly, annually, monthly, weekly, daily, hourly	A set of reoccurring times.
state	absent, present	If set to present, it will create the command. Absent will remove it.
cron_file	Null	If you have large banks of servers to maintain then sometimes it is better to have a pre-written crontab file.
backup	yes, no	Backs up the crontab file prior to being edited.

Managing Services with the systemd and service Modules

For managing services or reloading daemons, Ansible has the `systemd` and the service modules. Service offers a basic set of options start, stop, restart, enable. The systemd module offers more configuration options. Systemd will allow you to do a daemon-reload where the service module will not.

The service Module Example:

```
- name: start nginx
  service:
    name: nginx
    state: started"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Note
The init daemon is being replaced by systemd. So in a lot of cases systemd will be the better option.

The systemd Module Example:

```
- name: reload web server
  systemd:
    name: apache2
    state: reload
    daemon-reload: yes
```

The Reboot Module

Another well used Ansible Systems Module is `reboot`. Considered safer than using the shell module to initiate shutdown. While running a play the reboot module will shut down the managed host, then wait until it is back up again prior to carrying on with the play.

The reboot module Example:

```
- name: "Reboot after patching"
  reboot:
    reboot_timeout: 180

- name: force a quick reboot
  reboot:
```

The Shell and Command Module

Like the service and the systemd modules, the shell and the command can interchange some tasks. The command module is considered more secure but some environment variables are not available. Also, stream operators will not work. If you need to stream your commands then shell module will do.

The shell module example:

```
- name: Run a templated variable (always use quote filter to avoid injection)
    shell: cat {{ myfile|quote }}(1)
```

(1) To sanitize any variables, It is suggested that you use `{{ var | quote }}` instead of just `{{ var }}`

The command module example:

```
- name: This command only
  command: /usr/bin/scrape_logs.py arg1 arg2
  args:(1)
    chdir: scripts/
    creates: /path/to/script
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1. You can pass arguments into the form to provide the options.

Note

The command module is considered more secure because it is not affected by the users environment.

Gathering facts on the managed host will allow you to access the environment variables. There is a sublist called ansible_env which has all the environment variables inside it.

```
---
- name:
  hosts: webservers
  vars:
    local_shell:  "{{ ansible_env }}"(1)
  tasks:
    - name: Printing all the environment variables in Ansible
      debug:
        msg: "{{ local_shell }}"
```

1. You can isolate the variable you want to return by using the lookup plugin. `msg: "{{ lookup('env','USER','HOME','SHELL') }}"`

References

at - Schedule the execution of a command or script file via the at command – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/at_module.html

cron - Manage cron.d and crontab entries – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/cron_module.html

reboot - Reboot a machine – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/reboot_module.html

service - Run services on a machine – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/service_module.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing the Boot Process and Scheduled Processes

In this exercise, you will manage the startup process, schedule recurring jobs, and reboot managed hosts.

Outcomes

You should be able to use a playbook to:

- Schedule a `cron` job.
- Remove a single specific `cron` job from a `crontab` file.
- Schedule an `at` task.
- Set the default boot target on managed hosts.
- Reboot managed hosts.

Before You Begin

Run the `lab system-process start` script from `workstation` to configure the environment for the exercise. The script creates the `system-process` working directory, and downloads the Ansible configuration file and the host inventory file needed for the exercise.

```
[student@workstation ~]$ lab system-process start
```

Instructions

1. As the `student` user on `workstation`, change to the `/home/student/system-process` working directory.

```
[student@workstation ~]$ cd ~/system-process
[student@workstation system-process]$
```

2. Create a playbook, `create_crontab_file.yml`, in the current working directory. Configure the playbook to use the `cron` module to create the `/etc/cron.d/add-date-time` crontab file that schedules a recurring cron job. The job should run as the `devops` user every two minutes between `09:00` and `16:59` on `Monday` through `Friday`. The job should append the current date and time to the file /home/devops/my_datetime_cron_job

   2.1. Create a new playbook, `create_crontab_file.yml`, and add the lines needed to start the play. It should target the managed hosts in the `webservers` group and enable privilege escalation.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---
- name: Recurring cron job
  hosts: webservers
  become: true
```

2.2. Define a task that uses the `cron` module to schedule a recurring cron job.

Note

The `cron` module provides a `name` option to uniquely describe the crontab file entry and to ensure expected results. The description is added to the crontab file. For example, the `name` option is required if you are removing a crontab entry using `state=absent`. Additionally, the `name` option prevents a new crontab entry from always being created when the default state, `state=present`, is set.

```
  tasks:
    - name: Crontab file exists
      cron:
        name: Add date and time to a file
```

2.3. Configure the job to run every two minutes between `09:00` and `16:59` on `Monday` through `Friday`.

```
        minute: "*/2"
        hour: 9-16
        weekday: 1-5
```

2.4. Use the `cron_file` parameter to use the `/etc/cron.d/add-date-time` crontab file instead of an individual user's crontab in `/var/spool/cron/`. A relative path will place the file in `/etc/cron.d` directory. If the `cron_file` parameter is used, you must also specify the `user` parameter.

```
        user: devops
        job: date >> /home/devops/my_date_time_cron_job
        cron_file: add-date-time
        state: present
```

2.5. When completed, the playbook should appear as follows. Review the playbook for accuracy.

```
---
- name: Recurring cron job
  hosts: webservers
  become: true

  tasks:
    - name: Crontab file exists
      cron:
        name: Add date and time to a file
        minute: "*/2"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        hour: 9-16
        weekday: 1-5
        user: devops
        job: date >> /home/devops/my_date_time_cron_job
        cron_file: add-date-time
        state: present
```

2.6. Verify playbook syntax by running the `ansible-playbook --syntax-check create_crontab_file.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> create_crontab_file.yml

playbook: create_crontab_file.yml
```

2.7. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook create_crontab_file.yml

PLAY [Recurring cron job] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]

TASK [Crontab file exists] *****************************************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

2.8. Run an ad hoc command to verify that the `/etc/cron.d/add-date-time` cron file exists and its content is correct.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat /etc/cron.d/add-date-time"
servera.lab.example.com | CHANGED | rc=0 >>
#Ansible: Add date and time to a file
*/2 9-16 * * 1-5 devops date >> /home/devops/my_date_time_cron_job
```

3. Create a playbook, `remove_cron_job.yml`, in the current working directory. Configure the playbook to use the cron module to remove the `Add date and time to a file` cron job from the `/etc/cron.d/add-date-time` crontab file

3.1. Create a new playbook, `remove_cron_job.yml`, and add the following lines:

```
---
- name: Remove scheduled cron job
  hosts: webservers
  become: true

  tasks:
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: Cron job removed
      cron:
        name: Add date and time to a file
        user: devops
        cron_file: add-date-time
        state: absent
```

3.2. Verify playbook syntax by running the `ansible-playbook --syntax-check remove_cron_job.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> remove_cron_job.yml

playbook: remove_cron_job.yml
```

3.3. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook remove_cron_job.yml

PLAY [Remove scheduled cron job] ***************************************

TASK [Gathering Facts] *************************************************
ok: [servera.lab.example.com]

TASK [Cron job removed] ************************************************
changed: [servera.lab.example.com]

PLAY RECAP *************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

3.4. Run an ad hoc command to verify that the `/etc/cron.d/add-date-time` cron file continues to exist but the cron job has been removed.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat /etc/cron.d/add-date-time"
servera.lab.example.com | CHANGED | rc=0 >>
```

4. Create a playbook, `schedule_at_task.yml`, in the current working directory. Configure the playbook to use the `at` module to schedule a task that runs one minute in the future. The task should run the `date` command and redirect its output to the `/home/devops/my_at_date_time` file. Use the `unique: yes` option to ensure that if the command already exists in the `at` queue, a new task is not added.

4.1. Create a new playbook, `schedule_at_task.yml`, and add the following lines:

```
---
- name: Schedule at task
  hosts: webservers
  become: true
  become_user: devops
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  tasks:
    - name: Create date and time file
      at:
        command: "date > ~/my_at_date_time\n"
        count: 1
        units: minutes
        unique: yes
        state: present
```

4.2. Verify playbook syntax by running the `ansible-playbook -syntax-check schedule_at_task.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> schedule_at_task.yml

playbook: schedule_at_task.yml
```

4.3. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook schedule_at_task.yml

PLAY [Schedule at task] ********************************************

TASK [Gathering Facts] *********************************************
ok: [servera.lab.example.com]

TASK [Create date and time file] ***********************************
changed: [servera.lab.example.com]

PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

4.4. After waiting one minute for the `at` command to complete, run ad hoc commands to verify that the `/home/devops/my_at_date_time` file exists and has the correct contents.

```
[student@workstation system-process]$ ansible webservers -u devops \
> -a "ls -l my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
-rw-rw-r--. 1 devops devops 30 abr 17 06:15 my_at_date_time

[student@workstation system-process]$ ansible webservers -u devops \
> -a "cat my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
Thu Jul 22 13:24:34 PDT 2021
```

5. Create a playbook, `set_default_boot_target_graphical.yml`, in the current working directory. Configure the playbook to use the `file` module to change the symbolic link on managed hosts to reference the `graphical-target` boot target.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Note

In the following `file` module, the `src` parameter value is what the symbolic link references. The `dest` parameter value is the symbolic link.

5.1. Create a new playbook, `set_default_boot_target_graphical.yml`, and add the following lines:

```
---
- name: Change default boot target
  hosts: webservers
  become: true

  tasks:
    - name: Default boot target is graphical
      file:
        src: /usr/lib/systemd/system/graphical.target
        dest: /etc/systemd/system/default.target
        state: link
```

5.2. Verify the playbook syntax by running the `ansible-playbook --syntax-check set_default_boot_target_graphical.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_graphical.yml

playbook: set_default_boot_target_graphical.yml
```

5.3. Before running the playbook, run an ad hoc command to verify that the current default boot target is `multi-user.target`:

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

5.4. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_graphical.yml

PLAY [Change default boot target] ************************************

TASK [Gathering Facts] ***********************************************
ok: [servera.lab.example.com]

TASK [Default boot target is graphical] ******************************
changed: [servera.lab.example.com]
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

5.5. Run an ad hoc command to verify that the default boot target is now `graphical.target`.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

6. Create a playbook, `reboot_hosts.yml`, in the current working directory that reboots the managed hosts. It is not required to reboot a server after changing the default target. However, knowing how to create a playbook that reboots managed hosts may prove useful.

6.1. Create a new playbook, `reboot_hosts.yml`, and add the following lines:

```
---
- name: Reboot hosts
  hosts: webservers
  become: true

  tasks:
    - name: Hosts are rebooted
      reboot:
```

6.2. Verify the playbook syntax by running the `ansible-playbook --syntax-check reboot_hosts.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> reboot_hosts.yml

playbook: reboot_hosts.yml
```

6.3. Before running the playbook, run an ad hoc command to determine the timestamp of the last system reboot.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
         system boot  2021-07-22 14:34
```

6.4. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook reboot_hosts.yml

PLAY [Reboot hosts] ************************************************************

TASK [Gathering Facts] *********************************************************
ok: [servera.lab.example.com]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Hosts are rebooted] ***************************************************
changed: [servera.lab.example.com]

PLAY RECAP ******************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

6.5. Run an ad hoc command to determine the timestamp of the last system reboot. The timestamp displayed after the playbook runs should be later.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
         system boot  2021-07-22 14:52
```

6.6. Run a second ad hoc command to determine that the `graphical.target` boot target survived the reboot.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

7. To maintain consistency throughout the remaining exercises, change the default boot target back to its former setting, `multi-user.target`. Create a playbook, `set_default_boot_target_multi-user.yml`, in the current working directory. Configure the playbook to use the `file` module to change the symbolic link on managed hosts to reference the `multi-user.target` boot target.

7.1. Create a new playbook, `set_default_boot_target_multi-user.yml`, and add the following lines:

```
---
- name: Change default runlevel target
  hosts: webservers
  become: true

  tasks:
    - name: Default runlevel is multi-user target
      file:
        src: /usr/lib/systemd/system/multi-user.target
        dest: /etc/systemd/system/default.target
        state: link
```

7.2. Verify playbook syntax by running the `ansible-playbook --syntax-check set_default_boot_target_multi-user.yml` command. Correct any errors before moving to the next step.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_multi-user.yml

playbook: set_default_boot_target_multi-user.yml
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

7.3. Run the playbook.

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_multi-user.yml

PLAY [Change default runlevel target] ************************************

TASK [Gathering Facts] ***************************************************
ok: [servera.lab.example.com]

TASK [Default runlevel is multi-user target] *****************************
changed: [servera.lab.example.com]

PLAY RECAP ***************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

7.4. Run an ad hoc command to verify that the default boot target is now `multi-user.target`.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

Finish

On `workstation`, run the `lab system-process finish` script to clean up this exercise.

```
[student@workstation ~]$ lab system-process finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Storage

Objectives

After completing this section, you should be able to partition storage devices, configure LVM, format partitions or logical volumes, mount file systems, and add swap files or spaces.

Configuring Storage with Ansible Modules

Red Hat Ansible Automation Platform provides a collection of modules to configure storage devices on managed hosts. Those modules support partitioning devices, creating logical volumes, and creating and mounting filesystems.

The `parted` Module

The `parted` module supports the partition of block devices. This module includes the functionality of the `parted` command, and allows to create partitions with a specific size, flag, and alignment. The following table lists some of the parameters for the `parted` module.

Parameter name	Description
align	Configures partition alignment.
device	Block device.
flags	Flags for the partition.
number	The partition number.
part_end	Partition size from the beginning of the disk specified in `parted` supported units.
state	Creates or removes the partition.
unit	Size units for the partition information.

The following example creates a new partition of 10 GB.

```
- name: New 10GB partition
  parted:
    device: /dev/vdb (1)
    number: 1 (2)
    state: present (3)
    part_end: 10GB (4)
```

(1) Uses `vdb` as the block device to partition.

(2) Creates the partition number one.

(3) Ensures the partition is available.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4 Sets the partition size to 10 GB.

The `lvg` and `lvol` Modules

The `lvg` and `lvol` modules support the creation of logical volumes, including the configuration of physical volumes, and volume groups. The `lvg` takes as parameters the block devices to configure as the back end physical volumes for the volume group. The following table lists some of the parameters for the `lvg` module.

Parameter name	Description
pesize	The size of the physical extent. Must be a power of 2, or multiple of 128 KiB.
pvs	List of comma-separated devices to be configured as physical volumes for the volume group.
vg	The name of the volume group.
state	Creates or removes the volume.

The following task creates a volume group with a specific physical extent size using a block device as a back end.

```
- name: Creates a volume group
  lvg:
    vg: vg1 (1)
    pvs: /dev/vda1 (2)
    pesize: 32 (3)
```

1 The volume group name is `vg1`.

2 Uses `/dev/vda1` as the back end physical volume for the volume group.

3 Sets the physical extent size to 32.

In the following example, if the `vg1` volume group is already available with `/dev/vdb1` as a physical volume, the volume is enlarged adding a new physical volume with `/dev/vdc1`.

```
- name: Resize a volume group
  lvg:
    vg: vg1
    pvs: /dev/vdb1,/dev/vdc1
```

The `lvol` module creates logical volumes, and supports the resizing and shrinking of those volumes, and the filesystems on top of them. This module also supports the creation of snapshots for the logical volumes. The following table lists some of the parameters for the `lvol` module.

Parameter name	Description
lv	The name of the logical volume.
resizefs	Resizes the filesystem with the logical volume.
shrink	Enable logical volume shrink.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Parameter name	Description
size	The size of the logical volume.
snapshot	The name of the snapshot for the logical volume.
state	Create or remove the logical volume.
vg	The parent volume group for the logical volume.

The following task creates a logical volume of 2 GB.

```
- name: Create a logical volume of 2GB
  lvol:
    vg: vg1 (1)
    lv: lv1 (2)
    size: 2g (3)
```

(1) The parent volume group name is `vg1`.

(2) The logical volume name is `lv1`.

(3) The size of the logical volume is 2 GB.

The `filesystem` Module

The `filesystem` module supports both creating and resizing a filesystem. This module supports filesystem resizing for `ext2`, `ext3`, `ext4`, `ext4dev`, `f2fs`, `lvm`, `xfs`, and `vfat`. The following table lists some of the parameters for the `filesystem` module.

Parameter name	Description
dev	Block device name.
fstype	Filesystem type.
resizefs	Grows the filesystem size to the size of the block device.

The following example creates a filesystem on a partition.

```
- name: Create an XFS filesystem
  filesystem:
    fstype: xfs (1)
    dev: /dev/vdb1 (2)
```

(1) Uses the XFS filesystem.

(2) Uses the `/dev/vdb1` device.

The `mount` Module

The `mount` module supports the configuration of mount points on `/etc/fstab`. The following table lists some of the parameters for the `mount` module.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Parameter name	Description
fstype	Filesystem type.
opts	Mount options.
path	Mount point path.
src	Device to be mounted.
state	Specify the mount status. If set to `mounted`, the system mounts the device, and configures `/etc/fstab` with that mount information. To unmount the device and remove it from `/etc/fstab` use `absent`.

The following example mounts a device with an specific ID.

```
- name: Mount device with ID
  mount:
    path: /data (1)
    src: UUID=a8063676-44dd-409a-b584-68be2c9f5570 (2)
    fstype: xfs (3)
    state: present (4)
```

(1) Uses `/data` as the mount point path.

(2) Mounts the device with the `a8063676-44dd-409a-b584-68be2c9f5570` ID.

(3) Uses the XFS filesystem.

(4) Mounts the device and configures `/etc/fstab` accordingly.

The following example mounts the NFS share available at `172.25.250.100:/share` on the `/nfsshare` directory at the managed host.

```
- name: Mount NFS share
  mount:
    path: /nfsshare
    src: 172.25.250.100:/share
    fstype: nfs
    opts: defaults
    dump: '0'
    passno: '0'
    state: mounted
```

Configuring swap with Modules

Red Hat Ansible Automation Platform does not currently include modules to manage swap memory. To add swap memory to a system with Ansible with logical volumes you need to create a new volume group and logical volume with the `lvg` and `lvol` modules. When ready, you need to format as swap the new logical volume using the `command` module with the `mkswap` command. Finally, you need to activate the new swap device using the `command` module with the `swapon` command. Ansible includes the `ansible_swaptotal_mb` variable which includes the total swap memory. You can use this variable to trigger swap configuration and enablement when swap

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

memory is low. The following tasks, create a volume group and a logical volume for swap memory, format that logical volume as swap, and activates it.

```
- name: Create new swap VG
  lvg:
    vg: vgswap
    pvs: /dev/vda1
    state: present

- name: Create new swap LV
  lvol:
    vg: vgswap
    lv: lvswap
    size: 10g

- name: Format swap LV
  command: mkswap /dev/vgswap/lvswap
  when: ansible_swaptotal_mb < 128

- name: Activate swap LV
  command: swapon /dev/vgswap/lvswap
  when: ansible_swaptotal_mb < 128
```

Ansible Facts for Storage Configuration

Ansible uses facts to retrieve information to the control node about the configuration of the managed hosts. You can use the `setup` Ansible module to retrieve all the Ansible facts for a managed host.

```
[user@controlnode ~]$ ansible webservers -m setup
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
    }
```

The `filter` option for the `setup` module supports fine-grained filtering based on shell-style wildcards.

The `ansible_devices` element includes all the storage devices available on the managed host. The element for each storage device includes additional information like partitions or total size. The following example displays the `ansible_devices` element for a managed host with three storage devices: `sr0`, `vda`, and `vdb`.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_devices'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_devices": {
            "sr0": {
                "holders": [],
                "host": "IDE interface: Intel Corporation 82371SB PIIX3 IDE
 [Natoma/Triton II]",
                "links": {
                    "ids": [
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.


```
                        "ata-QEMU_DVD-ROM_QM00003"
                    ],
                    "labels": [],
                    "masters": [],
                    "uuids": []
                },
                "model": "QEMU DVD-ROM",
                "partitions": {},
                "removable": "1",
                "rotational": "1",
                "sas_address": null,
                "sas_device_handle": null,
                "scheduler_mode": "mq-deadline",
                "sectors": "2097151",
                "sectorsize": "512",
                "size": "1024.00 MB",
                "support_discard": "0",
                "vendor": "QEMU",
                "virtual": 1
            },
            "vda": {
                "holders": [],
                "host": "SCSI storage controller: Red Hat, Inc. Virtio block
device",
                "links": {
                    "ids": [],
                    "labels": [],
                    "masters": [],
                    "uuids": []
                },
                "model": null,
                "partitions": {
                    "vda1": {
                        "holders": [],
                        "links": {
                            "ids": [],
                            "labels": [],
                            "masters": [],
                            "uuids": [
                                "a8063676-44dd-409a-b584-68be2c9f5570"
                            ]
                        },
                        "sectors": "20969439",
                        "sectorsize": 512,
                        "size": "10.00 GB",
                        "start": "2048",
                        "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"
                    }
                },
                "removable": "0",
                "rotational": "1",
                "sas_address": null,
                "sas_device_handle": null,
                "scheduler_mode": "mq-deadline",
                "sectors": "20971520",
```

```
                "sectorsize": "512",
                "size": "10.00 GB",
                "support_discard": "0",
                "vendor": "0x1af4",
                "virtual": 1
            },
            "vdb": {
                "holders": [],
                "host": "SCSI storage controller: Red Hat, Inc. Virtio block
 device",
                "links": {
                    "ids": [],
                    "labels": [],
                    "masters": [],
                    "uuids": []
                },
                "model": null,
                "partitions": {},
                "removable": "0",
                "rotational": "1",
                "sas_address": null,
                "sas_device_handle": null,
                "scheduler_mode": "mq-deadline",
                "sectors": "10485760",
                "sectorsize": "512",
                "size": "5.00 GB",
                "support_discard": "0",
                "vendor": "0x1af4",
                "virtual": 1
            }
        }
    },
    "changed": false
}
```

The `ansible_device_links` element includes all the links available for each storage device. The following example displays the `ansible_device_links` element for a managed host with two storage devices, `sr0` and `vda1`, which have an associated ID.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_device_links'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_device_links": {
            "ids": {
                "sr0": [
                    "ata-QEMU_DVD-ROM_QM00003"
                ]
            },
            "labels": {},
            "masters": {},
            "uuids": {
                "vda1": [
                    "a8063676-44dd-409a-b584-68be2c9f5570"
                ]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
            }
        }
    },
    "changed": false
}
```

The `ansible_mounts` element includes information about the current mounted devices on the managed host, like the mounted device, the mount point, and the options. The following output displays the `ansible_mounts` element for a managed host with one active mount, `/dev/vda1` on the `/` directory.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_mounts'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_mounts": [
            {
                "block_available": 2225732,
                "block_size": 4096,
                "block_total": 2618619,
                "block_used": 392887,
                "device": "/dev/vda1",
                "fstype": "xfs",
                "inode_available": 5196602,
                "inode_total": 5242304,
                "inode_used": 45702,
                "mount": "/",
                "options": "rw,seclabel,relatime,attr2,inode64,noquota",
                "size_available": 9116598272,
                "size_total": 10725863424,
                "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"
            }
        ]
    },
    "changed": false
}
```

References

parted - Configure block device partitions – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/parted_module.html

lvg - Configure LVM volume groups – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/lvg_module.html

lvol - Configure LVM logical volumes – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/lvol_module.html

filesystem - Makes a filesystem – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/filesystem_module.html

mount - Control active and configured mount points – Ansible Documentation
https://docs.ansible.com/ansible/2.9/modules/mount_module.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Storage

In this exercise you will partition a new disk, create logical volumes and format them with XFS file systems, and mount them immediately and automatically at boot time on your managed hosts.

Outcomes

You should be able to:

- Use the `parted` module to configure block device partitions.
- Use the `lvg` module to manage LVM volume groups.
- Use the `lvol` module to manage LVM logical volumes.
- Use the `filesystem` module to create file systems.
- Use the `mount` module to control and configure mount points in `/etc/fstab`.

Before You Begin

Run the `lab system-storage start` script from `workstation` to configure the environment for the exercise. The script creates the `system-storage` project directory, and downloads the Ansible configuration file and the host inventory file needed for the exercise.

```
[student@workstation ~]$ lab system-storage start
```

Instructions

You are responsible for managing a set of web servers. A recommended practice for web server configuration is to store web server data on a separate partition or logical volume.

You will write a playbook to:

- Manage partitions of the `/dev/vdb` device
- Manage a volume group named `apache-vg` for web server data
- Create two logical volumes named `content-lv` and `logs-lv`, both backed by the `apache-vg` volume group
- Create an XFS file system on both logical volumes
- Mount the `content-lv` logical volume at `/var/www`
- Mount the `logs-lv` logical volume at `/var/log/httpd`

If the storage requirements for the web server change, update the appropriate playbook variables and re-execute the playbook. The playbook should be idempotent.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

1. As the `student` user on `workstation`, change to the `/home/student/system-storage` working directory.

```
[student@workstation ~]$ cd ~/system-storage
[student@workstation system-storage]$
```

2. Review the skeleton playbook file `storage.yml` and the associated variables file `storage_vars.yml` in the project directory. Execute the playbook.

   2.1. Review the `storage.yml` playbook.

```
---
- name: Ensure Apache Storage Configuration
  hosts: webservers
  vars_files:
    - storage_vars.yml
  tasks:
    - name: Correct partitions exist on /dev/vdb
      debug:
        msg: TODO
      loop: "{{ partitions }}"

    - name: Ensure Volume Groups Exist
      debug:
        msg: TODO
      loop: "{{  volume_groups }}"

    - name: Create each Logical Volume (LV) if needed
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
      when: true

    - name: Ensure XFS Filesystem exists on each LV
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"

    - name: Ensure the correct capacity for each LV
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"

    - name: Each Logical Volume is mounted
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
```

   The name of each task acts as an outline of the intended procedure to implement. In later steps, you will update and change these six tasks.

   2.2. Review the `storage_vars.yml` variables file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
---

partitions:
  - number: 1
    start: 1MiB
    end: 257MiB

volume_groups:
  - name: apache-vg
    devices: /dev/vdb1

logical_volumes:
  - name: content-lv
    size: 64M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/log/httpd
```

This file describes the intended structure of partitions, volume groups, and logical volumes on each web server. The first partition begins at an offset of 1 MiB from the beginning of the `/dev/vdb` device, and ends at an offset of 257 MiB, for a total size of 256 MiB.

Each web server has one volume group, named `apache-vg`, containing the first partition of the `/dev/vdb` device.

Each web server has two logical volumes. The first logical volume is named `content-lv`, with a size of 64 MiB, attached to the `apache-vg` volume group, and mounted at `/var/www`. The second logical volume is named `logs-lv`, with a size of 128 MiB, attached to the `apache-vg` volume group, and mounted at `/var/log/httpd`.

Note

The `apache-vg` volume group has a capacity of 256 MiB, because it is backed by the `/dev/vdb1` partition. It provides enough capacity for both of the logical volumes.

2.3. Execute the `storage.yml` playbook.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] *********************************

TASK [Gathering Facts] *****************************************************
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ********************************
ok: [servera.lab.example.com] => (item={u'start': u'1MiB', u'end': u'257MiB',
 u'number': 1}) => {
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "msg": "TODO"
}

...output omitted...

TASK [Each Logical Volume is mounted] ************************************
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'64M',
 u'mount_path': u'/var/www', u'name': u'content-lv'}) => {
    "msg": "TODO"
}
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'128M',
 u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'}) => {
    "msg": "TODO"
}

PLAY RECAP ***************************************************************
servera.lab.example.com    : ok=7    changed=0    unreachable=0    failed=0
```

3. Change the first task to use the `parted` module to configure a partition for each loop item. Each item describes an intended partition of the `/dev/vdb` device on each web server:

   number
   The partition number. Use this as the value of the `number` keyword for the `parted` module.

   start
   The start of the partition, as an offset from the beginning of the block device. Use this as the value of the `part_start` keyword for the `parted` module.

   end
   The end of the partition, as an offset from the beginning of the block device. Use this as the value of the `part_end` keyword for the `parted` module.

   The content of the first task should be:

```
- name: Correct partitions exist on /dev/vdb
  parted:
    device: /dev/vdb
    state: present
    number: "{{ item.number }}"
    part_start: "{{ item.start }}"
    part_end: "{{ item.end }}"
  loop: "{{ partitions }}"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4. Change the second task of the play to use the `lvg` module to configure a volume group for each loop item. Each item of the `volume_groups` variable describes a volume group that should exist on each web server:

   name
   The name of the volume group. Use this as the value of the `vg` keyword for the `lvg` module.

   devices
   A comma-separated list of devices or partitions that form the volume group. Use this as the value of the `pvs` keyword for the `lvg` module.

   The content of the second task should be:

```
- name: Ensure Volume Groups Exist
  lvg:
    vg: "{{ item.name }}"
    pvs: "{{ item.devices }}"
  loop: "{{  volume_groups }}"
```

5. Change the third task of the play to use the `lvol` module to create a logical volume for each item. Use the item's keywords to create the new logical volume:

   name
   The name of the logical volume. Use this as the value of the `lv` keyword for the `lvol` module.

   vgroup
   The name of the volume group that provides storage for the logical volume.

   size
   The size of the logical volume. The value of this keyword is any acceptable value for the `-L` option of the `lvcreate` command.

   Only execute the task if a logical volume does not already exist. Update the `when` statement to check that a logical volume does not exist with a name that matches the value of the item's `name` keyword.

   5.1. Change the third task to use the `lvol` module. Set the volume group name, logical volume name, and logical volume size using each item's keywords. The content of the third task is now:

```
- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
  when: true
```

   5.2. The Ansible fact `ansible_lvm` contains information about Logical Volume Management objects on each hosts. Use an ad hoc command to see the current set of logical volumes on the remote host:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": {},
            "pvs": {},
            "vgs": {}
        },
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false
}
```

The value of the `lvs` keyword indicates that there are no logical volumes on the remote host.

5.3. Execute the playbook to create the logical volumes on the remote host.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ********************************

TASK [Gathering Facts] ****************************************************
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] *******************************
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] *****************************************
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Create each Logical Volume (LV) if needed] **************************
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure XFS Filesystem exists on each LV] ****************************
ok: [servera.lab.example.com] => (item={...output omitted...}) => {
    "msg": "TODO"
}
...output omitted...
PLAY RECAP ****************************************************************
servera.lab.example.com    : ok=7    changed=3    unreachable=0    failed=0
```

5.4. Execute another ad hoc command to see the structure of the `ansible_lvm` variable when logical volumes exists on the remote host.

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": {
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
                "content-lv": {
                    "size_g": "0.06",
                    "vg": "apache-vg"
                },
                "logs-lv": {
                    "size_g": "0.12",
                    "vg": "apache-vg"
                }
            },
            "pvs": {(2)
                "/dev/vdb1": {
                    "free_g": "0.06",
                    "size_g": "0.25",
                    "vg": "apache-vg"
                }
            },
            "vgs": {(3)
                "apache-vg": {
                    "free_g": "0.06",
                    "num_lvs": "2",
                    "num_pvs": "1",
                    "size_g": "0.25"
                }
            }
        }
    },
    "changed": false
}
```

(1) The value of the `lvs` keyword is a key-value pair data structure. The keys of this structure are the names of any logical volumes on the host. This indicates that both the `content-lv` and `logs-lv` logical volumes exist. For each logical volume, the corresponding volume group is provided by the `vg` keyword.

(2) The `pvs` keyword contains information about physical volumes on the host. The information indicates that the `/dev/vdb1` partition belongs to the `apache-vg` volume group.

(3) The `vgs` keyword contains information about volume groups on the host.

5.5. Update the `when` statement to check that a logical volume does not exist with a name that matches the value of the item's `name` keyword. The content of the third task is now:

```
    - name: Create each Logical Volume (LV) if needed
      lvol:
        vg: "{{ item.vgroup }}"
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
      loop: "{{ logical_volumes }}"
      when: item.name not in ansible_lvm["lvs"]
```

6. Change the fourth task to use the `filesystem` module. Configure the task to ensure that each logical volume is formatted as an XFS file system. Recall that a logical volume is

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

associated with the logical device `/dev/<volume group name>/<logical volume name>`.

The content of the fourth task should be:

```
- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"
```

7. Configure the fifth task to ensure each logical volume has the correct storage capacity. If the logical volume increases in capacity, be sure to force the expansion of the volume's file system.

Warning

If a logical volume needs to decrease in capacity, this task will fail because an XFS file system does not support shrinking capacity.

The content of the fifth task should be:

```
- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"
```

8. Use the `mount` module in the sixth task to ensure that each logical volume is mounted at the corresponding mount path and persists after a reboot.

   The content of the sixth task should be:

```
- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    opts: noatime
    state: mounted
  loop: "{{ logical_volumes }}"
```

9. Review the completed `storage.yml` playbook. Execute the playbook and verify that each logical volume is mounted.

   9.1. Review the playbook:

```
---
- name: Ensure Apache Storage Configuration
  hosts: webservers
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
  vars_files:
    - storage_vars.yml
  tasks:
    - name: Correct partitions exist on /dev/vdb
      parted:
        device: /dev/vdb
        state: present
        number: "{{ item.number }}"
        part_start: "{{ item.start }}"
        part_end: "{{ item.end }}"
      loop: "{{ partitions }}"

    - name: Ensure Volume Groups Exist
      lvg:
        vg: "{{ item.name }}"
        pvs: "{{ item.devices }}"
      loop: "{{  volume_groups }}"

    - name: Create each Logical Volume (LV) if needed
      lvol:
        vg: "{{ item.vgroup }}"
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
      loop: "{{ logical_volumes }}"
      when: item.name not in ansible_lvm["lvs"]

    - name: Ensure XFS Filesystem exists on each LV
      filesystem:
        dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
        fstype: xfs
      loop: "{{ logical_volumes }}"

    - name: Ensure the correct capacity for each LV
      lvol:
        vg: "{{ item.vgroup }}"
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
        resizefs: yes
        force: yes
      loop: "{{ logical_volumes }}"

    - name: Each Logical Volume is mounted
      mount:
        path: "{{ item.mount_path }}"
        src: "/dev/{{ item.vgroup }}/{{ item.name }}"
        fstype: xfs
        opts: noatime
        state: mounted
      loop: "{{ logical_volumes }}"
```

9.2. Execute the playbook.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ***********************************

TASK [Gathering Facts] *******************************************************
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] **********************************
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] ********************************************
ok: [servera.lab.example.com] => (item={...output omitted...})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] *****************************
skipping: [servera.lab.example.com] => (item={...output omitted...})
skipping: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure XFS Filesystem exists on each LV] ********************************
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] *******************************
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Each Logical Volume is mounted] ****************************************
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

PLAY RECAP *******************************************************************
servera.lab.example.com    : ok=6    changed=2    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

A task is skipped during execution because the playbook was previously executed with the same variable values. The logical volumes did not need to be created.

9.3. Use an Ansible ad hoc command to run the `lsblk` command on the remote host. The output indicates the mount points for the logical volumes.

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                        11:0    1 1024M  0 rom
vda                       252:0    0   10G  0 disk
└─vda1                    252:1    0   10G  0 part /
vdb                       252:16   0    1G  0 disk
└─vdb1                    252:17   0  256M  0 part
  ├─apache--vg-content--lv 253:0    0   64M  0 lvm  /var/www
  └─apache--vg-logs--lv    253:1    0  128M  0 lvm  /var/log/httpd
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

10. Increase the capacity of the `content-lv` logical volume to 128 MiB, and the `logs-lv` logical volume to 256 MiB. This requires increasing the capacity of the `apache-vg` volume group.

Create a new partition with a capacity of 256 MiB and add it to the `apache-vg` volume group.

10.1. Edit the `partitions` variable definition in the `storage_vars.yml` file to add a second partition to the `/dev/vdb` device. The content of the `partitions` variable should be:

```
partitions:
  - number: 1
    start: 1MiB
    end: 257MiB
  - number: 2
    start: 257MiB
    end: 513MiB
```

10.2. Edit the `volume_groups` variable definition in the `storage_vars.yml` file. Add the second partition to list of devices backing the volume group. The content of the `volume_groups` variable should be:

```
volume_groups:
  - name: apache-vg
    devices: /dev/vdb1,/dev/vdb2
```

10.3. Double the capacity of each logical volume defined in the `storage_vars.yml` file. The content of the `logical_volumes` variable should be:

```
logical_volumes:
  - name: content-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 256M
    vgroup: apache-vg
    mount_path: /var/log/httpd
```

10.4. Execute the playbook. Verify the new capacity of each logical volume.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] *********************************

TASK [Gathering Facts] *****************************************************
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ********************************
ok: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={u'start': u'257MiB', u'end':
 u'513MiB', u'number': 2})
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Ensure Volume Groups Exist] ******************************************
changed: [servera.lab.example.com] => (item={u'name': u'apache-vg', u'devices':
 u'/dev/vdb1,/dev/vdb2'})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ***************************
skipping: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size':
 u'128M', u'mount_path': u'/var/www', u'name': u'content-lv'})
skipping: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size':
 u'256M', u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'})

TASK [Ensure XFS Filesystem exists on each LV] *****************************
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] *****************************
changed: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size':
 u'128M', u'mount_path': u'/var/www', u'name': u'content-lv'})
changed: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size':
 u'256M', u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'})

TASK [Each Logical Volume is mounted] **************************************
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

PLAY RECAP *****************************************************************
servera.lab.example.com    : ok=6    changed=3    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

The output indicates changes to the partitions and volume group on the remote host, and that both logical volumes were resized.

10.5. Use an Ansible ad hoc command to run the `lsblk` command on the remote host.

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                        11:0    1 1024M  0 rom
vda                       252:0    0   10G  0 disk
└─vda1                    252:1    0   10G  0 part /
vdb                       252:16   0    1G  0 disk
├─vdb1                    252:17   0  256M  0 part
| ├─apache--vg-content--lv 253:0    0  128M  0 lvm  /var/www
| └─apache--vg-logs--lv    253:1    0  256M  0 lvm  /var/log/httpd
└─vdb2                    252:18   0  256M  0 part
  ├─apache--vg-content--lv 253:0    0  128M  0 lvm  /var/www
  └─apache--vg-logs--lv    253:1    0  256M  0 lvm  /var/log/httpd
```

The output indicates that each logical volume is the correct size and mounted at the correct directory. Two entries exists for each logical volume because files stored on the logical volume may be physically located on either partition (`/dev/vdb1` or `/dev/vdb2`).

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finish

Run the `lab system-storage finish` command to cleanup the managed host.

```
[student@workstation ~]$ lab system-storage finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Managing Network Configuration

Objectives

After completing this section, you should be able to configure network settings and name resolution on managed hosts, and collect network-related Ansible facts.

Configuring Networking with the Network System Role

Red Hat Enterprise Linux 8 includes a collection of system Ansible roles to configure RHEL-based systems. The *rhel-system-roles* package installs those system roles which, for example, support the configuration of time synchronization or networking. You can list the currently installed system roles with the `ansible-galaxy list` command.

```
[user@controlnode ~]$ ansible-galaxy list
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
```

Roles are located in the `/usr/share/ansible/roles` directory. A role beginning with `linux-system-roles` is a symlink to the matching `rhel-system-roles` role.

The network system role supports the configuration of networking on managed hosts. This role supports the configuration of ethernet interfaces, bridge interfaces, bonded interfaces, VLAN interfaces, MacVLAN interfaces, and Infiniband interfaces. The network role is configured with two variables, `network_provider` and `network_connections`.

```
---
network_provider: nm
network_connections:
  - name: ens4
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
```

The `network_provider` variable configures the back end provider, either `nm` (NetworkManager) or `initscripts`. On Red Hat Enterprise Linux 8, the network role uses the `nm` (NetworkManager) as a default networking provider. The `initscripts` provider is used for RHEL 6 systems, and requires the `network` service to be available. The `network_connections` variable configures the different connections, specified as a list of dictionaries, using the interface name as the connection name.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The following table lists the options for the `network_connections` variable.

Option name	Description
name	Identifies the connection profile.
state	The runtime state of a connection profile. Either `up`, if the connection profile is active, or `down` if it is not.
persistent_state	Identifies if a connection profile is persistent. Either `present` if the connection profile is persistent, or `absent` if it is not.
type	Identifies the connection type. Valid values are `ethernet`, `bridge`, `bond`, `team`, `vlan`, `macvlan`, and `infiniband`.
autoconnect	Determines if the connection automatically starts.
mac	Restricts the connection to be used on devices with a specific MAC address.
interface_name	Restricts the connection profile to be used by a specific interface.
zone	Configures the FirewallD zone for the interface.
ip	Determines the IP configuration for the connection. Supports options like for example `address`, to specify a static IP address, or `dns` to configure a DNS server.

The following example uses some of the previous options:

```
network_connections:
- name: eth0 (1)
   persistent_state: present (2)
   type: ethernet (3)
   autoconnect: yes (4)
   mac: 00:00:5e:00:53:5d (5)
   ip:
     address:
       - 172.25.250.40/24 (6)
   zone: external (7)
```

(1) Uses `eth0` as the connection name.

(2) Makes the connection persistent. This is the default value.

(3) Sets the connection type to `ethernet`.

(4) Automatically starts the connection at boot. This is the default value.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

5. Restricts the connection usage to a device with that MAC address.
6. Configures the `172.25.250.40/24` IP address for the connection.
7. Configures the `external` zone as the FirewallD zone of the connection.

To use the network system role, you need to specify the role name under the `roles` clause in your playbook as follows:

```
- name: NIC Configuration
  hosts: webservers
  vars:
    network_connections:
      - name: ens4
        type: ethernet
        ip:
          address:
            - 172.25.250.30/24
  roles:
    - rhel-system-roles.network
```

You can specify variables for the network role with the `vars` clause, as in the previous example, or create a YAML file with those variables under the `group_vars` or `host_vars` directories, depending on your use case.

Configuring Networking with Modules

As an alternative to the `network` system role, Ansible includes modules which support the networking configuration on a system. The `nmcli` module supports the management of both network connections and devices. This module supports the configuration of both teaming and bonding for network interfaces, as well as IPv4 and IPv6 addressing.

The following table lists some of the parameters for the `nmcli` module.

Parameter name	Description
conn_name	Configures the connection name.
autoconnect	Enables automatic connection activation on boot.
dns4	Configures DNS servers for IPv4 (up to 3).
gw4	Configures the IPv4 gateway for the interface.
ifname	Interface to be bound to the connection.
ip4	IP address (IPv4) for the interface.
state	Enables or disables the network interface.
type	Type of device or network connection.

The following example configures a static IP configuration for a network connection and device.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- name: NIC configuration
  nmcli:
    conn_name: ens4-conn (1)
    ifname: ens4 (2)
    type: ethernet (3)
    ip4: 172.25.250.30/24 (4)
    gw4: 172.25.250.1 (5)
    state: present (6)
```

(1) Configures `ens4-conn` as the connection name.

(2) Binds the `ens4-conn` connection to the `ens4` network interface.

(3) Configures the network interface as `ethernet`.

(4) Configures the `172.25.250.30/24` IP address on the interface.

(5) Sets the gateway to `172.25.250.1`.

(6) Makes sure the connection is available.

The `hostname` module sets the hostname for a managed host without modifying the `/etc/hosts` file. This module uses the `name` parameter to specify the new hostname, as on the task shown below:

```
- name: Change hostname
  hostname:
    name: managedhost1
```

The `firewalld` module supports the management of FirewallD on managed hosts. This modules supports the configuration of FirewallD rules for services and ports. It also supports the zone management, including the association or network interfaces and rules to a specific zone.

The following task shows how to create a FirewallD rule for the `http` service on the default zone (`public`). The task configures the rule as permanent, and makes sure it is active.

```
- name: Enabling http rule
  firewalld:
    service: http
    permanent: yes
    state: enabled
```

This task configures the `eth0` in the `external` FirewallD zone.

```
- name: Moving eth0 to external
  firewalld:
    zone: external
    interface: eth0
    permanent: yes
    state: enabled
```

The following table lists some of the parameters for the `firewalld` module.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Parameter name	Description
interface	The interface name to manage with FirewallD.
port	Port or port range. Uses the port/protocol or port-port/protocol format.
rich_rule	Rich rule for FirewallD.
service	Service name to manage with FirewallD.
source	Source network to manage with FirewallD.
zone	The FirewallD zone.
state	Enables or disables a FirewallD configuration.
type	Type of device or network connection.

Ansible Facts for Network Configuration

Ansible uses facts to retrieve information to the control node about the configuration of the managed hosts. You can use the `setup` Ansible module to retrieve all the Ansible facts for a managed host.

```
[user@controlnode ~]$ ansible webservers -m setup
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
    }
```

All network interfaces for a managed host are available under the `ansible_interfaces` element. You can use the `gather_subset=network` parameter for the `setup` module to restrict the facts to those included in the `network` subset. The `filter` option for the `setup` module supports fine-grained filtering based on shell-style wildcards.

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_interfaces'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_interfaces": [
            "ens4",
            "lo",
            "ens3"
        ]
    },
    "changed": false
}
```

The previous command shows that three network interfaces are available on the managed host, `host.lab.example.com`: `lo`, `ens3`, and `ens4`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

You can retrieve additional information about the configuration for a network interface with the `ansible_`*`NIC_name`* filter for the `setup` module. For example, to retrieve the configuration for the `ens4` network interface, use the `ansible_ens4` filter.

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_ens4'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_ens4": {
            "active": true,
            "device": "ens4",
            "features": {
            },
            "hw_timestamp_filters": [],
            "ipv4": {
                "address": "172.25.250.30",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
            "ipv6": [
                {
                    "address": "fe80::5b42:8c94:1fc7:40ae",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "52:54:00:01:fa:0a",
            "module": "virtio_net",
            "mtu": 1500,
            "pciid": "virtio1",
            "promisc": false,
            "speed": -1,
            "timestamping": [
                "tx_software",
                "rx_software",
                "software"
            ],
            "type": "ether"
        }
    },
    "changed": false
}
```

The previous command displays additional configuration details like the IP address configuration both for IPv4 and IPv6, the associated device, and the type.

The following table lists some of the available facts for the `network` subset.

Fact name	Description
ansible_dns	Includes the DNS server(s) IP address, and the search domain(s).

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Fact name	Description
ansible_domain	Includes the subdomain for the managed host.
ansible_all_ipv4_addresses	Includes all the IPv4 addresses configured on the managed host.
ansible_all_ipv6_addresses	Includes all the IPv6 addresses configured on the managed host.
ansible_fqdn	Includes the FQDN for the managed host.
ansible_hostname	Includes the unqualified hostname, the string in the FQDN before the first period.
ansible_nodename	Includes the hostname for the managed host as reported by the system.

Note

Ansible also provides the `inventory_hostname` variable which includes the hostname as configured in Ansible's inventory file.

References

Knowledgebase: Red Hat Enterprise Linux (RHEL) System Roles
https://access.redhat.com/articles/3050101

Linux System Roles
https://linux-system-roles.github.io/

nmcli Module Documentation
https://docs.ansible.com/ansible/2.9/modules/nmcli_module.html

hostname Module Documentation
https://docs.ansible.com/ansible/2.9/modules/hostname_module.html

firewalld Module Documentation
https://docs.ansible.com/ansible/2.9/modules/firewalld_module.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Guided Exercise

Managing Network Configuration

In this exercise, you will adjust the network configuration of a managed host and collect information about it on a file created by a template.

Outcomes

You should be able to configure network settings and name resolution on managed hosts, and collect network-related Ansible facts.

Before You Begin

Run the `lab system-network start` script from `workstation` to configure the environment for the exercise. The script creates the `system-network` working directory, and downloads the Ansible configuration file and the host inventory file needed for the exercise.

```
[student@workstation ~]$ lab system-network start
```

Instructions

1. Review the inventory file at the `/home/student/system-network` directory.

   1.1. As the `student` user on `workstation`, change to the `/home/student/system-network` working directory.

```
[student@workstation ~]$ cd ~/system-network
[student@workstation system-network]$
```

   1.2. Verify that `servera.lab.example.com` is part of the `webservers` host group. This server has a spare network interface.

```
[student@workstation system-network]$ cat inventory
[webservers]
servera.lab.example.com
```

2. Use the `ansible-galaxy` command to verify that system roles are available. If no roles are available, you need to install the *rhel-system-roles* package.

```
[student@workstation system-network]$ ansible-galaxy list
# /usr/share/ansible/roles
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
 [WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

3. Create a playbook which uses the linux-system-roles.network role to configure the spare network interface `eth1` on `servera.lab.example.com` with the `172.25.250.30` IP address.

   3.1. Create a playbook, `playbook.yml`, with one play that targets the `webservers` host group. Include the `rhel-system-roles.network` role in the `roles` section of the play.

```
---
- name: NIC Configuration
  hosts: webservers

  roles:
    - rhel-system-roles.network
```

   3.2. Review the *Role Variables* section of the `README.md` file for the `rhel-system-roles.network` role. Determine the role variables to configure the `eth1` network interface with the `172.25.250.30` IP address.

```
[student@workstation system-network]$ cat \
> /usr/share/doc/rhel-system-roles/network/README.md
...output omitted...
Setting the IP configuration:
...output omitted...
```

   3.3. Create the `group_vars/webservers` subdirectory.

```
[student@workstation system-network]$ mkdir -pv group_vars/webservers
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/webservers'
```

   3.4. Create a new file `network.yml` to define role variables. Because these variable values apply to the hosts on the `webservers` host group, you need to create that file in the `group_vars/webservers` directory. Add variable definitions to support the configuration of the `eth1` network interface. The file now contains:

```
---
network_connections:
  - name: eth1
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
```

   3.5. Run the playbook to configure the secondary network interface on `servera`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation system-network]$ ansible-playbook playbook.yml

PLAY [NIC Configuration] ***************************************************

TASK [Gathering Facts] *****************************************************
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which services are running] ************
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which packages are installed] **********
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Print network provider] **********************
ok: [servera.lab.example.com] => {
    "msg": "Using network provider: nm"
}

TASK [rhel-system-roles.network : Install packages] ****************************
skipping: [servera.lab.example.com]

TASK [rhel-system-roles.network : Enable network service] **********************
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Configure networking connection profiles] ****
...output omitted...

changed: [servera.lab.example.com]

TASK [rhel-system-roles.network : Re-test connectivity] ************************
ok: [servera.lab.example.com]

PLAY RECAP *****************************************************************
servera.lab.example.com    : ok=7    changed=1    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

4. Use the Ansible `setup` module on an Ansible adhoc command to verify that the `eth1` network interface configuration on `servera` is correct.

   4.1. Use the `setup` Ansible module to list all the Ansible facts available for `servera`. Filter results for the `eth1` network interface with the `-a 'filter=`*`filter_string`*`'` option. Verify that the `eth1` network interface uses the `172.25.250.30` IP address. It may take up to a minute to configure the IP address.

```
[student@workstation system-network]$ ansible webservers -m setup \
> -a 'filter=ansible_eth1'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth1": {
...output omitted...
            "ipv4": {
                "address": "172.25.250.30",
                "broadcast": "172.25.250.255",
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
...output omitted...
```

Finish

On `workstation`, run the `lab system-network finish` script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab system-network finish
```

This concludes the guided exercise.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Automating Linux Administration Tasks

Performance Checklist

In this lab, you will configure and perform administrative tasks on managed hosts using a playbook.

Outcomes

You should be able to create playbooks for configuring on a managed host a software repository, users and groups, logical volumes, cron jobs, and additional network interfaces.

Before You Begin

On `workstation`, run the lab start script to confirm that the environment is ready for the lab to begin. The script creates the working directory, called `system-review`, and populates it with an Ansible configuration file, a host inventory, and lab files.

```
[student@workstation ~]$ lab system-review start
```

Instructions

1. Create and execute on the `webservers` host group a playbook which configures the Yum internal repository located at `http://materials.example.com/yum/repository`, and installs the *example-motd* package available in that repository. All RPM packages are signed with an organizational GPG key pair. The GPG public key is available at `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.
2. Create and execute on the `webservers` host group a playbook which creates the `webadmin` user group, and add two users to that group, `ops1` and `ops2`.
3. Create and execute on the `webservers` host group a playbook that uses the `/dev/vdb` device to create a volume group named `apache-vg`. This playbook also creates two logical volumes, named `content-lv` and `logs-lv`, both backed by the `apache-vg` volume group. Finally, it creates an XFS file system on each logical volume, and mounts the `content-lv` logical volume at `/var/www`, and the `logs-lv` logical volume at `/var/log/httpd`. The lab script populates two files in `~/system-review`, `storage.yml` which provides an initial skeleton for the playbook, and `storage_vars.yml` which provides values to all the variables required by the different modules.
4. Create and execute on the `webservers` host group a playbook which uses the `cron` module to create the `/etc/cron.d/disk_usage` crontab file that schedules a recurring cron job. The job should run as the `devops` user every two minutes between `09:00` and `16:59` on `Monday` through `Friday`. The job should append the current disk usage to the file `/home/devops/disk_usage`.
5. Create and execute on the `webservers` host group a playbook which uses the `linux-system-roles.network` role to configure with the `172.25.250.40/24` IP address the spare network interface, `eth1`.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Evaluation

Run `lab system-review grade` on `workstation` to grade your work.

```
[student@workstation ~]$ lab system-review grade
```

Finish

From `workstation`, run the `lab system-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab system-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Automating Linux Administration Tasks

Performance Checklist

In this lab, you will configure and perform administrative tasks on managed hosts using a playbook.

Outcomes

You should be able to create playbooks for configuring on a managed host a software repository, users and groups, logical volumes, cron jobs, and additional network interfaces.

Before You Begin

On `workstation`, run the lab start script to confirm that the environment is ready for the lab to begin. The script creates the working directory, called `system-review`, and populates it with an Ansible configuration file, a host inventory, and lab files.

```
[student@workstation ~]$ lab system-review start
```

Instructions

1. Create and execute on the `webservers` host group a playbook which configures the Yum internal repository located at `http://materials.example.com/yum/repository`, and installs the *example-motd* package available in that repository. All RPM packages are signed with an organizational GPG key pair. The GPG public key is available at `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

   1.1. As the `student` user on `workstation`, change to the `/home/student/system-review` working directory.

```
[student@workstation ~]$ cd ~/system-review
[student@workstation system-review]$
```

   1.2. Create the `repo_playbook.yml` playbook which runs on the managed hosts at the `webservers` host group. Add a task that uses the `yum_repository` module to ensure the configuration of the internal yum repository on the remote host. Ensure that:

   - The repository's configuration is stored in the file `/etc/yum.repos.d/example.repo`
   - The repository ID is `example-internal`
   - The base URL is `http://materials.example.com/yum/repository`
   - The repository is configured to check RPM GPG signatures
   - The repository description is `Example Inc. Internal YUM repo`

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

The playbook contains the following:

```
---
- name: Repository Configuration
  hosts: webservers
  tasks:
    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
        gpgcheck: yes
```

1.3. Add a second task to the play that uses the `rpm_key` module to ensure that the repository public key is present on the remote host. The repository public key URL is `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

The second task appears as follows:

```
    - name: Ensure Repo RPM Key is Installed
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present
```

1.4. Add a third task to install the *example-motd* package available in the Yum internal repository.

The third task appears as follows:

```
    - name: Install Example motd package
      yum:
        name: example-motd
        state: present
```

1.5. Execute the playbook:

```
[student@workstation system-review]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [Ensure Example Repo exists] **********************************************
changed: [serverb.lab.example.com]

TASK [Ensure Repo RPM Key is Installed] ****************************************
changed: [serverb.lab.example.com]

TASK [Install Example motd package] ********************************************
changed: [serverb.lab.example.com]
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=4    changed=3    unreachable=0    failed=0
```

2. Create and execute on the `webservers` host group a playbook which creates the `webadmin` user group, and add two users to that group, `ops1` and `ops2`.

2.1. Create a `vars/users_vars.yml` variable file, which defines two users, `ops1` and `ops2`, which belong to the `webadmin` user group. You may need to create the `vars` subdirectory.

```
[student@workstation system-review]$ mkdir vars
[student@workstation system-review]$ vi vars/users_vars.yml
---
users:
  - username: ops1
    groups: webadmin
  - username: ops2
    groups: webadmin
```

2.2. Create the `users.yml` playbook. Define a single play in the playbook that targets the `webservers` host group. Add a `vars_files` clause that defines the location of the `vars/users_vars.yml` filename. Add a task which uses the `group` module to create the `webadmin` user group on the remote host.

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  tasks:
    - name: Add webadmin group
      group:
        name: webadmin
        state: present
```

2.3. Add a second task to the playbook that uses the `user` module to create the users. Add a `loop: "{{ users }}"` clause to the task to loop through the variable file for every username found in the `vars/users_vars.yml` file. As the `name:` for the users, use the `item.username` the variable name. This way the variable file may contain additional information that might be useful for creating the users, such as the groups that the users should belong to. The second task contains the following:

```
    - name: Create user accounts
      user:
        name: "{{ item.username }}"
        groups: webadmin
      loop: "{{ users }}"
```

2.4. Execute the playbook:

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation system-review]$ ansible-playbook users.yml

PLAY [Create multiple local users] *********************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [Add webadmin group] ******************************************************
changed: [serverb.lab.example.com]

TASK [Create user accounts] ****************************************************
changed: [serverb.lab.example.com] => (item={'username': 'ops1', 'groups':
 'webadmin'})
changed: [serverb.lab.example.com] => (item={'username': 'ops2', 'groups':
 'webadmin'})

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0
```

3. Create and execute on the `webservers` host group a playbook that uses the `/dev/vdb` device to create a volume group named `apache-vg`. This playbook also creates two logical volumes, named `content-lv` and `logs-lv`, both backed by the `apache-vg` volume group. Finally, it creates an XFS file system on each logical volume, and mounts the `content-lv` logical volume at `/var/www`, and the `logs-lv` logical volume at `/var/log/httpd`. The lab script populates two files in `~/system-review`, `storage.yml` which provides an initial skeleton for the playbook, and `storage_vars.yml` which provides values to all the variables required by the different modules.

   3.1. Review the `storage_vars.yml` variables file.

```
[student@workstation system-review]$ cat storage_vars.yml
---

partitions:
  - number: 1
    start: 1MiB
    end: 257MiB

volume_groups:
  - name: apache-vg
    devices: /dev/vdb1

logical_volumes:
  - name: content-lv
    size: 64M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/log/httpd
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

This file describes the intended structure of partitions, volume groups, and logical volumes on each web server. The first partition begins at an offset of 1 MiB from the beginning of the `/dev/vdb` device, and ends at an offset of 257 MiB, for a total size of 256 MiB.

Each web server has one volume group, named `apache-vg`, containing the first partition of the `/dev/vdb` device.

Each web server has two logical volumes. The first logical volume is named `content-lv`, with a size of 64 MiB, attached to the `apache-vg` volume group, and mounted at `/var/www`. The second logical volume is named `content-lv`, with a size of 128 MiB, attached to the `apache-vg` volume group, and mounted at `/var/log/httpd`.

Note

The `apache-vg` volume group has a capacity of 256 MiB, because it is backed by the `/dev/vdb1` partition. It provides enough capacity for both of the logical volumes.

3.2. Change the first task in the `storage.yml` playbook to use the `parted` module to configure a partition for each loop item. Each item describes an intended partition of the `/dev/vdb` device on each web server:

number
The partition number. Use this as the value of the `number` keyword for the `parted` module.

start
The start of the partition, as an offset from the beginning of the block device. Use this as the value of the `part_start` keyword for the `parted` module.

end
The end of the partition, as an offset from the beginning of the block device. Use this as the value of the `part_end` keyword for the `parted` module.

The content of the first task should be:

```
- name: Correct partitions exist on /dev/vdb
  parted:
    device: /dev/vdb
    state: present
    number: "{{ item.number }}"
    part_start: "{{ item.start }}"
    part_end: "{{ item.end }}"
  loop: "{{ partitions }}"
```

3.3. Change the second task of the play to use the `lvg` module to configure a volume group for each loop item. Each item of the `volume_groups` variable describes a volume group that should exist on each web server:

name
The name of the volume group. Use this as the value of the `vg` keyword for the `lvg` module.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

devices
A comma-separated list of devices or partitions that form the volume group. Use this as the value of the `pvs` keyword for the `lvg` module.

The content of the second task should be:

```
- name: Ensure Volume Groups Exist
  lvg:
    vg: "{{ item.name }}"
    pvs: "{{ item.devices }}"
  loop: "{{  volume_groups }}"
```

3.4. Change the third task to use the `lvol` module. Set the volume group name, logical volume name, and logical volume size using each item's keywords. The content of the third task is now:

```
- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
```

3.5. Change the fourth task to use the `filesystem` module. Configure the task to ensure that each logical volume is formatted as an XFS file system. Recall that a logical volume is associated with the logical device `/dev/<volume group name>/<logical volume name>`.

The content of the fourth task should be:

```
- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"
```

3.6. Configure the fifth task to ensure each logical volume has the correct storage capacity. If the logical volume increases in capacity, be sure to force the expansion of the volume's file system.

Warning
If a logical volume needs to decrease in capacity, this task will fail because an XFS file system does not support shrinking capacity.

The content of the fifth task should be:

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: Ensure the correct capacity for each LV
      lvol:
        vg: "{{ item.vgroup }}"
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
        resizefs: yes
        force: yes
      loop: "{{ logical_volumes }}"
```

3.7. Use the `mount` module in the sixth task to ensure that each logical volume is mounted at the corresponding mount path and persists after a reboot.

The content of the sixth task should be:

```
    - name: Each Logical Volume is mounted
      mount:
        path: "{{ item.mount_path }}"
        src: "/dev/{{ item.vgroup }}/{{ item.name }}"
        fstype: xfs
        state: mounted
      loop: "{{ logical_volumes }}"
```

3.8. Execute the playbook to create the logical volumes on the remote host.

```
[student@workstation system-review]$ ansible-playbook storage.yml
PLAY [Ensure Apache Storage Configuration] ************************************

TASK [Gathering Facts] ********************************************************
ok: [serverb.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ***********************************
changed: [serverb.lab.example.com] => (item={'number': 1, 'start': '1MiB', 'end':
 '257MiB'})

TASK [Ensure Volume Groups Exist] *********************************************
changed: [serverb.lab.example.com] => (item={'name': 'apache-vg', 'devices': '/
dev/vdb1'})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ******************************
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M',
 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M',
 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Ensure XFS Filesystem exists on each LV] ********************************
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M',
 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M',
 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Ensure the correct capacity for each LV] ********************************
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
ok: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M',
 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
ok: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M',
 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Each Logical Volume is mounted] ****************************************
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M',
 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M',
 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

PLAY RECAP *******************************************************************
serverb.lab.example.com    : ok=7    changed=5    unreachable=0    failed=0
```

4. Create and execute on the `webservers` host group a playbook which uses the `cron` module to create the `/etc/cron.d/disk_usage` crontab file that schedules a recurring cron job. The job should run as the `devops` user every two minutes between `09:00` and `16:59` on `Monday` through `Friday`. The job should append the current disk usage to the file `/home/devops/disk_usage`.

    4.1. Create a new playbook, `create_crontab_file.yml`, and add the lines needed to start the play. It should target the managed hosts in the `webservers` group and enable privilege escalation.

```
---
- name: Recurring cron job
  hosts: webservers
  become: true
```

    4.2. Define a task that uses the `cron` module to schedule a recurring cron job.

Note

The `cron` module provides a `name` option to uniquely describe the crontab file entry and to ensure expected results. The description is added to the crontab file. For example, the `name` option is required if you are removing a crontab entry using `state=absent`. Additionally, when the default state, `state=present` is set, the `name` option prevents a new crontab entry from always being created, regardless of existing ones.

```
  tasks:
    - name: Crontab file exists
      cron:
        name: Add date and time to a file
```

    4.3. Configure the job to run every two minutes between `09:00` and `16:59` on `Monday` through `Friday`.

```
        minute: "*/2"
        hour: 9-16
        weekday: 1-5
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4.4. Use the `cron_file` parameter to use the `/etc/cron.d/disk_usage` crontab file instead of an individual user's crontab in `/var/spool/cron/`. A relative path will place the file in `/etc/cron.d` directory. If the `cron_file` parameter is used, you must also specify the `user` parameter.

```
        user: devops
        job: df >> /home/devops/disk_usage
        cron_file: disk_usage
        state: present
```

4.5. When completed, the playbook should appear as follows. Review the playbook for accuracy.

```
---
- name: Recurring cron job
  hosts: webservers
  become: true

  tasks:
    - name: Crontab file exists
      cron:
        name: Add date and time to a file
        minute: "*/2"
        hour: 9-16
        weekday: 1-5
        user: devops
        job: df >> /home/devops/disk_usage
        cron_file: disk_usage
        state: present
```

4.6. Run the playbook.

```
[student@workstation system-review]$ ansible-playbook create_crontab_file.yml
PLAY [Recurring cron job] ******************************************************

TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [Crontab file exists] *****************************************************
changed: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

5. Create and execute on the `webservers` host group a playbook which uses the `linux-system-roles.network` role to configure with the `172.25.250.40/24` IP address the spare network interface, `eth1`.

5.1. Use `ansible-galaxy` to verify that system roles are available. If not, you need to install the *rhel-system-roles* package.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation system-review]$ ansible-galaxy list
# /usr/share/ansible/roles
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
 [WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

5.2. Create a playbook, `network_playbook.yml`, with one play that targets the `webservers` host group. Include the `rhel-system-roles.network` role in the `roles` section of the play.

```
---
- name: NIC Configuration
  hosts: webservers

  roles:
    - rhel-system-roles.network
```

5.3. Create the `group_vars/webservers` subdirectory.

```
[student@workstation system-review]$ mkdir -pv group_vars/webservers
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/webservers'
```

5.4. Create a new file `network.yml` to define role variables. Because these variable values apply to the hosts on the `webservers` host group, you need to create that file in the `group_vars/webservers` directory. Add variable definitions to support the configuration of the `eth1` network interface. The file now contains:

```
[student@workstation system-review]$ vi group_vars/webservers/network.yml
---
network_connections:
  - name: eth1
    type: ethernet
    ip:
      address:
        - 172.25.250.40/24
```

5.5. Run the playbook to configure the secondary network interface.

```
[student@workstation system-review]$ ansible-playbook network_playbook.yml

PLAY [NIC Configuration] ***********************************************
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
TASK [Gathering Facts] *********************************************************
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Check which services are running] ************
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Check which packages are installed] **********
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Print network provider] **********************
ok: [serverb.lab.example.com] => {
    "msg": "Using network provider: nm"
}

TASK [rhel-system-roles.network : Install packages] ****************************
skipping: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Enable network service] **********************
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Configure networking connection profiles] ****
 [WARNING]: [002] <info>  #0, state:None persistent_state:present, 'eth1': add
 connection
eth1, 38d63afd-e610-4929-ba1b-1d38413219fb

changed: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Re-test connectivity] ************************
ok: [serverb.lab.example.com]

PLAY RECAP *********************************************************************
serverb.lab.example.com    : ok=7    changed=1    unreachable=0    failed=0
```

5.6. Verify that the `eth1` network interface uses the `172.25.250.40` IP address. It may take up to a minute to configure the IP address.

```
[student@workstation system-review]$ ansible webservers -m setup \
> -a 'filter=ansible_eth1'
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth1": {
...output omitted...
            "ipv4": {
                "address": "172.25.250.40",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
...output omitted...
```

Evaluation

Run `lab system-review grade` on `workstation` to grade your work.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ lab system-review grade
```

Finish

From `workstation`, run the `lab system-review finish` script to clean up the resources created in this lab.

```
[student@workstation ~]$ lab system-review finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Summary

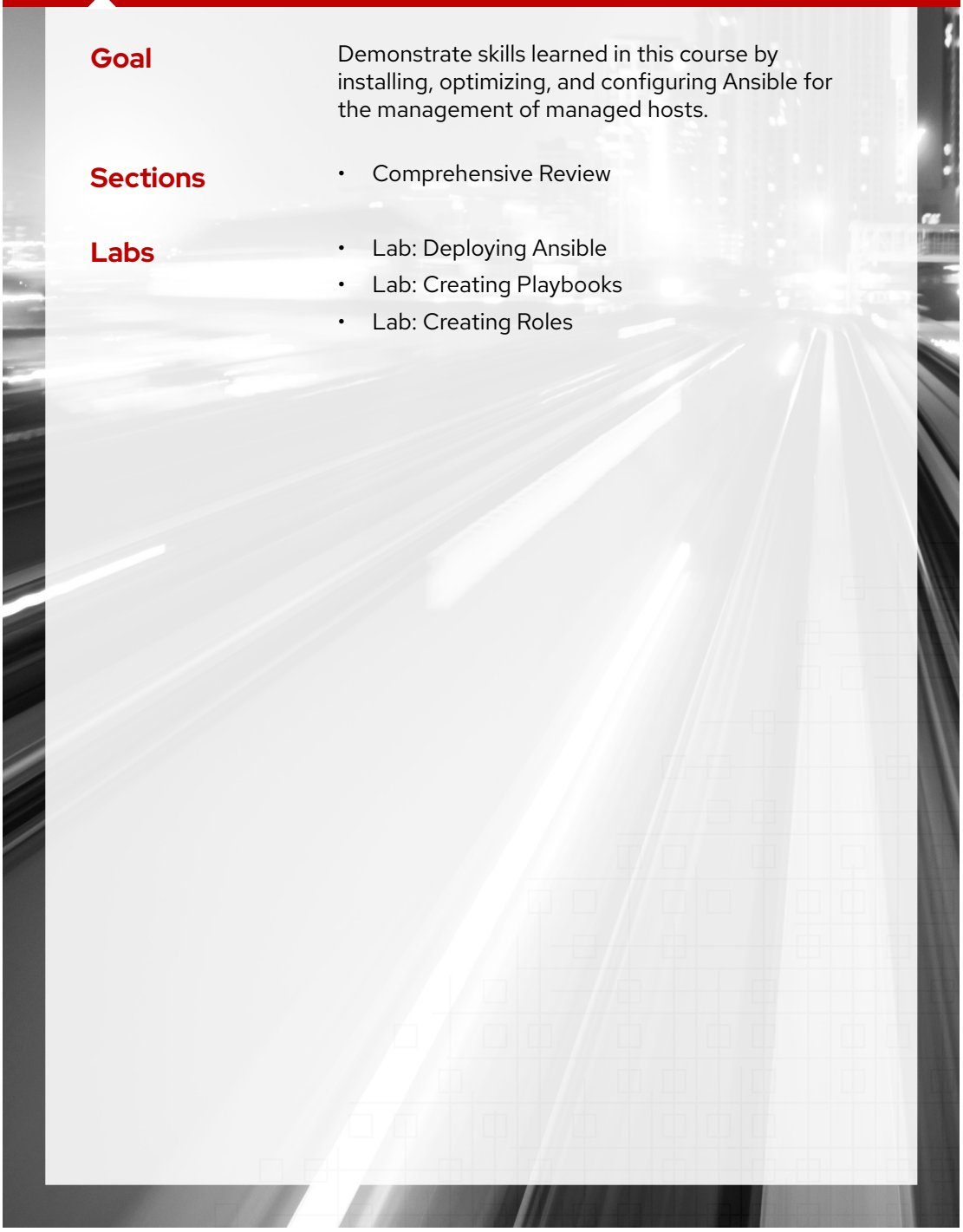
In this chapter, you learned:

- The `yum_repository` module configures a Yum repository on a managed host. For repositories that use public keys, you can verify that the key is available with the `rpm_key` module.
- The `user` and `group` modules create users and groups respectively on a managed host. You can configure authorized keys for a user with the `authorized_key` module.
- Cron jobs can be configured on managed hosts with the `cron` module.
- Ansible supports the configuration of logical volumes with the `lvg`, and `lvol` modules. The `parted` and `filesystem` modules support respectively the partition of devices and creation of filesystems.
- Red Hat Enterprise Linux 8 includes the network system role which supports the configuration of network interfaces on managed hosts.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Chapter 10

Comprehensive Review: Linux Automation with Ansible



Goal

Demonstrate skills learned in this course by installing, optimizing, and configuring Ansible for the management of managed hosts.

Sections

- Comprehensive Review

Labs

- Lab: Deploying Ansible
- Lab: Creating Playbooks
- Lab: Creating Roles

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Comprehensive Review

Objectives

After completing this section, you should be able to demonstrate proficiency with knowledge and skills learned in *Red Hat Enterprise Linux Automation with Ansible*.

Reviewing Red Hat System Administration III: Linux Automation with Ansible

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter.

Refer to earlier sections in the textbook for extra study.

Chapter 1, *Introducing Ansible*

Describe the fundamental concepts of Ansible and how it is used, and install Red Hat Ansible Automation Platform.

- Describe the motivation for automating Linux administration tasks with Ansible, fundamental Ansible concepts, and Ansible's basic architecture.
- Install Ansible on a control node and describe the distinction between community Ansible and Red Hat Ansible Automation Platform.

Chapter 2, *Implementing an Ansible Playbook*

Create an inventory of managed hosts, write a simple Ansible Playbook, and run the playbook to automate tasks on those hosts.

- Describe Ansible inventory concepts and manage a static inventory file.
- Describe where Ansible configuration files are located, how Ansible selects them, and edit them to apply changes to default settings.
- Run a single Ansible automation task using an ad hoc command and explain some use cases for ad hoc commands.
- Write a basic Ansible Playbook and run it using the `ansible-playbook` command.
- Write a playbook that uses multiple plays and per-play privilege escalation, and effectively use ansible-doc to learn how to use new modules to implement tasks for a play.

Chapter 3, *Managing Variables and Facts*

Write playbooks that use variables to simplify management of the playbook and facts to reference information about managed hosts.

- Create and reference variables that affect particular hosts or host groups, the play, or the global environment, and describe how variable precedence works.
- Encrypt sensitive variables using Ansible Vault, and run playbooks that reference Vault-encrypted variable files.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Reference data about managed hosts using Ansible facts, and configure custom facts on managed hosts.

Chapter 4, *Implementing Task Control*

Manage task control, handlers, and task errors in Ansible Playbooks.

- Use loops to write efficient tasks and use conditions to control when to run tasks.
- Implement a task that runs only when another task changes the managed host.
- Control what happens when a task fails, and what conditions cause a task to fail.

Chapter 5, *Deploying Files to Managed Hosts*

Deploy, manage, and adjust files on hosts managed by Ansible.

- Create, install, edit, and remove files on managed hosts, and manage permissions, ownership, SELinux context, and other characteristics of those files.
- Deploy files to managed hosts that are customized by using Jinja2 templates.

Chapter 6, *Managing Complex Plays and Playbooks*

Write playbooks for larger, more complex plays and playbooks.

- Write sophisticated host patterns to efficiently select hosts for a play or ad hoc command.
- Manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.

Chapter 7, *Simplifying Playbooks with Roles*

Use Ansible roles to develop playbooks more quickly and to reuse Ansible code.

- Describe what a role is, how it is structured, and how you can use it in a playbook.
- Write playbooks that take advantage of Red Hat Enterprise Linux System Roles to perform standard operations.
- Create a role in a playbook's project directory and run it as part of one of the plays in the playbook.
- Select and retrieve roles from Ansible Galaxy or other sources such as a Git repository, and use them in your playbooks.
- Obtain a set of related roles, supplementary modules, and other content from content collections, and use them in a playbook.

Chapter 8, *Troubleshooting Ansible*

Troubleshoot playbooks and managed hosts.

- Troubleshoot generic issues with a new playbook and repair them.
- Troubleshoot failures on managed hosts when running a playbook.

Chapter 9, *Automating Linux Administration Tasks*

Automate common Linux system administration tasks with Ansible.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

- Subscribe systems, configure software channels and repositories, enable module streams, and manage RPM packages on managed hosts.
- Manage Linux users and groups, configure SSH, and modify Sudo configuration on managed hosts.
- Manage service startup, schedule processes with at, cron, and systemd, reboot, and control the default boot target on managed hosts.
- Partition storage devices, configure LVM, format partitions or logical volumes, mount file systems, and add swap files or spaces.
- Configure network settings and name resolution on managed hosts, and collect network-related Ansible facts.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Deploying Ansible

In this review, you will install Ansible on `workstation`, 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 managed hosts.

Outcomes

You should be able to:

- Install Ansible.
- Use ad hoc commands to perform actions on managed hosts.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-deploy start` command. This script ensures that the managed hosts, `servera` and `serverb`, are reachable on the network. The script creates a lab subdirectory named `review-deploy` in the student's home directory.

```
[student@workstation ~]$ lab review-deploy start
```

Instructions

1. Install Ansible on `workstation` so that it can serve the control node.
2. On the control node, create an inventory file, `/home/student/review-deploy/inventory`, containing a group called `dev`. This group should consist of the managed hosts `servera.lab.example.com` and `serverb.lab.example.com`.
3. Create the Ansible configuration file in `/home/student/review-deploy/ansible.cfg`. The configuration file should reference the `/home/student/review-deploy/inventory` inventory file.
4. Execute an ad hoc command using privilege escalation to modify the contents of the `/etc/motd` file on `servera` and `serverb` to contain the string `Managed by Ansible\n`. Use `devops` as the remote user.
5. Execute an ad hoc command to verify that the contents of the `/etc/motd` file on `servera` and `serverb` are identical.

Evaluation

On `workstation`, run the `lab review-deploy grade` command to confirm success on this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab review-deploy grade
```

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Finish

On `workstation`, run the `lab review-deploy finish` command to clean up this exercise.

```
[student@workstation ~]$ lab review-deploy finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Deploying Ansible

In this review, you will install Ansible on `workstation`, 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 managed hosts.

Outcomes

You should be able to:

- Install Ansible.
- Use ad hoc commands to perform actions on managed hosts.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-deploy start` command. This script ensures that the managed hosts, `servera` and `serverb`, are reachable on the network. The script creates a lab subdirectory named `review-deploy` in the student's home directory.

```
[student@workstation ~]$ lab review-deploy start
```

Instructions

1. Install Ansible on `workstation` so that it can serve the control node.

```
[student@workstation ~]$ sudo yum install ansible
[sudo] password for student:
Loaded plugins: langpacks, search-disabled-repos
Resolving Dependencies
--> Running transaction check
...output omitted...
Is this ok [y/d/N]: y
...output omitted...
```

2. On the control node, create an inventory file, `/home/student/review-deploy/inventory`, containing a group called `dev`. This group should consist of the managed hosts `servera.lab.example.com` and `serverb.lab.example.com`.

   2.1. Change directory into the Ansible project directory, `/home/student/review-deploy`, created by the setup script.

```
[student@workstation ~]$ cd ~/review-deploy
```

   2.2. Create the `inventory` file with the following content.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

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

3. Create the Ansible configuration file in `/home/student/review-deploy/ansible.cfg`. The configuration file should reference the `/home/student/review-deploy/inventory` inventory file.

   Add the following entries to configure the inventory file `./inventory` as the inventory source. Save the changes and exit the text editor.

```
[defaults]
inventory=./inventory
```

4. Execute an ad hoc command using privilege escalation to modify the contents of the `/etc/motd` file on `servera` and `serverb` to contain the string `Managed by Ansible\n`. Use `devops` as the remote user.

```
[student@workstation review-deploy]$ ansible dev -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -b -u devops
servera.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "unconfined_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/...output omitted...",
    "state": "file",
    "uid": 0
}
serverb.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "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/...output omitted...",
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    "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 review-deploy]$ ansible dev -m command -a "cat /etc/motd"
servera.lab.example.com | CHANGED | rc=0 >>
Managed by Ansible

serverb.lab.example.com | CHANGED | rc=0 >>
Managed by Ansible
```

Evaluation

On `workstation`, run the `lab review-deploy grade` command to confirm success on this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab review-deploy grade
```

Finish

On `workstation`, run the `lab review-deploy finish` command to clean up this exercise.

```
[student@workstation ~]$ lab review-deploy finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Creating Playbooks

In this review, you will create three playbooks in the Ansible project directory, `/home/student/review-playbooks`. One playbook will ensure that *lftp* is installed on systems that should be FTP clients, one playbook will ensure that *vsftpd* is installed and configured on systems that should be FTP servers, and one playbook (`site.yml`) will run both of the other playbooks.

Outcomes

You should be able to:

- Create and execute playbooks to perform tasks on managed hosts.
- Utilize Jinja2 templates, variables, and handlers in playbooks.

Important

If you are having trouble with your `site.yml` playbook, make sure that both `ansible-vsftpd.yml` and `ftpclients.yml` use consistent indentation.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-playbooks start` command.

```
[student@workstation ~]$ lab review-playbooks start
```

Instructions

1. As the `student` user on `workstation`, create the inventory file `/home/student/review-playbooks/inventory`, containing `serverc.lab.example.com` in the `ftpclients` group, and `serverb.lab.example.com` and `serverd.lab.example.com` in the `ftpservers` group.
2. Create the Ansible configuration file, `/home/student/review-playbooks/ansible.cfg`, and populate it with the necessary entries to meet these requirements:
   - Configure the Ansible project to use the newly created inventory
   - Connect to managed hosts as the `devops` user
   - Utilize privilege escalation using `sudo` as the `root` user
   - Escalate privileges for each task by default
3. Create the playbook, `/home/student/review-playbooks/ftpclients.yml`, containing a play that targets hosts in the `ftpclients` inventory group and ensures that the *lftp* package is installed.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

4. Place the provided vsftpd configuration file, `vsftpd.conf.j2`, in the `templates` subdirectory.
5. Place the provided `defaults-template.yml` file in the `vars` subdirectory.
6. Create a `vars.yml` variable definition file in the `vars` subdirectory to define the following three variables and their values:

Variable	Value
vsftpd_package	vsftpd
vsftpd_service	vsftpd
vsftpd_config_file	/etc/vsftpd/vsftpd.conf

7. Using the previously created Jinja2 template and variable definition files, create a second playbook, `/home/student/review-playbooks/ansible-vsftpd.yml`, to configure the vsftpd service on the hosts in the `ftpservers` inventory group.
8. Create a third playbook, `/home/student/review-playbooks/site.yml`, and include the plays from the two playbooks created previously, `ftpclients.yml` and `ansible-vsftpd.yml`.
9. Execute the `/home/student/review-playbooks/site.yml` playbook to verify that it performs the desired tasks on the managed hosts.

Evaluation

As the `student` user on `workstation`, run the `lab review-playbooks grade` command to confirm success of this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab review-playbooks grade
```

Finish

Run the `lab review-playbooks finish` command to clean up the lab tasks on `serverb`, `serverc`, and `serverd`.

```
[student@workstation ~]$ lab review-playbooks finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Creating Playbooks

In this review, you will create three playbooks in the Ansible project directory, `/home/student/review-playbooks`. One playbook will ensure that *lftp* is installed on systems that should be FTP clients, one playbook will ensure that *vsftpd* is installed and configured on systems that should be FTP servers, and one playbook (`site.yml`) will run both of the other playbooks.

Outcomes

You should be able to:

- Create and execute playbooks to perform tasks on managed hosts.
- Utilize Jinja2 templates, variables, and handlers in playbooks.

Important

If you are having trouble with your `site.yml` playbook, make sure that both `ansible-vsftpd.yml` and `ftpclients.yml` use consistent indentation.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-playbooks start` command.

```
[student@workstation ~]$ lab review-playbooks start
```

Instructions

1. As the `student` user on `workstation`, create the inventory file `/home/student/review-playbooks/inventory`, containing `serverc.lab.example.com` in the `ftpclients` group, and `serverb.lab.example.com` and `serverd.lab.example.com` in the `ftpservers` group.

    1.1. Change directory into the Ansible project directory, `/home/student/review-playbooks`, created by the setup script.

```
[student@workstation ~]$ cd ~/review-playbooks
```

    1.2. Populate the `inventory` file with the following entries, and then save and exit.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[ftpservers]
serverb.lab.example.com
serverd.lab.example.com

[ftpclients]
serverc.lab.example.com
```

2. Create the Ansible configuration file, `/home/student/review-playbooks/ansible.cfg`, and populate it with the necessary entries to meet these requirements:

   - Configure the Ansible project to use the newly created inventory
   - Connect to managed hosts as the `devops` user
   - Utilize privilege escalation using `sudo` as the `root` user
   - Escalate privileges for each task by default

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

[privilege_escalation]
become_user = root
become_method = sudo
become = true
```

3. Create the playbook, `/home/student/review-playbooks/ftpclients.yml`, containing a play that targets hosts in the `ftpclients` inventory group and ensures that the *lftp* package is installed.

```
---
- name: Ensure FTP Client Configuration
  hosts: ftpclients

  tasks:
    - name: latest version of lftp is installed
      yum:
        name: lftp
        state: latest
```

4. Place the provided vsftpd configuration file, `vsftpd.conf.j2`, in the `templates` subdirectory.

   4.1. Create the `templates` subdirectory.

```
[student@workstation review-playbooks]$ mkdir -v templates
mkdir: created directory 'templates'
```

   4.2. Move the `vsftpd.conf.j2` file to the newly created `templates` subdirectory.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation review-playbooks]$ mv -v vsftpd.conf.j2 templates/
renamed 'vsftpd.conf.j2' -> 'templates/vsftpd.conf.j2'
```

5. Place the provided `defaults-template.yml` file in the `vars` subdirectory.

   5.1. Create the `vars` subdirectory.

```
[student@workstation review-playbooks]$ mkdir -v vars
mkdir: created directory 'vars'
```

   5.2. Move the `defaults-template.yml` file to the newly created `vars` subdirectory.

```
[student@workstation review-playbooks]$ mv -v defaults-template.yml vars/
renamed 'defaults-template.yml' -> 'vars/defaults-template.yml'
```

6. Create a `vars.yml` variable definition file in the `vars` subdirectory to define the following three variables and their values:

Variable	Value
vsftpd_package	vsftpd
vsftpd_service	vsftpd
vsftpd_config_file	/etc/vsftpd/vsftpd.conf

```
vsftpd_package: vsftpd
vsftpd_service: vsftpd
vsftpd_config_file: /etc/vsftpd/vsftpd.conf
```

7. Using the previously created Jinja2 template and variable definition files, create a second playbook, `/home/student/review-playbooks/ansible-vsftpd.yml`, to configure the vsftpd service on the hosts in the `ftpservers` inventory group.

```
---
- name: FTP server is installed
  hosts:
    - ftpservers
  vars_files:
    - vars/defaults-template.yml
    - vars/vars.yml

  tasks:
    - name: Packages are installed
      yum:
        name: "{{ vsftpd_package }}"
        state: present

    - name: Ensure service is started
      service:
        name: "{{ vsftpd_service }}"
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
        state: started
        enabled: true

    - name: Configuration file is installed
      template:
        src: templates/vsftpd.conf.j2
        dest: "{{ vsftpd_config_file }}"
        owner: root
        group: root
        mode: 0600
        setype: etc_t
      notify: restart vsftpd

    - name: firewalld is installed
      yum:
        name: firewalld
        state: present

    - name: firewalld is started and enabled
      service:
        name: firewalld
        state: started
        enabled: yes

    - name: FTP port is open
      firewalld:
        service: ftp
        permanent: true
        state: enabled
        immediate: yes

    - name: FTP passive data ports are open
      firewalld:
        port: 21000-21020/tcp
        permanent: yes
        state: enabled
        immediate: yes

  handlers:
    - name: restart vsftpd
      service:
        name: "{{ vsftpd_service }}"
        state: restarted
```

8. Create a third playbook, `/home/student/review-playbooks/site.yml`, and include the plays from the two playbooks created previously, `ftpclients.yml` and `ansible-vsftpd.yml`.

```
---
# FTP Servers playbook
- import_playbook: ansible-vsftpd.yml

# FTP Clients playbook
- import_playbook: ftpclients.yml
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

9. Execute the `/home/student/review-playbooks/site.yml` playbook to verify that it performs the desired tasks on the managed hosts.

```
[student@workstation review-playbooks]$ ansible-playbook site.yml
```

Evaluation

As the `student` user on `workstation`, run the `lab review-playbooks grade` command to confirm success of this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab review-playbooks grade
```

Finish

Run the `lab review-playbooks finish` command to clean up the lab tasks on `serverb`, `serverc`, and `serverd`.

```
[student@workstation ~]$ lab review-playbooks finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Lab

Creating Roles

In this review, you will convert the `ansible-vsftpd.yml` playbook into a role, and then use that role in a new playbook that will also run some additional tasks.

Outcomes

You should be able to:

- Create a role to configure the `vsftpd` service using tasks from an existing playbook.
- Include a role in a playbook, and execute the playbook.

Important

You may find it useful to debug your role by testing it in a playbook that does not contain the extra tasks or playbook variables listed above, but instead contains a play that only targets hosts in the group `ftpservers` and applies the role.

After confirming that a simplified playbook using only the role works just like the original `ansible-vsftpd.yml` playbook, you can build the complete `vsftpd-configure.yml` playbook by adding the additional variables and tasks specified above.

Important

If you are having trouble with your `site.yml` playbook, make sure that both `vsftpd-configure.yml` and `ftpclients.yml` use consistent indentation.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-roles start` command. This script ensures that the remote hosts are reachable on the network. The script also checks that Ansible is installed on `workstation`, creates a directory structure for the lab environment, and installs required lab files.

```
[student@workstation ~]$ lab review-roles start
```

Instructions

1. Change to the `review-roles` working directory. Configure the Ansible project to use the static inventory file `inventory`. Verify the inventory configuration using the `ansible-inventory` command.
2. Convert the `ansible-vsftpd.yml` playbook to the role `ansible-vsftpd`.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

3. Update the contents of the `roles/ansible-vsftpd/meta/main.yml` file.

Variable	Value
author	Red Hat Training
description	example role for RH294
company	Red Hat
license	BSD

4. Modify the contents of the `roles/ansible-vsftpd/README.md` file so that it provides pertinent information regarding the role. After modification, the file should contain the following.

```
ansible-vsftpd
=========
Example ansible-vsftpd role from Red Hat's "Linux Automation" (RH294)
course.

Role Variables
--------------

* defaults/main.yml contains variables used to configure the vsftpd.conf template
* vars/main.yml contains the name of the vsftpd service, the name of the RPM
package, and the location of the service's configuration file

Dependencies
------------

None.

Example Playbook
----------------

    - hosts: servers
      roles:
        - ansible-vsftpd

License
-------

BSD

Author Information
------------------

Red Hat (training@redhat.com)
```

5. Remove the unused directories from the new role.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

6. Create the new playbook `vsftpd-configure.yml`. It should contain the following.

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  vars:
    vsftpd_anon_root: /mnt/share/
    vsftpd_local_root: /mnt/share/

  roles:
    - ansible-vsftpd

  tasks:
    - name: /dev/vdb1 is partitioned
      parted:
        device: /dev/vdb
        number: 1
        label: gpt
        part_start: 1MiB
        part_end: 100%
        state: present

    - name: XFS file system exists on /dev/vdb1
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: yes

    - name: anon_root mount point exists
      file:
        path: '{{ vsftpd_anon_root }}'
        state: directory

    - name: /dev/vdb1 is mounted on anon_root
      mount:
        path: '{{ vsftpd_anon_root }}'
        src: /dev/vdb1
        fstype: xfs
        state: mounted
        dump: '1'
        passno: '2'
      notify: restart vsftpd

    - name: Make sure permissions on mounted fs are correct
      file:
        path: '{{ vsftpd_anon_root }}'
        owner: root
        group: root
        mode: '0755'
        setype: "{{ vsftpd_setype }}"
        state: directory

    - name: Copy README to the ftp anon_root
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    copy:
      dest: '{{ vsftpd_anon_root }}/README'
      content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"
      setype: '{{ vsftpd_setype }}'
```

7. Change the `site.yml` playbook to use the newly created `vsftpd-configure.yml` playbook instead of the `ansible-vsftpd.yml` playbook.

8. Verify that the `site.yml` playbook works as intended by executing it with `ansible-playbook`.

Evaluation

From `workstation`, run the `lab review-roles grade` command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab review-roles grade
```

Finish

Run the `lab review-roles finish` command to clean up the lab tasks on `servera` and `serverb`.

```
[student@workstation ~]$ lab review-roles finish
```

This concludes the lab.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Solution

Creating Roles

In this review, you will convert the `ansible-vsftpd.yml` playbook into a role, and then use that role in a new playbook that will also run some additional tasks.

Outcomes

You should be able to:

- Create a role to configure the `vsftpd` service using tasks from an existing playbook.
- Include a role in a playbook, and execute the playbook.

Important

You may find it useful to debug your role by testing it in a playbook that does not contain the extra tasks or playbook variables listed above, but instead contains a play that only targets hosts in the group `ftpservers` and applies the role.

After confirming that a simplified playbook using only the role works just like the original `ansible-vsftpd.yml` playbook, you can build the complete `vsftpd-configure.yml` playbook by adding the additional variables and tasks specified above.

Important

If you are having trouble with your `site.yml` playbook, make sure that both `vsftpd-configure.yml` and `ftpclients.yml` use consistent indentation.

Before You Begin

Log in to `workstation` as `student` using `student` as the password.

On `workstation`, run the `lab review-roles start` command. This script ensures that the remote hosts are reachable on the network. The script also checks that Ansible is installed on `workstation`, creates a directory structure for the lab environment, and installs required lab files.

```
[student@workstation ~]$ lab review-roles start
```

Instructions

1. Change to the `review-roles` working directory. Configure the Ansible project to use the static inventory file `inventory`. Verify the inventory configuration using the `ansible-inventory` command.

   1.1. Change to the `review-roles` working directory.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation ~]$ cd ~/review-roles
[student@workstation review-roles]$
```

1.2. Edit the `ansible.cfg` file, add the `inventory` directive in the `[defaults]` section, and set it to `./inventory`.

The `[defaults]` section of the `ansible.cfg` file looks like this:

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

1.3. Use the `ansible-inventory` command to verify the project inventory configuration:

```
[student@workstation review-roles]$ ansible-inventory --list all
{
    "_meta": {
        "hostvars": {}
    },
    "all": {
        "children": [
            "ftpclients",
            "ftpservers",
            "ungrouped"
        ]
    },
    "ftpclients": {
        "hosts": [
            "servera.lab.example.com",
            "serverc.lab.example.com"
        ]
    },
    "ftpservers": {
        "hosts": [
            "serverb.lab.example.com",
            "serverd.lab.example.com"
        ]
    }
}
```

2. Convert the `ansible-vsftpd.yml` playbook to the role `ansible-vsftpd`.

2.1. Create the `roles` subdirectory.

```
[student@workstation review-roles]$ mkdir -v roles
mkdir: created directory 'roles'
```

2.2. Using `ansible-galaxy`, create the directory structure for the new `ansible-vsftpd` role in the `roles` subdirectory.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
[student@workstation review-roles]$ cd roles
[student@workstation roles]$ ansible-galaxy init ansible-vsftpd
- Role ansible-vsftpd was created successfully
[student@workstation roles]$ cd ..
[student@workstation review-roles]$
```

2.3. Using `tree`, verify the directory structure created for the new role.

```
[student@workstation review-roles]$ tree roles
roles
└── ansible-vsftpd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
    │   └── main.yml
    ├── templates
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml

9 directories, 8 files
```

2.4. Replace the `roles/ansible-vsftpd/defaults/main.yml` file with the variable definitions in the `defaults-template.yml` file.

```
[student@workstation review-roles]$ mv -v defaults-template.yml \
> roles/ansible-vsftpd/defaults/main.yml
renamed 'defaults-template.yml' -> 'roles/ansible-vsftpd/defaults/main.yml'
```

2.5. Replace the `roles/ansible-vsftpd/vars/main.yml` file with the variable definitions in the `vars.yml` file.

```
[student@workstation review-roles]$ mv -v vars.yml \
> roles/ansible-vsftpd/vars/main.yml
renamed 'vars.yml' -> 'roles/ansible-vsftpd/vars/main.yml'
```

2.6. Use the `templates/vsftpd.conf.j2` file as a template for the `ansible-vsftpd` role.

```
[student@workstation review-roles]$ mv -v vsftpd.conf.j2 \
> roles/ansible-vsftpd/templates/
renamed 'vsftpd.conf.j2' -> 'roles/ansible-vsftpd/templates/vsftpd.conf.j2'
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2.7. Copy tasks from the `ansible-vsftpd.yml` playbook to the `roles/ansible-vsftpd/tasks/main.yml` file. The value of the `src` keyword in the `template` module task no longer needs to reference the `templates` subdirectory. The `roles/ansible-vsftpd/tasks/main.yml` file should contain the following when you finish.

```
---
# tasks file for ansible-vsftpd
- name: Packages are installed
  yum:
    name: '{{ vsftpd_package }}'
    state: present

- name: Ensure service is started
  service:
    name: '{{ vsftpd_service }}'
    state: started
    enabled: true

- name: Configuration file is installed
  template:
    src: vsftpd.conf.j2
    dest: '{{ vsftpd_config_file }}'
    owner: root
    group: root
    mode: '0600'
    setype: etc_t
  notify: restart vsftpd

- name: firewalld is installed
  yum:
    name: firewalld
    state: present

- name: firewalld is started and enabled
  service:
    name: firewalld
    state: started
    enabled: yes

- name: FTP port is open
  firewalld:
    service: ftp
    permanent: true
    state: enabled
    immediate: yes

- name: Passive FTP data ports allowed through the firewall
  firewalld:
    port: 21000-21020/tcp
    permanent: yes
    state: enabled
    immediate: yes
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

2.8. Copy the handlers from the `ansible-vsftpd.yml` playbook to the `roles/ansible-vsftpd/handlers/main.yml` file. The `roles/ansible-vsftpd/handlers/main.yml` file should contain the following when you finish.

```
---
# handlers file for ansible-vsftpd
- name: restart vsftpd
  service:
    name: "{{ vsftpd_service }}"
    state: restarted
```

3. Update the contents of the `roles/ansible-vsftpd/meta/main.yml` file.

Variable	Value
author	Red Hat Training
description	example role for RH294
company	Red Hat
license	BSD

3.1. Change the value of the `author` entry to `Red Hat Training`.

```
author: Red Hat Training
```

3.2. Change the value of the `description` entry to `example role for RH294`.

```
description: example role for RH294
```

3.3. Change the value of the `company` entry to `Red Hat`.

```
company: Red Hat
```

3.4. Change the value of the `license:` entry to `BSD`.

```
license: BSD
```

4. Modify the contents of the `roles/ansible-vsftpd/README.md` file so that it provides pertinent information regarding the role. After modification, the file should contain the following.

```
ansible-vsftpd
=========
Example ansible-vsftpd role from Red Hat's "Linux Automation" (RH294)
course.

Role Variables
--------------

* defaults/main.yml contains variables used to configure the vsftpd.conf template
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
* vars/main.yml contains the name of the vsftpd service, the name of the RPM
package, and the location of the service's configuration file

Dependencies
------------

None.

Example Playbook
----------------

    - hosts: servers
      roles:
        - ansible-vsftpd

License
-------

BSD

Author Information
------------------

Red Hat (training@redhat.com)
```

5. Remove the unused directories from the new role.

```
[student@workstation review-roles]$ rm -rvf roles/ansible-vsftpd/tests
removed 'roles/ansible-vsftpd/tests/inventory'
removed 'roles/ansible-vsftpd/tests/test.yml'
removed directory: 'roles/ansible-vsftpd/tests'
```

6. Create the new playbook `vsftpd-configure.yml`. It should contain the following.

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  vars:
    vsftpd_anon_root: /mnt/share/
    vsftpd_local_root: /mnt/share/

  roles:
    - ansible-vsftpd

  tasks:
    - name: /dev/vdb1 is partitioned
      parted:
        device: /dev/vdb
        number: 1
        label: gpt
        part_start: 1MiB
        part_end: 100%
        state: present
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

```
    - name: XFS file system exists on /dev/vdb1
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: yes

    - name: anon_root mount point exists
      file:
        path: '{{ vsftpd_anon_root }}'
        state: directory

    - name: /dev/vdb1 is mounted on anon_root
      mount:
        path: '{{ vsftpd_anon_root }}'
        src: /dev/vdb1
        fstype: xfs
        state: mounted
        dump: '1'
        passno: '2'
      notify: restart vsftpd

    - name: Make sure permissions on mounted fs are correct
      file:
        path: '{{ vsftpd_anon_root }}'
        owner: root
        group: root
        mode: '0755'
        setype: "{{ vsftpd_setype }}"
        state: directory

    - name: Copy README to the ftp anon_root
      copy:
        dest: '{{ vsftpd_anon_root }}/README'
        content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"
        setype: '{{ vsftpd_setype }}'
```

7. Change the `site.yml` playbook to use the newly created `vsftpd-configure.yml` playbook instead of the `ansible-vsftpd.yml` playbook.

```
---
# FTP Servers playbook
- import_playbook: vsftpd-configure.yml

# FTP Clients playbook
- import_playbook: ftpclients.yml
```

8. Verify that the `site.yml` playbook works as intended by executing it with `ansible-playbook`.

```
[student@workstation review-roles]$ ansible-playbook site.yml
```

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Evaluation

From `workstation`, run the `lab review-roles grade` command to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab review-roles grade
```

Finish

Run the `lab review-roles finish` command to clean up the lab tasks on `servera` and `serverb`.

```
[student@workstation ~]$ lab review-roles finish
```

This concludes the lab.

For use by La Pyayt lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Appendix A

Supplementary Topics

Goal Investigate supplementary topics not included in the official course.

Sections

- Examining Ansible Configuration Options

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Examining Ansible Configuration Options

Objectives

After completing this section, you should be able to use `ansible-config` to discover and investigate configuration options and to determine which options have been modified from the default settings.

Viewing Configuration Options

If you want to find out what options are available in the configuration file, use the `ansible-config list` command. It will display an exhaustive list of the available configuration options and their default settings. This list may vary depending on the version of Ansible that you have installed and whether you have any additional Ansible plugins on your control node.

Each option displayed by `ansible-config list` will have a number of key-value pairs associated with it. These key-value pairs provide information on how that option works. For example, the option `ACTION_WARNINGS` displays the following key-value pairs:

Key	Value	Purpose
`description`	[By default Ansible will issue a warning when received from a task action (module or action plugin). These warnings can be silenced by adjusting this setting to False.]	Describes what this configuration option is for.
`type`	`boolean`	What the type is for the option: `boolean` means true-false value.
`default`	`true`	The default value for this option.
`version_added`	`2.5`	The version of Ansible that added this option, for backward compatibility.
`ini`	`{ key: action_warnings, section: defaults }`	Which section of the INI-like inventory file contains this option, and the name of the option in the configuration file (`action_warnings`, in the `defaults` section).
`env`	`ANSIBLE_ACTION_WARNINGS`	If this environment variable is set, it will override any setting of the option made in the configuration file.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Determining Modified Configuration Options

When working with configuration files, you might want to find out which options have been set to values which are different from the built-in defaults.

You can do this by running the `ansible-config dump -v --only-changed` command. The `-v` option displays the location of the `ansible.cfg` file used when processing the command. The `ansible-config` command follows the same order of precedence mentioned previously for the `ansible` command. Output will vary depending on the location of the `ansible.cfg` file and which directory the `ansible-config` command is ran from.

In the following example, there is a single ansible configuration file located at `/etc/ansible/ansible.cfg`. The `ansible-config` command is first ran from student's home directory, then from a working directory with the same results:

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/
share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/
share/ansible/roles']
```

However, if you have a custom `ansible.cfg` file in your working directory, the same command will display information based on where it is ran from and the relative `ansible.cfg` file.

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/
share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ cat ansible.cfg
[defaults]
inventory = ./inventory
remote_user = devops

[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
Using /home/student/workingdirectory/ansible.cfg as config file
DEFAULT_HOST_LIST(/home/student/workingdirectory/ansible.cfg) = [u'/home/student/
workingdirectory/inventory']
DEFAULT_REMOTE_USER(/home/student/workingdirectory/ansible.cfg) = devops
```

References

`ansible-config`(1) man page

Configuration file: Ansible Documentation
https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Appendix B

Ansible Lightbulb Licensing

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.

Ansible Lightbulb License

Portions of this course were adapted from the Ansible Lightbulb project. The original material from that project is available from https://github.com/ansible/lightbulb under the following MIT License:

Copyright 2017 Red Hat, Inc.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For use by La Pyayt lapyayt2015 lapyayt@infratechmm.com Copyright © 2022 Red Hat, Inc.