





Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



Network with tens of thousands of community members



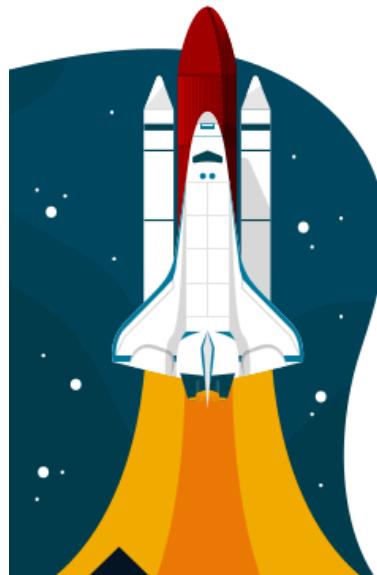
Engage in thousands of active conversations and posts



Join and interact with hundreds of certified training instructors



Unlock badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

Connect with your instructor - and your classmates - before, after, and during your training course.

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.



Developing Advanced Automation with Red Hat Ansible Automation Platform

Red Hat Ansible Automation Platform 2.2 DO374

Developing Advanced Automation with Red Hat Ansible

Automation Platform

Edition 2 20240606

Publication date 20240606

Authors: Michael Phillips, Hervé Quatremain, Karan Rai,
Alejandra Ramírez Palacios, Dallas Spohn, Morgan Weetman,
Mike Kelly, Antonio Marí, Steven Bonneville
Course Architect: Steven Bonneville
DevOps Engineer: Dan Kolepp
Editor: David O'Brien

© 2024 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are © 2024 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 [mailto: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, 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 American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP 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 a 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 the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors:Sajith Eyamkuzhy Sugathan, James Mighion, Richard Allred, Greg Hosler

Document Conventions	ix
Admonitions	ix
Inclusive Language	x
Introduction	xi
Developing Advanced Automation with Red Hat Ansible Automation Platform	xi
Orientation to the Classroom Environment	xii
Performing Lab Exercises	xv
1. Developing Playbooks with Ansible Automation Platform 2	1
Red Hat Ansible Automation Platform 2	2
Quiz: Red Hat Ansible Automation Platform 2	7
Running Playbooks with Automation Content Navigator	9
Guided Exercise: Running Playbooks with Automation Content Navigator	17
Managing Ansible Project Materials Using Git	22
Guided Exercise: Managing Ansible Project Materials Using Git	34
Implementing Recommended Ansible Practices	40
Guided Exercise: Implementing Recommended Ansible Practices	47
Lab: Developing Playbooks with Ansible Automation Platform	55
Summary	63
2. Managing Content Collections and Execution Environments	65
Reusing Content from Ansible Content Collections	66
Guided Exercise: Reusing Content from Ansible Content Collections	72
Finding and Installing Ansible Content Collections	77
Guided Exercise: Finding and Installing Ansible Content Collections	87
Selecting an Execution Environment	94
Guided Exercise: Selecting an Execution Environment	98
Lab: Managing Content Collections and Execution Environments	103
Summary	114
3. Running Playbooks with Automation Controller	115
Explaining the Automation Controller Architecture	116
Quiz: Explaining the Automation Controller Architecture	121
Running Playbooks in Automation Controller	123
Guided Exercise: Running Playbooks in Automation Controller	132
Lab: Running Playbooks with Automation Controller	137
Summary	143
4. Working with Ansible Configuration Settings	145
Examining the Ansible Configuration with Automation Content Navigator	146
Guided Exercise: Examining the Ansible Configuration with Automation Content Navigator	150
Configuring Automation Content Navigator	155
Guided Exercise: Configuring Automation Content Navigator	160
Lab: Working with Ansible Configuration Settings	167
Summary	173
5. Managing Inventories	175
Managing Dynamic Inventories	176
Guided Exercise: Managing Dynamic Inventories	183
Writing YAML Inventory Files	187
Guided Exercise: Writing YAML Inventory Files	194
Managing Inventory Variables	197
Guided Exercise: Managing Inventory Variables	206
Lab: Managing Inventories	212
Summary	221

6. Managing Task Execution	223
Controlling Privilege Escalation	224
Guided Exercise: Controlling Privilege Escalation	230
Controlling Task Execution	236
Guided Exercise: Controlling Task Execution	244
Running Selected Tasks	252
Guided Exercise: Running Selected Tasks	257
Optimizing Execution for Speed	260
Guided Exercise: Optimizing Execution for Speed	272
Lab: Managing Task Execution	276
Summary	285
7. Transforming Data with Filters and Plug-ins	287
Processing Variables Using Filters	288
Guided Exercise: Processing Variables Using Filters	297
Templating External Data Using Lookups	305
Guided Exercise: Templating External Data Using Lookups	311
Implementing Advanced Loops	316
Guided Exercise: Implementing Advanced Loops	323
Using Filters to Work with Network Addresses	329
Guided Exercise: Using Filters to Work with Network Addresses	336
Lab: Transforming Data with Filters and Plug-ins	346
Summary	355
8. Coordinating Rolling Updates	357
Delegating Tasks and Facts	358
Guided Exercise: Delegating Tasks and Facts	362
Configuring Parallelism	365
Guided Exercise: Configuring Parallelism	368
Managing Rolling Updates	374
Guided Exercise: Managing Rolling Updates	380
Lab: Coordinating Rolling Updates	391
Summary	399
9. Creating Content Collections and Execution Environments	401
Writing Ansible Content Collections	402
Guided Exercise: Writing Ansible Content Collections	410
Building a Custom Automation Execution Environment	417
Guided Exercise: Building a Custom Automation Execution Environment	425
Validating a Custom Execution Environment	433
Guided Exercise: Validating a Custom Execution Environment	438
Using Custom Content Collections and Execution Environments in Automation Controller	444
Guided Exercise: Using Custom Content Collections and Execution Environments in Automation Controller	452
Lab: Creating Content Collections and Execution Environments	460
Summary	472
10. Comprehensive Review	473
Comprehensive Review	474
Lab: Managing Inventory Variables to Use with Automation Content Navigator	477
Lab: Optimizing a Playbook for Large-scale Use	486
Lab: Creating and Using Ansible Content Collections and Automation Execution Environments	501

Document Conventions

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

Admonitions

Red Hat Training courses use the following admonitions:



References

These describe where to find external documentation that is relevant to a subject.



Note

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



Important

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



Warning

Do not ignore warnings. 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 that are covered in Red Hat Training courses. Red Hat appreciates your patience during this process.

Introduction

Developing Advanced Automation with Red Hat Ansible Automation Platform

Developing Advanced Automation with Red Hat Ansible Automation Platform (DO374) is designed for automation content developers to leverage the new, container-focused tools from Red Hat Ansible Automation Platform to efficiently develop automation that can be managed by the automation controller.

Course Objectives

- Leverage capabilities of the automation content navigator to develop Ansible Playbooks.
- Apply recommended practices for effective and efficient automation with Ansible.
- Use advanced features of Ansible to work with data, including filters and plugins.
- Perform automation operations as rolling updates.
- Create automation execution environments to bundle and distribute the dependencies needed by automation code.

Audience

- This course is designed for users who create recommended design patterns and operate automation practices at scale, including these roles:
 - DevOps engineers.
 - Linux system administrators.
 - Developers.
 - Other IT professionals with basic expertise using Red Hat Ansible Automation Platform to automate, provision, configure, and deploy applications and services in a Linux environment.

Prerequisites

- Red Hat Certified Engineer (RHCE) certification (EX294) on Red Hat Enterprise Linux 8 or later, or equivalent Ansible experience.

Orientation to the Classroom Environment

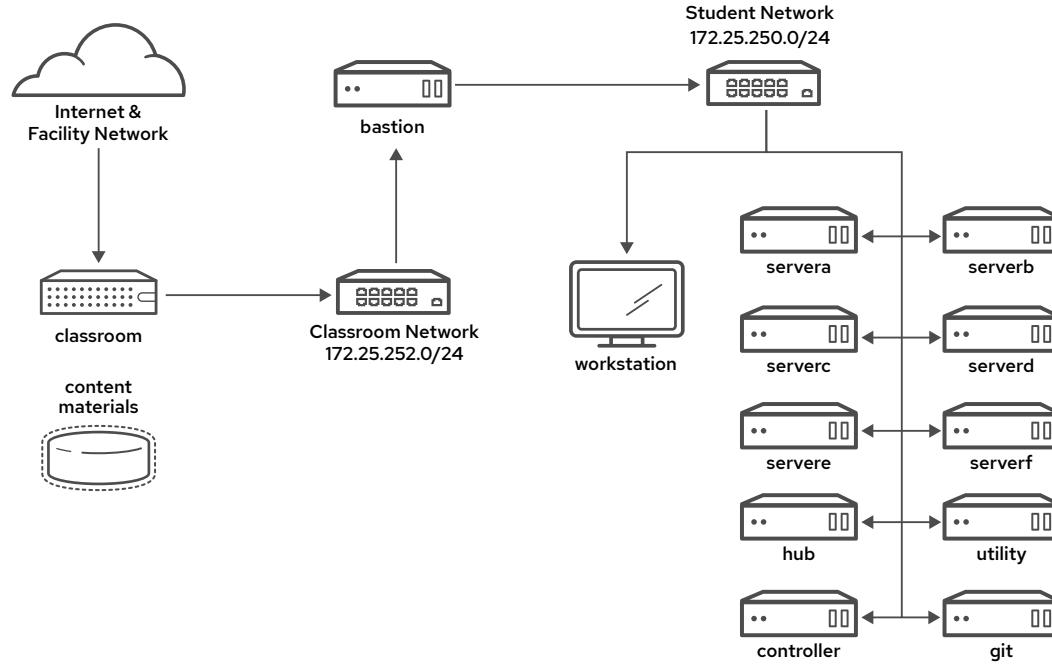


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is **workstation**. The system called **bastion** must always be running. Ten other machines are also used by students for these activities: **servera**, **serverb**, **serverc**, **serverd**, **servere**, **serverf**, **git**, **hub**, **controller**, and **utility**. All ten of these systems are in the `lab.example.com` DNS domain.

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

Classroom Machines

Machine name	IP addresses	Role
bastion.lab.example.com	172.25.250.254	Bridges the classroom and student networks
workstation.lab.example.com	172.25.250.9	Graphical workstation used for system administration
servera.lab.example.com	172.25.250.10	Managed server "A"
serverb.lab.example.com	172.25.250.11	Managed server "B"

Machine name	IP addresses	Role
serverc.lab.example.com	172.25.250.12	Managed server "C"
serverd.lab.example.com	172.25.250.13	Managed server "D"
servere.lab.example.com	172.25.250.14	Managed server "E"
serverf.lab.example.com	172.25.250.15	Managed server "F"
git.lab.example.com	172.25.250.5	GitLab server
hub.lab.example.com	172.25.250.6	Private automation hub server
controller.lab.example.com	172.25.250.7	Automation controller server
utility.lab.example.com	172.25.250.8	System with utility services required for classroom

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 are only 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.

Text Editors on the Student Workstation

The hands-on activities in this course require you to create and edit a number of text files. The text editors included with Red Hat Enterprise Linux are available in the software repository, but they are not all installed. Activities were developed assuming that most students use the Vim text editor, started at a shell prompt by using either the **vi** or **vim** command.

As an experiment based on customer input, the authors of this course have installed Visual Studio Code on **workstation.lab.example.com**. You can launch Visual Studio Code on **workstation** by selecting its icon from the **Activities** menu or by running the **code** command at a shell prompt.

Visual Studio Code is not included in Red Hat Enterprise Linux, but is a third-party product provided for free download by Microsoft at <https://code.visualstudio.com>. For this course, a number of extensions have also been installed. The most notable are "Ansible VS Code Extension" and "YAML Language Support", both developed by Red Hat.



Important

We do not guarantee that Visual Studio Code is available in the EX374 certification exam.

If you plan to take that exam, you should ensure that you are comfortable with a text editor included in Red Hat Enterprise Linux.

We have performed limited testing of Visual Studio Code in this lab environment. If you have issues, you might need to use a different editor to work around them.

Controlling Your Systems

In an instructor-led training classroom, you are assigned a physical computer (`foundationX.ilt.example.com`), which is used to access your virtual machines that run on that host. You are automatically logged in to the physical machine as the `kiosk` user, with the password `redhat`. The classroom systems that you work with are virtual machines running on that host.

On `foundationX`, you use the `rht-vmctl` command to work with the virtual machines. Run the commands in the following table as the `kiosk` user on `foundationX`. These commands can be used with any of the virtual machines.

rht-vmctl Commands

Action	Command
Start the virtual machine named <code>server</code> .	<code>rht-vmctl start server</code>
Display the "physical console" for the virtual machine named <code>server</code> , to log in and work on that system	<code>rht-vmview view server</code>
Reset the virtual machine named <code>server</code> to its initial course state and restart it.	<code>rht-vmctl reset server</code>

At the start of a lab exercise, if the instruction "reset your servera" appears, then run the `rht-vmctl reset servera` command on the `foundationX` system as the `kiosk` user. Likewise, if the instruction "reset your workstation" appears, then run the `rht-vmctl reset workstation` command on `foundationX` as the `kiosk` user.

Performing Lab Exercises

There are four types of lab activities that you might see in this course.

- A *guided exercise* is a hands-on practice exercise that follows a presentation section. It walks you through a procedure to perform, step-by-step.
- A *quiz* is typically used when checking knowledge-based learning, or when a hands-on activity is impractical for some other reason.
- An *end-of-chapter lab* is a gradable hands-on activity to help you check your learning. You are provided with a set of high-level steps to perform, based on the guided exercises in that chapter, but the steps do not walk you through every command. You are also given a solution that provides a step-by-step walk-through.
- A *comprehensive review lab* is used at the end of the course. It is also a gradable hands-on activity, but it might cover content from throughout the entire course. You are provided with a specification that details what you need to accomplish in the activity, but not the specific steps to do so. Again, you are given a solution that provides a step-by-step walk-through that meets the specification.

To prepare your lab environment at the start of each hands-on activity, run the `lab start` command with an activity name specified by the activity's instructions. Likewise, at the end of each hands-on activity, run the `lab finish` command with that same activity name to clean up after the activity. Each hands-on activity has a unique name within a course.

The syntax for running an exercise script is:

```
[student@workstation ~]$ lab action exercise
```

The *action* is a choice of `start`, `grade`, or `finish`. All exercises support `start` and `finish` actions. Only end-of-chapter labs and comprehensive review labs support the `grade` action.

start

The `start` action verifies the required resources to begin an exercise. It might include configuring settings, creating resources, ensuring prerequisite services are running, and verifying necessary outcomes from previous exercises. You can take an exercise at any time, even without taking the preceding exercises.

grade

For gradable activities, the `grade` action directs the `lab` command to evaluate your work, and displays a list of grading criteria with a `PASS` or `FAIL` status for each. To achieve a `PASS` status for all criteria, fix any failures and rerun the `grade` action.

finish

The `finish` action cleans up resources configured during the exercise. You can take an exercise as many times as you want.

The `lab` command supports tab completion. For example, to list all exercises that you can start, enter `lab start` and press the Tab key twice.

Chapter 1

Developing Playbooks with Ansible Automation Platform 2

Goal

Develop Ansible Playbooks with Red Hat Ansible Automation Platform 2 following recommended practices.

Objectives

- Describe the architecture of Red Hat Ansible Automation Platform 2 and how its new features help with Ansible automation development.
- Install automation content navigator and use it to run an existing playbook with a supported execution environment.
- Create and manage Ansible Playbooks in a Git repository, following recommended practices.
- Demonstrate and describe common recommended practices for developing and maintaining effective Ansible automation solutions.

Sections

- Red Hat Ansible Automation Platform 2 (and Quiz)
- Running Playbooks with Automation Content Navigator (and Guided Exercise)
- Managing Ansible Project Materials Using Git (and Guided Exercise)
- Implementing Recommended Ansible Practices (and Guided Exercise)

Lab

- Developing Playbooks with Ansible Automation Platform

Red Hat Ansible Automation Platform 2

Objectives

- Describe the architecture of Red Hat Ansible Automation Platform 2 and how its new features help with Ansible automation development.

Orientation to Red Hat Ansible Automation Platform 2

Red Hat Ansible Automation Platform 2 is the next evolution in automation from Red Hat. Featuring new tools, services, and capabilities that offer a whole new level of customization and control, it delivers an elevated automation experience that expands the boundaries of what is possible for your enterprise.

Red Hat Ansible Automation Platform 2 can help you achieve goals across many departments.

- Automate for velocity and accelerate business outcomes.
- Automate for collaboration and orchestrate across teams.
- Automate for growth and innovate at scale.

Red Hat Ansible Automation Platform 2 Components

Red Hat Ansible Automation Platform 2 includes a number of distinct components that together provide a complete and integrated set of automation tools and resources.

Ansible Core

Ansible Core provides the fundamental functionality used to run Ansible Playbooks. It defines the automation language that is used to write Ansible Playbooks in YAML text files. It provides the key functions such as loops, conditionals, and other Ansible imperatives needed for automation code. It also provides the framework and basic command-line tools to drive automation.

Red Hat Ansible Automation Platform 2.2 provides Ansible Core 2.13 in the `ansible-core` RPM package.

Ansible Content Collections

Historically, Ansible provided numerous modules as part of the core package; an approach referred to in the Ansible community as "batteries included". However, with the success and rapid growth of Ansible, the number of modules included with Ansible grew exponentially. This led to certain challenges with support, especially because users sometimes wanted to use earlier or later versions of modules than were packaged with a particular version of Ansible.

The upstream developers decided to reorganize most modules into separate *Ansible Content Collections*, made up of related modules, roles, and plug-ins that are supported by the same group of developers. Ansible Core itself is limited to a small set of modules provided by the `ansible.builtin` Ansible Content Collection, which is always part of Ansible Core.

This provides users with the flexibility to select different versions of collections, or different sets of collections, based on their needs. It also provides developers with the ability to update their modules on a separate cadence from Ansible itself.

Certain collections, *Red Hat Ansible Certified Content Collections*, are officially supported by Red Hat and its partners through Ansible Automation Platform.

Automation Content Navigator

Ansible Automation Platform 2 provides a new top-level tool to develop and test Ansible Playbooks, *automation content navigator* (`ansible-navigator`). This tool replaces and extends the functionality of several earlier command-line utilities, including `ansible-playbook`, `ansible-inventory`, and `ansible-config`.

In addition, automation content navigator separates the control node, on which you run Ansible, from the automation execution environment that runs it, by running your playbooks in a container. This separation makes it easier for you to provide a complete working environment for your automation code from deployment to production.

Automation Execution Environments

An *automation execution environment* is a container image that contains Ansible Core, Ansible Content Collections, and any Python libraries, executables, or other dependencies needed to run your playbook.

When you run a playbook with `ansible-navigator`, you can select an automation execution environment for it to use to run that playbook. When your code is working, you can provide the playbook and the automation execution environment to automation controller (formerly called Red Hat Ansible Tower) and know that automation controller has everything it needs to correctly run your playbook.

The default environment used in Ansible Automation Platform 2.2 provides Ansible Core 2.13 and many Red Hat Ansible Certified Content Collections to provide a user experience very similar to Ansible 2.9. Another advantage of automation execution environments is that you can also use them to run earlier versions of Ansible. Red Hat also supports an automation execution environment that provides Ansible 2.9 for compatibility with earlier versions.

Alternatively, you can use a new tool provided with Ansible Automation Platform 2 called `ansible-builder` to create your own custom execution environments.

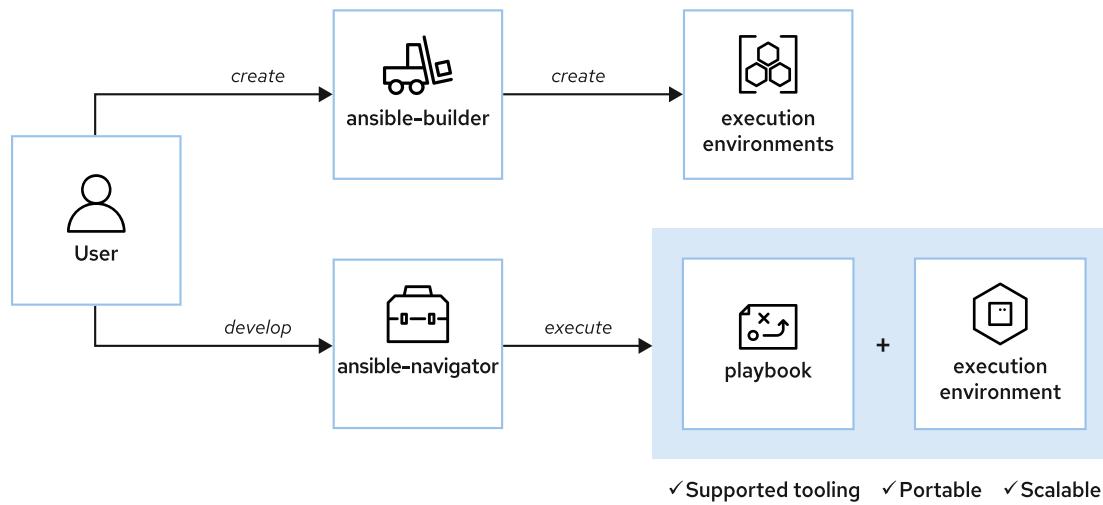


Figure 1.1: User experience: Adapting execution environments to your needs

Automation Controller

Automation controller, formerly called Red Hat Ansible Tower, is the component of Ansible Automation Platform that provides a central point of control to run your enterprise automation code. It provides a web UI and a REST API that can be used to configure, run, and evaluate your automation jobs.

In Red Hat Ansible Tower, the system was both the control node and the automation execution environment. If automation code required a different set of Python dependencies in the automation execution environment on Ansible Tower than the system provided by default, you had to manually set up a separate Python virtual environment (or "venv") for your automation code to use. This was difficult to manage, hard to distribute, required updates to the system running Ansible Tower, and scaled poorly.

The new automation controller design solves this challenge by separating the control node (providing the web UI and API) from the automation execution environments (now running in containers). You can deploy container-based automation execution environments from a container registry, and replacing them for a particular Ansible Playbook is a matter of a simple configuration change in the web UI. It is then easier to deploy the exact environment in which the developers expect their automation code to run.

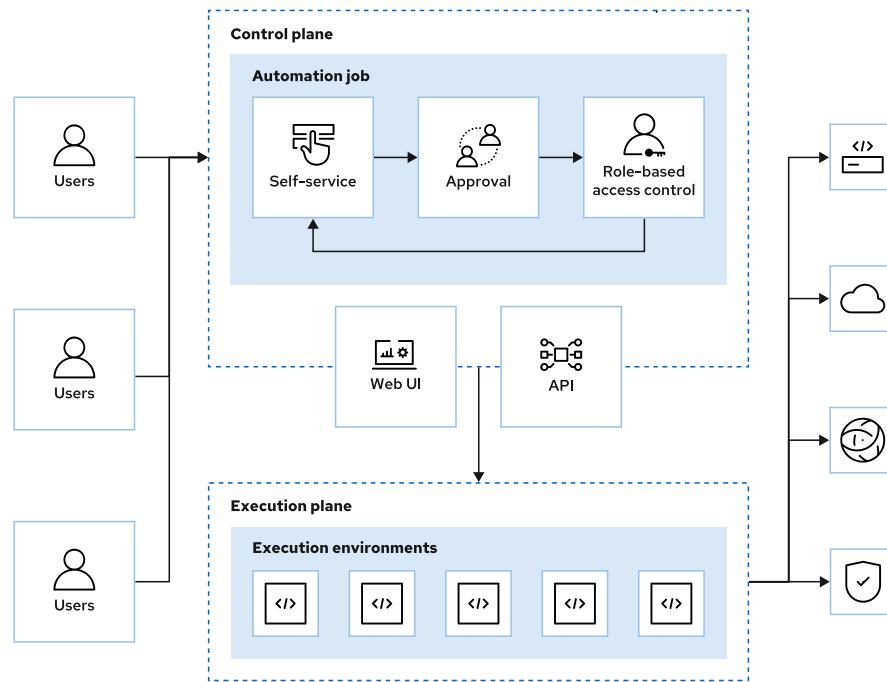


Figure 1.2: Components of automation controller

Automation controller can run automation execution environments on remote nodes (*execution nodes*), communicating with them over the network using a feature called *automation mesh*.

Automation Hub

Automation hub provides you with a way to manage and distribute automation content. A public service at console.redhat.com provides access to Red Hat Ansible Certified Content Collections that you can download and use with `ansible-galaxy` (for `ansible-navigator`) and with automation controller.

You can also set up a *private automation hub*. A private automation hub enables you to create your own curated set of Ansible Content Collections. It also provides a container registry that you can use for distributing your automation execution environments, if you do not already have one.

A private automation hub provides a centralized place for automation controller and the organization's Ansible developers to use to get automation content.

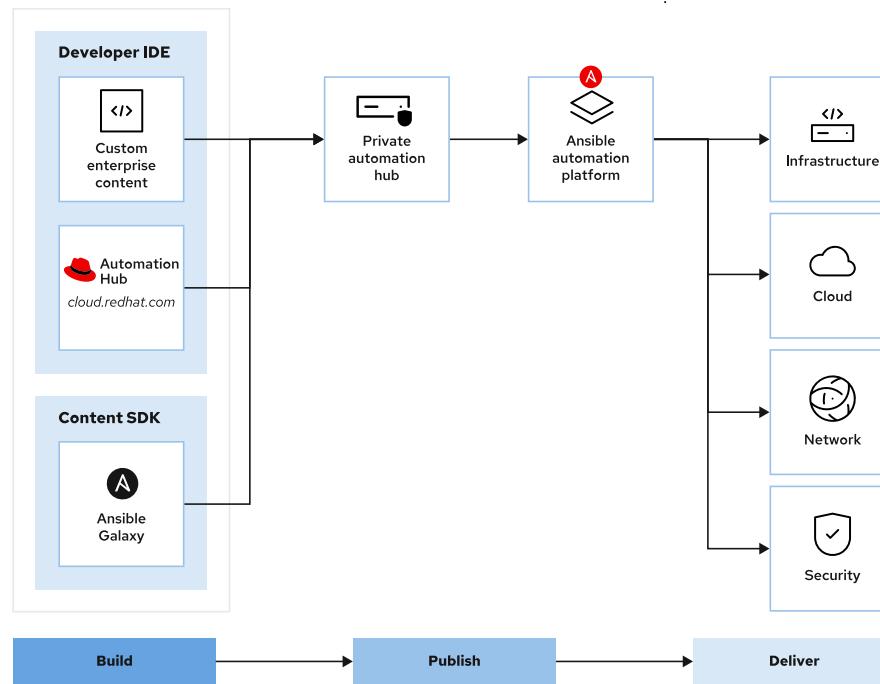


Figure 1.3: Creator experience: Working with Ansible Automation Platform

Hosted Services

In addition to the hosted automation hub at console.redhat.com, two other hosted services are available.

Red Hat Insights for Red Hat Ansible Automation Platform help you understand what automation code you are running and whether it is successful. You can also use it to evaluate the positive impact of automation on your organization.

Automation Analytics helps to provide better insight into the performance of your automation infrastructure. You can use it to analyze how you use automation and what modules, playbooks, and workflows you most frequently use.

Red Hat Ansible Automation Platform 2 Architecture

The following diagram shows the components of a Red Hat Ansible Automation Platform 2 deployment alongside the personas of some people who would use them.

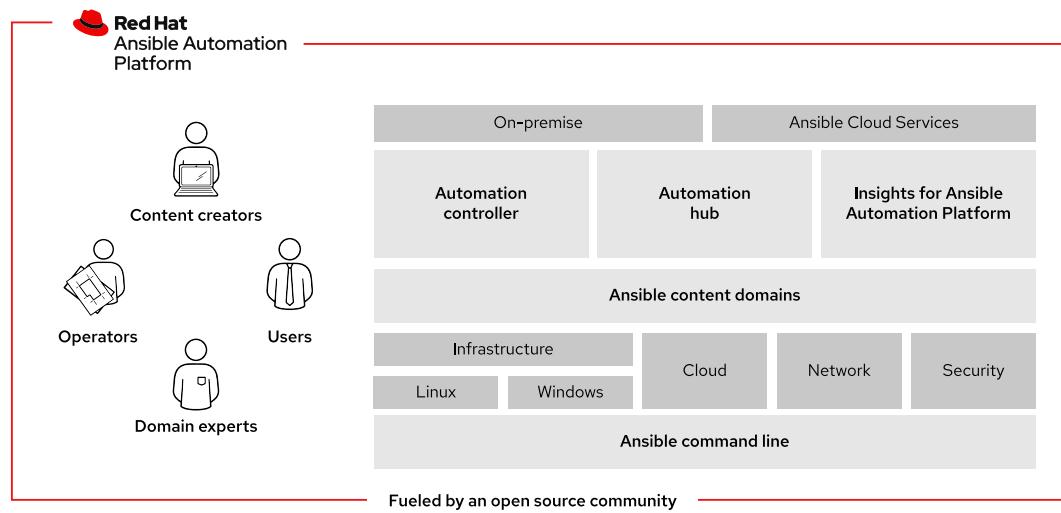


Figure 1.4: Architecture of Red Hat Ansible Automation Platform 2

Developing Playbooks with Ansible Automation Platform 2

Automation controller and automation content navigator both support automation execution environments. This simplifies the transition from developing playbooks to running them in production, because there is no need to use virtual environments or to install modules and their dependencies on automation controller. You can customize execution engines to contain everything that your organization requires to execute its playbooks.

Provided that automation content navigator and automation controller have access to the same automation execution environments, the transition from development to production for your playbooks should be almost seamless.



References

Red Hat Ansible Automation Platform

<https://www.redhat.com/en/technologies/management/ansible>

Introducing Ansible Automation Platform 2

<https://www.ansible.com/blog/introducing-ansible-automation-platform-2>

► Quiz

Red Hat Ansible Automation Platform 2

Choose the correct answers to the following questions:

- ▶ **1. Which component replaces commands such as `ansible-playbook` and `ansible-doc`?**
 - a. Automation hub
 - b. Ansible Core
 - c. Automation content navigator
 - d. Ansible Playbook editor

- ▶ **2. Which service provides automation content officially supported by Red Hat?**
 - a. Automation hub
 - b. Red Hat Enterprise Ansible Cloud
 - c. Ansible Galaxy
 - d. Ansible content platform

- ▶ **3. Automation content navigator separates which two functions to make it a complete working environment?**
 - a. Network management and host management
 - b. Content plane and configuration plane
 - c. Management network and production network
 - d. Control node and automation execution environment

- ▶ **4. Automation content can be hosted on-premise using which component?**
 - a. Private automation hub
 - b. Red Hat Insights for Red Hat Ansible Automation Platform
 - c. Automation Hub Repository
 - d. Red Hat Ansible cache

- ▶ **5. Which Ansible Content Collection is always included with `ansible-core`?**
 - a. `community.general`
 - b. `ansible.builtin`
 - c. `ansible.utils`
 - d. `ansible.posix`
 - e. `kubernetes.core`
 - f. `ansible.netcommon`

► Solution

Red Hat Ansible Automation Platform 2

Choose the correct answers to the following questions:

- ▶ **1. Which component replaces commands such as ansible-playbook and ansible-doc?**
 - a. Automation hub
 - b. Ansible Core
 - c. Automation content navigator
 - d. Ansible Playbook editor
- ▶ **2. Which service provides automation content officially supported by Red Hat?**
 - a. Automation hub
 - b. Red Hat Enterprise Ansible Cloud
 - c. Ansible Galaxy
 - d. Ansible content platform
- ▶ **3. Automation content navigator separates which two functions to make it a complete working environment?**
 - a. Network management and host management
 - b. Content plane and configuration plane
 - c. Management network and production network
 - d. Control node and automation execution environment
- ▶ **4. Automation content can be hosted on-premise using which component?**
 - a. Private automation hub
 - b. Red Hat Insights for Red Hat Ansible Automation Platform
 - c. Automation Hub Repository
 - d. Red Hat Ansible cache
- ▶ **5. Which Ansible Content Collection is always included with ansible-core?**
 - a. community.general
 - b. ansible.builtin
 - c. ansible.utils
 - d. ansible.posix
 - e. kubernetes.core
 - f. ansible.netcommon

Running Playbooks with Automation Content Navigator

Objectives

- Install automation content navigator and use it to run an existing playbook with a supported execution environment.

Automation Content Navigator

Automation content navigator (`ansible-navigator`) is a new tool in Red Hat Ansible Automation Platform 2, and is designed to make it easier for you to write playbooks that can be run in a reproducible manner. It combines the features formerly provided by `ansible-playbook`, `ansible-inventory`, `ansible-config`, and `ansible-doc` in one top-level interface.

Automation content navigator offers a new interactive mode with a text-based user interface (TUI), and can also be run using the `--mode stdout` (or `-m stdout`) option to provide output in the original format used by the earlier tools.

For example, to run a playbook with `ansible-playbook`, you might enter the following command:

```
[user@host ~]$ ansible-playbook playbook.yml -i inventory
```

You can use the `ansible-navigator` command to do the same thing and get output from the playbook in the same format, as follows:

```
[user@host ~]$ ansible-navigator run playbook.yml -i inventory -m stdout
```

However, if you omit the `-m stdout` option for the `ansible-navigator` command, it starts an interactive mode that summarizes the output of the playbook run in real time. This enables you to investigate the results in a TUI after the run completes.

Play name	Ok	Changed	Unreachable	Failed	Skipped	Ignored	In progress	Task count	Progress
0 Ensuring HAProxy is deployed	9	6	0	0	0	0	0	9	Complete
1 Ensuring Apache HTTP Server is deployed	14	8	0	0	0	0	0	14	Complete
2 Ensuring web content is deployed	4	2	0	0	0	0	0	4	Complete

^b/PgUp page up ^f/PgDn page down : scroll esc back [0-9] goto :help help Successful

Figure 1.5: Output of a playbook run with automation content navigator

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

In the preceding image, you can see that `ansible-navigator` ran a playbook containing three plays. The first play, "Ensuring HAProxy is deployed", ran nine tasks, six of which made changes. The second play, "Ensuring Apache HTTP Server is deployed", ran 14 tasks, eight of which made changes. The third play, "Ensuring web content is deployed", ran four tasks, two of which made changes. You can get more information about the tasks that were run by the first play by pressing 0, or by typing :0 in `ansible-navigator`:

Result	Host	Number	Changed	Task	Task action	Duration
0 Ok	demo.lab.example.com	0	False	Ensure HAProxy is installed and config	ansible.builtin.include_role	0s
1 Ok	demo.lab.example.com	1	True	Ensure the HAProxy packages are instal	ansible.builtin.yum	7s
2 Ok	demo.lab.example.com	2	True	Ensure HAProxy is started and enabled	ansible.builtin.service	2s
3 Ok	demo.lab.example.com	3	True	Ensure HAProxy configuration is set	ansible.builtin.template	1s
4 Ok	demo.lab.example.com	4	False	Ensure the firewall ports are opened	ansible.builtin.include_role	0s
5 Ok	demo.lab.example.com	5	False	Ensure the firewalld package is instal	ansible.builtin.yum	28s
6 Ok	demo.lab.example.com	6	True	Ensure the firewalld is configured	ansible.posix.firewalld	5s
7 Ok	demo.lab.example.com	7	True	reload haproxy	ansible.builtin.service	3s
8 Ok	demo.lab.example.com	8	True	reload firewalld	ansible.builtin.service	3s

^b/PgUp page up ^f/PgDn page down ↻ scroll esc back [0-9] goto :help help Successful

Figure 1.6: Details of a play run with automation content navigator

You can also get detailed information about particular tasks. In the preceding example, if you press 2 or type :2, then details of the "Ensure HAProxy is started and enabled" task are displayed, and you can use the arrow keys to scroll through the results:

Play name: Ensuring HAProxy is deployed:2
Task name: Ensure HAProxy is started and enabled
<u>CHANGED: demo.lab.example.com</u>
0 ---
1 duration: 1.851019
2 end: '2022-10-01T01:57:18.798249'
3 event_loop: null
4 host: demo.lab.example.com
5 play: Ensuring HAProxy is deployed
6 play_pattern: lb_servers
7 playbook: /home/student/.venv/labs/lib/python3.6/site-packages/do374/materials/labs/inventory-variables/site.yml
8 remote_addr: demo.lab.example.com
9 res:
10 ansible_no_log: null
11 changed: true
12 enabled: true
13 invocation:
14 module_args:
15 daemon_reexec: false
16 daemon_reload: false
17 enabled: true
18 force: null
19 masked: null

^b/PgUp page up ^f/PgDn page down ↻ scroll esc back - previous + next [0-9] goto :help help Successful

Figure 1.7: Details of a task in a play run with automation content navigator

To go back to previous pages or exit from `ansible-navigator` at the top-level page, press Esc.

Improving Portability with Automation Execution Environments

One new feature of Red Hat Ansible Automation Platform 2 is support for *automation execution environments*. An automation execution environment is a container image that includes Ansible Content Collections, their software dependencies, and a minimal Ansible engine that can run your playbooks and interact with Ansible. Both `ansible-navigator` and automation controller (formerly known as Red Hat Ansible Tower) use automation execution environments to simplify

the development, testing, and deployment of your automation code in a consistent Ansible environment.

Red Hat provides several supported automation execution environments consisting of slightly different Ansible deployments. You can also create your own customized automation execution environments.

With an automation execution environment, you can avoid creating multiple Python virtual environments on automation controller. Red Hat Ansible Tower required multiple Python virtual environments to support different combinations of Ansible, modules, Python libraries, and other software dependencies. This was hard to develop, hard to deploy, and hard to manage. Red Hat Ansible Automation Platform 2 uses multiple automation execution environments rather than multiple Python virtual environments. In Ansible Automation Platform, you can create the customized environment in a custom execution environment, test it with `ansible-navigator`, and distribute it to automation controller as a container image.

When you use `ansible-navigator` to run a playbook, and do not specify a particular automation execution environment, `ansible-navigator` automatically attempts to pull the default automation execution environment from `registry.redhat.io` if it is not already present on the system. For Red Hat Ansible Automation Platform 2.2, this default environment includes Ansible Core 2.13 and a standard set of Red Hat Ansible Certified Content Collections.

By default, the `ansible-navigator` command in Ansible Automation Platform 2.2 attempts to download and use the container image available at `registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest` if the automation execution environment is not specified in some other way.



Important

For `ansible-navigator` to pull the container image for the default automation execution environment (or any other execution environment) from the container registry on `registry.redhat.io`, it needs to authenticate. For this to succeed, you must be logged in to `registry.redhat.io` and have a valid subscription to Ansible Automation Platform.

Use the `podman login registry.redhat.io` command with your Customer Portal login credentials before running the `ansible-navigator` command. You only need to do this once per session.

If the container image has already been downloaded, then `ansible-navigator` can use that without needing to pull the container image again.

In this course, the classroom might not have internet access in some delivery modes. However, you can also download the supported container images in this course from a container registry on `hub.lab.example.com`. You can use the `podman login hub.lab.example.com` command to authenticate in the classroom environment using `admin` as the username and `redhat` as the password. After authenticating, you can manually download the available automation execution environments. The hands-on activities provide detailed instructions.

Selecting Alternative Automation Execution Environments

When you run automation content navigator, you can use the `--execution-environment-image` (`--eei`) option to select a specific container image to use as the automation execution environment, such as the `ee-supported-rhel8` container image. Container images are also published with tags so that you can select a specific version of the container image. For example,

you could specify the `ee-supported-rhel8:latest` container image. The `latest` tag indicates that you want the latest version of that image.

You can also use the `--pull-policy (- -pp)` option to control how `ansible-navigator` pulls container images. When the value of this option is set to `missing`, automation content navigator only pulls the container image if the image does not exist on the local system.

Installing Automation Content Navigator

You only need to install `ansible-navigator` on your control node. You do not need to install `ansible-navigator` on managed hosts.

You need a valid Red Hat Ansible Automation Platform subscription to install automation content navigator on your control node. If you are not yet using the product, trial entitlements are available through www.redhat.com and developer entitlements are available through the Red Hat Developer program.

If you have activated Simple Content Access for your organization, then you do not need to attach the entitlement to your system. The installation process is as follows:

- Register your system using Red Hat Subscription Manager.

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

- Enable the Red Hat Ansible Automation Platform 2 repository.

```
[root@host ~]# subscription-manager repos \
> --enable ansible-automation-platform-2.2-for-rhel-8-x86_64-rpms
```



Note

You do not need to run these `subscription-manager` commands in the classroom lab environment.

- Install the `ansible-navigator` RPM package.

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

Configuring Authentication to Managed Hosts

Automation content navigator needs to be able to log in to managed hosts and gain superuser privileges on those hosts. The easiest way to implement this is by using SSH key-based authentication to an account that allows privilege escalation through `sudo` without a password.

Preparing SSH Key-based Authentication

The initial part of the configuration of your managed hosts is identical to the procedure you use for the `ansible-playbook` command:

- Set a `remote_user` directive to the name of the user account you plan to use on the managed hosts in the `[defaults]` section of your Ansible configuration file on the control node.
- Use the `ssh-keygen` command to generate an SSH key pair for the user account that you use to run `ansible-navigator`.

- Install the public key of that key pair in the `~/.ssh/authorized_keys` file for the `remote_user` account on each managed host. You can use the `ssh-copy-id user@host` command to do this, where `user` is the remote user and `host` is one of the managed hosts.
- On the managed hosts, configure sudo to allow the remote user you specified to run all commands as any user without a password.

Providing Private Keys to the Automation Execution Environment

The main difference between authenticating `ansible-navigator` and `ansible-playbook` to managed hosts is that `ansible-navigator` runs the playbook inside a container that cannot access your `~/.ssh` directory. However, if you are running `ssh-agent` on your control node to store SSH private keys, when you run `ansible-navigator` it can automatically provide those keys to the execution environment.

If you logged in to the control node through the graphical desktop environment, `ssh-agent` is automatically started and the `ssh-add` command is automatically run to add your private keys to the agent. If any of your SSH private keys are passphrase protected, you are prompted for the password after you log in. Authentication then automatically works for all terminals in that graphical login session.

If you logged in to the control node through a text-based virtual console or remotely using `ssh`, you must start `ssh-agent` yourself by running the `eval $(ssh-agent)` command. In the same shell, you then run `ssh-add` and if necessary provide the passphrase for your private key. You can then run `ansible-navigator` in the same shell and authenticate.

Running Automation Content Navigator

Use the `ansible-navigator` command to run automation content navigator. If you run the command with no arguments, or as `ansible-navigator welcome`, it starts in interactive mode. This displays a help page describing the subcommands provided by the tool.

Many earlier Ansible commands can be replaced with an automation content navigator subcommand.

Earlier Ansible command	Automation content navigator subcommand
<code>ansible-config</code>	<code>ansible-navigator config</code>
<code>ansible-doc</code>	<code>ansible-navigator doc</code>
<code>ansible-inventory</code>	<code>ansible-navigator inventory</code>
<code>ansible-playbook</code>	<code>ansible-navigator run</code>

Automation content navigator also provides additional functionality. Most subcommands can be run from either the command line or from within an interactive automation content navigator session, but some commands do not support the `-m stdout` option.

The following table briefly describes the available subcommands.

Subcommand	Description
<code>collections</code>	Get information about installed collections.

Subcommand	Description
config	Examine the current Ansible configuration.
doc	Examine the Ansible documentation for a plug-in.
help	Display detailed help for ansible-navigator.
images	Examine an execution environment.
inventory	Explore an inventory.
log	Review the current log file.
open	Open the current page in a text editor.
replay	Replay a playbook artifact.
run	Run a playbook.

The help page displayed by `ansible-navigator welcome` is similar to the preceding table. When running a subcommand in an interactive session, type `:` to start the command, such as `:config`.

The following sections introduce some `ansible-navigator` subcommands.

Running Playbooks

Run playbooks using automation content navigator from the command line with the `ansible-navigator run` command or interactively with the `:run` subcommand. The `ansible-navigator run` command supports most of the same options as the `ansible-playbook` command.

When using the default `interactive` mode (rather than `stdout` mode), you can interactively examine playbook output, as previously discussed. For example, you might display details for a failed play or for a specific task in a play that failed.

Reviewing Previous Playbook Runs

Automation content navigator provides a new replay feature that displays the output of a previous playbook run. If playbook artifacts are enabled (the default), then the `ansible-navigator run` command generates an *artifact file*, which use names such as `site-artifact-2022-10-25T20:09:57.939910+00:00.json` or similar. The first part of the file name refers to the playbook name, and so `site` is used for a run using the `site.yml` playbook. The second part of the file name is the date and time with time zone offset from UTC (`+00:00` indicates that the time was recorded in UTC).

Review a previous playbook run by using either the `ansible-navigator replay` command from the command line or the `:replay` subcommand in interactive mode. Most replay files can be safely ignored or even deleted. If you plan to save a replay file, then consider renaming it so that you can more easily identify it.

One advantage of the replay file is that if the playbook fails then you can share the replay file with another developer or a support team. They can proceed with troubleshooting without needing to have access to all the files in the project directory, such as playbooks, roles, inventory files, variable files, and so on.

**Note**

Automation content navigator can prompt for passwords, but only if playbook artifacts are disabled. Configuration settings for automation content navigator, including the ability to disable playbook artifacts, are discussed later in the course.

Reading Documentation

Automation content navigator provides access to documentation for plug-ins, such as modules, lookups, and callbacks, much like the `ansible-doc` command.

One major difference from the `ansible-doc` command is that `ansible-navigator` does not support the `--list (-l)` option to list all plug-ins of a particular type. Instead you must explicitly specify the plug-in name.

For example, you can run `ansible-navigator doc ansible.posix.firewalld` from the command line to display documentation for the `ansible.posix.firewalld` module. You can do the same thing by running the `:doc ansible.posix.firewalld` subcommand from within an interactive automation content navigator session.

Getting Help

Automation content navigator provides multiple ways to get help. If you are running `ansible-navigator` in interactive mode, you can type `:help` to open a help page.

Alternatively, add the `--help (-h)` option to the `ansible-navigator` command. Using this option displays command syntax and describes command options.

**Important**

Using the `--help` option does not always display *all* options. For example, running the `ansible-navigator run --help` command displays the main options and then includes the following message:

...output omitted...

Note: 'ansible-navigator run' additionally supports the same parameters as the 'ansible-playbook' command. For more information about these, try 'ansible-navigator run --help-playbook --mode stdout'

Provided you have logged in to the classroom's private automation hub, the following command displays additional options:

```
[student@workstation ~]$ ansible-navigator run --help-playbook --mode stdout \
> --eei ee-supported-rhel8:latest
...output omitted...
```



References

Red Hat Ansible Automation Platform Creator Guide

https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.2/html-single/red_hat_ansible_automation_platform_creator_guide/index

Ansible Navigator Documentation

<https://ansible.readthedocs.io/projects/navigator/>

Ansible Navigator Creator Guide

https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.2/html-single/ansible_navigator_creator_guide/index

► Guided Exercise

Running Playbooks with Automation Content Navigator

Use automation content navigator to run a playbook and review the results of its execution.

Outcomes

- Use automation content navigator to run a playbook.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/develop-navigator/` directory.

```
[student@workstation ~]$ lab start develop-navigator
```

Instructions

- 1. Install the `ansible-navigator` package.

```
[student@workstation ~]$ sudo yum install ansible-navigator  
[sudo] password for student: student  
...output omitted...
```

- 2. Review the exercise playbook.

- 2.1. Change to the `/home/student/develop-navigator/` directory.

```
[student@workstation ~]$ cd ~/develop-navigator/
```

- 2.2. Use the `tree` command to view the files in the directory.

```
[student@workstation develop-navigator]$ tree  
. .  
└── inventory  
    └── intranet.yml  
    └── ansible.cfg  
0 directories, 3 files
```

- 2.3. Review the `intranet.yml` playbook.

```
---
```

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

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

    - name: firewalld enabled and running
      ansible.builtin.service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits http service
      ansible.posix.firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: true

    - name: httpd enabled and running
      ansible.builtin.service:
        name: httpd
        enabled: true
        state: started

- name: Test intranet web server
  hosts: localhost
  become: false
  tasks:
    - name: connect to intranet web server
      ansible.builtin.uri:
        url: http://servera.lab.example.com
        return_content: true
        status_code: 200
```

► 3. Run the `intranet.yml` playbook using automation content navigator.

- 3.1. Automation content navigator needs to pull the container image for its automation execution environment to your workstation from an automation hub or other container registry.

Use the `podman login` command to log in to the classroom private automation hub at `hub.lab.example.com`. Use `student` as your username and `redhat123` as your password.

(If you did not have a private automation hub, but had internet access, you could use your Customer Portal account to log in to the automation hub hosted by Red Hat at `registry.redhat.io` instead.)

```
[student@workstation develop-navigator]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 3.2. Use the `ansible-navigator run intranet.yml -m stdout --eei ee-supported-rhel8` command to run the `intranet.yml` playbook. The output presented by `stdout` mode is in the same format as output presented by the `ansible-playbook` command.

```
[student@workstation develop-navigator]$ ansible-navigator run intranet.yml \
> -m stdout --eei ee-supported-rhel8

-----
Execution environment image and pull policy overview
-----
Execution environment image name: ee-supported-rhel8:latest
Execution environment image tag: latest
Execution environment pull arguments: None
Execution environment pull policy: tag
Execution environment pull needed: True

-----
Updating the execution environment
-----
Running the command: podman pull ee-supported-rhel8:latest
Trying to pull hub.lab.example.com/ee-supported-rhel8:latest...
Getting image source signatures
Copying blob 3cbba8687c73 skipped: already exists
Copying blob 978c7c1baf28 skipped: already exists
Copying blob 858b1db62e17 skipped: already exists
Copying blob 91181fe42f7e skipped: already exists
Copying blob 4ace973aa1b1 skipped: already exists
Copying config c33e10e995 done
Writing manifest to image destination
Storing signatures
c33e10e995298d65bad08da35d7cd0beb660a2fe824aef460aebe14f889905d3

PLAY [Enable intranet services] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

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

TASK [test html page is installed] ****
```

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

```

changed: [servera.lab.example.com]

TASK [firewalld enabled and running] ****
ok: [servera.lab.example.com]

TASK [firewalld permits http service] ****
changed: [servera.lab.example.com]

TASK [httpd enabled and running] ****
changed: [servera.lab.example.com]

PLAY [Test intranet web server] ****

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

TASK [connect to intranet web server] ****
ok: [localhost]

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

```

- ▶ 4. Using the `ee-supported-rhel8` execution environment, run the `intranet.yml` playbook interactively from within automation content navigator.

4.1. Use the `ansible-navigator` command to launch automation content navigator.

```
[student@workstation develop-navigator]$ ansible-navigator \
> --eei ee-supported-rhel8
```

```

0|Welcome
1|-----
2|
3|Some things you can try from here:
4|- :collections           Explore available collections
5|- :config                Explore the current ansible config
6|- :doc <plugin>          Review documentation for a module
7|- :help                  Show the main help page
8|- :images                Explore execution environment imag
9|- :inventory -i <inventory> Explore an inventory
10|- :log                   Review the application log
11|- :lint <file or directory> Lint Ansible/YAML files (experimen
12|- :open                  Open current page in the editor
13|- :replay                Explore a previous run using a pla
14|- :run <playbook> -i <inventory> Run a playbook in interactive mode
15|- :settings              Review the current ansible-navigat
16|- :quit                  Quit the application
17|
18|happy automating,
19|
20|-winston

```

4.2. Type :run `intranet.yml` to interactively run the `intranet.yml` playbook.

```
Play name Ok Changed ... Failed Skipped Ignored ... Task count Progress
0|Enable intr 6      0 ...      0      0      0 ...          6 Complete
1|Test intr   2      0 ...      0      0      0 ...          2 Complete
```

`^f/PgUp` page up `^b/PgDn` page down `↑↓ scroll` `esc back` ... **SUCCESSFUL**

4.3. Type 0 to display details for the `Enable intranet services` play.

Result	Host	Number	Changed	Task	...
0 OK	servera	0	False	Gathering Facts	...
1 OK	servera	1	False	latest version of httpd and firewalld	...
2 OK	servera	2	False	test html page is installed	...
3 OK	servera	3	False	firewalld enabled and running	...
4 OK	servera	4	False	firewalld permits http service	...
5 OK	servera	5	False	httpd enabled and running	...

`^f/PgUp` page up `^b/PgDn` page down `↑↓ scroll` `esc back` ... **SUCCESSFUL**

4.4. Press ESC to return to the main playbook summary page, and then type 1 to display details for the `Test intranet web server` play.

Result	Host	Number	Changed	Task	Task action	...
0 OK	localhost	0	False	Gathering Facts	gather_facts	...
1 OK	localhost	1	False	connect to intranet	... ansible.builtin...	...

`^f/PgUp` page up `^b/PgDn` page down `↑↓ scroll` `esc back` ... **SUCCESSFUL**

Type :q to exit the `ansible-navigator` command.

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish develop-navigator
```

Managing Ansible Project Materials Using Git

Objectives

- Create and manage Ansible Playbooks in a Git repository, following recommended practices.

Defining Infrastructure as Code

One key DevOps concept is the idea of *infrastructure as code* (*IaC*). Instead of managing your infrastructure manually, you define and build your systems by running your automation code. Red Hat Ansible Automation Platform is a key tool that can help you implement this approach.

If Ansible projects are the code used to define the infrastructure, then a *version control system* (VCS) such as Git should be used to track and control changes to the code.

Version control also enables you to implement a lifecycle for the different stages of your infrastructure code, such as development, QA, and production. You can commit your changes to a *branch* and test those changes in noncritical development and QA environments. When you are confident with the changes, you can merge them to the main production code and apply the changes to your production infrastructure.

Introducing Git

Git is a *distributed version control system* (DVCS) that enables you to manage changes to files in a project collaboratively. Using Git provides many benefits:

- You can review and restore earlier versions of files.
- You can compare two versions of the same file to identify changes.
- You can record a log of who made what changes, and when those changes were made.
- Multiple users can collaboratively modify files, resolve conflicting changes, and merge the changes.

Using Git, you can start by *cloning* an existing shared project from a *remote repository*. Cloning a project creates a complete copy of the original remote repository as a *local repository*. This local copy has the entire history of the files in Git, and not just the latest snapshot of the project files.

You make edits in a *working tree*, which is a checkout of a single snapshot of the project. Next, you place the modified files in a staging area, ready to commit the differences to the local repository. At this point, no changes have been made to the shared remote repository.

When you are ready to share the completed work, you *push* the changes to the remote repository. Likewise, when you are ready to update the local repository with the latest changes to the remote repository, you *pull* the changes, which fetches them from the remote repository and merges them into the local repository.

To use Git effectively, you must be aware of the three possible states of a file in the working tree: *modified*, *staged*, or *committed*.

State	Description
Modified	You edited the copy of the file in the working tree and it is different from the latest version in the repository.
Staged	You added the modified file to a list of changed files to commit as a set, but the staged files have not been committed yet.
Committed	You committed the modified file to the local repository.

When you commit the file to the local repository, that file can be pushed to or pulled by a remote repository.



Figure 1.8: The four areas where Git manages files



Note

The presentation in this section assumes that you are using Git from the command line of a Bash shell. A number of programming editors and IDEs have integrated Git support, and although the configuration and UI details might differ, they provide a different front end to the same workflow.

Initial Git Configuration

Because Git users frequently modify projects with multiple contributors, Git records the user's name and email address on each of their commits. These values can be defined at a project level, but you can also set global defaults for a user. The `git config` command controls these settings.

Using `git config` with the `--global` option manages the default settings for all Git projects to which you contribute by saving the settings in your `~/.gitconfig` file.

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

If Bash is your shell, then another useful, optional setting, is to configure your prompt to automatically update to indicate the status of your working tree. The easiest way to do it is to use the `git-prompt.sh` script that ships with the `git` package.

To modify your shell prompt in this way, add the following lines to your `~/.bashrc` file.

```
source /usr/share/git-core/contrib/completion/git-prompt.sh
export GIT_PS1_SHOWDIRTYSTATE=true
export GIT_PS1_SHOWUNTRACKEDFILES=true
export PS1='[\u@\h \w$(declare -F __git_ps1 &>/dev/null && __git_ps1 " (%s)")]\$ '
```

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

If your current directory is in a Git working tree, then the name of the current Git branch for the working tree is displayed in parentheses. If you have untracked, modified, or staged files in your working tree, then the prompt could indicate one of the following meanings:

- (branch *) means that you have modified a tracked file.
- (branch +) means that you have modified a tracked file and used `git add` to stage it.
- (branch %) means that you have untracked files in your tree.
- Combinations of markers are possible, such as (branch *)+.

Starting the Git Workflow

When working on shared projects, you clone an existing upstream repository with the `git clone` command. The provided path name or URL determines which repository is cloned into the current directory. This process creates a working tree of files ready for revisions. Because the working tree is unmodified, it is initially in a clean state. A clean state means the working tree does not contain any modified, staged, or new files.

For example, the following command clones the `project.git` repository at `git.lab.example.com` by connecting using the SSH protocol and authenticating as the `git` user:

```
[user@host ~]$ git clone git@git.lab.example.com:project.git
```



Note

Another way to start the Git workflow is to create a new, private project with the `git init` command. When a project is started in this way, no remote repository is created.

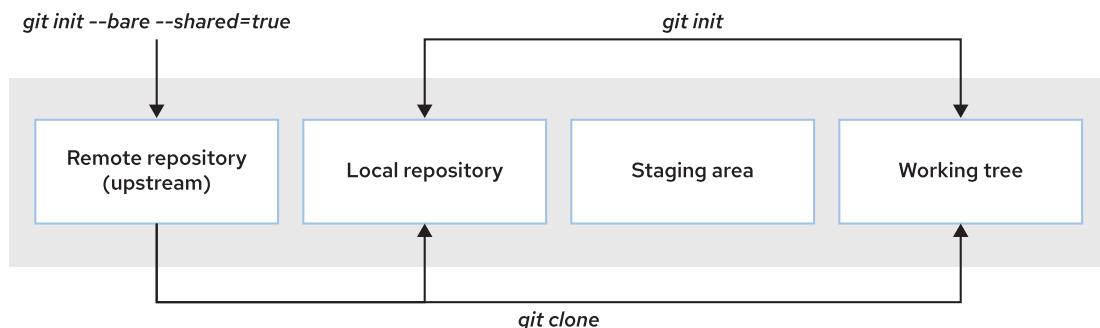
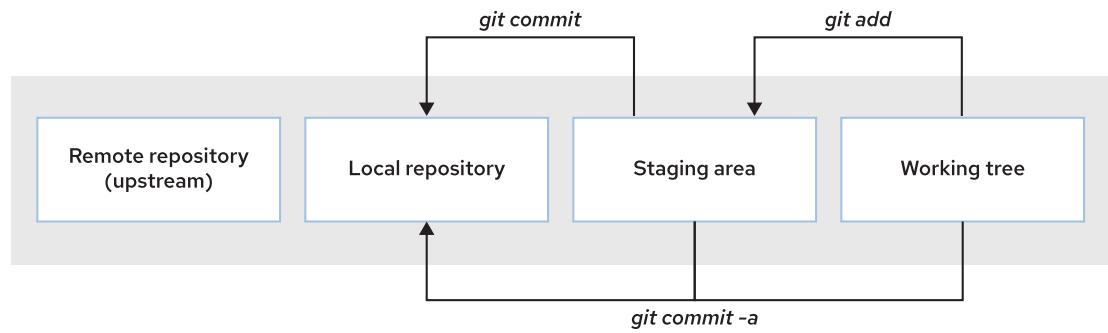


Figure 1.9: Git subcommands used to create a repository

Use `git init --bare` to create a *bare repository* on a server. Bare repositories do not have a local working tree. Therefore, you cannot use the bare repository to apply file changes.

Git servers usually contain bare repositories, because the working tree is not needed on the server. When you create a local copy of the repository, you usually need a working tree to make local changes, so you create a non-bare clone. The server must also be set up to allow users to clone, pull from, and push to the repository using either the HTTPS or SSH protocol.

As you work, you create new files and modify existing files in the working tree. This changes the working tree to a *dirty state*. The `git status` command displays detailed information about which files in the working tree are modified but unstaged, untracked (new), or staged for the next commit.

**Figure 1.10: Git subcommands used to add and update local repository content**

The `git add` command stages files, preparing them to be committed. Only files that are staged to the staging area are saved to the repository on the next commit.

**Note**

If you are working on two changes at the same time, then you can organize the files into two commits for better tracking of changes.

The `git rm` command removes a file from the working directory and also stages its removal from the repository on the next commit.

The `git reset` command removes a file from the staging area that has been added for the next commit. This command has no effect on the file's contents in the working tree.

The `git commit` command commits the staged files to the local Git repository. You must provide a log message that explains why you are saving the current set of staged files. Failure to provide a message aborts the commit. Log messages do not have to be long, but they should be meaningful so that they are useful.

**Important**

The `git commit` command by itself does not automatically commit changed files in the working tree.

The `git commit -a` command stages and commits all the *modified* files in one step. However, that command does not include any *untracked* (newly created) files in the directory. When you add a new file, you must explicitly run the `git add` command to stage the new file.



Note

Meaningful and concise commit messages are the key to maintaining a clear history for an Ansible project. There are many approaches to a good commit message, but most of these approaches agree on the following three points:

- The first line should be a short (usually fewer than 50 character) summary of the reason for the commit.
- A blank line follows the first line, and then the rest of the message must explain all the details, description of what was done, and the reasons for the commit.
- If available, then add references to an issue or feature tracker. These references expand the commit message with additional text, related people, or history.

The `git push` command uploads changes made to the local repository to the remote repository. One common way to coordinate work with Git is for all developers to push their work to the same, shared, remote repository.

Before Git pushes can work, you must define the default push method to use. The following command sets the default push method to the `simple` method. This is the safest option for beginners.

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

Use the `git pull` command to perform the following tasks:

- Fetch commits from the remote repository.
- Add those commits to the local directory.
- Merge changes into the files in the working tree.

Run the `git pull` command frequently to stay current with the changes that others are making to the project in the remote repository.



Note

An alternative approach is to use the `git fetch` command to download the changes from the remote repository to your local repository. You then use the `git merge` command to merge the changes in the tracking branch to your current branch.

Later in this section, you learn more about Git branches.

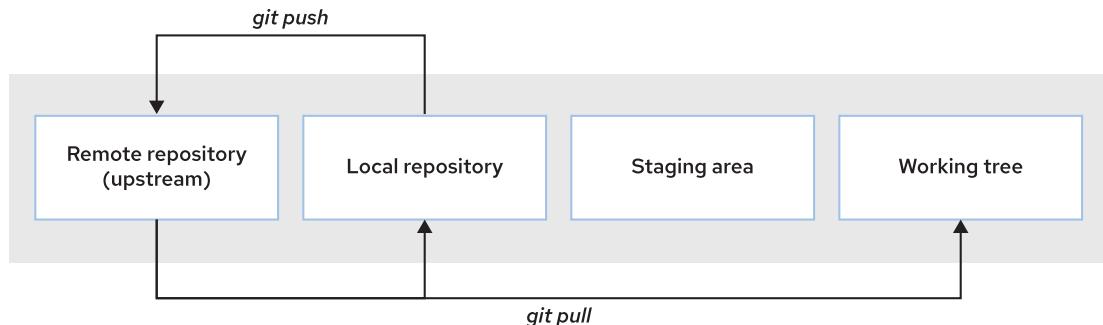


Figure 1.11: Git subcommands that interact with a remote repository

Examining the Git Log

Part of the point of a version control system is to track a history of commits. A commit hash identifies each commit.

The `git log` command displays the commit log messages with the associated ID hashes for each commit.

The `git show commit-hash` command shows what was in the change set for a particular commit hash. You do not need to use the entire hash with this command; only enough of it to uniquely identify a particular commit in the repository. These hashes can also be used to revert to earlier commits or otherwise explore the version control system's history.

Git Quick Reference

Command	Description
<code>git clone URL</code>	Clone an existing Git project from the remote repository at <i>URL</i> into the current directory.
<code>git status</code>	Display the status of files in the working tree.
<code>git add file</code>	Stage a new or changed file for the next commit.
<code>git rm file</code>	Stage removal of a file for the next commit.
<code>git reset</code>	Unstage files that have been staged for the next commit.
<code>git commit</code>	Commit the staged files to the local repository with a descriptive message.
<code>git push</code>	Push changes in the local repository to the remote repository.
<code>git pull</code>	Fetch updates from the remote repository to the local repository and merge them into the working tree.
<code>git revert commit-hash</code>	Create a new commit, undoing the changes in the referenced commit. You can use the commit hash that identifies the commit, although there are other ways to reference a commit.
<code>git init</code>	Create a new project.
<code>git log</code>	Display the commit log messages.
<code>git show commit-hash</code>	Shows what was in the change set for a particular commit hash.

Important

This is a highly simplified introduction to Git. It has made some assumptions and avoided discussion of the important topics of branches and merging. Some of these assumptions include:

- You cloned the local repository from a remote repository.
- You are using only one local branch.
- You configured the local branch to fetch from and push to a branch on the original remote repository.
- You provided write access to the remote repository, such that `git push` works.

Working with Branches and References

Branches enable you to make changes to your project, working on new development, without altering your main branch. Then, when you are satisfied with your changes, you can *merge* the changes in your branch to the main branch of development, integrating them.

A branch is essentially a pointer to a particular commit in your commit tree. Each commit contains the changes it made and information about how to reference it and what its relationship is to other commits in the commit tree. The information in a commit includes:

- A unique ID in the form of a 40-character hexadecimal string. This ID is the SHA-1 hash of the contents of the commit.
- A list of repository files that it changed, and the exact changes to each of them.
- The ID of the parent commit that defines the status of the repository before applying the commit in question.
- Information about the author and the creator (or *committer*) for the commit in the commit data.
- A list of *references*. A reference is like a named pointer to the commit. The most common references are *tags* and *branches*.

The `git commit` command generates a new commit with all changes added to the stage with the `git add` command. You can display a Git repository as a commit graph, and branches as references to commits.

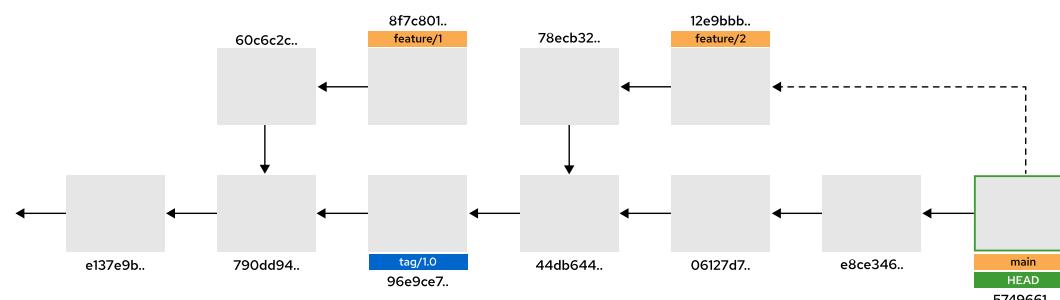


Figure 1.12: Sample Git repository

The preceding figure shows a sample Git repository containing 11 commits. The most recent commits are to the right; the arrows point to earlier commits. It has three branches (`main`, `feature/1`, and `feature/2`) and a single tag (`tag/1.0`).

The `HEAD` reference is the current commit in the local working tree. If you make a change in your working tree, stage it with the `git add` command, and commit it with the `git commit` command. A new commit is then created with the most recent commit as its parent, and `HEAD` moves to point to the new commit.

Creating Branches

Different branches in Git enable different work streams to evolve in parallel on the same Git repository. Commits for each work stream append only to that branch.

Use the `git branch` command to create a new branch from the current `HEAD` commit. This command creates a reference for the new branch, but it does not set the current `HEAD` to this branch. Use the `git checkout` command to move the `HEAD` to the appropriate branch.

In the preceding figure, the most recent commit for the `main` branch (and `HEAD` at that time) was commit `5749661`, which occurred at some point in the past. We are assuming you ran the `git branch feature/1` command, creating a branch named `feature/1`. Then, you ran `git checkout feature/1` to indicate that future commits should be made to that branch. Finally, you staged and committed two commits to the `feature/1` branch, commits `60c6c2c` and `8f7c801`. After running these commands, the `HEAD` was at commit `8f7c801`, and the new commits were added to the `feature/1` branch.

Next, you wanted to add commits to the `main` branch. After moving `HEAD` to the latest commit on that branch, by running the `git checkout main` command, two new commits were made: `96e9ce7` and `44bd644`. The `HEAD` pointed to `44bd644`, and commits were added to the `main` branch.

Merging Branches

When work is complete on a branch, you can *merge* the branch with the original branch. This enables you to work in parallel on new features and bug fixes, leaving the main branch free of incomplete or untested work.

In the preceding figure, it was determined that all changes from `feature/2` should be merged into the `main` branch. That is, someone wanted the most recent commit for `main` to be the result of all the commits from both branches.

This outcome was accomplished by first running the `git checkout main` command to make sure that `HEAD` was on the most recent commit on the `main` branch, `e8ce346`. Then, you ran the `git merge feature/2` command. This command created a new commit, `5749661`, which included all the commits from `feature/2` and all the commits from `main`. It moved `HEAD` to the latest commit.

Sometimes, changes on more than one branch cannot merge automatically because each branch makes changes to the same parts of the same files. This situation is called a *merge conflict* and you need to manually edit the affected files to resolve it.

**Important**

Some Git servers, such as GitHub, GitLab, and Bitbucket, allow you to protect branches from unwanted changes. As a best practice you might protect certain branches on the remote repository, such as the `main` branch, to avoid merging unwanted changes to them from other branches created by developers.

These servers also often implement a feature called *pull requests* or *merge requests*. This provides a mechanism that you or other code reviewers can use to review and approve requests before the branch can be merged with the protected branch on the remote server.

Merge strategies, conflict resolution, and protection of branches, are out of scope for this course.

Creating Branches from Earlier Commits

Use the `git checkout` command to move HEAD to any commit. For example, the command `git checkout 790dd94` moves HEAD to that commit. Then, you can create a new branch starting with that commit, using the `git branch` and `git checkout` commands. For example, you might choose to branch from a specific commit to ignore other recent changes to a branch.

You can create a branch and switch to it in a single step by running the `git checkout` command with the `-b` option:

```
[user@host ~]$ git checkout -b feature/3
Switched to a new branch 'feature/3'
```

**Note**

If you might need a commit in future, then you can use `git tag` to put a memorable label on it (such as `tag/1.0` in the example) and then run the `git checkout` command on that tag to switch to the desired commit.

Pushing Branches to Remote Repositories

Initially, any branches that you create only exist in your local repository. If you want to share them with users of the original remote repository that you cloned, then you must push them to the repository.

The most common way to do this is to use the `git push` command with the `--set-upstream` (or `-u`) option to create a branch on the remote repository. Your current local branch then tracks that branch.

For example, to push your new `feature/3` branch to your remote repository, run the following commands:

```
[user@host ~]$ git checkout feature/3
Switched to branch 'feature/3'
[user@host ~]$ git push --set-upstream origin feature/3
```

The remote `origin` repository normally refers to the Git repository that you originally cloned. This command indicates that you want to push the current branch to the `origin` repository, creating

feature/3. When the Git reviewers approve the push request, Git creates the **feature/3** branch in the remote `origin` repository.

Running the `git pull` command on this branch pulls and merges any commits from the remote repository into your local copy of the branch.

Structuring Ansible Projects in Git

Each Ansible project should have its own Git repository. The structure of the files in that repository should follow the directory layout recommended at https://docs.ansible.com/ansible/6/user_guide/sample_setup.html#sample-directory-layout. For example, the directory structure might appear as follows:

```

site.yml          # main playbook
webservers.yml   # playbook for webserver tier
dbservers.yml    # playbook for dbserver tier

collections/      # holds Ansible Content Collections in directories
  requirements.yml # specifies collections needed by this project

roles/
  common/          # this hierarchy represents a "role"
    tasks/          #
      main.yml       # <-- tasks file can include smaller files if warranted
    handlers/        #
      main.yml       # <-- handlers file
    templates/       # <-- files for use with the template resource
      ntp.conf.j2    # <----- templates end in .j2
    files/           #
      bar.txt        # <-- files for use with the copy resource
      myscript.sh    # <-- script files for use with the script resource
    vars/            #
      main.yml       # <-- variables associated with this role
    defaults/        #
      main.yml       # <-- default lower priority variables for this role
    meta/            #
      main.yml       # <-- role dependencies
...additional roles...

```

Roles and Ansible Content Collections

You can include roles and Ansible Content Collections in your Ansible project's Git repository. In this case each collection is generally stored in subdirectories of the `collections/` directory, and each role is stored in subdirectories of the `roles` directory. There might also be a `requirements.yml` file in either or both of those directories that specifies collections or roles that are used by this project, which the `ansible-galaxy` command can download as needed.

Including roles and collections in an Ansible project directory structure takes some thought to manage well. If your roles and collections are managed as part of the playbook, then keeping them with the playbook can make sense. However, most roles and collections are meant to be shared by multiple projects, and if each project has its own copy of each role or collection, then they could diverge from each other, losing some of the benefits of using them.

A role or collection should have its own Git repository. Some people try to include these repositories in their project repositories by using an advanced feature of Git called `submodules`.

Because submodules can be difficult to manage successfully, this is not a recommended approach.

A better approach is to set up `requirements.yml` files in your `roles/` and `collections/` directories and to use the `ansible-galaxy` command to populate this directory with the latest version of roles from automation hub, Ansible Galaxy, or their Git repositories before you run the project's playbooks.

**Note**

Automation controller automatically updates the project with roles and collections based on the `roles/requirements.yml` and `collections/requirements.yml` files included in the Ansible project.

Configuring Git to Ignore Files

Not every file and directory in your Ansible project directory should be tracked by Git. You can configure Git to ignore files and directories that should not be tracked or committed.

For example, the `ansible-navigator` command generates or updates an `ansible-navigator.log` file, creates an artifact file each time it runs a playbook, and creates a `.ssh` directory. None of these temporary files should be committed to the Git repository.

Likewise, if you use `requirements.yml` files with the `ansible-galaxy` command to populate the project's `roles` and `collections` directories, then you should configure Git to ignore the populated content.

To instruct Git to ignore specific files and paths in your project, add a `.gitignore` file at the top of your Ansible project directory. Document this in a `README` file at the top of your Ansible project directory.

Each line in a `.gitignore` file represents a pattern to match that determines whether Git ignores a file in the repository. Generally, lines that match file names indicate files that Git ignores. Lines that start with an exclamation mark (!) indicate files that Git should not ignore, even if they were matched by a previous pattern. Blank lines are permitted to improve readability, and any line that starts with a # is a comment.

Details on the pattern matching format for this file are documented in the `gitignore(5)` man page under "PATTERN FORMAT". Pay particular attention to the distinction between single asterisks (*) and double asterisks (**) in these pattern matching lines. (Double asterisks can match multiple directory levels.)

A sample `.gitignore` file could contain the following content:

```
roles/**  
!roles/requirements.yml  
collections/ansible_collections  
ansible-navigator.log  
*-artifact-*  
.ssh
```



References

gittutorial(7), git(1), and gitignore(5) man pages

- **Git**
<https://git-scm.com>
- **Pro Git by Scott Chacon and Ben Straub (free book)**
<https://git-scm.com/book/en/v2>
- **GitHub Getting Started Tutorial**
<http://try.github.io>
- **Learning Git Branching**
<http://learngitbranching.js.org>
- **Sample Directory Layout**
https://docs.ansible.com/ansible/6/user_guide/sample_setup.html#sample-directory-layout
- **How to Write a Git Commit Message**
<https://chris.beams.io/posts/git-commit/>

► Guided Exercise

Managing Ansible Project Materials Using Git

Clone an existing Git repository that contains an Ansible Playbook, make edits to files in that repository, commit the changes to your local repository, and push those changes to the original repository.

Outcomes

- Use basic Git commands to manage both new and modified files stored in an existing Git repository.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates the Git repository needed for the exercise.

```
[student@workstation ~]$ lab start develop-git
```

Instructions

- 1. Clone the `https://git.lab.example.com/student/develop-git.git` repository into the `/home/student/git-repos` directory and then create a new branch for this exercise.
- 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/develop-git.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/develop-git.git  
Cloning into 'develop-git'...  
...output omitted...  
[student@workstation git-repos]$ cd develop-git
```

- 1.3. Create the `exercise` branch.

```
[student@workstation develop-git]$ git checkout -b exercise  
Switched to a new branch 'exercise'
```

- 2. Configure the global Git settings for the `student` user to specify a username, an email address, and a default push method. Additionally, specify the username to use when connecting to `https://git.lab.example.com` and allow Git to remember your password for two hours.

2.1. Use the `git config` command to customize Git globally:

```
[student@workstation develop-git]$ git config --global user.name 'Student User'
[student@workstation develop-git]$ git config --global \
> user.email student@lab.example.com
[student@workstation develop-git]$ git config --global push.default simple
[student@workstation develop-git]$ git config --global \
> credential.https://git.lab.example.com.username student
[student@workstation develop-git]$ git config --global \
> credential.helper cache --timeout=7200
```

2.2. Verify the global configuration settings:

```
[student@workstation develop-git]$ git config --global -l
user.name=Student User
user.email=student@lab.example.com
push.default=simple
credential.https://git.lab.example.com.username=student
credential.helper=cache
```

- 3. Review the files in the project directory by using the `tree` command:

```
[student@workstation develop-git]$ tree
.
└── apache-setup.yml
└── templates
    ├── httpd.conf.j2
    └── index.html.j2

1 directory, 3 files
```

- The `apache-setup.yml` file is the playbook.
- The `httpd.conf.j2` template customizes the Apache server configuration.
- The `index.html.j2` template generates a basic index page for the server.

- 4. Modify the `apache-setup.yml` playbook to target the `web_servers` host group.

4.1. Create the `inventory` file in the project directory with the following content:

```
[web_servers]
serverd
```

4.2. Create an Ansible configuration file in the project directory that uses the `inventory` file and connects to remote hosts as the `devops` user.

The new `ansible.cfg` file should consist of the following content:

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

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

- 4.3. Modify the `apache-setup.yml` playbook to target the `web_servers` host group.
The top of the modified playbook contains the following content:

```
---
- name: Install web servers with WSGI interface
  hosts: web_servers
  become: true
  vars:
    httpd_packages:
      - httpd
      - python3-mod_wsgi
    apache_test_message: This is a test message
    apache_max_keep_alive_requests: 115
...output omitted...
```

- 4.4. Review the current status of the local Git repository.

```
[student@workstation develop-git]$ git status
On branch exercise
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   apache-setup.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ansible.cfg
        inventory

no changes added to commit (use "git add" and/or "git commit -a")
```

- 4.5. Use the `git add` command to stage all three files for the next commit.

```
[student@workstation develop-git]$ git add apache-setup.yml ansible.cfg \
> inventory
```

- 4.6. Verify that all three files are staged.

```
[student@workstation develop-git]$ git status
On branch exercise
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ansible.cfg
        modified:   apache-setup.yml
        new file:   inventory
```

- 4.7. Commit the changes with the message: Target the `web_servers` group

```
[student@workstation develop-git]$ git commit -m "Target the web_servers group"
[exercise cc04da0] Target the web_servers group
 3 files changed, 6 insertions(+), 1 deletion(-)
 create mode 100644 ansible.cfg
 create mode 100644 inventory
```

- 4.8. Push the locally committed changes to the remote server. If prompted, use Student@123 as the password.

```
[student@workstation develop-git]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 494 bytes | 494.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for exercise, visit:
remote:   https://git.lab.example.com/student/develop-git/-/merge_requests/new?
remote: merge_request%5Bsource_branch%5D=exercise
remote:
To https://git.lab.example.com/student/develop-git.git
 * [new branch]      exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

- 5. Run the apache-setup.yml playbook using automation content navigator and then verify the content displayed by the web server.

- 5.1. Use the podman login command to log in to the classroom private automation hub at hub.lab.example.com.

When prompted, enter student as the username and redhat123 as the password.

This step simulates a customer logging in to registry.redhat.io and ensures that the specified automation execution environment is pulled down to the local system if it does not already exist.

```
[student@workstation develop-git]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 5.2. Using the ee-supported-rhel8:latest execution environment, run the apache-setup.yml playbook.

```
[student@workstation develop-git]$ ansible-navigator run apache-setup.yml \
> --eei ee-supported-rhel8:latest

Play name          Ok  Changed ... Failed ... Task count  Progress
0|Install web servers with ...  8       7 ...       0 ...        8 Complete

^f/PgUp page up    ^b/PgDn page down    ↑↓ scroll    esc back ...  Successful
```

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

Press ESC to exit the `ansible-navigator` command.

- 5.3. Use the `curl` command to review the content displayed by the web server.

```
[student@workstation develop-git]$ curl serverd
This is a test message RedHat 8.4 <br>
Current Host: serverd <br>
Server list: <br>
serverd <br>
```

- 5.4. Review the status of the local Git repository. The `ansible-navigator` command created the `.ssh/` directory, the `ansible-navigator.log` file, and one artifact file for each time that you used the `ansible-navigator` command to run a playbook.

```
[student@workstation develop-git]$ git status
On branch exercise
Your branch is up to date with 'origin/exercise'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ansible-navigator.log
    apache-setup-artifact-2022-09-12T20:00:56.911737+00:00.json

nothing added to commit but untracked files present (use "git add" to track)
```

- 5.5. Create a new `.gitignore` file to ignore files and directories created by the `ansible-navigator` command. The `.gitignore` file contains the following content:

```
ansible-navigator.log
*-artifact-*
```

- 5.6. Review the status of the local Git repository. Files and directories that should not be committed are now ignored.

```
[student@workstation develop-git]$ git status
On branch exercise
Your branch is up to date with 'origin/exercise'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

- 5.7. Add and commit the new `.gitignore` with the message: `Ignore generated files`

```
[student@workstation develop-git]$ git add .gitignore
[student@workstation develop-git]$ git commit -m "Ignore generated files"
[exercise b780076] Ignore generated files
 1 file changed, 2 insertions(+)
 create mode 100644 .gitignore
```

- 5.8. Push the latest commits to the remote repository. If prompted, use Student@123 as the password.

```
[student@workstation develop-git]$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 323 bytes | 323.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for exercise, visit:
remote:   https://git.lab.example.com/student/develop-git/-/merge_requests/new?
remote: merge_request%5Bsource_branch%5D=exercise
remote:
To https://git.lab.example.com/student/develop-git.git
 6f87080..4ec3448  exercise -> exercise
```



Important

In a more realistic scenario, you would also create a merge request or a pull request so that changes could be incorporated into the main branch.

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish develop-git
```

Implementing Recommended Ansible Practices

Objectives

- Demonstrate and describe commonly recommended practices for developing and maintaining effective Ansible automation solutions.

The Effectiveness of Ansible

As you work with more advanced features and larger, more complex projects, it becomes more difficult to manage and maintain Ansible Playbooks, or to use them effectively.

This course introduces a number of advanced features of Red Hat Ansible Automation Platform. Using Ansible effectively is not just about features or tools, but about practices and organization.

To paraphrase Ansible developer Jeff Geerling, using Ansible effectively relies on three key practices:

- Keeping things simple
- Staying organized
- Testing often

Keeping Things Simple

One of the strengths of Ansible is its simplicity. Simple playbooks are easier to use, modify, and understand.

Keeping Your Playbooks Readable

Keep your playbooks well commented and easy to read. Use vertical white space and comments liberally. Always give plays and tasks meaningful names that make clear what the play or task is doing. These practices help document the playbook and make it easier to troubleshoot a failed playbook run.

YAML is not a programming language. It is good at expressing a list of tasks or items, or a set of key-value pairs. If you are struggling to express a complex control structure or conditional in your Ansible Playbook, then consider approaching the problem differently. Templates and Jinja2 filters provide features to process data in a variable, which might be a better approach to your problem.

Use native YAML syntax, not the "folded" syntax. For example, the following example is not a recommended format:

```
- name: Postfix is installed and updated
  ansible.builtin.yum: name=postfix state=latest
  notify: restart_postfix

- name: Postfix is running
  ansible.builtin.service: name=postfix state=started
```

The following syntax is easier for most people to read:

```

- name: Postfix is installed and updated
  ansible.builtin.yum:
    name: postfix
    state: latest
    notify: update_postfix

- name: Postfix is running
  ansible.builtin.service:
    name: postfix
    state: started

```

Use Existing Modules

When writing a new playbook, start with a basic playbook and, if possible, a static inventory. Use `ansible.builtin.debug` tasks as stubs as you build your design. When your playbook functions as expected, break up your playbook into smaller, logical components by using imports and includes.

Use special-purpose modules included with Ansible when you can, instead of `ansible.builtin.command`, `ansible.builtin.shell`, `ansible.builtin.raw`, or other similar modules. It is easier to make your playbook idempotent and easier to maintain if you use the modules designed for a specific task.

Many modules have a default `state` or other variables that control what they do. For example, the `yum` module currently assumes that the package you name should be present in most cases. However, you should explicitly specify what `state` you want. Doing so makes it easier to read the playbook, and protects you against changes in the module's default behavior in later versions of Ansible.

Adhering to a Standard Style

Consider having a standard "style" that your team follows when writing Ansible projects. For example, how many spaces do you indent? How do you want vertical white space used? How should tasks, plays, roles, and variables be named? What should get commented on and how? Having a consistent standard can help improve maintainability and readability.

Staying Organized

Well-organized standards can help with maintainability, troubleshooting, and auditing. Take advantage of Ansible organization features, such as roles, so that you can reuse them.

Following Conventions for Naming Variables

Variable naming can be significant because Ansible has a reasonably flat namespace. Use descriptive variables, such as `apache_tls_port` rather than a less explanatory variable such as `p`. In roles, it is a good practice to prefix role variables with the role name.

If the name of your role is `myapp` then prefix your variables with `myapp_` so that you can easily identify them from variables in other roles and the playbook.

Variable names should clarify contents. A name such as `apache_max_keepalive` clearly explains the meaning of the associated values. Prefix roles and group variables with the name of the role or group to which the variable belongs. `apache_port_number` is more error-resistant than `port_number`.

Standardizing the Project Structure

Use a consistent pattern when structuring the files of your Ansible project on a file system. The following is a good example:

```

├── dbservers.yml
└── inventories/
    ├── prod/
    │   ├── group_vars/
    │   ├── host_vars/
    │   └── inventory/
    └── stage/
        ├── group_vars/
        ├── host_vars/
        └── inventory/
└── roles/
    └── std_server/
└── site.yml
└── storage.yml
└── webservers.yml

```

The `site.yml` file is the main playbook, which includes or imports playbooks that perform specific tasks: `dbservers.yml`, `storage.yml`, and `webservers.yml`. Each role is located in a subdirectory in the `roles` directory, such as `std_server`. Two static inventories in the `inventories/prod` and `inventories/stage` directories provide separate inventory variable files so that you can select different sets of servers by changing the inventory that you use.

One of the playbook structure benefits is that you can divide up your extensive playbook into smaller files to make it more readable. Those smaller playbooks can contain plays for a specific purpose that you can run independently.

Using Dynamic Inventories

Use dynamic inventories whenever possible. Dynamic inventories enable central management of your hosts and groups from one primary source of truth. They also ensure that you have an updated list of hosts and groups before you run your playbooks. Dynamic inventories are especially powerful when used in conjunction with cloud providers, containers, and virtual machine management systems. Those systems might already have inventory information in a form that Ansible can consume.

If you cannot use dynamic inventories, then other tools can help you construct groups or extra information. For example, you can use the `group_by` module to generate group membership based on a fact. That group membership is valid for the rest of the playbook.

```

- name: Generate dynamic groups
  hosts: all
  tasks:
    - name: Generate dynamic groups based on architecture
      ansible.builtin.group_by:
        key: arch_"{{ ansible_facts['architecture'] }}"
    - name: Configure x86_64 systems
      hosts: arch_x86_64

```

```
tasks:  
  - name: First task for x86_64 configuration  
  ...output omitted...
```

Taking Advantage of Groups

Hosts can be members of many groups. Consider dividing your hosts into different categories based on different characteristics:

Geographical

Differentiate hosts from different regions, countries, continents, or data centers.

Environmental

Differentiate hosts dedicated to different stages of the software lifecycle, including development, staging, testing, and production.

Sites or services

Group hosts that offer or link to a subset of functions, such as a specific website, an application, or a subset of features.



Important

Remember that hosts inherit variables from all groups of which they are members. If two groups have different settings for the same variable, and a host is a member of both, then the value that is used is the last one loaded.

If there are differences in settings between two different groups that might be used at the same time, then take special care to determine how those variables should be set.

Using Roles and Ansible Content Collections for Reusable Content

Roles and collections help you keep your playbooks simple and enable you to save work by reusing standard code across projects. If you write your own roles and collections, keep them focused on a particular purpose or function, just like you do with playbooks. Make roles generic and configurable through variables so that you do not need to edit them when you use them with a different set of playbooks.

Use the `ansible-galaxy` command to initialize the directory hierarchy for your role or collection and provide initial template files. This makes it easier to share your content through private automation hub or on the community Ansible Galaxy website.

Take advantage of Red Hat Ansible Certified Content Collections. These are supported with your Ansible Automation Platform subscription, and provide many useful functions and features.

You can also examine the roles provided by the community through Ansible Galaxy. Be aware that these roles have varying levels of quality, so choose the ones that you use carefully.

Keep your roles in the `roles` subdirectory of your project, and your collections in the `collections` subdirectory of your project. (Your collections might contain roles, of course, and those should stay in their collection's directory.) Use the `ansible-galaxy` command to get roles from automation hub or a separate Git repository automatically.

Running Playbooks Centrally

To control access to your systems and audit Ansible activity, consider using a dedicated control node from which all Ansible Playbooks are run. Ideally, you should use automation controller.

System administrators should still have their own accounts on the system, as well as credentials to connect to managed hosts and escalate privileges, if needed. When a system administrator leaves, their SSH key can be removed from managed hosts' `authorized_keys` file and their `sudo` command privileges revoked, without impacting other administrators.

Consider using automation controller as this central host. Automation controller is included with your Red Hat Ansible Automation Platform subscription, and provides features that make it easier to control access to credentials, control playbook execution, simplify automation for users who are not comfortable with the Linux command line, as well as audit and track playbook runs. Later in this course, you learn about using automation controller. However, even if you do not use automation controller, using a central control node can be beneficial.

Building Automation Execution Environments

Create a custom automation execution environment if you need to frequently use specific Ansible Content Collections, especially if the collection has Python dependencies or system executables not included in the supported automation execution environments.

However, if you can use an existing automation execution environment to run your playbooks, you should consider doing that. Reusing automation execution environments can reduce maintenance effort and the number of execution environments that you must manage.

Performing Regular Testing

Test your playbooks and your tasks frequently during the development process, when the tasks run, and after the playbooks are in use.

Testing the Results of Tasks

If you need to confirm that a task succeeded, then verify the result of the task rather than trusting the return code of the module. There is more than one way to verify a task, depending on the module involved.

```
- name: Start web server
  ansible.builtin.service:
    name: httpd
    status: started

- name: Check web site from web server
  ansible.builtin.uri:
    url: http://{{ ansible_fqdn }}
    return_content: true
  register: example_webpage
  failed_when: example_webpage.status != 200
```

Using the Block and Rescue Directives to Recover or Roll Back

The `block` directive is helpful for grouping tasks; when used in conjunction with the `rescue` directive, it is useful when recovering from errors or failures.

```
- block:
  - name: Check web site from web server
    ansible.builtin.uri:
      url: http://{{ ansible_fqdn }}
      return_content: true
    register: example_webpage
    failed_when: example_webpage.status != 200

  rescue:
    - name: Restart web server
      ansible.builtin.service:
        name: httpd
        status: restarted
```

Developing Playbooks with the Latest Ansible Version

Even if you are not using the latest version of Ansible Core in production, you should routinely test your playbooks against the latest version of Ansible Core. This test helps you to avoid issues as Ansible modules and features evolve.

Automation execution environments can help make this easier by providing current and legacy versions of Ansible Core that you can use to test your playbooks.

If your playbooks print warnings or deprecation messages when they run, then you should pay attention to them and make adjustments. If a feature in Ansible Core is being deprecated or is changing, then the project provides deprecation notices for four minor releases before the feature gets removed or altered.

To prepare for changes and future updates to the upstream Ansible distribution, read the Ansible Porting Guides at https://docs.ansible.com/ansible-devel/porting_guides/porting_guides.html.



Important

Ansible 6 is not identical to Red Hat Ansible Automation Platform 2.2, but both use Ansible Core 2.13. Guidance in the upstream Porting Guides can be useful. However, you should use them with some caution.

Using Test Tools

Many commands and tools are available to help you test your playbooks. Use the `ansible-navigator run playbook --syntax-check -m stdout` command to validate the syntax of your playbook without running it.

The `ansible-lint` tool, added as a Technology Preview in Red Hat Ansible Automation Platform 2.2, parses your playbook and looks for possible issues. Not all the issues it reports result in playbook failures, but reported issues might indicate the presence of an error.



Note

Other useful tools are available that are not currently shipped in Red Hat Ansible Automation Platform 2; these tools might be included in Extra Packages for Enterprise Linux (EPEL), or can be obtained from upstream sources. Red Hat does not support these packages.

For example, the `yamllint` tool parses a YAML file and attempts to identify issues with the YAML syntax. This tool does not have direct knowledge of Ansible, but it can catch potential YAML syntax problems.



References

Ansible Documentation: Best Practices: Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_best_practices.html

Ansible Porting Guides

https://docs.ansible.com/ansible-devel/porting_guides/porting_guides.html

► Guided Exercise

Implementing Recommended Ansible Practices

Apply recommended practices to an existing Ansible automation project.

Outcomes

- Set up a web server on `servera.lab.example.com` and `serverb.lab.example.com`.
- Set up a database server on `serverc.lab.example.com` and `serverd.lab.example.com`.
- Identify points of improvement on Ansible projects.
- Implement recommended practices.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates a project directory for the exercise.

```
[student@workstation ~]$ lab start develop-practices
```

Instructions

- 1. The `lab start` command creates an Ansible project in the `~/develop-practices` directory. The `deploy.yml` playbook deploys the Apache HTTPD web server and the MariaDB database server.

Change to the project directory and review the `deploy.yml` playbook.

```
[student@workstation ~]$ cd ~/develop-practices/
[student@workstation develop-practices]$ cat deploy.yml
---
- name: Deploy a web application
  hosts: webservers
  vars:
    service: httpd
    fw_service: http
    package:
      - httpd

  tasks:
    - name: Install the webserver
      ansible.builtin.yum: name={{ package }} state=latest
```

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

```
- name: Deploy web content
ansible.builtin.template:
  src: templates/index.html.j2
  dest: "{{ web_dest }}"

- name: Start the webserver
ansible.builtin.service:
  name: "{{ service }}"
  state: started
  enabled: true

- name: Enable the {{ fw_service }} service in the firewall
ansible.posix.firewalld:
  service: "{{ fw_service }}"
  state: enabled
  permanent: true
  immediate: true

- hosts:
  - serverc.lab.example.com
  - serverd.lab.example.com
vars:
  service: mariadb
  fw_service: mysql
  package:
    - mariadb-server

tasks:

- name: Install the database server
ansible.builtin.yum:
  name: "{{ package }}"
  state: latest

- name: Start the database server
ansible.builtin.shell:
  cmd: systemctl enable --now {{ service }}

- name: Enable the {{ fw_service }} service in the firewall
ansible.posix.firewalld:
  service: "{{ fw_service }}"
  state: enabled
  permanent: true
  immediate: true

- name: Test web application
hosts: webservers

tasks:

#TODO: Add test
```

- 2. The second play installs the MariaDB database server on the `serverc.lab.example.com` and `serverd.lab.example.com` hosts. Create a name

for that play, and replace its list of hosts with a group so that it can be updated more easily, as follows:

- 2.1. In the `~/develop-practices/inventory` file, create the `dbservers` group, consisting of `serverc.lab.example.com` and `serverd.lab.example.com`. The updated `~/develop-practices/inventory` file should now consist of the following content:

```
...output omitted...
[webservers]
servera.lab.example.com
serverb.lab.example.com

[dbservers]
serverc.lab.example.com
serverd.lab.example.com
```

- 2.2. In the `~/develop-practices/deploy.yml` playbook, change the second play to run against the `dbservers` group, and give that unnamed play the name `Deploy database servers`. The beginning of the updated second play in `~/develop-practices/deploy.yml` should now contain the following content:

```
...output omitted...
- name: Deploy database servers
  hosts: dbservers
  vars:
    service: mariadb
    fw_service: mysql
    package:
      - mariadb-server
...output omitted...
```

- 3. Update the variables used in the first two plays of the `deploy.yml` playbook so that how they are defined and named follow better practices.

- 3.1. In the first play in the `~/develop-practices/deploy.yml` playbook, move the `service`, `fw_service`, and `package` variables to the `~/develop-practices/group_vars/webservers` group variable file. Rename them to `web_service`, `web_fw_service`, and `web_package`, respectively.

The `~/develop-practices/group_vars/webservers` file should now contain the following content:

```
---
web_dest: /var/www/html/index.html
web_service: httpd
web_fw_service: http
web_package:
  - httpd
```

- 3.2. In the first play in the `~/develop-practices/deploy.yml` file, remove the definitions of the `service`, `fw_service`, and `package` variables in `vars`.

The `~/develop-practices/deploy.yml` file should now consist of the following content:

```
---  
- name: Deploy a web application  
  hosts: webservers  
  
  tasks:  
  
    - name: Install the webserver  
      ansible.builtin.yum: name={{ package }} state=latest  
...output omitted...
```

- 3.3. In the first play in the `~/develop-practices/deploy.yml` file, update the variable names used in the following tasks:

- Install the webserver
- Start the webserver
- Enable the `{{ fw_service }}` service in the firewall

The updated tasks should now consist of the following content:

```
- name: Install the webserver  
  ansible.builtin.yum: name={{ web_package }} state=latest  
...output omitted...  
  
- name: Start the webserver  
  ansible.builtin.service:  
    name: "{{ web_service }}"  
    state: started  
    enabled: true  
  
- name: Enable the {{ web_fw_service }} service in the firewall  
  ansible.posix.firewalld:  
    service: "{{ web_fw_service }}"  
    state: enabled  
    permanent: true  
    immediate: true
```

- 3.4. Create the `~/develop-practices/group_vars/dbservers` file.

Move the `service`, `fw_service`, and `package` variables from the second play into the `~/develop-practices/group_vars/dbservers` file with the names `db_service`, `db_fw_service`, and `db_package`, respectively.

The `~/develop-practices/group_vars/dbservers` file should now contain the following content:

```
---  
db_service: mariadb  
db_fw_service: mysql  
db_package:  
  - mariadb-server
```

- 3.5. In the `~/develop-practices/deploy.yml` playbook, remove the `service`, `fw_service`, and `package` variables from `vars` in the second play, because they are now defined in the `dbservers` group variable file using new names.

The `~/develop-practices/deploy.yml` file should now consist of the following content:

```
...output omitted...
- name: Deploy database servers
  hosts: dbservers

  tasks:
...output omitted...
```

- 3.6. In the `~/develop-practices/deploy.yml` playbook, in the second play, update the variable names used in the following tasks:

- Install the database server
- Start the database server
- Enable the `{{ fw_service }}` service in the firewall

The updated tasks should now consist of the following content:

```
- name: Install the database server
  ansible.builtin.yum:
    name: "{{ db_package }}"
    state: latest

- name: Start the database server
  ansible.builtin.shell:
    cmd: systemctl enable --now {{ db_service }}

- name: Enable the {{ db_fw_service }} service in the firewall
  ansible.posix.firewalld:
    service: "{{ db_fw_service }}"
    state: enabled
    permanent: true
    immediate: true
```

- 4. The `Install the webserver` task in the first play uses folded syntax for the `ansible.builtin.yum` module. Change the task to use native YAML syntax.

The updated task should now consist of the following content:

```
- name: Install the webserver
  ansible.builtin.yum:
    name: "{{ web_package }}"
    state: latest
```

- 5. The `Start the webserver` and `Enable the {{ web_fw_service }} service in the firewall` tasks both use indentation that is not standard with the rest of the playbook. Update the indentation in the tasks.

The updated tasks should now consist of the following content:

```
- name: Start the webserver
ansible.builtin.service:
  name: "{{ web_service }}"
  state: started
  enabled: true

- name: Enable the {{ web_fw_service }} service in the firewall
ansible.posix.firewalld:
  service: "{{ web_fw_service }}"
  state: enabled
  permanent: true
  immediate: true
```

- ▶ 6. The `Start the database server` task uses the `ansible.builtin.shell` module to run the `systemctl` command to enable and start a service. Change this task to use the `ansible.builtin.service` module instead.

The updated task should now consist of the following content:

```
- name: Start the database server
ansible.builtin.service:
  name: "{{ db_service }}"
  state: started
  enabled: true
```

- ▶ 7. The third play in the `~/develop-practices/deploy.yml` playbook has no tasks. Add `block` and `rescue` directives to that play that restart the web service on the hosts in the `webservers` group if the web application is unavailable.

The updated third play should now consist of the following content:

```
- name: Test web application
hosts: webservers

tasks:

  - name: Web application test
    block:
      - name: Check web application
        ansible.builtin.uri:
          url: http://{{ item }}
        loop: "{{ groups['webservers'] }}"

    rescue:
      - name: Restart web server
        ansible.builtin.service:
          name: "{{ web_service }}"
          state: restarted
```

- 8. Review your playbook. The completed ~/develop-practices/deploy.yml playbook should consist of the following content:

```
---
```

```
- name: Deploy a web application
  hosts: webservers

  tasks:

    - name: Install the webserver
      ansible.builtin.yum:
        name: "{{ web_package }}"
        state: latest

    - name: Deploy web content
      ansible.builtin.template:
        src: templates/index.html.j2
        dest: "{{ web_dest }}"

    - name: Start the webserver
      ansible.builtin.service:
        name: "{{ web_service }}"
        state: started
        enabled: true

    - name: Enable the {{ web_fw_service }} service in the firewall
      ansible.posix.firewalld:
        service: "{{ web_fw_service }}"
        state: enabled
        permanent: true
        immediate: true

- name: Deploy database servers
  hosts: dbservers

  tasks:

    - name: Install the database server
      ansible.builtin.yum:
        name: "{{ db_package }}"
        state: latest

    - name: Start the database server
      ansible.builtin.service:
        name: "{{ db_service }}"
        state: started
        enabled: true

    - name: Enable the {{ db_fw_service }} service in the firewall
      ansible.posix.firewalld:
        service: "{{ db_fw_service }}"
        state: enabled
        permanent: true
        immediate: true
```

```
- name: Test web application
  hosts: webservers

  tasks:

    - name: Web application test
      block:
        - name: Check web application
          ansible.builtin.uri:
            url: http://{{ item }}
          loop: "{{ groups['webservers'] }}"

  rescue:
    - name: Restart web server
      ansible.builtin.service:
        name: "{{ web_service }}"
        state: restarted
```

- 9. Run the `deploy.yml` playbook using automation content navigator.

```
[student@workstation develop-practices]$ ansible-navigator run deploy.yml \
> --eei ee-supported-rhel8 --pp missing

Play name           Ok  Changed ... Failed ... Task count  Progress
0|Deploy a web application  10      8 ...      0 ...          10  Complete
1|Deploy database servers    8      6 ...      0 ...          8   Complete
2|Test web application       4      0 ...      0 ...          4   Complete

^b/PgUp page up  ^f/PgDn page down  ↑↓ scroll  esc ...          Successful
```

Press ESC to exit from the `ansible-navigator` command.

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish develop-practices
```

▶ Lab

Developing Playbooks with Ansible Automation Platform

Improve an existing Ansible Playbook to follow recommended practices, commit the changes to its Git repository, and test it with automation content navigator.

Outcomes

- Create roles with appropriate names.
- Implement style guidelines for task and variable names in roles and playbooks.
- Commit changes to a Git repository.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository at <https://git.lab.example.com/student/develop-review.git>. The Git repository contains a `site.yml` playbook that uses a role to configure the number of seconds that GRUB waits before proceeding with the boot process. You can push changes to this repository by using `Student@123` as the Git password.

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

Instructions

1. Clone the <https://git.lab.example.com/student/develop-review.git> Git repository into the `/home/student/git-repos` directory and then create a new branch named `exercise`.
2. Ensure that plays and tasks have names associated with them. Add a name to the play in the `site.yml` playbook and add names to the two tasks in the `roles/test/tasks/main.yml` file.
3. The `site.yml` playbook uses the `test` role to deploy the configuration changes. This role does not have a suitable name. Rename this role to `grub` and then update the `site.yml` playbook accordingly.
4. Change the two role variables names (`timeout` and `persistent`) to use the `grub_` prefix. Update the `roles/grub/defaults/main.yml`, `group_vars/lb_servers.yml`, and `roles/grub/tasks/main.yml` files to use the new variable names. Although not evaluated, you might update the `roles/grub/README.md` file to reflect changes made in this exercise.
5. Test the `~/git-repos/develop-review/site.yml` playbook. Using the `ee-supported-rhel8` execution environment, verify that the `site.yml` file runs without any errors. Automation execution environments are available from `hub.lab.example.com` by using `student` as the username and `redhat123` as the password.

6. Push all of your changes to the `exercise` branch of the remote Git repository.

Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures. Make sure to commit and push any additional changes to the remote Git repository and then rerun the command until successful.

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

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Developing Playbooks with Ansible Automation Platform

Improve an existing Ansible Playbook to follow recommended practices, commit the changes to its Git repository, and test it with automation content navigator.

Outcomes

- Create roles with appropriate names.
- Implement style guidelines for task and variable names in roles and playbooks.
- Commit changes to a Git repository.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository at `https://git.lab.example.com/student/develop-review.git`. The Git repository contains a `site.yml` playbook that uses a role to configure the number of seconds that GRUB waits before proceeding with the boot process. You can push changes to this repository by using `Student@123` as the Git password.

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

Instructions

1. Clone the `https://git.lab.example.com/student/develop-review.git` Git repository into the `/home/student/git-repos` directory and then create a new branch named `exercise`.
 - 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/develop-review.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/develop-review.git  
Cloning into 'develop-review'...  
...output omitted...  
[student@workstation git-repos]$ cd develop-review
```

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

- 1.3. Create the `exercise` branch.

```
[student@workstation develop-review]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

2. Ensure that plays and tasks have names associated with them. Add a name to the play in the `site.yml` playbook and add names to the two tasks in the `roles/test/tasks/main.yml` file.

- 2.1. Edit the `site.yml` playbook to include a name for the play.

```
---
- name: Sets the GRUB timeout
  hosts: all
  become: true
  roles:
    - role: test
```

- 2.2. Edit the `roles/test/tasks/main.yml` file to add a name to each task.

```
---
- name: Sets persistent GRUB timeout
  ansible.builtin.lineinfile:
    path: /etc/default/grub
    regexp: "^\$GRUB_TIMEOUT="
    line: "GRUB_TIMEOUT={{ timeout | int }}"
  when: persistent | bool == true

- name: Sets temporary GRUB timeout
  ansible.builtin.lineinfile:
    path: /boot/grub2/grub.cfg
    regexp: "^\$set timeout="
    line: "set timeout={{ timeout | int }}"
```

3. The `site.yml` playbook uses the `test` role to deploy the configuration changes. This role does not have a suitable name. Rename this role to `grub` and then update the `site.yml` playbook accordingly.

- 3.1. Edit the related variables to reflect the new role location. Edit the `site.yml` file to show the new role.

```
---
- name: Sets the GRUB timeout
  hosts: all
  become: true
  roles:
    - role: grub
```

- 3.2. Rename the role by moving the `roles/test` subdirectory to `roles/grub`.

**Note**

Use the `git mv` command to rename the `test` role to the `grub` role. The `git mv` command renames the file or directory *and* automatically stages the change.

```
[student@workstation develop-review]$ git mv roles/test roles/grub
```

4. Change the two role variables names (`timeout` and `persistent`) to use the `grub_` prefix. Update the `roles/grub/defaults/main.yml`, `group_vars/lb_servers.yml`, and `roles/grub/tasks/main.yml` files to use the new variable names. Although not evaluated, you might update the `roles/grub/README.md` file to reflect changes made in this exercise.

- 4.1. Edit the `roles/grub/defaults/main.yml` file to add the `grub_` prefix.

```
---
grub_timeout: 0
grub_persistent: true
```

- 4.2. Edit the `group_vars/lb_servers.yml` to add the `grub_` prefix.

```
---
grub_timeout: 30
grub_persistent: true
```

- 4.3. Edit the `roles/grub/tasks/main.yml` file.

```
---
- name: Sets persistent GRUB timeout
  ansible.builtin.lineinfile:
    path: /etc/default/grub
    regexp: "^\$GRUB_TIMEOUT="
    line: "GRUB_TIMEOUT={{ grub_timeout | int }}"
    when: grub_persistent | bool == true

- name: Sets temporary GRUB timeout
  ansible.builtin.lineinfile:
    path: /boot/grub2/grub.cfg
    regexp: "^\$set timeout="
    line: "set timeout={{ grub_timeout | int }}"
```

- 4.4. (Optional) Update the `roles/grub/README.md` file to change the role and variable names. The `README.md` file provides information on how to use the role, but the file is not used by the role itself. The grading script does not evaluate this file.

...output omitted...

* `grub_timeout`: An integer used to set the GRUB timeout (in seconds). If a user does not interrupt the boot process within this timeout, then the system uses the default kernel with the default boot options. Defaults to a value of `0`.

Chapter 1 | Developing Playbooks with Ansible Automation Platform 2

* `grub_persistent`: A boolean that controls whether a change to the GRUB timeout is persistent or temporary. A persistent change also updates the `/etc/default/grub` file so that the new timeout will be applied, even if students install a new kernel. Defaults to a value of `true`.

Example Playbook

```
-----
- name: Sets the GRUB timeout
  hosts: all
  become: true
  roles:
    - role: grub
      grub_timeout: 15
      grub_persistent: true
...output omitted...
```

5. Test the ~/git-repos/develop-review/site.yml playbook. Using the ee-supported-rhel8 execution environment, verify that the site.yml file runs without any errors. Automation execution environments are available from hub.lab.example.com by using student as the username and redhat123 as the password.
 - 5.1. Run the site.yml playbook using the ee-supported-rhel8 execution environment. Verify that the playbook has no errors.

```
[student@workstation develop-review]$ ansible-navigator run site.yml \
> --eei ee-supported-rhel8 --pp missing -m stdout

PLAY [Sets the GRUB timeout] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]
ok: [serverc.lab.example.com]

TASK [grub : Sets persistent GRUB timeout] ****
ok: [serverb.lab.example.com]
changed: [servera.lab.example.com]
ok: [serverc.lab.example.com]

TASK [grub : Sets temporary GRUB timeout] ****
changed: [servera.lab.example.com]
ok: [serverb.lab.example.com]
ok: [serverc.lab.example.com]

PLAY RECAP ****
servera.lab.example.com    : ok=3    changed=2    unreachable=0    failed=0    ...
serverb.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0    ...
serverc.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0    ...
```

6. Push all of your changes to the exercise branch of the remote Git repository.

- 6.1. Display changes made to the local Git repository:

```
[student@workstation develop-review]$ git status
On branch exercise
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:  roles/test/README.md -> roles/grub/README.md
    renamed:  roles/test/defaults/main.yml -> roles/grub/defaults/main.yml
    renamed:  roles/test/meta/main.yml -> roles/grub/meta/main.yml
    renamed:  roles/test/tasks/main.yml -> roles/grub/tasks/main.yml

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified: group_vars/lb_servers.yml
    modified: roles/grub/README.md
    modified: roles/grub/defaults/main.yml
    modified: roles/grub/tasks/main.yml
    modified: site.yml
```

6.2. Add the files you created and edited to the staging area:

```
[student@workstation develop-review]$ git add group_vars/ roles/ site.yml
```

6.3. Commit the staged files to the local Git repository:

```
[student@workstation develop-review]$ git commit -m 'Changes for exercise'
...output omitted...
```

6.4. Push the changes from the local Git repository to the **exercise** branch on the remote Git repository. If prompted, use **Student@123** as the Git password.

```
[student@workstation develop-review]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
```

6.5. Verify that your changes have been pushed to the **exercise** branch:

```
[student@workstation develop-review]$ git status
On branch exercise
Your branch is up to date with 'origin/exercise'.

nothing to commit, working tree clean
```

Evaluation

As the **student** user on the **workstation** machine, use the **lab** command to grade your work. Correct any reported failures. Make sure to commit and push any additional changes to the remote Git repository and then rerun the command until successful.

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

Finish

On the **workstation** machine, change to the student user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Red Hat Ansible Automation Platform 2 simplifies development and use of Ansible Playbooks at scale, by introducing new features such as Ansible Content Collections and automation execution environments.
- Automation content navigator (`ansible-navigator`) extends and can replace many previous Ansible commands, including `ansible-playbook`.
- An Ansible Content Collection provides a set of related Ansible modules, roles, and plug-ins that are supported by the same group of software developers.
- Automation execution environments are container images that contain a specific version of Ansible Core, Ansible Content Collections, and any Python libraries, executables, or other dependencies needed to run playbooks.
- You should manage your automation content in a version control system, such as Git, which enables you to track and control changes to your automation code.
- Implementing recommended practices facilitates collaboration and reduces potential problems.

Chapter 2

Managing Content Collections and Execution Environments

Goal

Run playbooks that use content collections not included in `ansible-core`, either from an existing execution environment or by downloading them from automation hub.

Objectives

- Describe how Ansible Content Collections are used to distribute modules and plug-ins, and create plays that use content from them.
- Search automation hub for Ansible Content Collections, and install them from the command line by name or by using a `requirements.yml` file.
- Identify the automation execution environments provided by Red Hat and select the correct one for your use case.

Sections

- Reusing Content from Ansible Content Collections (and Guided Exercise)
- Finding and Installing Ansible Content Collections (and Guided Exercise)
- Selecting an Execution Environment (and Guided Exercise)

Lab

- Managing Content Collections and Execution Environments

Reusing Content from Ansible Content Collections

Objectives

- Describe how Ansible Content Collections are used to distribute modules and plug-ins, and create plays that use content from them.

Defining Ansible 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.

You can use collections to separate core Ansible code updates from updates to modules and plug-ins. Vendors and developers can then 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.

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 a later one) or choose between a version of a collection supported by Red Hat and one provided by the community.

Upstream Ansible unbundled most modules from the core Ansible code in Ansible Base 2.10 and Ansible Core 2.11 and placed them in collections. Red Hat Ansible Automation Platform 2.2 provides automation execution environments based on Ansible Core 2.13 that inherit this feature.

Organizing Ansible Content 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 maintain are in the `community` namespace, and have names such as `community.crypto`, `community.postgresql`, and `community.rabbitmq`. Collections that Red Hat maintain and support might use the `redhat` namespace, and have names such as `redhat.rhv`, `redhat.satellite`, and `redhat.insights`.

Names of namespaces are limited to ASCII lowercase letters, numbers, and underscores, must be at least two characters long, and must not start with an underscore.

Using Ansible Content Collections

The automation execution environments that Red Hat provides already include some collections. You can install additional collections on your local system or create a custom automation execution environment that incorporates these collections. Other sections of the course present those subjects in more detail.

Accessing Ansible Content Collection Documentation

Use the `ansible-navigator collections` command to list the collections available in automation execution environments. The following output shows the collections available in the `ee-supported-rhel8` automation execution environment:

Name	Version	Shadowed	Type	Path
0 amazon.aws	3.2.0	False	contained	/usr/share/ansible/collec
1 ansible.builtin	2.13.3	False	contained	/usr/lib/python3.9/site-p
2 ansible.controller	4.2.1	False	contained	/usr/share/ansible/collec
3 ansible.netcommon	3.0.0	False	contained	/usr/share/ansible/collec
4 ansible.network	1.2.0	False	contained	/usr/share/ansible/collec
5 ansible.posix	1.3.0	False	contained	/usr/share/ansible/collec
6 ansible.security	1.0.0	False	contained	/usr/share/ansible/collec
7 ansible.utils	2.6.1	False	contained	/usr/share/ansible/collec
8 ansible.windows	1.9.0	False	contained	/usr/share/ansible/collec
9 ansible.yang	1.0.0	False	contained	/usr/share/ansible/collec
10 arista.eos	5.0.0	False	contained	/usr/share/ansible/collec
<i>...output omitted...</i>				
21 redhat.insights	1.0.7	False	contained	/usr/share/ansible/collec
22 redhat.openshift	2.1.0	False	contained	/usr/share/ansible/collec
<i>...output omitted...</i>				

To list the modules and plug-ins for a collection, type the associated collection number. If the collection number is greater than nine, type a colon before the collection number, for example, :21, for line 21, and then press **Enter**. The following output shows the modules available in the `redhat.insights` collection.

Redhat.insights	Type	Added	Deprecated	Description
0 compliance	role	Unknown	Unknown	Install and configure Red
1 insights	inventory	None	False	insights inventory source
2 insights_client	role	Unknown	Unknown	Install and configure Red
3 insights_config	module	None	False	This module handles initi
4 insights_register	module	None	False	This module registers the

Enter the module number to access its documentation. The `ansible-navigator collections` command renders the documentation in the YAML format by default. You can use the `--mode stdout` (or `-m stdout`) option of the `ansible-navigator doc` command to render the documentation in plain text. For example, the following command displays the documentation of the `insights_register` module:

```
[user@host ~]$ ansible-navigator doc redhat.insights.insights_register \
> --mode stdout
> REDHAT.INSIGHTS.INSIGHTS_REGISTER (/usr/share/ansible/collections/ansible_>
This module will check the current registration status,
```

```
unregister if needed, and then register the insights client  
(and update the display_name if needed)
```

OPTIONS (= is mandatory):

- **display_name**
This option is here to enable registering with a display_name outside of using a configuration file. Some may be used to doing it this way so I left this in as an optional parameter.
[Default: (null)]
type: str
- **force_reregister**
This option should be set to true if you wish to force a reregister of the insights-client. Note that this will remove the existing machine-id and create a new one. Only use this option if you are okay with creating a new machine-id.
[Default: False]
type: bool
...output omitted...

Using Ansible Content Collections in Playbooks

To use a module or a role from a collection, refer to it with its *fully qualified collection name (FQCN)*. For example, use `redhat.insights.insights_register` to refer to the `insights_register` module.

The play in the following playbook includes a task that calls the `insights_register` module.

```
---  
- name: Register new systems  
  hosts: db.example.com  
  
  tasks:  
    - name: Ensure the new system is registered with Red Hat Insights  
      redhat.insights.insights_register:  
        state: present  
        force_reregister: true
```

The play in 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  
      ansible.builtin.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
```

The play in the following playbook uses the k8s lookup plug-in from the `kubernetes.core` collection:

```
---
- name: Fetch the namespaces from Kubernetes
  hosts: localhost

  tasks:
    - name: Retrieve the namespaces as a list
      ansible.builtin.set_fact:
        ns: "{{ lookup('kubernetes.core.k8s', kind='Namespace') }}"
```

Finding Ansible Content Collections

If you plan to update legacy playbooks that use modules moved to collections, then you need to identify in which collection these modules are now available.

The `https://github.com/ansible/ansible/blob/devel/lib/ansible/config/ansible_builtin_runtime.yml` file maps earlier module names from Ansible 2.9 to their new FQCNs.

For example, the following extract from that file shows that the `acl` module is now part of the `ansible.posix` collection, which uses `ansible.posix.acl` as its FQCN.

```
...output omitted...
acl:
  redirect: ansible.posix.acl
  synchronize:
    redirect: ansible.posix.synchronize
  at:
    redirect: ansible.posix.at
  authorized_key:
    redirect: ansible.posix.authorized_key
  mount:
    redirect: ansible.posix.mount
  seboolean:
    redirect: ansible.posix.seboolean
  selinux:
    redirect: ansible.posix.selinux
...output omitted...
```

This redirection mechanism is also used by many Ansible Content Collections to translate short names to FQCNs, so that playbooks written for Ansible 2.9 (Red Hat Ansible Automation Platform 1.2) require less immediate migration work.

**Note**

For playbooks that do not use Ansible Content Collections, you can manage the migration process by using the ee-29-rhel8 automation execution environment. That automation execution environment provides Ansible 2.9, which does not require collections.

Using the Built-in Ansible Content Collection

Ansible always includes a special collection named `ansible.builtin`. This collection includes a set of common modules, such as `copy`, `template`, `file`, `yum`, `command`, and `service`.

You can use the short names of these modules in your playbooks. For example, you can use `file` to refer to the `ansible.builtin.file` module. This mechanism allows many existing Ansible Playbooks to work without modification, although you might need to install additional collections for modules that are not included in the `ansible.builtin` special collection.

However, Red Hat recommends that you use the FQCN notation to prevent 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

Knowledgebase: "Where Did a Module Go During the Ansible 2.9 to 2.10 Content Migration?"

<https://access.redhat.com/solutions/5295121>

Using Collections – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/collections_using.html

The Future of Ansible Content Delivery

<https://www.ansible.com/blog/the-future-of-ansible-content-delivery>

Getting Started With Ansible Content Collections

<https://www.ansible.com/blog/getting-started-with-ansible-collections>

► Guided Exercise

Reusing Content from Ansible Content Collections

Write a playbook that uses modules from an Ansible Content Collection.

Outcomes

- Identify the Ansible Content Collection that provides a specific module.
- Create a task that specifies a module by its fully qualified collection name.
- Use the `collections` keyword in a playbook.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that a project directory is created with the required files for this exercise.

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

Instructions

- 1. Change to the `/home/student/manage-reusing/` directory and attempt to run the `basic-web.yml` playbook by using the `ansible-playbook` command.
- Change to the `/home/student/manage-reusing/` directory and list the contents of the directory.

```
[student@workstation ~]$ cd ~/manage-reusing/  
[student@workstation manage-reusing]$ ls  
ansible.cfg basic-web.yml inventory
```

- Use the `ansible-playbook` command to attempt to run the `basic-web.yml` playbook. The attempt fails with an error message.

```
[student@workstation manage-reusing]$ ansible-playbook basic-web.yml  
ERROR! couldn't resolve module/action 'firewalld'. This often indicates a  
misspelling, missing collection, or incorrect module path.  
...output omitted...
```

- Attempt to find the `firewalld` module in the default modules installed by the `ansible-core` package. The command does not return output indicating that the module is not found.

```
[student@workstation manage-reusing]$ ansible-doc -l | grep firewalld
```



Important

The `ansible-playbook` command provided by Red Hat Ansible Automation Platform 2 is based on the Ansible Core 2.13 package. It does not include any Ansible Content Collections other than `ansible.builtin`, which is why the preceding two substeps fail. Your control node also does not have any additional Ansible Content Collections installed. The `firewalld` module was moved into a separate collection after Ansible 2.9.

- ▶ 2. Use automation content navigator with the default `ee-supported-rhel8` execution environment to identify the Ansible Content Collection that provides the `firewalld` module.
 - 2.1. Use the `podman login` command to log in to the classroom private automation hub at `hub.lab.example.com`. Log in as `student` with `redhat123` as the password. This step simulates a customer logging in to `registry.redhat.io` and ensures that the specified automation execution environment is pulled down to the local system if it does not already exist.

```
[student@workstation manage-reusing]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 2.2. List the module documentation available within the `ee-supported-rhel8` automation execution environment. Notice the colon at then end of the output which means that the output is extensive.

```
[student@workstation manage-reusing]$ ansible-navigator doc -l \
> --eei ee-supported-rhel8 --pp missing -m stdout
add_host                                     Add a host (and alternatively a group) to the ...
amazon.aws.aws_az_info                      Gather information about availability zones in ...
amazon.aws.aws_caller_info                  Get information about the user and account ...
amazon.aws.aws_s3                           manage objects in S3
amazon.aws.cloudformation                 Create or delete an AWS CloudFormation stack
...output omitted...
:
```

- 2.3. Type `/firewalld` to search the module documentation for `firewalld`. The `ansible.posix` content collection provides the `firewalld` module.

<code>add_host</code>	Add a host (and alternatively a group) to the ...
<code>amazon.aws.aws_az_info</code>	Gather information about availability zones in ...
<code>amazon.aws.aws_caller_info</code>	Get information about the user and account ...
<code>amazon.aws.aws_s3</code>	manage objects in S3
<code>amazon.aws.cloudformation</code>	Create or delete an AWS CloudFormation stack
<code>...output omitted...</code>	
<code>/firewalld</code>	

The fully qualified collection name for the `firewalld` module is `ansible.posix.firewalld`.

```
ansible.posix.firewalld      Manage arbitrary ports/services with firewalld
ansible.posix.firewalld_info  Gather information about firewalld
...output omitted...
```

Press q to exit the `ansible-navigator` command.

- 3. Create a copy of the `basic-web.yml` playbook to a file named `basic-web-fqcn.yml` and then update the new file to use fully qualified collection names for each module.

- 3.1. Create a copy of the `basic-web.yml` playbook named `basic-web-fqcn.yml`.

```
[student@workstation manage-reusing]$ cp basic-web.yml basic-web-fqcn.yml
```

- 3.2. Update the `basic-web-fqcn.yml` playbook to use the fully qualified collection names for each module. The `ansible.posix` collection provides the `firewalld` module. The `ansible.builtin` collection provides the other modules. The modified `basic-web-fqcn.yml` playbook should consist of the following content:

```
---
- name: Configure a basic web server
  hosts: servere.lab.example.com
  become: true
  tasks:
    - name: Install software
      ansible.builtin.yum:
        name:
          - httpd
          - firewalld
        state: latest

    - name: Start and enable services
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
        enabled: true
      loop:
        - httpd
        - firewalld

    - name: Open access to http
      ansible.posix.firewalld:
        service: http
        immediate: true
        permanent: true
        state: enabled

    - name: Configure simple index.html
      ansible.builtin.copy:
        content: "Hello world from {{ ansible_facts['fqdn'] }}.\n"
        dest: /var/www/html/index.html
```

**Note**

Although not currently required, Red Hat recommends that you use the fully qualified collection name.

- ▶ 4. Use the ee-supported-rhel8 automation execution environment to verify that the basic-web-fqcn.yml playbook completes successfully.

- 4.1. Run the basic-web-fqcn.yml playbook.

```
[student@workstation manage-reusing]$ ansible-navigator run \
> basic-web-fqcn.yml --eei ee-supported-rhel8 --pp missing

Play name          Ok  Changed ... Failed ... Task count  Progress
0|Configure a basic web server   5      4 ...      0 ...      5 Complete
^f/PgUp page up    ^b/PgDn page down    ↑ scroll    esc back ... Successful
```

When the Progress column shows **Successful**, press ESC to exit from the ansible-navigator command.

- 4.2. Use the curl command to display web content on servere.lab.example.com.

```
[student@workstation manage-reusing]$ curl servere.lab.example.com
Hello world from servere.lab.example.com.
```

- ▶ 5. Create a copy of the basic-web.yml playbook named basic-web-keyword.yml. Modify the basic-web-keyword.yml playbook so that it defines the included collections.

- 5.1. Create a copy of the basic-web.yml playbook.

```
[student@workstation manage-reusing]$ cp basic-web.yml basic-web-keyword.yml
```

- 5.2. Define the ansible.builtin and ansible.posix collections in the basic-web-keyword.yml playbook using the collections keyword. Configure the play to target serverf.lab.example.com instead of servere.lab.example.com. The top of the modified playbook should consist of the following content:

```
---
- name: Configure a basic web server
  hosts: serverf.lab.example.com
  become: true
  collections:
    - ansible.builtin
    - ansible.posix
  tasks:
    ...output omitted...
```

- ▶ 6. Use the ee-supported-rhel8 automation execution environment to verify that the basic-web-keyword.yml playbook completes successfully.

6.1. Run the `basic-web-keyword.yml` playbook.

```
[student@workstation manage-reusing]$ ansible-navigator run \
> basic-web-keyword.yml --eei ee-supported-rhel8 --pp missing

Play name          Ok  Changed ... Failed ... Task count  Progress
0|Configure a basic web server  5      4 ...      0 ...          5 Complete

^f/PgUp page up   ^b/PgDn page down    ↑↓ scroll    esc back ... Successful
```

When the Progress column shows **Successful**, press ESC to exit from the `ansible-navigator` command.

6.2. Use the `curl` command to display web content on `serverf.lab.example.com`.

```
[student@workstation manage-reusing]$ curl serverf.lab.example.com
Hello world from serverf.lab.example.com.
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Finding and Installing Ansible Content Collections

Objectives

- Search automation hub for Ansible Content Collections, and install them from the command line by name or by using a requirements.yml file.

Sources for Ansible Content Collections

Two sources provide Ansible Content Collections: automation hub, which is officially supported by Red Hat; and Ansible Galaxy, which is provided by the open source community.

Automation hub

Automation hub hosts Red Hat Certified Ansible Content Collections. These collections are officially supported by Red Hat and its partners for customers. Red Hat reviews, maintains, updates, and fully supports these collections. For example, the redhat.rhv, redhat.satellite, redhat.insights, and cisco.ios collections are available on this platform.

You need a valid Red Hat Ansible Automation Platform subscription to access automation hub. Use the automation hub web UI at <https://console.redhat.com/ansible/automation-hub/> to list and access the collections.

Ansible Galaxy

Ansible Galaxy is a public site that hosts unsupported collections submitted by various Ansible developers and users. 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.

In addition to these two distribution platforms, you can also deploy a private automation hub on-premise to store your custom collections or to provide a curated list of collections to your teams. You can also retrieve collections directly from their Git repositories.

Finding Collections on Automation Hub

Use the automation hub web UI at <https://console.redhat.com/ansible/automation-hub/> to browse its Ansible Content Collections.

Navigate to **Automation Hub > Collections** to list the collections. Use the **Search** field to access collections by keywords.

Provider	Collection Name	Status	Description	Modules	Roles	Plugins
Red Hat	satellite	Certified	Provided by Red Hat, Inc. Ansible Modules to manage Satellite installations	65	11	3
IBM	ibm_zos_cics	Certified	Provided by IBM The Red Hat Ansible Certified Content for IBM Z CICS collect...	5	0	0
IBM	ibm_zosmf	Certified	Provided by IBM Ansible collection consisting of modules and roles to work w...	2	4	0
NGINX	nginx_controller	Certified	Provided by NGINX Supporting NGINX Controller 3.x and later in your pipelines.	0	16	0
NetApp	azure	Certified	Provided by NetApp Azure NetApp Files (ANF)	4	0	0
SEIKO	smartcs	Certified	Provided by Seiko Solutions Inc. Ansible Network Collection for Seiko SmartCS devices	4	0	5

Figure 2.1: Listing collections in automation hub

You can also navigate to Automation Hub > Partners to browse the collections by partners.

Installing Ansible Content Collections

Your automation execution environment might not include the Ansible Content Collection that you want to use in your playbook. If your collection is not included in the automation execution environment, then you must make it available.

One method is to install the collection in the directory containing your Ansible Playbook before running the `ansible-navigator` command. Normally you create a `collections` subdirectory in that directory and put your collection in that subdirectory.

Installing Collections from the Command Line

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@host ~]$ ansible-galaxy collection install community.crypto
```

Chapter 2 | Managing Content Collections and Execution Environments

By default, the `ansible-galaxy` command tries to download the collection from Ansible Galaxy. Configuring `ansible-galaxy` to use automation hub or a private automation hub to get collections is covered later in this section.

The `ansible-galaxy` command can also install a collection from a local or a remote `.tar` file or a Git repository.

```
[user@host ~]$ ansible-galaxy collection install /tmp/community-dns-1.2.0.tar.gz
[user@host ~]$ ansible-galaxy collection install \
> http://www.example.com/redhat-insights-1.0.5.tar.gz
[user@host ~]$ ansible-galaxy collection install \
> git@github.com:ansible-collections/community.mysql.git
```

When Ansible runs a playbook, it looks for collections in a `collections/` directory in the same directory as the playbook. If it cannot find the required collection in that location, then it uses the `collections_paths` directive from the `ansible.cfg` configuration file. That directive specifies a colon-separated list of paths on the system where Ansible looks for installed collections.



Important

If you modify the `collections_paths` directive in an Ansible project's `ansible.cfg` file, it is important that you keep `/usr/share/ansible/collections` as one of the directories containing collections.

The Ansible configuration file for the project is used by the automation execution environment, and the collections included in the execution environment are in the `/usr/share/ansible/collections` directory inside the container.

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.

To install a collection in a different directory, use the `--collections-path` (or `-p`) option:

```
[user@host ~]$ ansible-galaxy collection install community.postgresql \
> -p ~/myproject/collections/
```

 **Important**

The `ansible-galaxy` command was originally written for `ansible-playbook` users, for whom the control node and the automation execution environment are the same system. The `ansible-galaxy` command installed files in the `/usr/share/ansible/collections` directory on the control node so that the `ansible-playbook` command could use them.

Red Hat Ansible Automation Platform 2 separates automation execution environments from the control node so that they run in containers. These container-based automation execution environments have limited access to the control node's file systems. When you run the playbook in the container, the `collections_paths` directive uses the `/usr/share/ansible/collections` directory inside the container, not on the control node. Therefore, if you install your collections in the `~/.ansible/collections` or `/usr/share/ansible/collections` directories on the control node, the automation execution environment cannot use them.

However, automation execution environments do have access to the directory that hosts the playbook that they are running. If you install your collections in a `./collections` directory in the same directory as your playbook, then the automation execution environment can access them.

Installing Collections with a Requirements File

The `collections/requirements.yml` file is a list of the collections needed to run playbooks in your Ansible project, just like you can create a `roles/requirements.yml` file to list the roles 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, and they can use `ansible-galaxy` to manually install them. Automation controller detects the `collections/requirements.yml` and automatically installs the collections it specifies before running your playbooks.

The following `collections/requirements.yml` file lists several collections to install. You can target a specific version of an Ansible Content Collection or provide local or remote `.tar` files or Git repositories as the source of the collection.

```
---
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

  - name: git@github.com:ansible-collections/community.mysql.git
```

You can process that file with the `ansible-galaxy` command to install those collections. Use the `--requirements-file` (or `-r`) option to provide the `requirements.yml` file to the command.

```
[user@host ~]$ ansible-galaxy collection install -r requirements.yml
```

Listing Installed Collections

The `ansible-navigator collections` command starts an interactive text-based user interface (TUI) that lists the collections installed in the current automation execution environment.

Name	Version	Shadowed	Type	Path
0 amazon.aws	3.2.0	False	contained	/usr/share/ansible/collection/0 amazon.aws
1 ansible.builtin	2.13.0	False	contained	/usr/lib/python3.9/site-packages/ansible/builtin
2 ansible.controller	4.2.0	False	contained	/usr/share/ansible/collection/2 ansible.controller
3 ansible.netcommon	3.0.0	False	contained	/usr/share/ansible/collection/3 ansible.netcommon
4 ansible.network	1.2.0	False	contained	/usr/share/ansible/collection/4 ansible.network
5 ansible.posix	1.3.0	False	contained	/usr/share/ansible/collection/5 ansible.posix
6 ansible.security	1.0.0	False	contained	/usr/share/ansible/collection/6 ansible.security
7 ansible.utils	2.6.1	False	contained	/usr/share/ansible/collection/7 ansible.utils
8 ansible.windows	1.9.0	False	contained	/usr/share/ansible/collection/8 ansible.windows
9 ansible.yang	1.0.0	False	contained	/usr/share/ansible/collection/9 ansible.yang
10 arista.eos	5.0.0	False	contained	/usr/share/ansible/collection/10 arista.eos
11 cisco.asa	3.0.0	False	contained	/usr/share/ansible/collection/11 cisco.asa
12 cisco.ios	3.0.0	False	contained	/usr/share/ansible/collection/12 cisco.ios
13 cisco.iosxr	3.0.0	False	contained	/usr/share/ansible/collection/13 cisco.iosxr
14 cisco.nxos	3.0.0	False	contained	/usr/share/ansible/collection/14 cisco.nxos
15 cloud.common	2.1.1	False	contained	/usr/share/ansible/collection/15 cloud.common

If you enter the line number of a particular collection, `ansible-navigator` displays detailed information about the content of that collection in the TUI.

For example, if `ansible-navigator` displays the preceding output, and you type 5, then the following output is displayed:

Ansible.posix	Type	Added	Deprecated	Description
0 acl	module	1.0.0	False	Set and retrieve file AC
1 at	module	1.0.0	False	Schedule the execution o
2 authorized_key	module	1.0.0	False	Adds or removes an SSH a
3 cgroup_perf_recap	callback	None	False	Profiles system activity
4 csh	shell	None	False	C shell (/bin/csh)
5 debug	callback	None	False	formatted stdout/stderr
6 firewalld	module	None	False	Manage arbitrary ports/s
7 firewalld_info	module	None	False	Gather information about
8 fish	shell	None	False	fish shell (/bin/fish)
9 json	callback	None	False	Ansible screen output as
10 mount	module	1.0.0	False	Control active and confi
11 patch	module	1.0.0	False	Apply patch files using
12 profile_roles	callback	None	False	adds timing information
13 profile_tasks	callback	None	False	adds time information to
14 seboolean	module	1.0.0	False	Toggles SELinux booleans
15 selinux	module	1.0.0	False	Change policy and state

From that page, you could type 1 to get documentation and usage examples for the `ansible.posix.at` module:

```
Image: ansible.posix.at
Description: Schedule the execution of a command or script file via
0|---
1|additional_information: {}
2|collection_info:
3|  authors:
4|    - Ansible (github.com/ansible)
5|  dependencies: {}
6|  description: Ansible Collection targeting POSIX and POSIX-ish plat
7|  documentation: https://github.com/ansible-collections/ansible.posi
8|  homepage: https://github.com/ansible-collections/ansible.posix
9|  issues: https://github.com/ansible-collections/ansible.posix
10| license: []
11| license_file: COPYING
12| name: ansible.posix
13| namespace: ansible
14| path: /usr/share/ansible/collections/ansible_collections/ansible/po
15| readme: README.md
^f/PgUp page ^b/PgDn page do↑↓ scroesc ba- previo+ ne[0-9] go:help he
```

You can also use the `ansible-galaxy collection list` command to list the collections available in the directories specified by your current `collections_paths` Ansible configuration directive on your control node. It does not list collections that you might have installed in directories not specified by that directive.

```
[user@host ~]$ ansible-galaxy collection list

# /home/user/.ansible/collections/ansible_collections
Collection      Version
-----
ansible.posix   1.3.0
community.crypto 1.9.5
community.dns    1.2.0
community.mysql   2.1.0
redhat.insights  1.0.7

# /usr/share/ansible/collections/ansible_collections
Collection      Version
-----
redhat.rhel_system_roles 1.0.1
```



Important

The `ansible-galaxy collection list` command lists the Ansible Content Collections installed in the directory specified by the `collections_paths` directive on the *control node*. This might include collections installed in your Ansible project directory, if that location is listed in your `collections_paths` directive. You must use the `ansible-navigator collections` command to list the Ansible Content Collections in your current automation execution environment. This does not include any collections that your Ansible project brings into the automation execution environment when a playbook in the project is run.

Neither command lists the `ansible.builtin` collection, which is always available.

Configuring Collection Sources

By default, the `ansible-galaxy` command uses Ansible Galaxy at `https://galaxy.ansible.com/` to download collections. You can configure additional distribution platforms in the `ansible.cfg` configuration file.

Installing Collections from Automation Hub

For the `ansible-galaxy` command to also use 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://console.redhat.com/api/automation-hub/ ②
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token ③
token=eyJh...Jf0o ④

[galaxy_server.galaxy]
url=https://galaxy.ansible.com/
```

- ① Lists all the repositories that the `ansible-galaxy` command must use, in order. For each name that you define, add a `[galaxy_server.name]` section to provide the connection parameters. Because 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 automation hub, then the `ansible-galaxy` command uses Ansible Galaxy to retrieve it.
- ② Provides the URL of the distribution platform.
- ③ Provides the URL for authentication.
- ④ Specifies the authentication token associated with your account. Required to access automation hub.

To retrieve the preceding URLs and generate a token, log in to the automation hub web UI and navigate to **Automation Hub > Connect to Hub**.

The screenshot shows the Red Hat Hybrid Cloud Console interface. At the top, there's a navigation bar with a menu icon, the Red Hat logo, and the text "Red Hat Hybrid Cloud Console". A dropdown menu labeled "All apps and services" is open. Below the navigation, there's a section titled "Offline token" with a "Load token" button. Another section titled "Manage tokens" has a note about revoking tokens via the "offline API token management" page. A "Server URL" section contains a text input field with the URL "https://console.redhat.com/api/automation-hub/" and a copy icon. A note below it says "Note: this URL contains all collections in Hub. To connect to your organization's sync repository use the URL found on Repository Management." A "SSO URL" section contains a text input field with the URL "https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token" and a copy icon.

Figure 2.2: Generating a token

Instead of a token, you can use the `username` and `password` parameters to provide your customer portal username and password.

```
...output omitted...
[galaxy_server.automation_hub]
url=https://console.redhat.com/api/automation-hub/
username=operator1
password=Sup3r53cR3t
...output omitted...
```

You might not want to expose your credentials or your token 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:

```
ANSIBLE_GALAXY_SERVER_SERVERID_KEY=value
```

SERVERID

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.serverid]` section.

KEY

Key name in uppercase. For example, use `TOKEN` for the token key. The following example provides the `token` key as an environment variable:

```
[user@host ~]$ cat ansible.cfg
...output omitted...
[galaxy_server.automation_hub]
url=https://console.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token
[user@host ~]$ export \
> ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN='eyJh...Jf0o'
[user@host ~]$ ansible-galaxy collection install redhat.insights
```

**Note**

If you develop and test playbooks on your local system, then you can define these `ANSIBLE_GALAXY_SERVER_*` variables in your `~/.bashrc` file. If you do so, then protect the file because the variables contain your authentication credentials or your authentication token.

Installing Collections from Private Automation Hub

Configuring `ansible-galaxy` to get content from a private automation hub is similar to how it is configured to use automation hub. The following example includes a new section that configures `ansible-galaxy` to use a private automation hub installed on `hub.example.com`.

```
...output omitted...
[galaxy]
server_list = my_hub, automation_hub, galaxy

[galaxy_server.my_hub]
url=https://hub.example.com/api/galaxy/content/rh-certified/
token=e8e4...b0c2

[galaxy_server.automation_hub]
url=https://console.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token

[galaxy_server.galaxy]
url=https://galaxy.ansible.com/
```

Notice that you do not need the `auth_url` directive for private automation hub.

Use the private automation hub web UI to retrieve the URL and generate the token.

Navigate to **Collections > Repository Management** to retrieve the URL. Private automation hub can host collections that you have synchronized from automation hub, from Ansible Galaxy, or that you have created for your teams. Select the URL corresponding to the type of collection you want to install.

	Distributio...	Repositor...	Cont...	Last u...	Repo URL	CLI configuration
1	community	community	120	9 days ago	https://hub.la...	[galaxy]server_list = ...
2	published	published	0	11 days ago	https://hub.la...	[galaxy]server_list = ...
	rejected	rejected	0	11 days ago	https://hub.la...	[galaxy]server_list = ...
3	rh-certified	rh-certified	74	9 days ago	https://hub.la...	[galaxy]server_list = ...

- ① The **community** repository uses the collections that you have synchronized from Ansible Galaxy.
- ② The **published** repository uses custom collections that you have installed and approved.
- ③ The **rh-certified** repository uses the collections that you have synchronized from automation hub.

To generate a token, either click **Get token** on the **Repo Management** page or navigate to **Collections > API token management** and then click **Load token**.

Installing and deploying private automation hub is beyond the scope of this course.



References

Using Collections – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/collections_using.html

► Guided Exercise

Finding and Installing Ansible Content Collections

Configure ansible-galaxy and use it to install Ansible Content Collections.

Outcomes

- Configure various collection sources on a system to install collections.
- Install collections with the ansible-galaxy collection command.
- Install multiple collections using a requirements.yml file.

Before You Begin

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command creates an Ansible project in the /home/student/manage-finding/ directory on workstation.

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

Instructions

- 1. Browse the Ansible Content Collections available from the classroom's private automation hub. Access the details for one of the collections that Red Hat provides and supports, ansible.netcommon.
 - 1.1. Use a browser to navigate to the private automation hub at <https://hub.lab.example.com>, and then log in as student using redhat123 as the password.
 - 1.2. Navigate to Collections > Collections to browse the Ansible Content Collections. Because the ansible.netcommon collection is a Red Hat Certified Ansible Content Collection, select Red Hat Certified from the Filter by repository menu.
 - 1.3. Search for the collection by entering netcommon in the search field. When the search completes, click the netcommon tile.
 - 1.4. Notice that the UI displays an ansible-galaxy collection install ansible.netcommon command that you can use to install the collection. Click the Documentation tab to access the list of modules and plug-ins available with this collection.
- 2. On the workstation machine, configure the Ansible project in the /home/student/manage-finding/ directory so that the ansible-galaxy command retrieves collections from the classroom's private automation hub. Enable access to the rh-certified and community repositories.

- 2.1. Change to the /home/student/manage-finding/ directory.

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

**Important**

You need to run the ansible-galaxy commands in this exercise from the /home/student/manage-finding/ directory, which contains the Ansible project. This helps ensure that you use the correct configuration file and install Ansible Content Collections to the correct location.

- 2.2. From the private automation hub web UI, navigate to **Collections > Repository Management**. This page has the parameters that you need for configuring the ansible.cfg file.

Click the **Copy to clipboard** icon in the **CLI configuration** column and the **rh-certified** row.

Do not close your web browser window.

**Note**

You might need to collapse the web UI navigation bar or zoom out in your web browser to see the **Copy to clipboard** icon for the **CLI configuration** column.

- 2.3. Modify the ansible.cfg file to append the lines that you copied to the clipboard in the preceding step. The modified ansible.cfg file now contains the following content:

```
[defaults]
inventory = ./inventory

[galaxy]
server_list = rh-certified_repo

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=<put your token here>
```

- 2.4. Return to the **Repo Management** page in the private automation hub web UI. Click the **Copy to clipboard** icon in the **CLI configuration** column and the **community** row.

Do not close your web browser window.

- 2.5. Modify the ansible.cfg file to append the lines that you copied to the clipboard in the preceding step. The modified ansible.cfg file now contains the following content:

```
[defaults]
inventory = ./inventory

[galaxy]
server_list = rh-certified_repo
```

```
[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=<put your token here>

[galaxy]
server_list = community_repo

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=<put your token here>
```

You added a second [galaxy] section to the file, and there needs to be only one [galaxy] section. You correct that problem in the next step.

- 2.6. Update the ansible.cfg file so that it only contains one [galaxy] section. Update the first [galaxy] section so that it lists both repositories, rh-certified_repo and community_repo. Remove the second [galaxy] section. The modified ansible.cfg file now contains the following content.

```
[defaults]
inventory = ./inventory

[galaxy]
server_list = rh-certified_repo, community_repo

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=<put your token here>

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=<put your token here>
```

- 2.7. Return to the private automation hub web UI. Navigate to **Collections > API token management** and then click **Load token**. Copy the API token.
- 2.8. Using the copied token, update both token lines in the ansible.cfg file. Your token is probably different from the one displayed in this example. Save and close the file when done.

```
[defaults]
inventory = ./inventory

[galaxy]
server_list = rh-certified_repo, community_repo

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=c4f21af7090f0f9cb74d3c3f9e1748884ecdc180

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=c4f21af7090f0f9cb74d3c3f9e1748884ecdc180
```

- 3. Create a collections directory within the project directory, and then install the ansible.netcommon collection into the new directory.

3.1. Create the collections directory.

```
[student@workstation manage-finding]$ mkdir collections
```

- 3.2. Use the ansible-galaxy command to install the ansible.netcommon collection into the collections directory.

```
[student@workstation manage-finding]$ ansible-galaxy collection install \
> ansible.netcommon -p collections/
Starting galaxy collection install process
[WARNING]: The specified collections path '/home/student/manage-finding/
collections' is not part of the configured Ansible
collections paths '/home/student/.ansible/collections:/usr/share/ansible/
collections'. The installed collection won't be picked up
in an Ansible run.
Process install dependency map
Starting collection install process
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
rh-certified/collections/artifacts/ansible-netcommon-4.1.0.tar.gz to /home/
student/.ansible/tmp/ansible-local-29962sr162r7w/tmprrxsyau18/ansible-
netcommon-4.1.0-eduihb5k
[WARNING]: The GnuPG keyring used for collection signature verification was not
configured but signatures were provided by the
Galaxy server to verify authenticity. Configure a keyring for ansible-galaxy to
use or disable signature verification. Skipping
signature verification.
Installing 'ansible.netcommon:4.1.0' to '/home/student/manage-finding/collections/
ansible_collections/ansible/netcommon'
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
rh-certified/collections/artifacts/ansible-utils-2.8.0.tar.gz to /home/
student/.ansible/tmp/ansible-local-29962sr162r7w/tmprrxsyau18/ansible-utils-2.8.0-
hn1_8a18
ansible.netcommon:4.1.0 was installed successfully
Installing 'ansible.utils:2.8.0' to '/home/student/manage-finding/collections/
ansible_collections/ansible/utils'
ansible.utils:2.8.0 was installed successfully
```

**Note**

You might see two **WARNING** messages in the output that you can safely ignore. The first one is solved when you configure the `collections_paths` directive in the next step. The second warning occurs because the Ansible Galaxy server is providing GnuPG digital signatures that can be used to verify the downloaded content, but your client is not configured to validate those signatures.

The preceding output indicates that the `ansible-galaxy` command downloaded the collection from `https://hub.lab.example.com`, the classroom private automation hub.

Notice that the `ansible.utils` collection is installed automatically as a dependency of the `ansible.netcommon` collection.

- 3.3. Modify the `ansible.cfg` file so that it explicitly looks for collections in the project `collections` directory. The top of the modified file now contains the following content:

```
[defaults]
inventory = ./inventory
collections_paths = ./collections:/usr/share/ansible/collections
...output omitted...
```

- 3.4. In the `/home/student/manage-finding` directory, use the `ansible-galaxy` command to list the collections installed on the control node that are specified by the current `collections_paths` directive.

```
[student@workstation manage-finding]$ ansible-galaxy collection list

# /home/student/manage-finding/collections/ansible_collections
Collection      Version
-----
ansible.netcommon 4.1.0
ansible.utils     2.8.0

# /usr/share/ansible/collections/ansible_collections
Collection      Version
-----
redhat.rhel_system_roles 1.16.2
```



Note

Although other Ansible Content Collections might be installed on the system, the `ansible-galaxy` command only lists collections in the directories specified by the `collections_paths` directive. The versions of the `ansible.netcommon`, `ansible.utils`, and `redhat.rhel_system_roles` collections installed on your system might be different than the versions displayed here.

4. Install version 2.1.0 of the `redhat.rhv` collection. Make sure that you run `ansible-galaxy` from a prompt in the Ansible project's `/home/student/manage-finding` directory. Because you updated the project's `ansible.cfg` file to look first for collections in the `./collections` directory, you do not need to use the `-p` option.

```
[student@workstation manage-finding]$ ansible-galaxy collection install \
> redhat.rhv:2.1.0
...output omitted...
Installing 'redhat.rhv:2.1.0' to '/home/student/manage-finding/collections/
ansible_collections/redhat/rhv'
redhat.rhv:2.1.0 was installed successfully
```

5. Install collections specified in a requirements file. Include the existing `ansible.netcommon` and `redhat.rhv` collections, and then add two additional collections, `community.crypto` and `ansible.controller`.

- 5.1. Create a file named `requirements.yml` in the `collections` directory, and add the following content:

```
---
collections:
  - name: ansible.netcommon

  - name: redhat.rhv
    version: 2.1.0

  - name: community.crypto
    version: 2.7.0

  - name: ansible.controller
    version: 4.2.1
```

- 5.2. Save and close the file when done.

- 5.3. In the `/home/student/manage-finding` directory, install the collections by using the `ansible-galaxy` command. Add the `-r` option to specify the location of the requirements file.

When you run the command, it skips installing the `ansible.netcommon`, `redhat.rhv`, and `ansible.utils` collections because they are already installed.

```
[student@workstation manage-finding]$ ansible-galaxy collection install \
> -r collections/requirements.yml
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
community/collections/artifacts/community-crypto-2.7.0.tar.gz to /home/
student/.ansible/tmp/ansible-local-32771_upchxep/tmp36d_hwh7/community-
crypto-2.7.0-fx5pgw7w
'ansible.netcommon:4.1.0' is already installed, skipping.
'redhat.rhv:2.1.0' is already installed, skipping.
'ansible.utils:2.8.0' is already installed, skipping.
Installing 'community.crypto:2.7.0' to '/home/student/manage-finding/collections/
ansible_collections/community/crypto'
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
rh-certified/collections/artifacts/ansible-controller-4.2.1.tar.gz to /
home/student/.ansible/tmp/ansible-local-32771_upchxep/tmp36d_hwh7/ansible-
controller-4.2.1-89anvds8
community.crypto:2.7.0 was installed successfully
[WARNING]: The GnuPG keyring used for collection signature verification was not
configured but signatures were provided by the
Galaxy server to verify authenticity. Configure a keyring for ansible-galaxy to
use or disable signature verification. Skipping
signature verification.
Installing 'ansible.controller:4.2.1' to '/home/student/manage-finding/
collections/ansible_collections/ansible/controller'
ansible.controller:4.2.1 was installed successfully
```

- 5.4. In the Ansible project's /home/student/manage-finding directory, list the collections installed on the control node.

```
[student@workstation manage-finding]$ ansible-galaxy collection list

# /home/student/manage-finding/collections/ansible_collections
Collection          Version
-----
ansible.controller 4.2.1
ansible.netcommon  4.1.0
ansible.utils       2.8.0
community.crypto    2.7.0
redhat.rhv          2.1.0

# /usr/share/ansible/collections/ansible_collections
Collection          Version
-----
redhat.rhel_system_roles 1.16.2
```

Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Selecting an Execution Environment

Objectives

- Identify the automation execution environments provided by Red Hat and select the correct one for your use case.

Automation Execution Environments

An *automation execution environment* is a container image that includes Ansible Content Collections, their software dependencies, and a minimal Ansible engine that can run your playbooks.

By using an automation execution environment, you can use the same portable environment to develop your Ansible Playbooks on one system and run them on another. This streamlines and simplifies the development process and helps to ensure predictable, reproducible results.

The automation execution environment is where your Ansible Playbook runs. You normally use a tool such as `ansible-navigator` to run a playbook, but the playbook runs inside the container rather than directly on your system.

An automation execution environment consists of the following components:

- Ansible Core (or Ansible)
- Ansible Content Collections to supplement Ansible Core
- Python and any other dependencies of Ansible Core and the included collections
- Ansible Runner to run your playbooks

Automation execution environments are prepared by the execution environment builder (`ansible-builder`), and can be pushed to any container registry.



Note

A subsequent chapter covers how to create custom automation execution environments.

Selecting a Supported Automation Execution Environment

Red Hat Ansible Automation Platform 2.2 provides three prebuilt automation execution environments:

Short name	Container image	Description
Minimal	<code>ee-minimal-rhel8</code>	A minimal automation execution environment based on Ansible Core 2.13.

Short name	Container image	Description
Supported	ee-supported-rhel8	An automation execution environment based on Ansible Core 2.13 that includes Red Hat Ansible Certified Content Collections and their software dependencies.
Compatibility	ee-29-rhel8	An automation execution environment based on Ansible 2.9.

The minimal automation execution environment only includes the `ansible.builtin` Ansible Content Collection. You usually reserve it as a starting point to build custom automation execution environments.

The supported automation execution environment includes some collections that Red Hat supports. Automation content navigator and automation controller use the supported automation execution environment by default. Use this automation execution environment with your new playbooks.

The compatibility automation execution environment is based on Ansible 2.9, which was released before modules were unbundled from the core Ansible package into collections. This is useful when you want to run playbooks written for Ansible 2.9 (or Red Hat Ansible Automation Platform 1.2) that you have not yet migrated to Red Hat Ansible Automation Platform 2.

Red Hat hosts those three automation execution environments in its container registry at `registry.redhat.io`, under the `ansible-automation-platform-22` namespace:

- `registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest`
- `registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest`
- `registry.redhat.io/ansible-automation-platform-22/ee-29-rhel8:latest`

You can browse the registry catalog at <https://catalog.redhat.com/software/containers/search?q=ansible-automation-platform-22>.



Note

Container images, such as automation execution environments, can have multiple tags that point to the same container image. In the classroom environment, each of the three automation execution environments can be pulled from `hub.lab.example.com` using the `latest` tag.

Inspecting Automation Execution Environments

Use automation content navigator to list and inspect the automation execution environments available on the system. To do so, run the `ansible-navigator images` command:

```
[user@host ~]$ ansible-navigator images
  Image           Tag     Execution environment  Created      Size
0|ee-29-rhel8    latest   True                  4 months ago  815 MB
1|ee-minimal-rhel8  latest   True                  4 months ago  265 MB
2|ee-supported-rhel8  latest   True                  4 months ago  1.05 GB

^f/PgUp page up ^b/PgDn page down ↑ scroll esc back [0-9] goto :help help
```

You can type the number of the automation execution environment that you want to inspect. For example, press 2 to review the details of the supported automation execution environment:

Image: ee-supported-rhel8:latest	Description
0 Image information	Information collected from image inspectio
1 General information	OS and python version information
2 Ansible version and collections	Information about ansible and ansible coll
3 Python packages	Information about python and python packag
4 Operating system packages	Information about operating system package
5 Everything	All image information

```
^f/PgUp page up ^b/PgDn page down ↑ scroll esc back [0-9] goto :help help
```

You can view further details on a specific aspect of the automation execution environment by selecting the associated number. In the preceding example, if you press 2 or enter :2, then you get the list of the available collections in that automation execution environment.

Using Automation Execution Environments with Automation Content Navigator

The `ansible-navigator run` command runs your playbooks in an automation execution environment.

This command uses the supported automation execution environment by default, but you can use the `--execution-environment-image` (or `-eei`) option to specify a different environment. The following example shows how to run a playbook using the compatibility automation execution environment:

```
[user@host ~]$ ansible-navigator run oldplaybook.yml --eei \
> registry.redhat.io/ansible-automation-platform-22/ee-29-rhel8:latest
```

If the container image is already available on your system, then you can use its short image name: `--eei ee-29-rhel8:latest` in the preceding example.

If the container image is not already available on your system, `ansible-navigator` tries to pull it from the container registry. Use the `podman login` command to ensure that you are authenticated to the registry. In the preceding example, at the beginning of your session, run the `podman login registry.redhat.io` command and provide your Customer Portal credentials to authenticate to the registry.

You can control how `ansible-navigator` pulls container images by using the `--pull-policy` (or `--pp`) option with one of the following arguments:

Option	Description
always	Always pulls the image.
missing	Only pulls the image if it is not already available locally.
never	Never pulls the image.
tag	Pulls the image if the image tag is <code>latest</code> or if it is not already available locally (the default).

**Note**

Instead of using the `--eei` option, you can create an `ansible-navigator.yml` configuration file to define the automation execution environment to use by default. A later chapter describes that configuration file in more detail.

**References****What Are Automation Execution Environments?**

<https://www.redhat.com/en/technologies/management/ansible/automation-execution-environments>

What's New in Ansible Automation Platform 2: Automation Execution Environments

<https://www.ansible.com/blog/whats-new-in-ansible-automation-platform-2-automation-execution-environments>

► Guided Exercise

Selecting an Execution Environment

Ensure that the execution environments you need are installed on your development workstation and use them to run playbooks.

Outcomes

- Identify the automation execution environment that includes a specific Ansible Content Collection.
- Run a playbook using that automation execution environment.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/manage-selecting/` directory.

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

Instructions

- 1. Identify the Ansible Content Collection that is required to run the prebuilt playbook.
- 1.1. From a terminal, change to the `/home/student/manage-selecting/` directory and review the `acl_info.yml` playbook.

```
[student@workstation ~]$ cd ~/manage-selecting
[student@workstation manage-selecting]$ cat acl_info.yml
---
- name: Retrieving ACLs using the ansible.posix collection
  hosts: web_servers
  gather_facts: false

  tasks:
    - name: Obtain the ACLs for /webserver_data directory
      ansible.posix.acl:
        path: /webserver_data/
        register: acl_info

    - name: Print the ACLs obtained for /webserver_data directory
      ansible.builtin.debug:
        var: acl_info['acl']
```

- 1.2. Identify the modules that the `acl_info.yml` playbook uses:

- `ansible.posix.acl`

- `ansible.builtin.debug`

To successfully run the playbook, you need the `ansible.posix.acl` module, which is part of the `ansible.posix` Ansible Content Collection.



Note

The `ansible.builtin.debug` module is part of the `ansible.builtin` collection that is included in the `ansible-core` RPM package.

- 2. Find the automation execution environment that includes the `ansible.posix` Ansible Content Collection.

- 2.1. Use the `podman login` command to log in to the classroom private automation hub at `hub.lab.example.com`.

This step simulates logging in to `registry.redhat.io`, but uses the private automation hub in the classroom environment. That private automation hub includes the three automation execution environments provided by Red Hat Ansible Automation Platform 2.

```
[student@workstation manage-selecting]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 2.2. Use the `ansible-navigator images` command to pull the minimal automation execution environment and analyze it.

```
[student@workstation manage-selecting]$ ansible-navigator images \
> --eei hub.lab.example.com/ee-minimal-rhel8
-----
Execution environment image and pull policy overview
-----
Execution environment image name: hub.lab.example.com/ee-minimal-rhel8:latest
Execution environment image tag: latest
Execution environment pull arguments: None
Execution environment pull policy: tag
Execution environment pull needed: True
...output omitted...
```

After a few seconds, automation content navigator displays output that lists the `ee-minimal-rhel8` automation execution environment.

Image	Tag	Execution environment	...
0 ee-minimal-rhel8	latest	True	...

- 2.3. Type 0 to inspect this automation execution environment, and then type 2 to select the Ansible version and collections option.

Chapter 2 | Managing Content Collections and Execution Environments

	Image: ee-minimal-rhel8:latest (primary)	Description
0	Image information	Information collected from image...
1	General information	OS and python version information
2	Ansible version and collections	Information about ansible and an...
3	Python packages	Information about python and pyt...
4	Operating system packages	Information about operating syst...
5	Everything	All image information

- 2.4. The resulting output indicates that the minimal automation execution environment does not contain any additional Ansible Content Collections.

```
Image: ee-minimal-rhel8:latest (primary) (Information about ansible and ...)
0|---
1|ansible:
2| collections:
3|   details: {}
4|   errors:
5|   - |-
6|     ERROR! - None of the provided paths were usable. Please specify a valid ...
7|   version:
8|   details: ansible [core 2.13.4]
```

Type :q and then press Enter to exit the ansible-navigator command. The ee-minimal-rhel8 automation execution environment is not useful for running this playbook.

- 2.5. Use the ansible-navigator command to pull down and inspect the supported automation execution environment.

```
[student@workstation manage-selecting]$ ansible-navigator images \
> --eei hub.lab.example.com/ee-supported-rhel8
-----
Execution environment image and pull policy overview
-----
Execution environment image name:    hub.lab.example.com/ee-supported-rhel8:...
Execution environment image tag:      latest
Execution environment pull arguments: None
Execution environment pull policy:    tag
Execution environment pull needed:   True
...output omitted...
```

- 2.6. Automation content navigator shows both automation execution environments. Type 1 to inspect the supported one and then type the number for the Ansible version and collections entry.

NAME	Tag	Execution environment	...
0 ee-minimal-rhel8	latest	True	...
1 ee-supported-rhel8	latest	True	...

- 2.7. The resulting output indicates that the supported automation execution environment contains the ansible.posix Ansible Content Collection.

```
Image: ee-supported-rhel8:latest (primary) (Information about ansible and ...
0|---
1|ansible:
2| collections:
3|   details:
4|     amazon.aws: 3.2.0
5|     ansible.controller: 4.2.1
6|     ansible.netcommon: 3.1.1
7|     ansible.network: 1.2.0
8|     ansible.posix: 1.3.0
9|     ansible.security: 1.0.0
...output omitted...
```

- 2.8. Type :q and then press Enter to exit the `ansible-navigator` command. The `ee-supported-rhel8` automation execution environment contains the `ansible.posix` Ansible Content Collection that you need. You can run this playbook using this automation execution environment.
- ▶ 3. Use automation content navigator to run the `acl_info.yml` playbook.
- 3.1. Demonstrate that the *minimal* automation execution environment does not work with the `acl_info.yml` playbook because it does not include the `ansible.posix` collection. Use the `ansible-navigator` command to run the `acl_info.yml` playbook using the *minimal* automation execution environment.
As expected, the playbook cannot resolve the `ansible.posix.acl` module, causing it to fail.

```
[student@workstation manage-selecting]$ ansible-navigator run acl_info.yml \
> -m stdout --eei ee-minimal-rhel8 --pp missing
ERROR! couldn't resolve module/action 'ansible.posix.acl'. This often indicates a misspelling, missing collection, or incorrect module path.
```

The error appears to be in '/home/student/manage-selecting/acl_info.yml': line 7, column 7, but may be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

```
tasks:
  - name: Obtain the ACLs for /webserver_data directory
    ^ here
```

- 3.2. Use the `ansible-navigator` command to run the `acl_info.yml` playbook using the `ee-supported-rhel8:latest` automation execution environment. This automation execution environment contains the `ansible.posix` collection and the command succeeds.

```
[student@workstation manage-selecting]$ ansible-navigator run acl_info.yml \
> -m stdout --eei ee-supported-rhel8 --pp missing
PLAY [Retrieving ACLs using the ansible.posix collection] ****
TASK [Obtain the ACLs for /webserver_data directory] ****
```

```
ok: [serverc.lab.example.com]
ok: [serverb.lab.example.com]

TASK [Print the ACLs obtained for /webserver_data directory] ****
ok: [serverc.lab.example.com] => {
    "acl_info['acl']": [
        "user::rwx",
        "group::r-x",
        "other::r-x",
        "default:user::rwx",
        "default:group::r-x",
        "default:mask::rwx",
        "default:other::r-x"
    ]
}
ok: [serverb.lab.example.com] => {
    "acl_info['acl']": [
        "user::rwx",
        "group::r-x",
        "other::r-x",
        "default:user::rwx",
        "default:group::r-x",
        "default:mask::rwx",
        "default:other::r-x"
    ]
}

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

Finish

On the workstation machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

▶ Lab

Managing Content Collections and Execution Environments

Run a playbook that needs an Ansible Content Collection in a particular execution environment, and another playbook that needs an Ansible Content Collection that is not provided by an execution environment.

Outcomes

- Identify Ansible Content Collections inside automation execution environment images.
- Run a playbook that requires a collection from an automation execution environment.
- Install a collection from automation hub.
- Run a playbook that requires a collection from automation hub.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/manage-review/` directory on the workstation machine and obtains the files necessary to complete this lab.

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

Instructions

1. Change into the `/home/student/manage-review/` directory and review the `cockpit.yml` playbook. Make note of the Ansible Content Collections that the playbook requires.
2. Use the `ansible-navigator` command to inspect the automation execution environment images available on your local system. Determine which one you need to run the `cockpit.yml` playbook.
3. Run the `cockpit.yml` playbook by using the `ansible-navigator` command with the correct automation execution environment image.
4. Log in to the web console at <https://servera.lab.example.com:9090> to confirm that the playbook correctly configured the web console. Accept the insecure TLS certificate and then log in as the student user. Use `student` as the password.
5. Review the `create_samples_db.yml` playbook. Make note of the collections that it requires. Find those collections in the private automation hub.
You can access the private automation hub web UI at <https://hub.lab.example.com>. Use `student` as the username and `redhat123` as the password.
6. Configure the `/home/student/manage-review/ansible.cfg` file so that the `ansible-galaxy` command installs collections from private automation hub.

7. Create the `/home/student/manage-review/collections/` directory. In this directory, install any collections that the `create_samples_db.yml` playbook requires.
8. Use the `ansible-navigator` command to run the `create_samples_db.yml` playbook.
9. Confirm that the playbook correctly configured a MariaDB database on the `serverb` machine. You can confirm that the playbook created the `samples` database by running the `sudo mysql -e "SHOW DATABASES"` command. The password for the student user is `student`.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Managing Content Collections and Execution Environments

Run a playbook that needs an Ansible Content Collection in a particular execution environment, and another playbook that needs an Ansible Content Collection that is not provided by an execution environment.

Outcomes

- Identify Ansible Content Collections inside automation execution environment images.
- Run a playbook that requires a collection from an automation execution environment.
- Install a collection from automation hub.
- Run a playbook that requires a collection from automation hub.

Before You Begin

As the student user on the workstation machine, use the lab command to prepare your system for this exercise.

This command creates the /home/student/manage-review/ directory on the workstation machine and obtains the files necessary to complete this lab.

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

Instructions

1. Change into the /home/student/manage-review/ directory and review the cockpit.yml playbook. Make note of the Ansible Content Collections that the playbook requires.
 - 1.1. Change into the /home/student/manage-review/ directory.

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

- 1.2. Review the cockpit.yml playbook.

```
[student@workstation manage-review]$ cat cockpit.yml
---
- name: Deploy cockpit
  hosts: servera.lab.example.com
  become: true

  tasks:
    - name: Ensure cockpit is installed
      ansible.builtin.yum:
```

```

name: cockpit
state: present

- name: Ensure the cockpit service is started and enabled
  ansible.builtin.systemd:
    name: cockpit
    state: started
    enabled: true

- name: Ensure Cockpit network traffic is allowed
  ansible.posix.firewalld:
    service: cockpit
    permanent: true
    state: enabled
    immediate: true

```

Notice that the `ansible.posix.firewalld` module requires the `ansible.posix` collection.

2. Use the `ansible-navigator` command to inspect the automation execution environment images available on your local system. Determine which one you need to run the `cockpit.yml` playbook.

- 2.1. List the available automation execution environment images.

```
[student@workstation manage-review]$ ansible-navigator images
  Image           Tag     Execution environment   Created      Size
0|ee-minimal-rhel8    latest    True            3 months ago  294 MB
1|ee-supported-rhel8  latest    True            3 months ago  1.68 GB
```

Type 1 to select the `ee-supported-rhel8` image.

- 2.2. The TUI displays information about the `ee-supported-rhel` image.

```

Image: ee-supported-rhel8:latest  Description
0|Image information          Information collected from image inspection
1|General information        OS and python version information
2|Ansible version and collections  Information about ansible and ansible ...
3|Python packages             Information about python and python packages
4|Operating system packages  Information about operating system packages
5|Everything                  All image information

```

Type 2 to view the collections available in the `ee-supported-rhel8` image.

- 2.3. Notice that the `ee-supported-rhel8` image provides the `ansible.posix` collection that you need to run the `cockpit.yml` playbook.

```

Image: ee-supported-rhel8:latest (Information about ansible and ansible
collections)
0|---
1|ansible:
2|  collections:
3|    details:
4|      amazon.aws: 3.2.0
5|      ansible.controller: 4.2.1

```

```
6|     ansible.netcommon: 3.1.1
7|     ansible.network: 1.2.0
8|     ansible.posix: 1.3.0
9|     ansible.security: 1.0.0
10|    ansible.utils: 2.6.1
...output omitted...
```

Type :q and then press Enter to exit the ansible-navigator command.

3. Run the cockpit.yml playbook by using the ansible-navigator command with the correct automation execution environment image.

- 3.1. Use the ansible-navigator run command with the ee-supported-rhel8 image to run the cockpit.yml playbook.

```
[student@workstation manage-review]$ ansible-navigator run cockpit.yml \
> --eei ee-supported-rhel8
...output omitted...
```

Press Esc to exit the ansible-navigator command after the playbook run is complete.

4. Log in to the web console at <https://servera.lab.example.com:9090> to confirm that the playbook correctly configured the web console. Accept the insecure TLS certificate and then log in as the student user. Use student as the password.

- 4.1. Open a web browser on the workstation machine and navigate to <https://servera.lab.example.com:9090>.

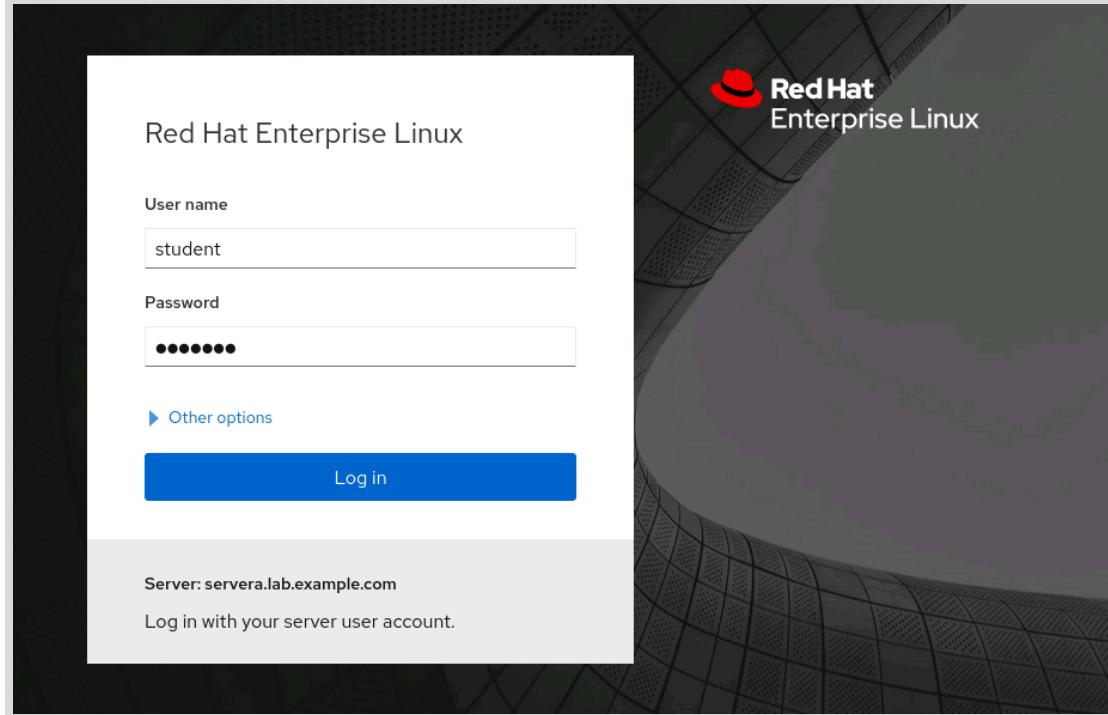


Figure 2.3: Accessing the web console login page

- 4.2. Log in with student as the username and student as the password. The playbook correctly configured the web console if the login process is successful.

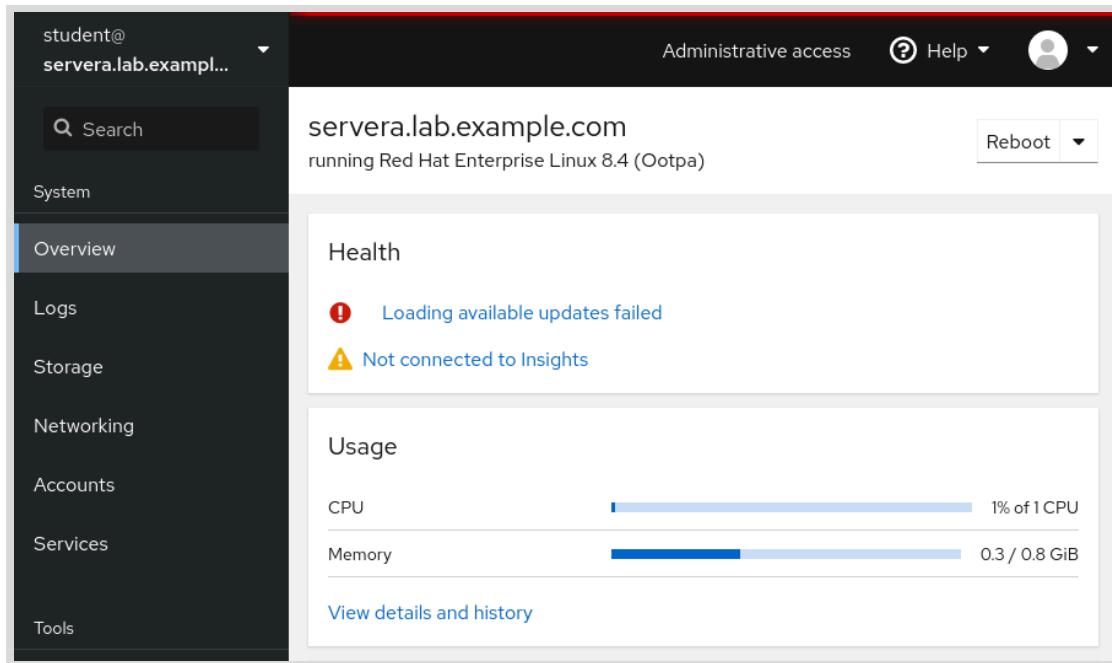


Figure 2.4: Accessing the Overview page

5. Review the `create_samples_db.yml` playbook. Make note of the collections that it requires. Find those collections in the private automation hub.

You can access the private automation hub web UI at <https://hub.lab.example.com>. Use `student` as the username and `redhat123` as the password.

5.1. Review the `create_samples_db.yml` playbook.

```
[student@workstation manage-review]$ cat create_samples_db.yml
---
- name: Create samples database
  hosts: serverb.lab.example.com
  become: true

  tasks:
    - name: Ensure MariaDB is installed
      ansible.builtin.yum:
        name:
          - mariadb
          - mariadb-server
        state: present

    - name: Ensure the mariadb service is started and enabled
      ansible.builtin.systemd:
        name: mariadb
        state: started
        enabled: true

    - name: Ensure the samples database exists
      community.mysql.mysql_db:
        name: samples
        state: present
```

Notice that the playbook requires the `community.mysql` collection.

- 5.2. Open a web browser and navigate to `https://hub.lab.example.com`. Use `student` as the username and `redhat123` as the password.

Navigate to **Collections > Collections**. Change the **Filter by repository** selection to **Community** and search for `mysql`.

The screenshot shows the Red Hat Ansible Galaxy interface. At the top, there's a navigation bar with the Red Hat logo and a user dropdown for "Student User". Below it, a search bar has "Community" selected as the filter. A search input field contains "mysql", and a magnifying glass icon is to its right. Below the search bar, there's a "Clear all filters" button. The main area is titled "Collections" and displays three search results:

- mysql** (Provided by community): MySQL collection for Ansible. It shows 0 Modules, 0 Roles, 0 Plugins, and 0 Dependencies.
- podman** (Provided by containers): Podman container Ansible modules. It shows 0 Modules, 0 Roles, 0 Plugins, and 0 Dependencies.
- ah_configuration** (Provided by redhat_cop): Ansible content that interacts with the Ansible Automation H... It shows 0 Modules, 0 Roles, 0 Plugins, and 0 Dependencies.

Figure 2.5: Searching collections

- 5.3. Click the `mysql` collection to obtain the `ansible-galaxy` installation command.

Chapter 2 | Managing Content Collections and Execution Environments

The screenshot shows the Red Hat Ansible Galaxy web interface. At the top, there's a navigation bar with the Red Hat logo and a 'Student User' dropdown. Below it, a search bar says 'Filter by repository' and has a 'Community' dropdown. The URL shows 'Namespaces > community > mysql'. The main content area is titled 'mysql' and shows a version '3.5.1 updated 24 days ago (latest)'. It includes tabs for 'Install', 'Documentation', 'Contents', 'Import log', and 'Dependencies'. Below these are sections for 'Install', 'MySQL collection for Ansible', and 'License'. Under 'Installation', there's a command input field containing 'ansible-galaxy collection install community.mysql' with a note: 'Note: Installing collections with ansible-galaxy is only supported in ansible 2.9+'. A 'Download tarball' button is also present. Requirements are listed as 'Requires Ansible >=2.9.10'.

Figure 2.6: Retrieving the command to install a collection

6. Configure the /home/student/manage-review/ansible.cfg file so that the ansible-galaxy command installs collections from private automation hub.
 - 6.1. From the private automation hub web UI, navigate to **Collections > Repository Management**. Use the **CLI configuration** section from the community repository to construct the /home/student/manage-review/ansible.cfg file.

The screenshot shows the 'Repo Management' page in the Red Hat Ansible Galaxy web interface. It has a 'Get token' button at the top right. Below it, there are tabs for 'Local' and 'Remote', with 'Local' selected. A table lists repositories:

Distributio...	Repositor...	Cont...	Last upd...	Repo URL	CLI configuration
community	community	194	24 days ago	https://hub.lab.exa...	[galaxy] server_list...
published	published	0	24 days ago	https://hub.lab.exa...	[galaxy] server_list...

Figure 2.7: Listing the private automation hub repositories

- 6.2. Add the repository configurations to the /home/student/manage-review/ansible.cfg file.

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

[galaxy]
server_list = community_repo

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=<put your token here>
```

- 6.3. From the private automation hub web UI, navigate to Collections > API token management and then click Load token. Click the Copy to clipboard icon.

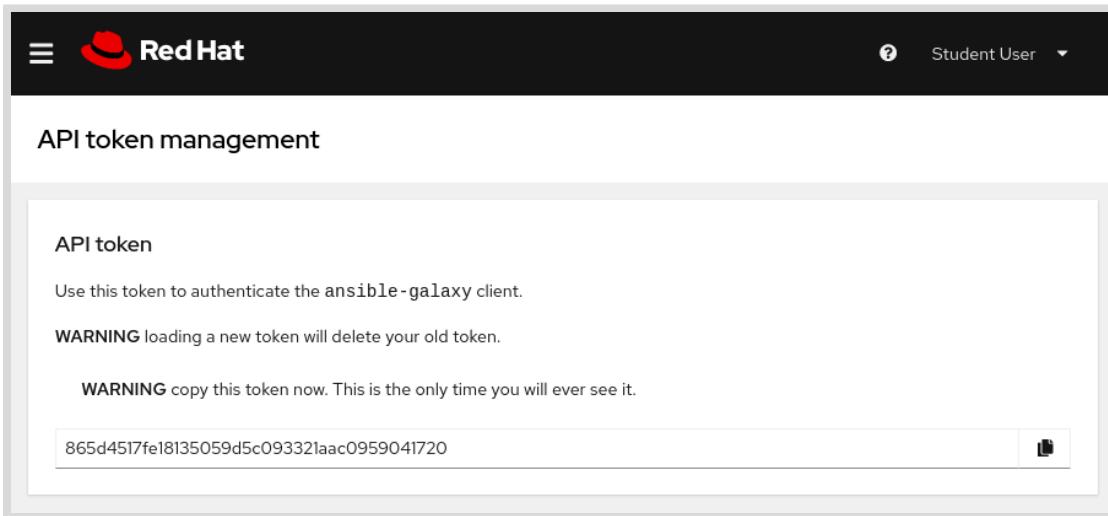


Figure 2.8: Generating a token

- 6.4. Edit the /home/student/manage-review/ansible.cfg file and then paste the token from the clipboard as the value for the token parameters. Your token is different from the one displayed in the following example.

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

[galaxy]
server_list = community_repo

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=865d4517fe18135059d5c093321aac0959041720
```

7. Create the /home/student/manage-review/collections/ directory. In this directory, install any collections that the create_samples_db.yml playbook requires.
- 7.1. Create the /home/student/manage-review/collections/ directory.

```
[student@workstation manage-review]$ mkdir collections
```

Chapter 2 | Managing Content Collections and Execution Environments

- 7.2. Install the `community.mysql` collection into the `/home/student/manage-review/collections/` directory.

```
[student@workstation manage-review]$ ansible-galaxy collection install \
> community.mysql -p collections/
...output omitted...
```

8. Use the `ansible-navigator` command to run the `create_samples_db.yml` playbook.

- 8.1. Use the `ansible-navigator run` command to run the playbook.

```
[student@workstation manage-review]$ ansible-navigator run create_samples_db.yml
...output omitted...
```

Press Esc to exit the `ansible-navigator` command after the playbook run is complete.

9. Confirm that the playbook correctly configured a MariaDB database on the `serverb` machine. You can confirm that the playbook created the `samples` database by running the `sudo mysql -e "SHOW DATABASES"` command. The password for the `student` user is `student`.

- 9.1. Connect to `serverb` as the `student` user.

```
[student@workstation manage-review]$ ssh serverb
...output omitted...
[student@serverb ~]$
```

- 9.2. Confirm that the `samples` database exists.

```
[student@serverb ~]$ sudo mysql -e "SHOW DATABASES"
[sudo] password for student: student
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| samples         |
+-----+
```

- 9.3. Log out of the `serverb` machine.

```
[student@serverb ~]$ logout
Connection to serverb closed.
[student@workstation manage-review]$
```

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- You can use the `ansible-navigator doc --mode stdout` command to access the module documentation provided by Ansible Content Collections that are available to your automation execution environment.
- You can use the `ansible-navigator images` command to inspect automation execution environments and to list the collections and other resources and collection dependencies provided by an automation execution environment's container image.
- Ansible Playbooks should use fully qualified collection names (FQCNs) to refer to modules, roles, and plug-ins provided by Ansible Content Collections.
- The `ansible-galaxy collection install` command installs Ansible Content Collections on the local system. The `--collections-path` (or `-p`) option specifies the installation directory.
- You can use the `collections/requirements.yml` file to list the Ansible Content Collections that are required for the project.
- Automation execution environments can access the Ansible Content Collections that are installed in the `collections/` subdirectory in the directory that contains the playbook.
- The `ee-supported-rhel8` automation execution environment is used by default by automation content navigator and automation controller, and includes selected Red Hat Certified Ansible Content Collections in addition to the `ansible.builtin` collection.
- The `ee-minimal-rhel8` automation execution environment only provides the `ansible.builtin` Ansible Content Collection, but you can also use Ansible Content Collections from your project's `collections/` directory.
- You can use the `ee-29-rhel8` automation execution environment for playbooks that require Ansible 2.9.

Chapter 3

Running Playbooks with Automation Controller

Goal

Explain what automation controller is and demonstrate how to use it to run playbooks that you developed with automation content navigator.

Objectives

- Describe the architecture and use cases of the automation controller component of Red Hat Ansible Automation Platform.
- Navigate and describe the automation controller web UI, and successfully launch a job using a job template, project, credential, and inventory.

Sections

- Explaining the Automation Controller Architecture (and Quiz)
- Running Playbooks in Automation Controller (and Guided Exercise)

Lab

- Running Playbooks with Automation Controller

Explaining the Automation Controller Architecture

Objectives

- Describe the architecture and use cases of the automation controller component of Red Hat Ansible Automation Platform.

Introduction to Automation Controller

Red Hat Ansible Automation Platform 2 includes a component called *automation controller*, which was called *Red Hat Ansible Tower* in earlier versions of Ansible Automation Platform. Automation controller provides a centralized hub that you can use to run your Ansible automation code.

Enterprise IT organizations need a way to define and embed automation workflows for other tools and processes. They also need reliable and scalable automation execution, and a centralized system that supports auditing.

With automation controller, companies can automate with confidence and reduce automation drift and variance across the enterprise by standardizing how automation deploys in one centralized location.

Automation controller provides a framework for running and managing Ansible efficiently on an enterprise scale. Automation controller maintains organization security by introducing features such as a centralized web UI for playbook management, role-based access control (RBAC), and centralized logging and auditing. You can integrate an enterprise's existing workflows and tool sets, such as enabling continuous integration and deployment, by using the automation controller REST API. Automation controller provides mechanisms to enable the centralized use and control of machine credentials and other secrets without exposing the credentials or secrets to the end users of automation controller.

Automation Controller Architecture

The introduction of automation execution environments in Red Hat Ansible Automation Platform 2 decouples the automation controller control plane from its execution environment.

Red Hat Ansible Tower 3.8 and earlier tightly coupled the execution environment to the system that ran Ansible Tower. Sometimes, this tight coupling required you to manage the dependencies of various modules needed to run Ansible Playbooks. If two playbooks required different environments, then you needed to create one or more Python virtual environments on the Ansible Tower system to manage the different dependencies and requirements. Some enterprises ended up with tens or hundreds of Python virtual environments.

Automation controller improves this architecture significantly. Instead of using the system executables and Python installation or virtual environment, automation controller uses automation execution environments. These environments are container images that you can pull from a central container registry, install on automation controller, and manage through a web UI. If needed, then you can create custom automation execution environments. After confirming that a custom automation execution environment works with your automation code, you can publish the container image to a container registry and then make the new or updated automation execution environment available to automation controller. Using the same automation execution

environment helps ensure that automation code runs consistently on both your system and in automation controller.

Compare the previous Red Hat Ansible Tower architecture shown in Figure 3.1 to the updated automation controller architecture shown in Figure 3.2.

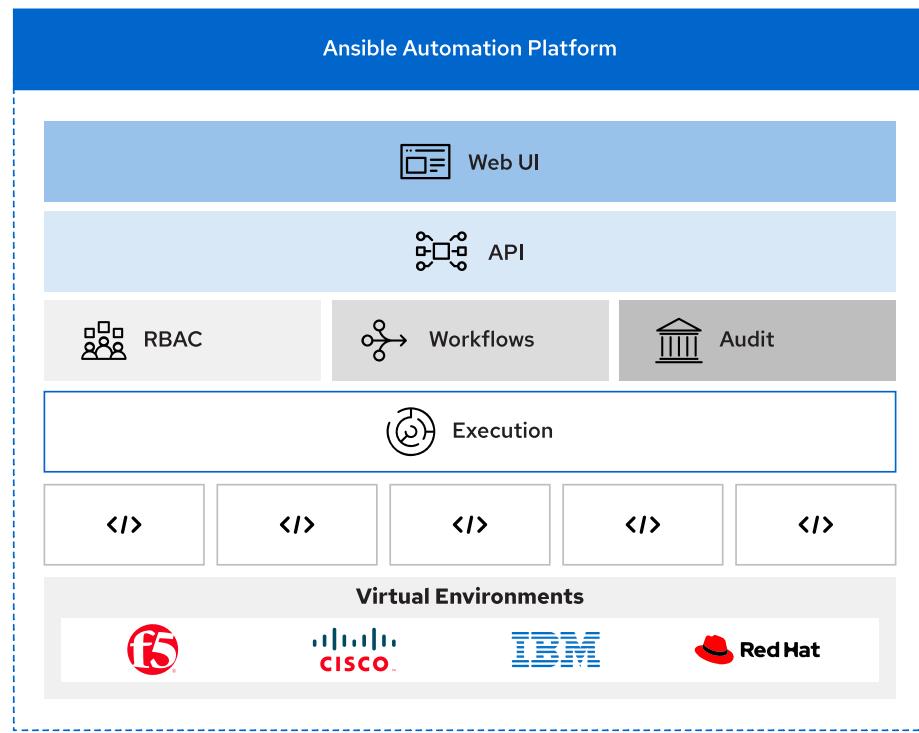


Figure 3.1: Architecture of Ansible Tower 3.8

- Centralized, monolithic application
- Control node contains the control plane and the execution plane
- Poor scalability from a rigid architecture

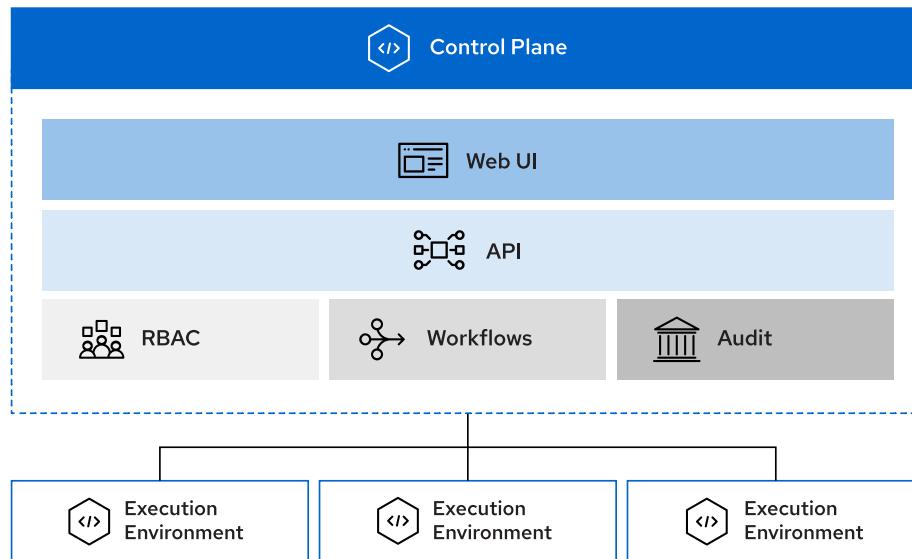


Figure 3.2: Architecture of Ansible Automation Platform 2

- Decentralized, modular application
- Control plane and execution plane decoupled
- Containerized virtual environments
- Scale as needed, on-demand by using container orchestrators

This design enables automation controller to run the control plane (with the web UI and API) on the automation controller system, and run automation execution environments on other machines closer to the managed systems, enabling increased efficiency and scaling.

Automation Controller Features

Automation controller offers many features for controlling, securing, and managing Ansible in an enterprise environment, such as:

Visual Dashboard

The automation controller web UI displays a dashboard that provides a summary of an enterprise's entire Ansible environment. You can use the automation controller dashboard to review the current status of hosts and inventories, and the results of recent job executions. Automation controller's upgraded web UI provides better security and performance and improves observability with new filtering capabilities and distinct views.

Role-based Access Control (RBAC)

Automation controller uses a role-based access control (RBAC) system, which maintains security when streamlining user access management. You can assign access to automation controller objects, such as organizations, projects, and inventories.

Graphical Inventory Management

You can use the automation controller web UI to create inventories and then add host groups and hosts to the inventories. Alternatively, you can update inventories from an external inventory source such as public cloud providers, local virtualization environments, an organization's custom configuration management database (CMDB), or content in a Git repository.

Task Manager and Job Scheduling

Use automation controller to schedule playbook execution and updates from external data sources either on a one-time or recurring basis. This means that routine tasks can run unattended, and is especially useful for tasks such as backup routines, which are ideally executed during operational off-hours.

Real-time and Historical Job Status Reporting

When you initiate a playbook run in automation controller, the web UI displays the playbook output and execution results in real time. The results of previously executed jobs and scheduled job runs are also available from automation controller for auditing or review.

User-triggered Automation

Users with appropriate permissions can launch job templates or workflow job templates with a single click of an icon.

Credential Management

Automation controller centrally manages authentication credentials. This means that you can run Ansible plays on managed hosts, synchronize information from dynamic inventory sources, and import Ansible project content from version control systems.

Automation controller encrypts the passwords or keys provided so that automation controller users cannot retrieve them. Even if a user has privileges to use or edit a credential, automation controller does not expose the current values of the credential passwords or keys.

Centralized Logging and Auditing

Automation controller logs all playbook and remote command execution. This provides the ability to audit when each job was executed and by whom. In addition, automation controller can integrate its log data into third-party logging aggregation solutions, such as Splunk and Sumologic.

Integrated Notifications

Automation controller can notify you when job executions start and whether the job succeeds or fails. Automation controller can deliver notifications to many applications, including email, Grafana, IRC, Mattermost, Slack, Rocket.Chat, Twilio, and Webhooks.

Multiplaybook Workflows

Complex operations often involve the serial execution of multiple playbooks. Automation controller multiplaybook workflows can chain together numerous playbooks to facilitate the implementation of complex routines involving provisioning, configuration, deployment, and orchestration. An intuitive workflow editor also helps model multiplaybook workflows.

Browsable RESTful API

The automation controller's RESTful API exposes every automation controller feature available through the web UI. The browsable format of the API makes it self-documenting and lets you look up information on how to use the API.



References

Automation Controller User Guide v4.2.1

<https://docs.ansible.com/automation-controller/4.2.1/html/userguide/index.html>

What Is Automation Controller?

<https://www.redhat.com/en/technologies/management/ansible/automation-controller>

For more information, refer to *What's New in Ansible Automation Controller 4.0* at
<https://access.redhat.com/articles/6184841>

► Quiz

Explaining the Automation Controller Architecture

Choose the correct answers to the following questions:

► 1. **Which statement about automation controller is accurate?**

- a. Automation controller is not included with Red Hat Ansible Automation Platform 2.
- b. Automation controller is a modular application that has decoupled the control plane from the execution plane by using containerization.
- c. Automation controller is a command-line tool that you use to manage Ansible Playbooks in Git repositories.
- d. Automation controller is a cloud-based application accessed through `console.redhat.com`.

► 2. **Which feature is not part of automation controller?**

- a. A wizard for creating playbooks.
- b. Logs that record the results and output of playbook execution.
- c. Role-based access control (RBAC).
- d. A RESTful API.

► 3. **Which statement is true about how automation controller manages credentials?**

- a. Users who can use credentials stored in automation controller can read the secrets stored in them.
- b. Credentials cannot be used to authenticate to a version control system to import Ansible project content.
- c. Automation controller encrypts the secrets stored in credentials so that automation controller users can use them but not read them.
- d. Automation controller cannot manage credentials.

► 4. **Which statement about improvements to the automation controller architecture is true?**

- a. You can create playbooks in the web interface.
- b. You can use Python virtual environments to manage playbook dependencies.
- c. You can use automation execution environments to contain playbook dependencies, separating the control plane from the execution plane.
- d. You can identify failed deployments and automatically launch jobs by using AI.

► Solution

Explaining the Automation Controller Architecture

Choose the correct answers to the following questions:

► 1. Which statement about automation controller is accurate?

- a. Automation controller is not included with Red Hat Ansible Automation Platform 2.
- b. Automation controller is a modular application that has decoupled the control plane from the execution plane by using containerization.
- c. Automation controller is a command-line tool that you use to manage Ansible Playbooks in Git repositories.
- d. Automation controller is a cloud-based application accessed through `console.redhat.com`.

► 2. Which feature is not part of automation controller?

- a. A wizard for creating playbooks.
- b. Logs that record the results and output of playbook execution.
- c. Role-based access control (RBAC).
- d. A RESTful API.

► 3. Which statement is true about how automation controller manages credentials?

- a. Users who can use credentials stored in automation controller can read the secrets stored in them.
- b. Credentials cannot be used to authenticate to a version control system to import Ansible project content.
- c. Automation controller encrypts the secrets stored in credentials so that automation controller users can use them but not read them.
- d. Automation controller cannot manage credentials.

► 4. Which statement about improvements to the automation controller architecture is true?

- a. You can create playbooks in the web interface.
- b. You can use Python virtual environments to manage playbook dependencies.
- c. You can use automation execution environments to contain playbook dependencies, separating the control plane from the execution plane.
- d. You can identify failed deployments and automatically launch jobs by using AI.

Running Playbooks in Automation Controller

Objectives

- Navigate and describe the automation controller web UI, and successfully launch a job using a job template, project, credential, and inventory.

Exploring Resources in Automation Controller

Automation controller provides a centralized location for organizations to run their Ansible Playbooks.

Several resources must exist before you can create a job template to test a playbook in automation controller.

Typically, these resources include:

- A machine credential used to connect to the managed hosts.
- A source control credential used to download and synchronize remote content, such as from a Git repository.
- A project that specifies the location of content, such as playbooks.
- An inventory with at least one host.

Although automation controller provides tremendous functionality, this course focuses on creating the basic resources needed to launch a job template. Detailed information about individual resources, including options and assigning permissions to those resources, is outside the scope of this course.

Creating Credential Resources

Automation controller accesses many resources that might require separate credentials. The following list briefly describes some credential types and their intended purpose. Descriptions of additional credential types and their intended purpose can be found in the reference links.

Ansible Galaxy/Automation Hub API Token

Create the credentials resource so that automation controller can download content collections and roles from Ansible Galaxy, automation hub, and private automation hub.

After creating the credential, you must enable the credential for an organization. During this process, you can specify a precedence by ordering the credentials.



Note

If you install automation controller and private automation hub at the same time, then the installer creates automation controller credentials for the community, published, and Red Hat certified repositories available from private automation hub. The installer also enables these credentials for the Default automation controller organization.

Container Registry

Create this type of credential if you need to authenticate before you can pull down a container image from a container registry, such as `registry.redhat.io` or a private automation hub.



Note

If you install automation controller and private automation hub at the same time, then the installer creates an automation controller container registry credential for private automation hub.

GitHub Personal Access Token

GitHub no longer supports password-based authentication to synchronize projects using the HTTPS protocol. To continue using HTTPS (rather than SSH), create and use a GitHub personal access token, and then create an automation controller credential that uses the personal access token.

Machine

Automation controller can use this credential type to access and make changes to the managed hosts. This credential specifies a username and either a password or the content of the user's SSH private key. If the credential should allow privilege escalation, specify the privilege escalation username (typically `root`) and specify a privilege escalation password if necessary.



Note

In organizations that mandate against storing passwords, you can configure automation controller to prompt for passwords when using the credential.

Source Control

Automation controller can use this credential type to synchronize project resources from a remote repository. Specify a username and either a password or the user's SSH private key.

Vault

Use vault credentials to decrypt files that have been encrypted with Ansible vault.

Listing Credentials

Navigate to **Resources > Credentials** to list credentials. Automation controller displays the name and credential type for each credential.

Click the name of any credential to display credential details.

Figure 3.3: List of existing credentials

Creating a Machine Credential

To create a new machine credential, navigate to **Resources > Credentials** and click **Add**. Specify a name for the credential and choose the machine credential type. Specify additional settings and then click **Save**.

Setting	Description
Username	Automation controller connects to the managed hosts as this user (similar to the <code>remote_user</code> setting in an <code>ansible.cfg</code> file).
Password	The password associated with the user. Enter the password, or prompt for the password when the credential is used. A password or SSH private key is required. Using a password does not work if a managed host prevents password-based authentication over SSH.
SSH Private Key	This is the associated private key for a public SSH key that has already been copied to the managed host. A password or SSH private key is required.
Private Key Passphrase	If the private SSH key is password protected, then enter the passphrase, or prompt for the passphrase when the credential is used.
Privilege Escalation Method	The method used for privilege escalation (the equivalent of the <code>become_method</code> setting in an <code>ansible.cfg</code> file).
Privilege Escalation Username	The user to become for tasks that require privilege escalation (the equivalent of the <code>become_user</code> setting in an <code>ansible.cfg</code> file).

Setting	Description
Privilege Escalation Password	If the user requires a password for privilege escalation, then enter the password, or prompt for the password when the credential is used.

Creating a Source Control Credential

To create a new source control credential, navigate to **Resources > Credentials** and click **Add**. Specify a name for the credential and choose the source control credential type.

Setting	Description
Username	Automation controller connects to the source control repository as this user.
Password	The password associated with the user. The source control repository might disable password-based authentication. In that case, use the SCM private key.
SCM Private Key	This is the associated private key for a public SSH key that has already been copied to the managed host.
Private Key Passphrase	If the private SCM key is password-protected, then enter the passphrase.

Creating Project Resources

Automation controller uses project resources to provide access to Ansible Playbooks. If a project uses source control, such as Git or Subversion, then automation controller synchronizes content from a remote repository.

Navigate to **Resources > Projects** to list existing projects. Click the **>** icon to the left of any project name to expand project information. Synchronize projects that use source control from the **Projects** page.

Use the **Revision** column to assess if your project is current. For a Git repository, the revision string matches a commit hash.

Name	Status	Type	Revision	Actions
Controller Playbooks Project	Successful	Git	d339950	
Demo Project	Sync for revision			

Figure 3.4: List of existing projects

To create a new project, navigate to **Resources > Projects** and click **Add**.

Projects can specify a default execution environment to use for job templates. If not specified, then the project defaults to the defined default execution environment for automation controller. Job templates can choose a different execution environment.

Choose a source control credential type, such as Git, specify additional settings, and then click **Save**.

Setting	Description
Source Control URL	The URL used to clone the remote repository.
Source Control Branch/Tag/Commit	(Optional) A specific branch, tag, or commit for the repository. Use the Allow Branch Override option so that job templates can use a different branch, tag, or commit.
Source Control Credential	An <i>existing</i> credential that can be used to synchronize the remote repository.

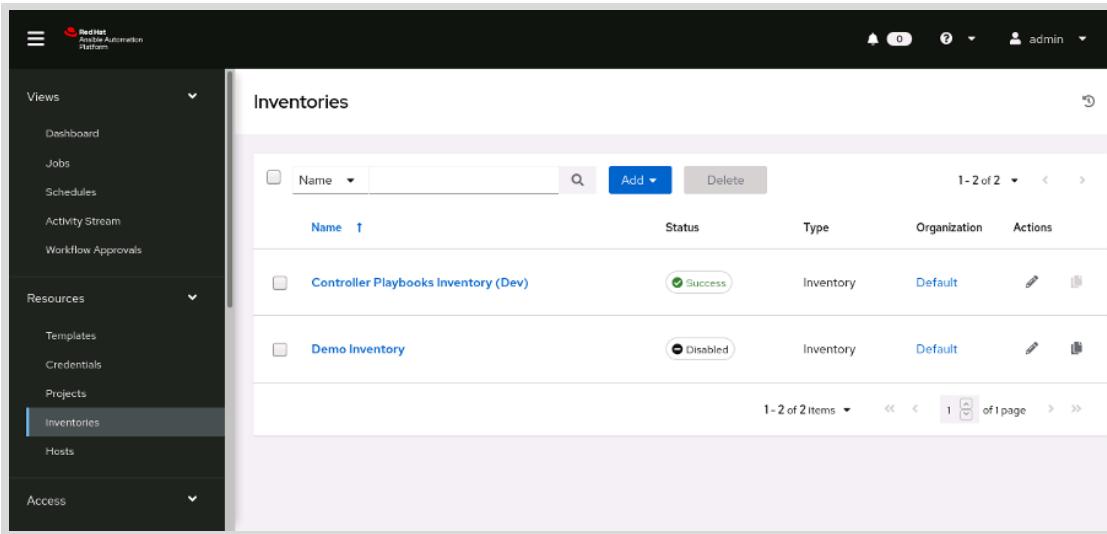
Automation controller automatically attempts to synchronize new projects. A project synchronization can fail for many reasons, including:

- The project specifies an incorrect source control URL.
- The project fails to specify, or specifies the wrong source control credential.
- The source control credential specifies the wrong username, password, or SCM private key.

Creating Inventory Resources

Similar to a local inventory file or directory, you can create inventory resources within automation controller. Inventories can contain groups of hosts as well as ungrouped hosts. You can configure variables that apply to the entire inventory (similar to the `all` group), to a specific group (similar to using the `groups_vars` directory), and to a specific host (similar to using the `host_vars` directory). When creating a job template, you must specify an inventory resource to use. This is similar to specifying the `--inventory (-i)` option to the `ansible-navigator` command.

Automation controller can contain many inventories, both static and dynamic. Inventories that contain dynamic content display the status of the most recent inventory synchronization attempt. Static inventories display **Disabled** in the **Status** column.



The screenshot shows the Red Hat Ansible Automation Platform interface. The left sidebar has a 'Views' dropdown, followed by a list of resources: Dashboard, Jobs, Schedules, Activity Stream, Workflow Approvals, Resources (Templates, Credentials, Projects), Inventories (selected), Hosts, and Access. The main area is titled 'Inventories' and contains a table with two rows. The columns are Name, Status, Type, Organization, and Actions. Row 1: 'Controller Playbooks Inventory (Dev)' with Status 'Success', Type 'Inventory', Organization 'Default', and Actions (Edit, Delete). Row 2: 'Demo Inventory' with Status 'Disabled', Type 'Inventory', Organization 'Default', and Actions (Edit, Delete). The bottom of the table shows '1-2 of 2 items' and navigation links.

Figure 3.5: List of existing inventories

To create an inventory, navigate to **Resources > Inventories** and click **Add > Add inventory**. Specify a name for the inventory and click **Save**.

After creating the inventory, populate the inventory with groups and hosts. You can manually add the groups and hosts or you can automatically populate them by using an existing inventory file in the project repository.

Manually Adding Groups and Hosts

To manually add a group, navigate to the **Groups** tab and then click **Add**. Enter a name for the group and then click **Save**.

To manually add an ungrouped host, navigate to the **Hosts** tab for the inventory and click **Add**. Enter a name for the host (frequently the fully qualified domain name for the host) and then click **Save**. You can add an ungrouped host to a group at any time.

To manually add a host to a group, navigate to the **Groups** tab for the inventory, click the group name, and then click the **Hosts** tab for the group. You can either associate an existing host within the inventory or add a new host to the group. When adding a new host, enter a name for the host (frequently the fully qualified domain name for the host) and then click **Save**.

The breadcrumb navigation displays the inventory name and the group name. In the following example, the `serverb.lab.example.com` host is being added to the `apac` group in the `Web Servers (Prod)` inventory.

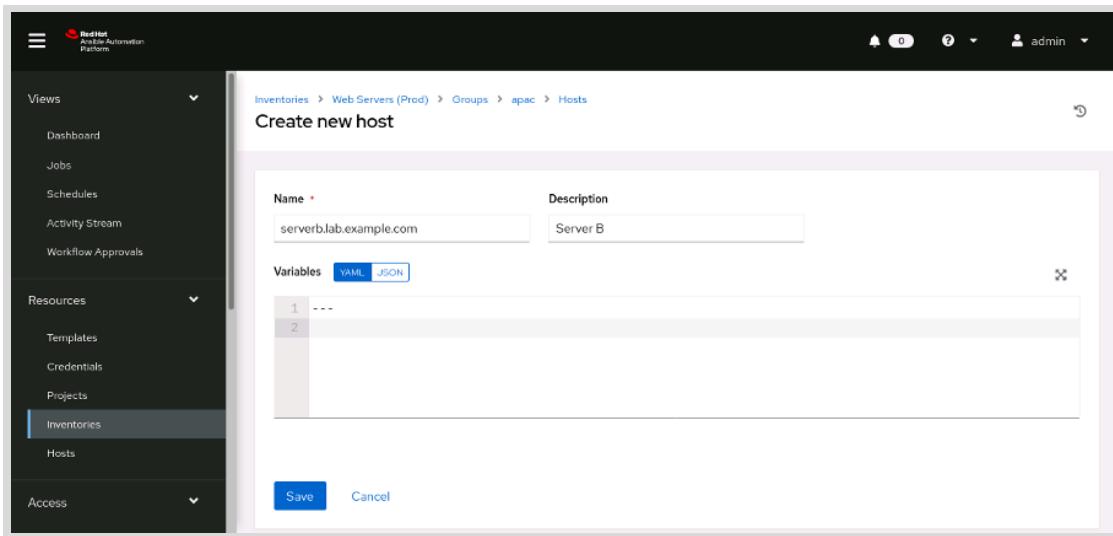


Figure 3.6: Adding a new host to a group

Populating Groups and Hosts by Using a Project Inventory File

If a project already contains an inventory file, then you can use the file to automatically populate groups and hosts within automation controller.

Within an existing group, navigate to the **Sources** tab and then click **Add**. Enter a name for the source and then select **Sourced from a Project** from the **Source** menu. Choose an existing project, select an inventory file that exists within the project, and then click **Save**.

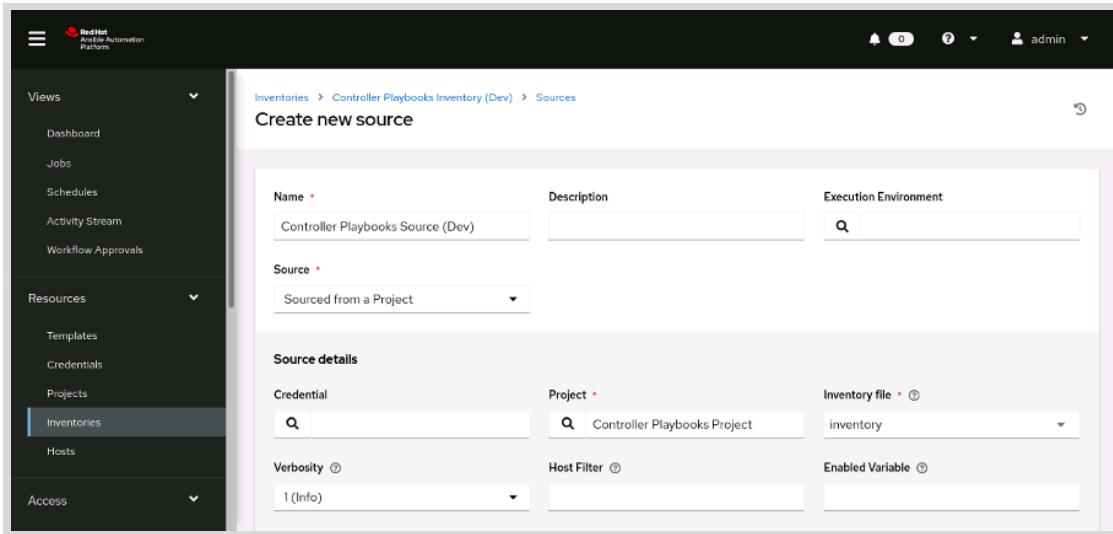


Figure 3.7: Adding a new inventory source from a project

Within an existing group, navigate to the **Sources** tab and then click the **Start sync process** icon for the source. A successful inventory synchronization creates the hosts and groups defined in the selected inventory file.

Creating Job Template Resources

A job template is the equivalent of a specific way of running a playbook. Among other things, a job template includes an inventory and might specify variables to override, or tags to either use or skip.

Chapter 3 | Running Playbooks with Automation Controller

The **Templates** page lists existing templates. Expand template information by clicking the **>** icon to the left of any template name. Initiate a new job run by clicking the **Launch Template** icon.

Figure 3.8: List of existing job templates

To create a new job template, navigate to **Resources > Templates** and then click **Add > Add job template**. At a minimum, enter a name, select a project, and select a playbook. You can either choose an existing inventory or prompt for an inventory when the job template is launched. Job templates should specify necessary credentials, such as the machine credential that automation controller needs to connect to the managed hosts.



Important

Organizations that forbid storing credentials, such as passwords or passphrases, should configure job templates to prompt for credentials. Any job template that prompts for passwords (or other settings) must be run *interactively*.

Job templates include several additional optional fields. Many of these options have corresponding command-line options that can be used when running a playbook with the `ansible-navigator` command. You can specify these options as part of the job template, or be prompted for customizations when the job template is launched.

Job template setting or option	Equivalent command-line option
Inventory	<code>--inventory (-i)</code>
Execution Environment	<code>--execution-environment-image (--eei)</code>
Variables	<code>--extra-vars (-e)</code>
Forks	<code>--forks (-f)</code>
Limit	<code>--limit (-l)</code>
Verbosity	<code>--verbose (-v)</code>
Job Tags	<code>--tags (-t)</code>

Job template setting or option	Equivalent command-line option
Skip Tags	--skip-tags
Privilege Escalation	--become (-b)

Launching and Reviewing Jobs

When you launch a job template, automation controller uses an execution environment to run the playbook using all the supplied information. If you configured the job template to prompt for information, such as an inventory, credentials, or variables, then automation controller prompts you for this information. Similarly, if the credentials used by your job template prompt for passwords or passphrases, then automation controller prompts you for this information. Job templates that prompt for any information must be run interactively.

Navigate to **Views > Jobs** to review job output. In addition to the output of playbook runs, the **Jobs** page displays inventory synchronization jobs and source control update jobs. Each job contains a number to show the order in which automation controller ran the job.

Expand job information by clicking the **>** icon to the left of any job name. Expanded details include who launched the job, which inventory was used, and which execution environment was used. Click a job name to see the full job output.

Name	Status	Type	Start Time	Finish Time	Actions
6 - Basic Web	Successful	Playbook Run	11/10/2021, 10:05:48 AM	11/10/2021, 10:06:04 AM	
4 - Controller Playbooks Inventory (Dev) - Controller Playbooks Source (Dev)	Successful	Inventory Sync	11/10/2021, 10:04:00 AM	11/10/2021, 10:04:08 AM	
1 - Controller Playbooks Project	Successful	Source Control Update	11/10/2021, 10:02:23 AM	11/10/2021, 10:02:44 AM	

Figure 3.9: List of jobs



References

Automation Controller User Guide

<https://docs.ansible.com/automation-controller/4.2.1/html/userguide/index.html>

► Guided Exercise

Running Playbooks in Automation Controller

Navigate through the automation controller web UI and launch a job.

Outcomes

- Create source control and machine credentials.
- Create a project.
- Create an inventory that uses an inventory file from the project.
- Create a new job template.
- Launch a job template and review the output generated by the job.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the Git repository used by this exercise.

```
[student@workstation ~]$ lab start controller-playbooks
```

Instructions

This exercise provides you with an Ansible project in a Git repository that contains a working playbook to deploy a simple web server.

- ▶ 1. Automation controller must have a *source control credential* to access content from a Git repository. From the automation controller web UI, create a source control credential named `Git Project Credential`. Configure this credential to use authentication information for the `git.lab.example.com` server.
 - 1.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.
 - 1.2. Navigate to **Resources > Credentials** and click **Add**. Create the credential with the following settings and then click **Save**.

Field	Value
Name	Git Project Credential
Organization	Default
Credential Type	Source Control
Username	student
SCM Private Key	Content of the /home/student/.ssh/gitlab_rsa file.

**Note**

If you choose to browse for the file, then right-click anywhere in the directory navigation and select **Show Hidden Files**. With this option enabled you can see the .ssh directory in the /home/student directory.

- ▶ 2. To run the playbook against each managed host, automation controller must have a *machine credential* that contains authentication information. Create a machine credential named **Devops Machine Credential** that contains authentication information for the managed hosts in the classroom environment.
 - 2.1. Navigate to **Resources > Credentials** and click **Add**. Create the credential with the following settings and then click **Save**.

Field	Value
Name	Devops Machine Credential
Organization	Default
Credential Type	Machine
Username	devops
SSH Private Key	Content of the /home/student/.ssh/lab_rsa file.
Privilege Escalation Method	sudo
Privilege Escalation Username	root

**Note**

If you choose to browse for the file, then right-click anywhere in the directory navigation and select **Show Hidden Files**. This lets you see the .ssh directory in the /home/student directory.

- ▶ 3. Create a project named **Controller Playbooks Project**. This project points to a Git repository that contains a playbook and an inventory file. Configure the project to specify a default automation execution environment.

Chapter 3 | Running Playbooks with Automation Controller

- 3.1. Navigate to **Resources > Projects** and then click **Add**. Create the project with the following settings and then click **Save**.

Field	Value
Name	Controller Playbooks Project
Organization	Default
Execution Environment	Automation Hub Default execution environment
Source Control Type	Git
Source Control URL	git@git.lab.example.com:student/controller-playbooks.git
Source Control Credential	Git Project Credential
Options	Allow Branch Override

After you click **Save**, automation controller immediately attempts to synchronize the project. The project synchronization should succeed.

**Important**

If synchronization fails, ensure that the project specifies the correct source control URL and credentials and then attempt the project synchronization again.

- ▶ 4. Create an inventory for this exercise named **Controller Playbooks Inventory (Dev)** using the **inventory** file in the project repository.

Managing your inventory file in a Git repository enables you to benefit from storing your changes in a version control system. You can also update it by editing the file rather than manually updating inventory information in the automation controller web UI.

- 4.1. Navigate to **Resources > Inventories** and then click **Add > Add inventory**.

- 4.2. Create a new inventory with the following settings and then click **Save**.

Field	Value
Name	Controller Playbooks Inventory (Dev)
Organization	Default

- 4.3. From the **Sources** tab, click **Add**. Create a new inventory source with the following settings and then click **Save**.

**Note**

After selecting the project, you might see the following error message: That value was not found. Please enter or select a valid value. The error message disappears when you select the inventory file.

Field	Value
Name	Controller Playbooks Source (Dev)
Source	Sourced from a Project
Project	Controller Playbooks Project
Inventory file	inventory
Options	Update on project update

- 4.4. Click **Sources** in the breadcrumb navigator and then click the **Start sync process** icon for the **Controller Playbooks Source (Dev)** source. The status updates to indicate that the synchronization succeeded.
 - 4.5. Click the **Hosts** tab. The inventory source imported the `serverf.lab.example.com` host.
 - 4.6. Click the **Groups** tab. The inventory source imported the **web** group.
 - 4.7. Click **web** and then click the **Hosts** tab. The **web** group contains the `serverf.lab.example.com` host.
- ▶ 5. Create a job template named **Basic Web**. Configure the job template to run the `basic-web-fqcn.yml` playbook using the default automation execution environment for the project. The playbook ensures that a web server and its content are deployed on hosts in the **web** inventory host group.
- 5.1. Navigate to **Resources > Templates** and then click **Add > Add job template**. Create the new job template with the following settings, and then click **Save**.

Field	Value
Name	Basic Web
Inventory	Controller Playbooks Inventory (Dev)
Project	Controller Playbooks Project
Playbook	<code>basic-web-fqcn.yml</code>
Credentials	Devops Machine Credential

- 5.2. Click **Launch** to run the job template. The resulting job successfully runs the playbook.
- 5.3. Navigate to **Views > Jobs** and look for a numbered job with **Basic Web** in the name. By default, automation controller lists the most recent jobs at the top. The **Basic Web** job displays a status of **Successful**.
- 5.4. *(Optional)* Click the name of the **Basic Web** job to review the job output. This is the same output that you initially saw when you launched the job.

- 5.5. Verify that the job successfully deployed a web server on `serverf.lab.example.com`. From a terminal window, run the `curl` command to access web content at `http://serverf.lab.example.com`.

```
[student@workstation ~]$ curl http://serverf.lab.example.com
Hello world from serverf.lab.example.com.
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish controller-playbooks
```

▶ Lab

Running Playbooks with Automation Controller

Launch a job with automation controller that runs a playbook using a specific execution environment, and then review the job results.

Outcomes

- Create automation controller resources, such as credentials, projects, and inventories.
- Create a new job template and then launch a job from the job template.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/controller-review` directory, prepares automation controller and private automation hub for the exercise, and creates a Git repository that includes a playbook to test your work.

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

Instructions

This lab provides you with a Git repository that contains a working playbook to deploy a simple web server. Create all resources from the automation controller web UI located at <https://controller.lab.example.com>. Log in as the `admin` user with `redhat` as the password.

1. Create a source control credential named `reviewgitcred`. The credential must belong to the `Default` organization. The GitLab user for the credential is `student`. Use the private key stored in the `/home/student/.ssh/gitlab_rsa` file on the `workstation` machine as the SCM private key.
2. Create a machine credential named `reviewmachinecred`. The credential must belong to the `Default` organization. Configure this credential for the `devops` user. Use the SSH private key stored in the `/home/student/.ssh/lab_rsa` file. Use `sudo` as the privilege escalation method and `root` as the privilege escalation username.
3. Create an inventory named `reviewinventory`. The inventory must belong to the `Default` organization. Add the `servera.lab.example.com` host to this inventory.
4. Create a project named `reviewproject`. The project must belong to the `Default` organization. This project must use the `git@git.lab.example.com:student/controller-review.git` Git repository and use the `reviewgitcred` source control credential to authenticate to the repository.
5. Create a job template named `reviewtemplate`. Configure the job template to use the `reviewinventory` inventory, the `reviewproject` project, the `webserver.yml` playbook, and the `reviewmachinecred` machine credential. Configure the job template to use the Automation Hub `Default` execution environment execution environment.

Launch a job that uses the `reviewtemplate` job template. If you completed the exercise correctly, then navigating to `http://servera.lab.example.com` displays the following message:

```
Successful playbook run from automation controller.
```

Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the workstation machine, change to the student user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Running Playbooks with Automation Controller

Launch a job with automation controller that runs a playbook using a specific execution environment, and then review the job results.

Outcomes

- Create automation controller resources, such as credentials, projects, and inventories.
- Create a new job template and then launch a job from the job template.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/controller-review` directory, prepares automation controller and private automation hub for the exercise, and creates a Git repository that includes a playbook to test your work.

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

Instructions

This lab provides you with a Git repository that contains a working playbook to deploy a simple web server. Create all resources from the automation controller web UI located at <https://controller.lab.example.com>. Log in as the `admin` user with `redhat` as the password.

1. Create a source control credential named `reviewgitcred`. The credential must belong to the `Default` organization. The GitLab user for the credential is `student`. Use the private key stored in the `/home/student/.ssh/gitlab_rsa` file on the `workstation` machine as the SCM private key.
 - 1.1. Navigate to <https://controller.lab.example.com> and log in as the `admin` user with `redhat` as the password.
 - 1.2. Navigate to **Resources > Credentials** and click **Add**. Create the credential with the following settings and then click **Save**.

Field	Value
Name	reviewgitcred
Organization	Default
Credential Type	Source Control
Username	student
SCM Private Key	Content of the /home/student/.ssh/gitlab_rsa file.

**Note**

If you choose to browse for the file, then right-click anywhere in the directory navigation area and select **Show Hidden Files**. Viewing hidden files lets you see the .ssh directory in the /home/student directory.

2. Create a machine credential named **reviewmachinecred**. The credential must belong to the **Default** organization. Configure this credential for the **devops** user. Use the SSH private key stored in the /home/student/.ssh/lab_rsa file. Use **sudo** as the privilege escalation method and **root** as the privilege escalation username.
 - 2.1. Navigate to **Resources > Credentials** and click **Add**. Create the credential with the following settings and then click **Save**.

Field	Value
Name	reviewmachinecred
Organization	Default
Credential Type	Machine
Username	devops
SSH Private Key	Content of the /home/student/.ssh/lab_rsa file.
Privilege Escalation Method	sudo
Privilege Escalation Username	root

3. Create an inventory named **reviewinventory**. The inventory must belong to the **Default** organization. Add the **servera.lab.example.com** host to this inventory.
 - 3.1. Navigate to **Resources > Inventories** and then click **Add > Add Inventory**. Create the inventory with the following settings and then click **Save**.

Field	Value
Name	reviewinventory
Organization	Default

- 3.2. Click the **Hosts** tab and then click **Add**. Add the `servera.lab.example.com` host, and then click **Save**.
4. Create a project named `reviewproject`. The project must belong to the `Default` organization. This project must use the `git@git.lab.example.com:student/controller-review.git` Git repository and use the `reviewgitcred` source control credential to authenticate to the repository.
- 4.1. Navigate to **Resources > Projects** and then click **Add**. Create the project with the following settings and then click **Save**.

Field	Value
Name	<code>reviewproject</code>
Organization	<code>Default</code>
Source Control Type	<code>Git</code>
Source Control URL	<code>git@git.lab.example.com:student/controller-review.git</code>
Source Control Credential	<code>reviewgitcred</code>

After clicking **Save**, automation controller immediately attempts to synchronize the project. The project synchronization succeeds.



Important

If the project synchronization fails, then ensure that the project specifies the correct source control URL and credential and then attempt the project synchronization again.

5. Create a job template named `reviewtemplate`. Configure the job template to use the `reviewinventory` inventory, the `reviewproject` project, the `webserver.yml` playbook, and the `reviewmachinecred` machine credential. Configure the job template to use the `Automation Hub Default execution environment` execution environment.

Launch a job that uses the `reviewtemplate` job template. If you completed the exercise correctly, then navigating to `http://servera.lab.example.com` displays the following message:

Successful playbook run from automation controller.

- 5.1. Navigate to **Resources > Templates** and then click **Add > Add job template**. Create the new job template with the following settings and then click **Save**.

Field	Value
Name	reviewtemplate
Inventory	reviewinventory
Project	reviewproject
Execution Environment	Automation Hub Default execution environment
Playbook	webserver.yml
Credentials	reviewmachinecred

- 5.2. Click **Launch** to create a job from the job template. The resulting job runs the playbook.



Important

If the job fails, then ensure that the job template specifies the correct inventory and machine credential. Ensure that the machine credential uses the correct values and then launch a new job from the job template.

- 5.3. Click the **Details** tab. Notice that the job displays **Successful** as its status.

- 5.4. Verify the web content updated by the job. Either use a web browser to navigate to <http://servera.lab.example.com> or run the following curl command:

```
[student@workstation ~]$ curl http://servera.lab.example.com
Successful playbook run from automation controller.
```

Evaluation

As the **student** user on the workstation machine, use the **lab** command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- Automation controller provides a centralized location that you can use to run your Ansible automation and review the results of automation runs.
- An automation controller *project* specifies the location of a Git repository that stores Ansible code and, if needed, a source control credential to authenticate to the Git repository.
- An automation controller *job template* specifies the inventory, machine credentials, execution environment, project, and playbook to use to run your Ansible code on managed hosts.
- You can use the automation controller web UI to launch automation jobs from a job template and to review the results of jobs.
- Using `ansible-navigator` to test Ansible Playbooks in an execution environment can help streamline the process of setting up those playbooks to run successfully on automation controller.

Chapter 4

Working with Ansible Configuration Settings

Goal

Examine and adjust the configuration of Ansible and automation content navigator to simplify development and to troubleshoot issues.

Objectives

- Browse the current Ansible configuration by using automation content navigator.
- Change configuration settings for automation content navigator with its configuration file, and determine where the configuration file is located.
- Examining the Ansible Configuration with Automation Content Navigator (and Guided Exercise)
- Configuring Automation Content Navigator (and Guided Exercise)
- Working with Ansible Configuration Settings

Sections

Lab

Examining the Ansible Configuration with Automation Content Navigator

Objectives

- Browse the current Ansible configuration by using automation content navigator.

Inspecting the Ansible Configuration in Interactive Mode

The `ansible-navigator config` command displays the current Ansible configuration used by the `ansible-navigator run` command in interactive mode. You can also use `ansible-navigator config` to determine the source of current `ansible-navigator` configuration settings.

Knowing the source of `ansible-navigator` configuration settings can help you determine why a particular setting in your current Ansible configuration has a specific value.

For example, configuration settings can come from the following sources:

- Specific environment variables.
- A configuration file specified by the `ANSIBLE_CONFIG` environment variable.
- Hard-coded default values.

You can use the `ansible-navigator config` command to examine your configuration, retrieve the current and default configuration settings, and access documentation for each configuration parameter.

The `ansible-navigator config` command is useful for troubleshooting configuration issues. For example, this command can help you locate the issue when a configuration parameter does not work as expected or when Ansible does not take a parameter into account. It also displays the actual value that Ansible uses for each parameter and from which source it retrieves that value, configuration file, or environment variable.

The following screen capture shows the output of the `ansible-navigator config` command:

Name	Default	Source	Current
0 Action warnings	True	default	True
1 Agnostic become prompt	True	default	True
<i>...output omitted...</i>			
44 Default ask pass	True	default	False
45 Default ask vault pass	True	default	False
46 Default become	False	/home/.../ansible.cfg	True
47 Default become ask pass	False	/home/.../ansible.cfg	False
48 Default become exe	True	default	None
<i>...output omitted...</i>			
61 Default forks	False	env	100
62 Default gathering	True	default	implicit
<i>...output omitted...</i>			

Chapter 4 | Working with Ansible Configuration Settings

In the preceding example, each line describes an Ansible configuration parameter. The columns provide the following information:

Name

The Name column displays the internal names that Ansible uses for parameters. These names are not the same as the names that you use in the `ansible.cfg` configuration file. For example, the `Default_become` parameter corresponds to the `become` parameter in the `ansible.cfg` configuration file. You can display that name mapping by typing the number of the parameter you are interested in, prefixed by a colon if that number is greater than 9.

Default

The Default column indicates whether the parameter is using its default value (when `True`) or is using a value that you explicitly set in an `ansible.cfg` configuration file or through an environment variable (when `False`). To help you identify changed parameters, the `ansible-navigator` command displays default parameter settings in green and those that have changed from the defaults in yellow.

Source

When a parameter is not using its default value, the Source column indicates how the parameter has been set. The Source column gives the full path to the configuration file from which Ansible retrieved the parameter's value. If the column displays the `env` keyword, then an environment variable set that parameter's value.

Current

The Current column displays the actual value of the parameter.

Searching for Specific Configuration Parameters

The filter function of the `ansible-navigator` command enables you to search for specific parameters. Type `:filter` (or `:f`) followed by a regular expression to perform a search in the command output. For example, to locate the `Default_forks` parameter, you could use the `:f forks` filter command.

This function is not limited to searching for parameter names. You can search for any pattern in the `ansible-navigator` output. For example, you can use the `:f ansible.cfg` filter command to list all the parameters defined in an `ansible.cfg` configuration file. Remember that regular expressions are case-sensitive.

Accessing Parameter Details

To get the details of a parameter, type its number, prefixed by a colon if that number is greater than 9. The following example displays the details of the `Default_forks` parameter.

```
Default forks (current: 100) (default: 5)
0|---
1|current_config_file: None
2|current_value: 100 ①
3|default: false
4|default_value: 5 ②
5|description: Maximum number of forks Ansible will use to execute tasks on
target hosts.
6|env:
7|- name: ANSIBLE_FORKS ③
8|ini:
9|- key: forks ④
10| section: defaults
```

Chapter 4 | Working with Ansible Configuration Settings

```

11|name: Default forks
12|option: DEFAULT_FORKS
13|source: env
14|type: integer
15|via: ANSIBLE_FORKS

```

- ➊ The actual parameter value is 100.
- ➋ The parameter's default value is 5. Ansible only uses that default value if you do not override it in the `ansible.cfg` configuration file or through the environment variable.
- ➌ The name of the environment variable that you can use to set the parameter value is `ANSIBLE_FORKS`.
- ➍ You can also set the parameter's value in an `ansible.cfg` configuration file by setting the `forks` parameter in the `[defaults]` section.

Inspecting the Local Configuration

By default, the `ansible-navigator` command uses an automation execution environment to perform its tasks. If your Ansible project does not provide the `ansible.cfg` configuration file, then the `ansible-navigator` command uses the `/etc/ansible/ansible.cfg` file that the automation execution environment supplies.

The command does not use the `/etc/ansible/ansible.cfg` or the `~/.ansible.cfg` configuration files from the local system when you use automation execution environments. To process those local files, use the `--execution-environment false` (or `--ee false`) option to prevent the command from using automation execution environments.



Important

Because you use `ansible-navigator` to run your playbook with an automation execution environment instead of the local control node environment, you usually inspect the automation execution environment configuration.

Inspecting the Ansible Configuration in Standard Output Mode

In addition to its interactive mode, the `ansible-navigator config` command can list the configuration parameters on the standard output and then exit. Add the `--mode stdout` (or `-m stdout`) option to the command to use that mode.

In non-interactive mode, the command works like the `ansible-config` command and requires a subcommand:

- The `list` subcommand lists all the Ansible configuration parameters. The resulting output is a static list that describes each parameter. The command does not report the current parameter values.
- The `dump` subcommand lists all the Ansible configuration parameters and their current values.
- The `view` subcommand displays the contents of the `ansible.cfg` configuration file that Ansible is using. This subcommand is helpful to confirm that Ansible is using the expected configuration file.

The following example lists the configuration parameters and their current values:

```
[user@host ansible-project]$ ansible-navigator config -m stdout dump
ACTION_WARNINGS(default) = True
AGNOSTIC_BECOME_PROMPT(default) = True
...output omitted...
DEFAULT_ASK_VAULT_PASS(default) = False
DEFAULT_BECOME(/home/user/ansible-project/ansible.cfg) = True
DEFAULT_BECOME_ASK_PASS(/home/user/ansible-project/ansible.cfg) = False
DEFAULT_BECOME_EXE(default) = None
...output omitted...
DEFAULT_FORCE_HANDLERS(default) = False
DEFAULT_FORKS(env: ANSIBLE_FORKS) = 100
DEFAULT_GATHERING(default) = implicit
...output omitted...
```



References

Review Your Ansible Configuration with Automation Content Navigator

https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.0-ea/html-single/ansible_navigator_creator_guide/index#assembly-review-config-navigator_ansible-navigator

Configuring Ansible – Ansible Documentation

https://docs.ansible.com/ansible/6/installation_guide/intro_configuration.html

► Guided Exercise

Examining the Ansible Configuration with Automation Content Navigator

Browse the current configuration settings for running Ansible Playbooks by using automation content navigator.

Outcomes

- Use the `ansible-navigator` command to review your current Ansible configuration.
- Compare your custom configuration to default Ansible settings.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/config-ansible` directory.

```
[student@workstation ~]$ lab start config-ansible
```

Instructions

- 1. Use automation content navigator to browse the current configuration and determine which settings use their default values.
- 1.1. Use the `ansible-navigator config` command.

```
[student@workstation ~]$ ansible-navigator config \
> --eei hub.lab.example.com/ee-supported-rhel8 --pp missing
```

You can explore the current configuration settings for automation content navigator.

Name	Default	Source	Current
0 Action warnings	True	default	True
1 Agnostic become prompt	True	default	True
2 Allow world readable tmpfiles	True	default	False
3 Ansible connection path	True	default	None
<i>...output omitted...</i>			

- 1.2. Search for the `Default host list` option. Type `:f host` and then press Enter.

Name	Default	Source	Current
0 Default host list	True	default	['/etc/ansible/hosts']
1 Display skipped hosts	True	default	True
<i>...output omitted...</i>			

Chapter 4 | Working with Ansible Configuration Settings

Notice that line 0 in your terminal output is green and the value in the **Default** column is **True**, indicating that it is a default setting.

13. Press 0 to view more details for the `Default host list` option.

```
Default host list (current/default: ['/etc/ansible/hosts'])
0|---
1|current_config_file: None
2|current_value:
3|- /etc/ansible/hosts
4|default: true
5|default_value: /etc/ansible/hosts
6|description: Comma separated list of Ansible inventory sources
7|env:
8|- name: ANSIBLE_INVENTORY
9|expand_relative_paths: true
10|ini:
11|- key: inventory
12| section: defaults
13|name: Default host list
14|option: DEFAULT_HOST_LIST
15|source: default
16|type: pathlist
17|via: default
18|yaml:
19| key: defaults.inventory
```

Exit from `ansible-navigator` by pressing ESC twice, or type :q and then press Enter.

- ▶ 2. Customize your current Ansible configuration and use automation content navigator to identify some changes that you made to the default settings. Ansible configuration settings specified within an `ansible.cfg` file in your current directory take precedence over the default Ansible configuration settings.

 - 2.1. Change to the `/home/student/config-ansible` directory and review the `ansible.cfg` configuration file.

```
[student@workstation ~]$ cd ~/config-ansible
[student@workstation config-ansible]$ cat ansible.cfg
[defaults]
inventory=inventory
remote_user=devops
```

- 2.2. Use the `ansible-navigator config` command.

```
[student@workstation config-ansible]$ ansible-navigator config \
> --eei hub.lab.example.com/ee-supported-rhel8 --pp missing
```

- 2.3. Search for the `Default host list` option again. Type :f `host` and then press Enter.

Chapter 4 | Working with Ansible Configuration Settings

Name	Default	Source	Current
0 Default host list	False	/home/student/config-a['/home/student/config-	
1 Display skipped hosts	True	default	True
...output omitted...			

Notice that line 0 in your terminal output is yellow and the value in the Default column is False, indicating that it is not a default setting. This time the Default host list option has the following displayed values:

Field	Value
Default	False
Source	/home/student/config-ansible/ansible.cfg
Current	['/home/student/config-ansible/inventory']

- 2.4. Press 0 to view more details for the Default host list option in the custom configuration.

```
Default host list (current: ['/home/student/config-ansible/inventory']) ...
0|---
1|current_config_file: /home/student/config-ansible/ansible.cfg
2|current_value:
3|- /home/student/config-ansible/inventory
4|default: false
5|default_value: /etc/ansible/hosts
6|description: Comma separated list of Ansible inventory sources
7|env:
8|- name: ANSIBLE_INVENTORY
9|expand_relative_paths: true
10|ini:
11|- key: inventory
12| section: defaults
13|name: Default host list
14|option: DEFAULT_HOST_LIST
15|source: /home/student/config-ansible/ansible.cfg
16|type: pathlist
17|via: /home/student/config-ansible/ansible.cfg
18|yaml:
19| key: defaults.inventory
```

Exit from the ansible-navigator command by typing :q and then pressing Enter.

- 3. Use automation content navigator to search for a specific configuration setting, determine if the setting is customized, and discover the source of any customizations.

- 3.1. Use the ansible-navigator config command.

```
[student@workstation config-ansible]$ ansible-navigator config \
> --eei hub.lab.example.com/ee-supported-rhel8 --pp missing
```

Chapter 4 | Working with Ansible Configuration Settings

- 3.2. Search for the `Default ask pass` setting. Type :f ask and then press Enter.

Name	Default	Source	Current
0 Default ask pass	True	default	False
1 Default ask vault pass	True	default	False
<i>...output omitted...</i>			

Notice that line 0 in your terminal output is green and the value in the `Default` column is `True`. This means that the displayed values are the default settings for the `Default ask pass` option.

Type :q and then press Enter to exit from the `ansible-navigator` command.

- 3.3. In the `ansible.cfg` file, add the `ask_pass=true` line to the `[defaults]` section. The resulting `ansible.cfg` file has the following content.

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

- 3.4. Use the `ansible-navigator config` command again and search for the `Default ask pass` setting. Type :f ask and then press Enter.

Name	Default	Source	Current
0 Default ask pass	False	/home/student/config-ansible/a	True
1 Default ask vault pass	True	default	False
<i>...output omitted...</i>			

Notice that line 0 in your terminal output is yellow and the value in the `Default` column is `False`. The displayed values correspond to the custom configuration for the `Default ask pass` option.

Exit the `ansible-navigator` command by pressing ESC.

- 3.5. Change the value of the `ask_pass` setting in the `ansible.cfg` file to `false`. The resulting `ansible.cfg` file should read as follows:

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

- 3.6. Use the `ansible-navigator config` command again and search for the `Default ask pass` setting. Type :f ask and then press Enter.

Name	Default	Source	Current
0 Default ask pass	False	/home/student/config-ansible/a	False
1 Default ask vault pass	True	default	False
<i>...output omitted...</i>			

The `Default ask pass` setting has a value of `False` if it is not overridden. Although this value matches the value configured in your `ansible.cfg` file, line 0 in your terminal output remains yellow and the value in the `Default` column is `False`. The `ansible.cfg` file in your current directory overrides the `Default ask pass`

setting, even if the file uses the same value as the default value for the setting. The displayed values correspond to the custom configuration for the `Default ask pass` option:

Field	Value
Default	<code>False</code>
Source	<code>/home/student/config-ansible/ansible.cfg</code>
Current	<code>False</code>

Exit the `ansible-navigator` command by pressing `ESC`.

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish config-ansible
```

Configuring Automation Content Navigator

Objectives

- Change configuration settings for automation content navigator with its configuration file, and determine where the configuration file is located.

Format of the Settings File

You can create a configuration file (or *settings* file) for `ansible-navigator` to override the default values of its configuration settings. A configuration file is useful, for example, if you want to use a different automation execution environment from the default one, and you do not want to enter the correct `--eei` option every time you run the `ansible-navigator` command.

- The settings file can be in JSON or YAML format.
- For settings in JSON format, the extension must be `.json`.
- For settings in YAML format, the extension must be `.yml` or `.yaml`.

This course focuses on the YAML-formatted configuration file.

Locating the Settings File

Automation content navigator looks for a settings file in the following order and uses the first file that it finds:

- If the `ANSIBLE_NAVIGATOR_CONFIG` environment variable is set, then use the configuration file at the location the variable specifies.
- An `ansible-navigator.yml` file in your current Ansible project directory.
- A `~/.ansible-navigator.yml` file (in your home directory). Notice that it has a "dot" at the start of its file name.

Each project can have its own automation content navigator settings file. Common configuration parameters can be names of automation execution environment images required for the project, Ansible configuration parameters such as forks, log levels, artifact creation, and so on.



Important

The project directory and home directory can only contain one settings file each. If there is an `ansible-navigator.yml` file and an `ansible-navigator.yaml` or `ansible-navigator.json` file in the same directory, it results in an error.

Selecting a Settings File to Use

If you have an `ansible-navigator.yml` file in your Ansible project directory, it is stored in version control with the rest of your project, and overrides any configuration file in the user's home directory. This also means that other users of that project use that same configuration file.

If you have a `~/.ansible-navigator.yml` file, it is used only if there is no other configuration file available. You can use it to specify default settings to use if the Ansible project you are working with has no preferred configuration settings.

Chapter 4 | Working with Ansible Configuration Settings

You should only specify an automation content navigator configuration file with the `ANSIBLE_NAVIGATOR_CONFIG` environment variable if you want to override all other configuration files.

Generating a Settings File

You can use the `ansible-navigator settings` command to generate settings files. If you run that command with the `--sample` option, then the command outputs a sample `ansible-navigator` configuration file in YAML format. If you run that command with the `--effective` option instead, then the command outputs a configuration file that matches the current effective configuration for `ansible-navigator`. In either case, you could redirect the output of the command into a file to save it, edit it, and use it.

The `--sample` option generates output where most lines begin with a number sign (#), indicating that they are commented out. This output also contains comments that describe the settings. Many comments include samples and display valid setting values. If you want to use a commented-out line, then to preserve the correct indentation of the YAML file, you must remove the number sign at the beginning of the line *and* one blank space. Removing the additional space conforms to the best practice of using two spaces for indentation and ensures that the uncommented lines match the existing indentation of the `logging` top-level setting.

The `--effective` option generates more condensed output and does not include any comments. Many options that you can pass to the command, such as the `--eei` and `--pp` options are configured in the output. When running this command, the `mode` key always has a value of `stdout`. If you do not need this value, then you can change the value after generating the sample file. You can delete any of the lines that you do not want to explicitly configure, but if you remove a parent key, then you must remove all its child keys. Because the command correctly indents all the output, this method generates a working configuration that you can immediately use.



Important

If you redirect the output of the `ansible-navigator settings` command into a file named `ansible-navigator.yml` in the current directory, then the `ansible-navigator` command attempts to use the configuration file and ultimately fails. Instead, redirect the command output to a file in a different directory, such as to the `/tmp/ansible-navigator.yml` file, or to a file with a different name in the current directory, such as `sample.yml`.

After the command completes, you can copy or move the generated file to the desired directory with the correct name.

```
[user@host ~]$ ansible-navigator settings --effective --pp missing \
> --eei ee-supported-rhel8 > /tmp/ansible-navigator.yml
```

The generated file starts with an `ansible-navigator` key, which contains subkeys representing the top-level configuration categories and settings. Top-level settings include `ansible` for Ansible-specific settings, and `execution-environment` for the automation execution environment settings that `ansible-navigator` uses.

```
...
ansible-navigator:
...output omitted...
execution-environment:
```

```
container-engine: podman
enabled: true
image: ee-supported-rhel8:latest
pull:
  policy: missing
...output omitted...
```

Setting a Default Automation Execution Environment

Use the `execution-environment` key to set default values for a number of the options that `ansible-navigator` accepts to control automation execution environments.

- `image` specifies the container image to use for the automation execution environment, and sets the default for the `--execution-environment-image` (or `--eei`) option.
- `pull` specifies when and how to pull the container image. Configuring the `policy` key is the equivalent to using the `--pull-policy` (or `--pp`) option. You can also pass additional arguments, such as disabling TLS verification.
- `enabled` specifies whether to use an automation execution environment or not and defaults to `True`.

As an example, consider the following `ansible-navigator` command:

```
[user@host ~]$ ansible-navigator run site.yml --eei ee-29-rhel8 --pp always
```

You can eliminate the `--eei` and `--pp` options if you create an `ansible-navigator.yml` file, which contains the following content, in the same directory as the `site.yml` file:

```
---
ansible-navigator:
  execution-environment:
    image: ee-29-rhel8:latest
    pull:
      policy: always
```

That would convert the preceding command to the following:

```
[user@host ~]$ ansible-navigator run site.yml
```

Specify the other advanced options to adjust the exact behavior of the automation execution environment.

Setting the Default Mode to Standard Output

The `mode` key takes the same options as the `--mode (-m)` option, and defaults to `interactive` mode. If you prefer to work in `stdout` mode, you can specify this with an `ansible-navigator.yml` file that has the following content:

```
---
ansible-navigator:
  mode: stdout
```

Chapter 4 | Working with Ansible Configuration Settings

With this configuration, you need to specify the `-m interactive` option explicitly if you want to run in interactive mode.

Disabling Playbook Artifacts

The basic design of `ansible-navigator` assumes that you cannot provide interactive input to the playbook while it is running. This makes sense because that is also true when you run a playbook in automation controller.

Automation content navigator also records *playbook artifact* files for each run of the playbook. These files record information about the playbook run and can be used to review the results of the run when it completes, reviewed to troubleshoot issues, or kept for compliance purposes. You can review the contents of these files with the `ansible-navigator replay filename` command. However, the contents of these files might contain sensitive information about the playbook run, especially if you are providing authentication-related input to the playbook.

If you frequently run playbooks with interactive password prompts or playbooks that use modules that prompt you for input, you can disable the generation of playbook artifacts with a configuration file such as the following example. Alternatively, you can temporarily disable playbook artifacts by running the `ansible-navigator run` command with the `--pae false` option.

```
---
```

```
ansible-navigator:
  playbook-artifact:
    enable: false
```

Overview of an Example Settings File

The following `ansible-navigator.yml` file configures some common settings:

```
---
```

```
ansible-navigator:
  ansible:
    config:
      path: ./ansible.cfg
```

```
  editor: ①
    command: /bin/emacs
```

```
  execution-environment: ②
    image: ee-supported-rhel8:latest
    pull:
      policy: missing
```

```
  playbook-artifact: ③
    enable: false
```

- ① You can configure `ansible-navigator` to use the text editor of your choice.
- ② By default, the `ansible-navigator` command looks for the `registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest` image. You can configure it to use a specific automation execution environment by defining the `execution-environment` section in the settings file.

- ③ To enable the `ansible-navigator` command to prompt for passwords, you can disable playbook artifacts.

More information about the available options and how to use them is available in the references at the end of this section.



References

Automation Content Navigator Subcommand Settings

https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.1/html-single/ansible_navigator_creator_guide/index#assembly-settings-navigator_ansible-navigator/

ansible-navigator Settings

<https://ansible.readthedocs.io/projects/navigator/settings/>

► Guided Exercise

Configuring Automation Content Navigator

Change settings for automation content navigator by creating and editing its configuration file.

Outcomes

- Configure automation content navigator by using an `ansible-navigator.yml` file.
- Identify the sequence of locations that automation content navigator checks for its configuration file.
- Configure automation content navigator to prompt the user for an Ansible Vault password.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/config-navigator` directory.

```
[student@workstation ~]$ lab start config-navigator
```

Instructions

- 1. Review the Ansible configuration files, inventories, and the playbook in the `/home/student/config-navigator` directory. Modify the Ansible configuration files to use the `secret-pass` vault password file.
- 1.1. Change to the `/home/student/config-navigator/` directory.

```
[student@workstation ~]$ cd ~/config-navigator/
```

- 1.2. Display the contents of the `ansible1.cfg` and `inventory1` files. Notice how the `ansible1.cfg` configuration file sets the inventory location to `inventory1`.

```
[defaults]
inventory = inventory1
```

The `inventory1` file defines the `devservers` host group with `serverb.lab.example.com` as the only managed host.

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

Chapter 4 | Working with Ansible Configuration Settings

- 1.3. Display the contents of the `ansible2.cfg` and `inventory2` files. Notice how the `ansible2.cfg` configuration file sets the inventory location to `inventory2`.

```
[defaults]
inventory = inventory2
```

The `inventory2` file defines the `devservers` host group with `servera.lab.example.com` as the only managed host.

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

- 1.4. Display the contents of the `create_users.yml` playbook.

```
---
- name: create user accounts in devservers
  hosts: devservers
  become: true
  remote_user: devops
  gather_facts: false
  vars_files:
    - secret.yml

  tasks:
    - name: Creating user from secret.yml
      ansible.builtin.user:
        name: "{{ username }}"
        password: "{{ pwhash }}"
```

This playbook uses the variables defined in the `secret.yml` encrypted file.

- 1.5. Use the `ansible-vault view` command to display the contents of the `secret.yml` file using the `secret-pass` vault password file to decrypt the `secret.yml` file.

```
[student@workstation config-navigator]$ cat secret-pass
redhat

[student@workstation config-navigator]$ ansible-vault view secret.yml \
> --vault-password-file=secret-pass
username: ansibleuser
pwhash: $6$j ... xhP1
```

- 1.6. In the `ansible1.cfg` file, specify `secret-pass` as the vault password file. The `ansible1.cfg` file should now consist of the following content:

```
[defaults]
inventory = inventory1
vault_password_file = secret-pass
```

- 2. Create the `ansible-navigator.yml` settings file and configure it so that `ansible-navigator` uses the `ansible1.cfg` Ansible configuration file, the

Chapter 4 | Working with Ansible Configuration Settings

`hub.lab.example.com/ee-supported-rhel8:latest` automation execution environment, and standard output mode.

- 2.1. Use the `ansible-navigator settings --effective` command to generate the `sample.yml` settings file. You can use this file as a starting point to configure automation content navigator.

```
[student@workstation config-navigator]$ ansible-navigator settings --effective \
> -m stdout --eei hub.lab.example.com/ee-supported-rhel8:latest \
> --pp never > sample.yml
```

Important

When you redirect the output of the `ansible-navigator settings --effective` command to a file in the current working directory, you must redirect it to a file with a name other than `ansible-navigator.yml` or the command fails.

- 2.2. Rename the `sample.yml` file to `ansible-navigator.yml`.

```
[student@workstation config-navigator]$ mv -v sample.yml ansible-navigator.yml
renamed 'sample.yml' -> 'ansible-navigator.yml'
```

- 2.3. Open the `ansible-navigator.yml` settings file in a text editor, and set the `ANSIBLE_CONFIG` environment variable to `ansible1.cfg` by editing the `environment-variables` subsection of the `execution-environment` section:

```
execution-environment:
  container-engine: podman
  enabled: true
  environment-variables:
    set:
      ANSIBLE_CONFIG: /home/student/config-navigator/ansible1.cfg
```

- 2.4. Examine the `ansible-navigator.yml` settings file and notice that the `ansible-navigator` command set the `image` key using the value you specified with the `--eei` option.

```
image: hub.lab.example.com/ee-supported-rhel8:latest
```

- 2.5. Examine the `ansible-navigator.yml` settings file and notice that the `ansible-navigator` command set the `policy` key using the value you specified with the `--pp` option.

```
pull:
  policy: never
```

The updated `ansible-navigator.yml` file now contains the following content:

```
...
ansible-navigator:
...output omitted...
  execution-environment:
```

Chapter 4 | Working with Ansible Configuration Settings

```

container-engine: podman
enabled: true
environment-variables:
  set:
    ANSIBLE_CONFIG: /home/student/config-navigator/ansible1.cfg
image: hub.lab.example.com/ee-supported-rhel8:latest
pull:
  policy: never
...output omitted...

```

**Important**

Be careful with indentation of the lines in the `ansible-navigator.yml` file. The preceding example shows correct indentation for each line.

- ▶ 3. Create a similar `~/.ansible-navigator.yml` settings file, but set the `ANSIBLE_CONFIG` environment variable to `ansible2.cfg`.

Run the `create_users.yml` playbook and confirm that the configuration file in the directory for its project has precedence over the `~/.ansible-navigator.yml` file in your home directory.

- 3.1. Copy the existing automation content navigator settings file to `~/.ansible-navigator.yml`.

```
[student@workstation config-navigator]$ cp ansible-navigator.yml \
> ~/.ansible-navigator.yml
```

- 3.2. Edit the new `~/.ansible-navigator.yml` settings file. Set the `ANSIBLE_CONFIG` environment variable to `ansible2.cfg`. After editing and removing all comments, the `~/.ansible-navigator.yml` file contains the following content:

```

---
ansible-navigator:
...output omitted...
execution-environment:
  container-engine: podman
  enabled: true
  environment-variables:
    set:
      ANSIBLE_CONFIG: /home/student/config-navigator/ansible2.cfg
  image: hub.lab.example.com/ee-supported-rhel8:latest
  pull:
    policy: never
...output omitted...

```

- 3.3. Run the `create_users.yml` playbook by using the `ansible-navigator` command.

```
[student@workstation config-navigator]$ ansible-navigator run create_users.yml
PLAY [create user accounts in devservers] ****
```

```
TASK [Creating user from secret.yml] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com : ok=1    changed=1    unreachable=0    failed=0
skipped=0    rescued=0   ignored=0
```

**Note**

When running commands, you do not need to specify any settings that are already defined in the automation content navigator settings file.

The fact that the playbook ran against `serverb` implies that the `./ansible-navigator.yml` file has precedence over `~/.ansible-navigator.yml` file.

- ▶ 4. Verify that you can use `~/.ansible-navigator.yml` as the settings file for automation content navigator when there is no configuration file in the local project directory, or when you specify it as the value of the `ANSIBLE_NAVIGATOR_CONFIG` environment variable.

- 4.1. Back up the `ansible-navigator.yml` file by moving it to `backup-an.yml`. Run the `create_users.yml` playbook by using the `ansible-navigator` command with the `--pae false` and `--ask-vault-pass` options.

This time, the playbook ran against the `servera` server, because there was no settings file in the project directory and automation content navigator found the settings file in your home directory.

```
[student@workstation config-navigator]$ mv ansible-navigator.yml backup-an.yml
[student@workstation config-navigator]$ ansible-navigator run create_users.yml \
> --pae false --ask-vault-pass
Vault password: redhat

PLAY [create user accounts in devservers] ****

TASK [Creating user from secret.yml] ****
changed: [servera.lab.example.com]

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

**Note**

The `--ask-vault-pass` option is used because the `vault_password_file` option is not specified in the `ansible2.cfg` file. This option also requires the use of the `--pae false` option to temporarily disable playbook artifacts.

- 4.2. Restore the local automation content navigator settings file and set the `ANSIBLE_NAVIGATOR_CONFIG` environment variable to `~/.ansible-navigator.yml`.

Chapter 4 | Working with Ansible Configuration Settings

```
[student@workstation config-navigator]$ mv backup-an.yml ansible-navigator.yml
[student@workstation config-navigator]$ export \
> ANSIBLE_NAVIGATOR_CONFIG=~/ansible-navigator.yml
```

- 4.3. Run the `create_users.yml` playbook by using the `ansible-navigator` command. As expected, the playbook ran against the `servera` server, even when the local `./ansible-navigator.yml` settings file was present, because the `ANSIBLE_NAVIGATOR_CONFIG` environment variable takes precedence.

```
[student@workstation config-navigator]$ ansible-navigator run create_users.yml \
> --pae false --ask-vault-pass
Vault password: redhat

PLAY [create user accounts in devservers] ****
TASK [Creating user from secret.yml] ****
ok: [servera.lab.example.com]

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

- 4.4. Unset the `ANSIBLE_NAVIGATOR_CONFIG` environment variable so that the project's `./ansible-navigator.yml` settings file is used for the remaining exercises.

```
[student@workstation config-navigator]$ unset ANSIBLE_NAVIGATOR_CONFIG
```

- ▶ 5. You might prefer to enter the password to decrypt the `secret.yml` file as user input when you run the `create_users.yml` playbook instead of using the `--pae false` option every time. Prepare the Ansible and automation content navigator settings files to prompt for the Ansible Vault password.
- 5.1. Remove the line for `vault_password_file` in the `ansible1.cfg` file. The `ansible1.cfg` file should now consist of the following content:

```
[defaults]
inventory = inventory1
```

- 5.2. In the `ansible-navigator.yml` file, disable creating playbook artifacts by setting the `enable` key to `false`.

```
---
ansible-navigator:
...output omitted...
execution-environment:
  container-engine: podman
  enabled: true
  environment-variables:
    set:
      ANSIBLE_CONFIG: /home/student/config-navigator/ansible1.cfg
  image: hub.lab.example.com/ee-supported-rhel8:latest
```

Chapter 4 | Working with Ansible Configuration Settings

```
pull:  
  policy: never  
...output omitted...  
logging:  
  append: true  
  file: /home/student/config-navigator/ansible-navigator.log  
  level: warning  
mode: stdout  
playbook-artifact:  
  enable: false  
  save-as: '{playbook_dir}/{playbook_name}-artifact-{time_stamp}.json'  
...output omitted...
```



Important

Disabling playbook artifact creation enables you to run the playbook interactively. Running interactively is useful when you want to use options such as `--ask-vault-pass` on the command line.

However, disabling artifact creation removes the ability to run `ansible-navigator replay`.

- 5.3. Run the `create_users.yml` playbook using the `ansible-navigator` command. This time, specify the `--ask-vault-pass` option to prompt for the Ansible Vault password.

```
[student@workstation config-navigator]$ ansible-navigator run create_users.yml \  
> --ask-vault-pass  
Vault password: redhat  
  
PLAY [create user accounts in devservers] ****  
  
TASK [Creating user from secret.yml] ****  
ok: [serverb.lab.example.com]  
  
PLAY RECAP ****  
serverb.lab.example.com : ok=1    changed=0    unreachable=0    failed=0  
  skipped=0   rescued=0    ignored=0
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish config-navigator
```

▶ Lab

Working with Ansible Configuration Settings

Configure automation content navigator to use a specific image by default and then use it to examine the current Ansible configuration.

Outcomes

- Configure automation content navigator by using an `ansible-navigator.yml` file.
- Configure Ansible by using an `ansible.cfg` file.
- Use the `ansible-navigator` command to review your current Ansible configuration.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/config-review` directory.

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

Instructions

- Change to the `/home/student/config-review` directory for your Ansible project and configure the automation content navigator settings file to use the `hub.lab.example.com/ee-supported-rhel8:latest` image by default, and set the pull policy to `missing`.
- Because the `install-web.yml` playbook requires prompting for the privilege escalation password, you must disable creating playbook artifacts. Configure the automation content navigator settings file to disable creating playbook artifacts.
- Use automation content navigator to determine how to specify the maximum number of forks Ansible uses to execute tasks on target hosts. Identify the name of the variable and to which section the variable belongs. Modify the `~/config-review/ansible.cfg` Ansible configuration file to set the maximum number of forks to 15. Use automation content navigator to verify the changes.
- Use the `ansible-navigator` command with the `-m stdout` and `--ask-become-pass` options to run the `install-web.yml` playbook. When prompted, enter `student` for the privilege escalation password. Verify that the playbook runs successfully.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Working with Ansible Configuration Settings

Configure automation content navigator to use a specific image by default and then use it to examine the current Ansible configuration.

Outcomes

- Configure automation content navigator by using an `ansible-navigator.yml` file.
- Configure Ansible by using an `ansible.cfg` file.
- Use the `ansible-navigator` command to review your current Ansible configuration.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/config-review` directory.

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

Instructions

- Change to the `/home/student/config-review` directory for your Ansible project and configure the automation content navigator settings file to use the `hub.lab.example.com/ee-supported-rhel8:latest` image by default, and set the `pull` policy to `missing`.
 - Change to the `/home/student/config-review` directory.

```
[student@workstation ~]$ cd ~/config-review
```

- Edit the `~/config-review/ansible-navigator.yml` automation content navigator settings file. Change the value of the `image` directive to `hub.lab.example.com/ee-supported-rhel8:latest`, and in the `pull` section change the value of the `policy` directive to `missing`.

Use the following `grep` command to display lines that do not begin with a number sign. The output of the `grep` command displays the expected content after you modify the `ansible-navigator.yml` file:

```
[student@workstation config-review]$ grep -v '^#' ansible-navigator.yml
---
ansible-navigator:
  execution-environment:
    enabled: True
```

Chapter 4 | Working with Ansible Configuration Settings

```
image: hub.lab.example.com/ee-supported-rhel8:latest
pull:
  policy: missing
logging:
  level: critical
playbook-artifact:
  enable: True
```

2. Because the `install-web.yml` playbook requires prompting for the privilege escalation password, you must disable creating playbook artifacts. Configure the automation content navigator settings file to disable creating playbook artifacts.

- 2.1. Edit the `~/config-review/ansible-navigator.yml` automation content navigator settings file. In the `playbook-artifact` section, change the value of the `enable` directive to `False`.

The output of the `grep` command displays the expected content after you modify the `ansible-navigator.yml` file:

```
[student@workstation config-review]$ grep -v '^#' ansible-navigator.yml
---
ansible-navigator:
  execution-environment:
    enabled: True
    image: hub.lab.example.com/ee-supported-rhel8:latest
    pull:
      policy: missing
  logging:
    level: critical
  playbook-artifact:
    enable: False
```

3. Use automation content navigator to determine how to specify the maximum number of forks Ansible uses to execute tasks on target hosts. Identify the name of the variable and to which section the variable belongs. Modify the `~/config-review/ansible.cfg` Ansible configuration file to set the maximum number of forks to 15. Use automation content navigator to verify the changes.

- 3.1. Run the `ansible-navigator config` command with no arguments.

```
[student@workstation config-review]$ ansible-navigator config

  Name          Default Source Current
0|Action warnings        True   default True
1|Agnostic become prompt True   default True
2|Allow world readable tmpfiles True   default False
3|Ansible connection path True   default None
...output omitted...
```

- 3.2. Type `:f forks` and then press `Enter` to filter the configuration settings for the `forks` pattern.

Name	Default	Source	Current
0 Default forks	True	default	5

Chapter 4 | Working with Ansible Configuration Settings

- 3.3. Press 0 to inspect the `Default forks` configuration.

```
Default forks (current/default: 5)
0|---
1|current_config_file: /home/student/config-review/ansible.cfg
2|current_value: 5
3|default: true
4|default_value: 5
5|description: Maximum number of forks Ansible will use to execute tasks on target
6|env:
7|- name: ANSIBLE_FORKS
8|ini:
9|- key: forks ①
10| section: defaults ②
11|name: Default forks
12|option: DEFAULT_FORKS
13|source: default
14|type: integer
15|via: default
```

- ① The `forks` key matches the name of the variable in the `ansible.cfg` file.
- ② The `defaults` section matches the name of the section that the variable belongs to in the `ansible.cfg` file.

Exit from `ansible-navigator` by pressing ESC twice.

- 3.4. Edit the `~/config-review/ansible.cfg` Ansible settings file and change the `forks` setting to 15.

The following `grep` command does not display lines that begin with a number sign, blank lines, and lines that begin with a semicolon. The output of the `grep` command displays the expected content after you modify the `ansible.cfg` file:

```
[student@workstation config-review]$ grep -Ev '^#|^$|^;' ansible.cfg
[defaults]
forks=15
inventory = inventory
[privilege_escalation]
...output omitted...
```

- 3.5. To verify the setting, run the `ansible-navigator config` command with no arguments.

```
[student@workstation config-review]$ ansible-navigator config

      Name          Default Source Current
0|Action warnings        True   default True
1|Agnostic become prompt True   default True
2|Allow world readable tmpfiles True   default False
3|Ansible connection path True   default None
...output omitted...
```

- 3.6. Type `:f forks` and then press Enter to filter the configuration settings for the `forks` pattern.

Chapter 4 | Working with Ansible Configuration Settings

Name	Default	Source	Current
0 Default forks	False	/home/student/config-review/ansible.cfg	15

Exit from ansible-navigator by pressing ESC.

4. Use the `ansible-navigator` command with the `-m stdout` and `--ask-become-pass` options to run the `install-web.yml` playbook. When prompted, enter student for the privilege escalation password. Verify that the playbook runs successfully.
 - 4.1. Use the `ansible-navigator` command with the `-m stdout` and `--ask-become-pass` options to run the `install-web.yml` playbook.

```
[student@workstation config-review]$ ansible-navigator run install-web.yml \
> -m stdout --ask-become-pass
BECOME password: student

PLAY [Install web server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install httpd package] ****
changed: [servera.lab.example.com]

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

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- The `ansible-navigator config` command helps you explore and analyze the current configuration used by the `ansible-navigator run` command.
- Automation content navigator can use only `ansible.cfg` files that it can see from inside the automation execution environment. These files include the `/etc/ansible/ansible.cfg` file in the execution environment container and the `ansible.cfg` file in your project directory.
- You can configure automation content navigator settings by using the file defined by the `ANSIBLE_NAVIGATOR_CONFIG` environment variable, the `ansible-navigator.yml` configuration file in your current directory, or the `~/.ansible-navigator.yml` configuration file in your home directory.
- The `ansible-navigator settings --sample` command generates a sample `ansible-navigator.yml` configuration file.
- The `ansible-navigator settings --effective` command generates a configuration file that reflects your current effective configuration, including options specified on the command line, values of environment variables, and defaults.

Chapter 5

Managing Inventories

Goal

Manage inventories by using advanced features of Ansible.

Objectives

- Describe what dynamic inventories are, and install and use an existing script or plug-in as an Ansible dynamic inventory source.
- Write static inventory files in YAML format.
- Structure host and group variables by using multiple files per host or group, and use special variables to override the host, port, or remote user that Ansible uses for a specific host.

Sections

- Managing Dynamic Inventories (and Guided Exercise)
- Writing YAML Inventory Files (and Guided Exercise)
- Managing Inventory Variables (and Guided Exercise)

Lab

- Managing Inventories

Managing Dynamic Inventories

Objectives

- Describe what dynamic inventories are, and install and use an existing script or plug-in as an Ansible dynamic inventory source.

Generating Inventories Dynamically

The static inventory files you have worked with so far are easy to write and are convenient for managing small infrastructures. However, these static lists require manual administration to keep them up-to-date. This can be inconvenient or challenging, especially when an organization wants to run the playbooks against hosts that are dynamically created in a virtual or cloud computing environment.

In that scenario, it is useful to use a dynamically generated inventory.

Dynamic inventories are scripts which, when run, dynamically determine which hosts and host groups should be in the inventory, based on information from an external source. These can include the API for cloud providers, Cobbler, LDAP directories, or other third-party configuration management database (CMDB) software. Using a dynamically generated inventory is a recommended practice in a large and rapidly changing IT environment, where systems are frequently deployed, tested, and then removed.

Ansible supports two types of dynamically generated inventories:

- Inventory plug-ins
- Inventory scripts

Inventory Plug-ins

An inventory plug-in is a piece of Python code that generates an inventory object from a dynamic source. Ansible ships with inventory plug-ins for a number of external inventory sources, including:

- Amazon Web Services EC2
- Google Compute Engine
- Microsoft Azure Resource Manager
- Red Hat OpenStack Platform
- VMware vCenter
- Red Hat Satellite 6
- Red Hat Virtualization
- Red Hat OpenShift

Ansible provides inventory plug-ins through Ansible Content Collections. Third-party content collections provide additional inventory plug-ins for specific products.

Developing inventory plug-ins is beyond the scope of this course.

Using Inventory Plug-ins

For most inventory plug-ins you must prepare a configuration file in YAML format. This configuration file usually provides the connection parameters that the plug-in requires to access the external source.

For example, the following `satellite.yml` configuration file provides the connection parameters to access a Red Hat Satellite Server installation.

```
plugin: redhat.satellite.foreman ①
url: https://satellite.example.com ②
user: ansibleinventory
password: Sup3r53cr3t
host_filters: 'organization="Development"' ③
```

- ① The required `plugin` keyword indicates which plug-in to use. You must provide its fully qualified collection name (FQCN).
- ② The connection parameters are specific to the plug-in. This example provides the Red Hat Satellite Server URL, the username, and the password to use to access the API endpoint.
- ③ Some plug-ins provide mechanisms to filter the list of hosts. This example uses the `host_filters` parameter to only retrieve the hosts in the Development Red Hat Satellite organization.

Every inventory plug-in has documentation with examples. You can access that documentation with the `ansible-navigator doc` command. Use the `--type inventory` (or `-t inventory`) option to look up documentation for inventory plug-ins.

Use the `--list` (or `-l`) option to list all the inventory plug-ins available in your automation execution environment:

```
[user@host ~]$ ansible-navigator doc --mode stdout --type inventory --list
...output omitted...
ini                      Uses an Ansible INI file as inventory source
script                   Executes an inventory script that returns JSON
redhat.satellite.foreman Foreman inventory source
toml                     Uses a specific TOML file as an inventory source
yaml                     Uses a specific YAML file as an inventory source
```

Notice that the command only lists the plug-ins from installed Ansible Content Collections. By default, those are the content collections installed in the execution environment, or the collections in the directories you define in your `ansible.cfg` configuration file by setting the `collections_paths` directive.

To access the documentation for a specific plug-in, use the `ansible-navigator doc` command and provide the FQCN for the plug-in as an argument:

```
[user@host ~]$ ansible-navigator doc --mode stdout --type inventory \
> redhat.satellite.foreman
```

To run a playbook that targets hosts in that dynamic inventory, use the `ansible-navigator run` command with the `--inventory` (or `-i`) option to provide the plug-in configuration file:

```
[user@host ~]$ ansible-navigator run --inventory ./satellite.yml my_playbook.yml
```

Remember that you also use that same option for static inventory files. Because Ansible uses the `--inventory` option for different inventory formats, it parses the file you provide in the following order by default:

1. If the file is executable, then Ansible uses it as an inventory script, as described in the following section.
2. Otherwise, Ansible tries to parse the given file as a configuration file for inventory plug-ins.
3. If that fails, then Ansible uses the file as a static inventory file.

You can control that order by setting the `enable_plugins` directive under the `inventory` section in your `ansible.cfg` configuration file.

Developing Inventory Scripts

An inventory script is a program that collects information from an external source and returns the inventory in JSON format. You can write the custom program in any programming language as long as it returns inventory information in JSON format.



Important

Red Hat recommends developing inventory plug-ins instead of inventory scripts.

Use inventory scripts when you already have legacy scripts that you want to reuse or when you do not want to use Python for your development.

The `ansible-navigator inventory` command is a helpful tool for learning how to write Ansible inventories in JSON format. This command takes an inventory file and returns it in JSON format.

To display the contents of an inventory file in JSON format, use the `--list` option. You can also add the `--inventory` (or `-i`) option to specify the location of the inventory file to process instead of the default inventory set by the current Ansible configuration.

The following example runs the `ansible-navigator inventory` command to process an inventory file in INI format and output it in JSON format.

```
[user@host ~]$ cat inventory
monitor.example.com

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

[databases]
db1.example.com
db2.example.com

[user@host ~]$ ansible-navigator inventory --mode stdout -i inventory --list
{
    "_meta": {
        "hostvars": {}
    }
}
```

```

},
"all": {
    "children": [
        "databases",
        "ungrouped",
        "webservers"
    ]
},
"databases": {
    "hosts": [
        "db1.example.com",
        "db2.example.com"
    ]
},
"ungrouped": {
    "hosts": [
        "monitor.example.com"
    ]
},
"webservers": {
    "hosts": [
        "web1.example.com",
        "web2.example.com"
    ]
}
}
}

```

To develop your own dynamic inventory script, refer to [Inventory scripts](#) [https://docs.ansible.com/ansible/6/dev_guide/developing_inventory.html#developing-inventory-scripts] in the *Ansible Developer Guide*.

Start the script with an appropriate interpreter line (for example, `#!/usr/bin/python`) and make sure that it is executable so that Ansible can run it.

The script must support the `--list` and `--host managed-host` options.

When called with the `--list` option, the script must print a JSON dictionary of all the hosts and groups in the inventory.

In its simplest form, a group is a list of managed hosts. In the following example, the `webservers` group includes the `web1.example.com` and `web2.example.com` hosts. The `databases` group includes the `db1.example.com` and `db2.example.com` hosts.

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

Alternatively, each group value can be a JSON dictionary containing a list of managed hosts, a list of child groups, and group variables that might be set. The following example shows the JSON output for a more complex dynamic inventory. The `boston` group has two child groups, `backup` and `ipa`, three managed hosts, and a group variable set (`example_host: false`).

Chapter 5 | Managing Inventories

```
{  
    "boston": {  
        "children": [  
            "backup",  
            "ipa"  
        ],  
        "vars": {  
            "example_host": false  
        },  
        "hosts": [  
            "server1.example.com",  
            "server2.example.com",  
            "server3.example.com"  
        ]  
    },  
    "backup": [  
        "server4.example.com"  
    ],  
    "ipa": [  
        "server5.example.com"  
    ]  
}
```

The script must also support the `--host managed-host` option. When you run the script with that option, it must print a JSON dictionary consisting of the variables associated with the host, or an empty JSON dictionary if there are no variables for that host.

```
[user@host ~]$ ./inventoryscript --host server5.example.com  
{  
    "ntpserver": "ntp.example.com",  
    "dnsserver": "dns.example.com"  
}
```



Note

For each host in the inventory, Ansible calls the script with the `--host` option to retrieve the host variables. When the inventory is very long, calling the script for each host can consume a lot of time and system resources.

To improve performance, you can directly provide the variables for all the hosts when the script is called with the `--list` option. To do so, the script must return a top-level element called `_meta`, which lists all the hosts and their variables. In that case, Ansible does not make the `--host` calls.

See the [Tuning the External Inventory Script](https://docs.ansible.com/ansible/6/dev_guide/developing_inventory.html#tuning-the-external-inventory-script) [https://docs.ansible.com/ansible/6/dev_guide/developing_inventory.html#tuning-the-external-inventory-script] documentation for more information.

Using Inventory Scripts

You use dynamic inventory scripts just like static inventory text files. Specify the location of the inventory script either in the `ansible.cfg` file or by using the `--inventory` (or `-i`) option.

Chapter 5 | Managing Inventories

If the inventory file is executable, then Ansible treats it as a dynamic inventory program and attempts to run it to generate the inventory. If the file is not executable, then Ansible treats it as a static inventory.

```
[user@host ~]$ ansible-navigator run --inventory ./inventoryscript my_playbook.yml
```

Managing Multiple Inventories

Ansible supports the use of multiple inventories in the same run. If the location of the inventory is a directory (whether set by the `-i` option, the value of the `inventory` parameter, or in some other way), then all inventory files included in the directory, either static or dynamic, are combined to determine the inventory. The executable files within that directory are used to retrieve dynamic inventories, and the other files are used as static inventories or configuration files for inventory plug-ins.

Inventory files should not depend on other inventory files or scripts in order to resolve. For example, if a static inventory file specifies that a particular group should be a child of another group, then it also needs to have a placeholder entry for that group, even if all members of that group come from the dynamic inventory. Consider the `cloud-east` group in the following example:

```
[cloud-east]  
  
[servers]  
test.example.com  
  
[servers:children]  
cloud-east
```

This ensures that no matter the order in which inventory files are parsed, they are all internally consistent.



Important

The order in which inventory files are parsed is not specified by the documentation. Currently, when multiple inventory files exist, they are parsed in alphabetical order. If one inventory source depends on information from another inventory source, then the order in which they are loaded determines if the inventory file works as expected or throws an error. Therefore, it is important to make sure that all files are self-consistent to avoid unexpected errors.

Ansible ignores files in an inventory directory if they end with certain suffixes. This can be controlled with the `inventory_ignore_extensions` directive in the Ansible configuration file. That directive defaults to `.pyc`, `.pyo`, `.swp`, `.bak`, `~`, `.rpm`, `.md`, `.txt`, `.rst`, `.orig`, `.ini`, `.cfg`, and `.retry`.



References

Ansible Documentation: Working with Dynamic Inventory

https://docs.ansible.com/ansible/6/user_guide/intro_dynamic_inventory.html

Ansible Documentation: Inventory Plugins

<https://docs.ansible.com/ansible/6/plugins/inventory.html>

Ansible Documentation: Developing Dynamic Inventory

https://docs.ansible.com/ansible/6/dev_guide/developing_inventory.html

► Guided Exercise

Managing Dynamic Inventories

Install custom scripts that dynamically generate a list of inventory hosts.

Outcomes

- Install and use existing dynamic inventory scripts.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command ensures that Ansible is installed on the workstation machine and also prepares the `/home/student/inventory-dynamic` working directory for this exercise.

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

Instructions

- 1. On the workstation machine, change to the `/home/student/inventory-dynamic` directory.

View the contents of the `ansible.cfg` Ansible configuration file. The configuration file sets the inventory location to `~/inventory-dynamic/inventory`.

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

- 2. Create the `/home/student/inventory-dynamic/inventory` directory.

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

- 3. Use the `tree` command to list the files in the current directory.

```
[student@workstation inventory-dynamic]$ tree  
. .  
└── ansible.cfg  
└── ansible-navigator.yml  
└── hosts ①  
└── inventory  
└── inventorya.py ②  
└── inventoryw.py ③
```

1 directory, 5 files

Chapter 5 | Managing Inventories

- ➊ The `hosts` static inventory file defines the `servers` group, which is a parent group of the `webservers` group.
 - ➋ The `inventorya.py` dynamic inventory script defines the `webservers` group, which includes the `servera.lab.example.com` host.
 - ➌ The `inventoryw.py` dynamic inventory script defines the `workstation.lab.example.com` host.
- ▶ 4. Move the Python scripts and the `hosts` file into the `inventory` directory.

```
[student@workstation inventory-dynamic]$ mv *.py hosts inventory/
```

- ▶ 5. Use the following `ansible-navigator inventory` command to list the hosts in the `webservers` group. Use the `inventorya.py` script as the inventory source and add the `--graph` option to display the result as a graph. The command fails with a permission error on the `inventorya.py` file.

```
[student@workstation inventory-dynamic]$ ansible-navigator inventory \
> --mode stdout -i inventory/inventorya.py --graph webservers
[WARNING]: * Failed to parse /home/student/inventory-
dynamic/inventory/inventorya.py with script plugin: problem running
/home/student/inventory-dynamic/inventory/inventorya.py --list ([Errno 13]
Permission denied: '/home/student/inventory-dynamic/inventory/inventorya.py')
[WARNING]: * Failed to parse /home/student/inventory-
dynamic/inventory/inventorya.py with ini plugin: /home/student/inventory-
dynamic/inventory/inventorya.py:3: Expected key=value host variable assignment,
got: subprocess
[WARNING]: Unable to parse /home/student/inventory-
dynamic/inventory/inventorya.py as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
usage: ansible-inventory [-h] [--version] [-v] [-i INVENTORY]
                           [--vault-id VAULT_IDS]
                           [--ask-vault-password | --vault-password-file
VAULT_PASSWORD_FILES]
                           [--playbook-dir BASEDIR] [-e EXTRA_VARS] [--list]
                           [--host HOST] [--graph] [-y] [--toml] [--vars]
                           [--export] [--output OUTPUT_FILE]
                           [host|group]
...output omitted...
```

Note, too, that Ansible tries first to process the file as an inventory script and then tries to parse it as a static inventory file in INI format, which also fails. Remember that Ansible tries different plug-ins in a specific order to process inventory files.

- ▶ 6. Inspect the current mode of the `inventorya.py` script. Change the mode of all Python scripts to be executable.

```
[student@workstation inventory-dynamic]$ ls -l inventory/inventorya.py
-rw-rw-r--. 1 student student 640 Oct 20 05:26 inventory/inventorya.py
[student@workstation inventory-dynamic]$ chmod 755 inventory/*.py
```

Chapter 5 | Managing Inventories

- ▶ 7. Run the `inventorya.py` script with the `--list` option. The script displays the hosts in the `webservers` group.

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

- ▶ 8. Run the `inventoryw.py` script with the `--list` option. The script displays the `workstation.lab.example.com` host.

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

- ▶ 9. Display the content of the `inventory/hosts` file. Notice that the file references the `webservers` group that the dynamic inventory defines.

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

- ▶ 10. Use the following `ansible-navigator inventory` command to list the hosts in the `webservers` group. Add the `--graph` option to display the result as a graph. The command fails with an error about the `webservers` group being undefined.

```
[student@workstation inventory-dynamic]$ ansible-navigator inventory \
> --mode stdout --graph webservers
[WARNING]: * Failed to parse /home/student/inventory-dynamic/inventory/hosts
with yaml plugin: We were unable to read either as JSON nor YAML, these are the
errors we got from each: JSON: Expecting value: line 1 column 2 (char 1)
Syntax Error while loading YAML.  found unexpected ':'  The error appears to
be in '/home/student/inventory-dynamic/inventory/hosts': line 1, column 9, but
may be elsewhere in the file depending on the exact syntax problem.  The
offending line appears to be:  [servers:children]           ^ here
[WARNING]: * Failed to parse /home/student/inventory-dynamic/inventory/hosts
with ini plugin: /home/student/inventory-dynamic/inventory/hosts:2: Section
[servers:children] includes undefined group: webservers
[WARNING]: Unable to parse /home/student/inventory-dynamic/inventory/hosts as
an inventory source
@webservers:
  |--servera.lab.example.com
```

**Important**

If the dynamic inventory script that provides the host group is named so that it sorts before the static inventory references it, then you do not see this error. However, if the host group is ever removed from the dynamic inventory, and you do not have a placeholder, then the static inventory references a missing host group and the error causes the parsing of the inventory to fail.

- ▶ 11. To avoid this problem, the static inventory must have a placeholder entry that defines an empty `webservers` host group. It is important for the static inventory to define any host

Chapter 5 | Managing Inventories

group it references, because it is possible that it could dynamically get removed from the external source, causing this error.

Edit the `inventory/hosts` file and add an empty `[webservers]` section.

```
[webservers]
```

```
[servers:children]
webservers
```

- ▶ 12. Rerun the `ansible-navigator inventory` command that lists the hosts in the `webservers` group. It works without any errors.

```
[student@workstation inventory-dynamic]$ ansible-navigator inventory \
> --mode stdout --graph webservers
@webservers:
|--servera.lab.example.com
```

- ▶ 13. Rerun the `ansible-navigator inventory` command that lists all the hosts in the `inventory`. Use the `--graph` option but do not provide a group name.

```
[student@workstation inventory-dynamic]$ ansible-navigator inventory \
> --mode stdout --graph
@all:
|--@servers:
| |--@webservers:
| | |--servera.lab.example.com
|--@ungrouped:
| |--workstation.lab.example.com
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Writing YAML Inventory Files

Objectives

- Write static inventory files in YAML format.

Inventory Plug-ins

Ansible uses plug-ins to support different formats of inventory files. Plug-ins are pieces of code that enhance the functionality of Ansible. By writing inventory plug-ins, Ansible developers can quickly support new formats and methods of generating inventory data. For example, Ansible manages static inventory files in INI format and dynamic inventory scripts with plug-ins.

Most inventory plug-ins are enabled by default. You can enable additional plug-ins by setting the `enable_plugins` directive under the `inventory` section in your `ansible.cfg` configuration file.

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml
```

The preceding example shows the default list. When Ansible parses inventory sources, it tries the plug-ins in the order they appear in the `enable_plugins` directive.

The `script` plug-in provides support for dynamic inventory scripts. The `ini` plug-in provides support for INI format. The other plug-ins get inventory information from files in other formats or other sources. Details are available at <https://docs.ansible.com/ansible/6/plugins/inventory.html>.

Some Ansible Content Collections come with additional inventory plug-ins. For example, the `amazon.aws` content collection provides the `aws_ec2` inventory plug-in to retrieve inventory hosts from Amazon Web Services EC2.

Writing YAML Static Inventory Files

You can use the `yaml` inventory plug-in to write static inventory files in YAML format. This plug-in is enabled by default.

The following example shows a static inventory file in INI format:

```
[lb_servers]
proxy.example.com

[web_servers]
web1.example.com
web2.example.com

[backend_server_pool]
appsrv-[a:e].example.com
```

This static inventory has three groups:

Chapter 5 | Managing Inventories

- The `lb_servers` group includes the `proxy.example.com` host.
- The `web_servers` group includes the `web1.example.com` and `web2.example.com` hosts.
- The `backend_server_pool` group uses an alphabetical range to include five hosts, `appsrv-a`, `appsrv-b`, `appsrv-c`, `appsrv-d`, and `appsrv-e`. The following example defines the same inventory in YAML format:

```
lb_servers:
  hosts:
    proxy.example.com:
web_servers:
  hosts:
    web1.example.com:
    web2.example.com:
backend_server_pool:
  hosts:
    appsrv-[a:e].example.com:
```

YAML inventories use blocks to organize related configuration items. Each block begins with the name of a group followed by a colon. Everything indented below the group name belongs to that group.

If indented below a group name, then the `hosts` keyword starts a block of hostnames. All server names indented below the `hosts` keyword belong to this group. These servers themselves form their own groups, and must therefore end in a colon.

To define nested groups, use the `children` keyword in a group block. This keyword starts a list of subgroups that are members of this group. Those member subgroups can have their own `hosts` and `children` blocks.

One advantage that the YAML format has over the INI format is that it organizes both the lists of servers and the list of nested groups in the same place in the static inventory file.

**Important**

The `all` group implicitly exists at the top level and includes the rest of the inventory as its children. You can explicitly list it in your YAML inventory file, but it is not required.

```
all:
  children:
    lb_servers:
      hosts:
        proxy.example.com:
    web_servers:
      hosts:
        web1.example.com:
        web2.example.com:
    backend_server_pool:
      hosts:
        appsrv-[a:e].example.com:
```

Static inventories in INI format often include hosts that are not members of any group.

```
notinagroup.example.com  
  
[mailservers]  
mail1.example.com
```

Ansible automatically puts those hosts in the special `ungrouped` group. In YAML, explicitly assign them to the `ungrouped` group.

```
all:  
  children:  
    ungrouped:  
      notinagroup.example.com:  
        mailservers:  
          mail1.example.com:
```

Setting Inventory Variables

You can set inventory variables directly in an inventory file in YAML format, just like you can in an inventory file in INI format.



Note

It is often a best practice to avoid storing variables in static inventory files.

Experienced Ansible developers prefer to use static inventory files only to list the hosts that Ansible manages, and to organize them in groups. The variables and their values are stored in the `host_vars` or `group_vars` files for the inventory.

However, you might want to define variables such as `ansible_port` or `ansible_connection` in the same file as the inventory itself, to keep this information in one place. If you set variables in too many places, then it is harder to remember where a particular variable is defined.

In a group block, use the `vars` keyword to set group variables directly in a YAML inventory file.

The following static inventory file in INI format sets the `smtp_relay` variable for all the hosts in `monitoring` group.

```
[monitoring]  
watcher.example.com  
  
[monitoring:vars]  
smtp_relay=smtp.example.com
```

The equivalent static inventory file in YAML format is as follows:

```
monitoring:  
  hosts:  
    watcher.example.com:  
  vars:  
    smtp_relay: smtp.example.com
```

Chapter 5 | Managing Inventories

In YAML, to set a variable for a specific host instead of a group, indent the variable under that host. The following static inventory file in INI format sets the `ansible_connection` variable for the `localhost` host.

```
[databases]
maria1.example.com
localhost ansible_connection=local
maria2.example.com
```

The equivalent static inventory file in YAML format is as follows:

```
databases:
hosts:
  maria1.example.com:
    localhost:
      ansible_connection: local
  maria2.example.com:
```

Converting a Static Inventory File in INI Format to YAML

You can use the `ansible-navigator inventory` command to help you convert an inventory file in INI format into YAML format. However, this is not the intended purpose of this command. The command is designed to display the entire configured inventory as Ansible sees it, and results can differ from those reported by the original inventory file. The `ansible-navigator inventory` command parses and tests the format of the inventory files, but it does not attempt to validate whether a hostname in the inventory exists.



Note

Remember that a name in an inventory file is not necessarily a valid hostname, but is used by the playbook to refer to a specific managed host. The name of that managed host could have an `ansible_host` host variable set that points to the real name or IP address of the host. The name in the inventory file can be a simplified alias for the purposes of the playbook.

The following example shows a static inventory file in INI format, called `origin_inventory`, which you want to convert to YAML. Notice that the `web1.example.com` host belongs to two groups: `web_servers` and `dc1`.

```
[lb_servers]
proxy.example.com

[web_servers]
web1.example.com
web2.example.com

[web_servers:vars]
http_port=8080

[backend_server_pool]
appsrv-[a:e].example.com
```

```
[dc1]
web1.example.com
appsrv-e.example.com
```

Use the following `ansible-navigator inventory --mode stdout -i origin_inventory \> --list --yaml` command to convert the `origin_inventory` file to YAML.

```
[user@host ~]$ ansible-navigator inventory --mode stdout -i origin_inventory \
> --list --yaml
all:
  children:
    backend_server_pool:
      hosts:
        appsrv-a.example.com: {}
        appsrv-b.example.com: {}
        appsrv-c.example.com: {}
        appsrv-d.example.com: {}
        appsrv-e.example.com: {}
dc1:
  hosts:
    appsrv-e.example.com: {}
    web1.example.com:
      http_port: 8080
  lb_servers:
    hosts:
      proxy.example.com: {}
  ungrouped: {}
  web_servers:
    hosts:
      web1.example.com: {}
      web2.example.com:
        http_port: 8080
```

Notice that the command converts the `http_port` group variable to a host variable for the two members of the `web_servers` group.

Also, for the `web1.example.com` host, the variable is only set for the first occurrence of that host in the inventory; that is, when the host is declared in the `dc1` group. This is incorrect because this is not how the original inventory works. If you create a playbook that targets the `web_servers` group, then Ansible does not set the `http_port` variable for the `web1.example.com` host as it would with the original inventory.

It is therefore important to carefully review and fix the resulting YAML file. In this example, you would define the `http_port` variable in the `web_servers` group. You could also define the hosts in the `backend_server_pool` by a range.

```
all:
  children:
    backend_server_pool:
      hosts:
        appsrv-[a:e].example.com: {}
dc1:
  hosts:
```

Chapter 5 | Managing Inventories

```

appsrv-e.example.com: {}
web1.example.com: {}

lb_servers:
  hosts:
    proxy.example.com: {}

ungrouped: {}

web_servers:
  hosts:
    web1.example.com: {}
    web2.example.com: {}

vars:
  http_port: 8080

```

**Note**

Some group or host lines in the YAML output end with braces, `{}`. Those braces indicate that the group does not have any members or group variables, or that the host has no host variables. They do not need to be included.

Using the `ansible-navigator inventory` command to convert large inventory files can save time, but you must use it with caution. It works better when your original static inventory file does not directly declare inventory variables, but gets them from external files in `host_vars` and `group_vars`.

Troubleshooting YAML Files

If you have trouble with static inventory files in YAML format, then consider these points about YAML syntax.

Protecting a Colon Followed by a Space

Remember that in an unquoted string, a colon followed by a space causes an error. YAML interprets this as starting a new element in a dictionary.

The following example causes an error because the second colon is not protected:

```
title: Ansible: Best Practices
```

The following strings are correct because they are protected by quotes or they do not have a space after the colons:

```
title: 'Ansible: Best Practices'
double: "Ansible: Best Practices"
fine: Not:a:problem
```

Protecting a Variable That Starts a Value

Ansible performs variable replacement with the `{{ variable }}` syntax, but anything beginning with an opening brace, `{`, in YAML is interpreted as the start of a dictionary. Thus, you must enclose the variable placeholder with double quotes: `name: "{{ variable }}" rest of the value`.

In general, when using any of the reserved characters, [] {} > , | * & ! % # ` @ , you should use double quotes around the value.

Knowing the Difference Between a String and a Boolean or Float

Booleans and floating point numbers used as values for a variable must not be quoted. Quoted values are treated as strings. The following example sets the `active` parameter to the `true` Boolean value, and sets the `default_answer` parameter to the `true` string.

```
active: true
default_answer: "true"
```

The following example sets the `temperature` parameter to a floating point value, and sets the `version` parameter to a string.

```
temperature: 36.5
version: "2.0"
```



References

Ansible Documentation: Inventory Plugins

<https://docs.ansible.com/ansible/6/plugins/inventory.html>

Ansible Documentation: How to Build Your Inventory

https://docs.ansible.com/ansible/6/user_guide/intro_inventory.html

Ansible Documentation: YAML Syntax

https://docs.ansible.com/ansible/6/reference_appendices/YAMLSyntax.html

► Guided Exercise

Writing YAML Inventory Files

Convert an inventory file from INI format to YAML format.

Outcomes

- Replace an Ansible inventory file in INI format with its YAML equivalent.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares an Ansible project in `/home/student/inventory-yaml/` on the `workstation` machine.

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

Instructions

- 1. The `lab` command provides an inventory file in INI format in the `/home/student/inventory-yaml/` directory. Change to that directory, create a copy of the `inventory` file and save it as `inventory.yml`.
- 1.1. Change to the `/home/student/inventory-yaml/` directory.

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

- 1.2. Copy the existing static inventory file to `inventory.yml`.

```
[student@workstation inventory-yaml]$ cp inventory inventory.yml
```

- 2. Edit the new inventory file, `inventory.yml`, to convert it to YAML format.

- 2.1. Convert the groups to YAML format.

- Remove the brackets that surround the group names and end each group name with a colon.
- Beneath each group name, add the `hosts:` keyword on a line indented by two spaces relative to the group name.
- Indent the hosts in each group by four spaces relative to the group name and add a colon at the end of each hostname.

The exact number of spaces does not matter as long as you keep a consistent indentation.

For example, the following group definition in INI format:

```
[active_web_servers]
server[b:c].lab.example.com
```

becomes:

```
active_web_servers:
hosts:
server[b:c].lab.example.com:
```

Repeat the operation for the `all_servers` and `inactive_web_servers` groups.

- 2.2. Convert the definition of the child groups of the `web_servers` group. The file in INI format lists the child groups under the `[web_servers:children]` section.

- Beneath the group name, add the `children:` keyword on a line indented by two spaces relative to the group name.
- Add each child group on a line indented by four spaces relative to the group name and add a colon at the end of each group name.

For example, the following section:

```
[web_servers:children]
active_web_servers
inactive_web_servers
```

becomes:

```
web_servers:
children:
active_web_servers:
inactive_web_servers:
```

- 2.3. Finally, convert the `all_servers:children` group to YAML format by putting the child groups in the `children` section.

```
[all_servers:children]
web_servers
```

becomes:

```
all_servers:
hosts:
servera.lab.example.com:
children:
web_servers:
```

The resulting `inventory.yml` file should consist of the following content:

```
active_web_servers:
hosts:
server[b:c].lab.example.com:
```

```
inactive_web_servers:  
  hosts:  
    server[d:f].lab.example.com:  
  
web_servers:  
  children:  
    active_web_servers:  
    inactive_web_servers:  
  
all_servers:  
  hosts:  
    servera.lab.example.com:  
  children:  
    web_servers:
```

- 3. To test your inventory in YAML format, use the `test-ping-all-servers.yml` playbook that the exercise provides. That playbook targets the `all_servers` group and uses the `ping` module to test the connection to all the servers.

Use the `ansible-navigator` command to run the `test-ping-all-servers.yml` playbook. Add the `--inventory` (or `-i`) option to provide your inventory file in YAML format.

```
[student@workstation inventory-yaml]$ ansible-navigator run -i inventory.yml \  
> --mode stdout test-ping-all-servers.yml  
  
PLAY [Pinging all servers] *****  
  
TASK [Ensure the hosts are reachable] *****  
ok: [serverb.lab.example.com]  
ok: [serverc.lab.example.com]  
ok: [serverd.lab.example.com]  
ok: [servere.lab.example.com]  
ok: [servera.lab.example.com]  
ok: [serverf.lab.example.com]  
...output omitted...
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Managing Inventory Variables

Objectives

- Structure host and group variables by using multiple files per host or group, and use special variables to override the host, port, or remote user that Ansible uses for a specific host.

The Basic Principles of Variables

You can use variables to write flexible and reusable tasks, roles, and playbooks. You can also use them to specify differences in configuration between different systems. You can set variables in many places, including:

- In the `defaults/main.yml` and `vars/main.yml` files for a role.
- In the inventory file, either as a host variable or a group variable.
- In a variable file in the `group_vars/` or `host_vars/` subdirectories of the playbook or inventory.
- In a play, role, or task.

As you define and manage variables in a project, plan to follow these principles:

Keep it simple

Even though you can define Ansible variables in many ways, try to define them by using only a couple of different methods and in only a few places.

Do not repeat yourself

If a set of systems has a common configuration, then organize them into a group and set inventory variables for them in a file in the `group_vars/` directory. This way, you do not have to define the same settings on a host-by-host basis, and when you do have to modify a variable for that group of systems you can do it by updating the variable file only once.

Organize variables in small, readable files

If you have a large project with many host groups and variables, then split the variable definitions into multiple files. To make it easier to find particular variables, group related variables into a file and give the file a meaningful name.

Remember that you can use subdirectories instead of files in the `host_vars/` and `group_vars/` directories. For example, you could have a `group_vars/webserver/` directory for the `webserver` group, and that directory could contain a file called `firewall.yml`, which contains only variables related to firewall configuration. That directory could also contain other group variable files for configuring other components of the servers in the group.

Variable Merging and Precedence

When you define the same variable in multiple ways, Ansible uses precedence rules to choose a value for that variable. After processing all the variable definitions, Ansible generates a set of merged variables for each host at the start of each task.

Chapter 5 | Managing Inventories

When Ansible merges variables, if there are two definitions for the same variable in different places, then it uses the value that has the highest precedence.

Precedence order is covered in the *Variable Precedence: Where Should I Put a Variable?* documentation at https://docs.ansible.com/ansible/6/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable. The following discussion breaks this down into more detail, from the lowest to the highest precedence.

Determining Command-line Option Precedence

Options that you pass to `ansible-navigator run` on the command line, other than `--extra-vars` (or `-e`) for extra variables, have the lowest precedence. For example, you can set the remote user with the `--user` (or `-u`) option to override the configuration file, but setting `ansible_user` at a higher precedence overrides it.

Determining Role Default Precedence

Variables set by a role in the `rolename/defaults/main.yml` file have a low precedence so that they are easy to override. These variables provide some reasonable default values and users often override them to configure the role.

Determining Host and Group Variable Precedence

You can set host-specific and group-specific variables in a number of places. You can set them relative to the location of your inventory or your playbook, by collecting host facts, or by reading facts from a cache.

The precedence order for these variables is as follows, from lowest to highest:

- Group variables (in ascending order of precedence):
 - Set *directly* in an inventory file or by a dynamic inventory script
 - For `all` set in the `inventory group_vars/all` file or subdirectory
 - For `all` set in the `playbook group_vars/all` file or subdirectory
 - For other groups set in the `inventory group_vars/` subdirectory
 - For other groups set in the `playbook group_vars/` subdirectory
- Host variables (in ascending order of precedence):
 - Set *directly* in an inventory file or by a dynamic inventory script
 - Set in the `inventory host_vars/` subdirectory
 - Set in the `playbook host_vars/` subdirectory
- Host facts and cached facts

The biggest source of confusion here is the distinction between the `group_vars/` and `host_vars/` subdirectories relative to the inventory and those relative to the playbook. This difference does not exist if you use a static inventory file in the same directory where you run the playbook.

If you have `group_vars/` and `host_vars/` subdirectories in the same directory as your playbook, then Ansible automatically includes those group and host variables.

When you use a flat inventory file in a directory other than the one the playbook is in, then Ansible also automatically includes the `group_vars/` and `host_vars/` directories in the inventory file directory. However, the variables that Ansible includes from the playbook directory override them if there is a conflict. For example, if you use `/etc/ansible/hosts` as your inventory, then Ansible uses the `/etc/ansible/group_vars/` directory as another group variable directory.

Chapter 5 | Managing Inventories

When you use an inventory directory that contains multiple inventory files, then Ansible includes the `group_vars/` and `host_vars/` subdirectories of your inventory directory.

Consider the following tree structure:

```
.  
└── ansible.cfg  
└── group_vars/  
    └── all  
└── inventory/  
    ├── group_vars/  
    │   └── all  
    ├── phoenix-dc  
    └── singapore-dc  
└── playbook.yml
```

The `playbook.yml` file is the playbook. The `ansible.cfg` file configures the `inventory` directory as the inventory source, and `phoenix-dc` and `singapore-dc` are two static inventory files.

The `inventory/group_vars/all` file is inside the inventory directory and defines variables for all the hosts.

The `group_vars/all` file, in the same directory as the playbook, also loads variables for all the hosts. In the case of a conflict, these settings override `inventory/group_vars/all`.



Note

The value of this distinction is that you can set default values for variables that are bundled with the inventory files in a location that is shared by all your playbooks.

You can still override the settings for those inventory variables in individual playbook directories.

You can set host or group variables directly in inventory files (such as `phoenix-dc` in the preceding example), but it is generally not a good practice. It is easier to find all the variable settings that apply to a host or group if you group them in files that only contain settings for that host or group. If you have to also examine the inventory files, then that can be more time-consuming and error prone, especially for a large inventory.

Determining Play Variable Precedence

The next category of variables is those that you set in the playbook as part of a play, task, role parameter, or that you include or import. These have higher precedence than host or group variables, role defaults, and command-line options other than `--extra-vars` (or `-e`).

The precedence order for these variables is as follows, from lowest to highest:

- Set by the `vars` section of a play
- Set by prompting the user with a `vars_prompt` section in a play
- Set from a list of external files by the `vars_files` section of a play
- Set by a role `rolename/vars/main.yml` file
- Set for the current block with a `vars` section of that block

Chapter 5 | Managing Inventories

- Set for the current task with a `vars` section of that task
- Loaded dynamically with the `include_vars` module
- Set for a specific host by using either the `set_fact` module or by using `register` to record the result of task execution on a host
- Parameters set for a role in the playbook when loaded by the `role` section of a play or by using the `include_role` module
- Set by a `vars` section on tasks included with the `include_tasks` module

Notice that the normal `vars` section in a play has the lowest precedence in this category. There are many ways to override those settings if necessary. Variables set here override host-specific and group-specific settings in general.

Using `vars_prompt` is not a recommended practice. To operate correctly, you need to configure `ansible-navigator` to allow interaction with the `ansible-navigator run` command while it is running. The `vars_prompt` directive is also not compatible with automation controller.

The `vars_files` directive is useful for organizing large lists of variables that are not host or group specific. These variables are organized by function into separate files. It can also help you separate sensitive variables into a separate file that you can encrypt with Ansible Vault. You can keep this separate from variables that are not sensitive and that you do not need to encrypt.

You can set variables that apply only to a specific block or task. These values override the play variables and inventory variables. You should use these sparingly, because they can make the playbook more complex.

```
- name: Task with a local variable definition
  vars:
    taskvariable: task
  ansible.builtin.debug:
    var: taskvariable
```

Notice that variables that you load by using `include_vars` have a high precedence, and can override variables set for roles and specific blocks and tasks. Often, you might want to use `vars_files` instead if you do not want to override those values with your external variable files.

The `set_fact` module and the `register` directive both set host-specific information, either a fact or the results of task execution on that host. Note that `set_fact` sets a high precedence value for that variable for the rest of the playbook run, but if you cache that fact then Ansible stores it in the fact cache at normal fact precedence (lower than play variables).

Determining the Precedence of Extra Variables

Extra variables that you set by using the `--extra-vars` (or `-e`) option of the `ansible-navigator run` command always have the highest precedence. This is useful so that you can override the global setting for a variable in your playbook from the command line without editing any of the Ansible project files.

Separating Variables from Inventory

The inventory sources define the hosts and host groups that Ansible uses, whether they are static files or a dynamic inventory script. If you are managing your inventory as static files, then you can define variables in the inventory files, in the same files in which you define your host and host

Chapter 5 | Managing Inventories

group lists. However, this is not the best practice. As your environment grows in both size and variety, the inventory file becomes large and difficult to read.

In addition, you probably want to migrate to dynamic inventory sources rather than static inventory files to make it easier to manage your inventory. You might also still want to manage inventory variables statically, however, separately from or in addition to the output from the dynamic inventory script.

A better approach is to move variable definitions from the inventory file into separate variable files, one for each host group. Each variable file is named after a host group, and contains variable definitions for the host group.

```
[user@host project]$ tree -F group_vars
group_vars/
└── db_servers.yml
└── lb_servers.yml
└── web_servers.yml
```

This structure makes it easier to locate configuration variables for any of the host groups: `db_servers`, `lb_servers`, or `web_servers`. Provided that each host group does not contain too many variable definitions, the above organizational structure is sufficient. As playbook complexity increases, however, even these files can become long and difficult to understand.

An even better approach for large, diverse environments is to create subdirectories for each host group under the `group_vars/` directory. Ansible parses any YAML files in these subdirectories and associates the variables with a host group based on the parent directory.

```
[user@host project2]$ tree -F group_vars
group_vars/
└── db_servers/
    ├── 3.yml
    ├── a.yml
    └── myvars.yml
└── lb_servers/
    ├── 2.yml
    ├── b.yml
    └── something.yml
└── web_servers/
    └── nothing.yml
```

In the preceding example, variables in the `myvars.yml` file are associated with the `db_servers` host group, because the file is in the `group_vars/db_servers/` subdirectory. The file names in this example, however, make it difficult to know where to find a particular variable.

If you use this organizational structure for variables, then group variables with a common theme into the same file, and use a file name that indicates that common theme. When a playbook uses roles, a common convention is to create variable files named after each role.

A project organized according to this convention might be as follows:

```
[user@host project3]$ tree -F group_vars
group_vars/
└── all/
    └── common.yml
```

Chapter 5 | Managing Inventories

```
└── db_servers/
    ├── mysql.yml
    └── firewall.yml
└── lb_servers/
    ├── firewall.yml
    ├── haproxy.yml
    └── ssl.yml
└── web_servers/
    ├── firewall.yml
    ├── webapp.yml
    └── apache.yml
```

With this organizational structure for a project, you can quickly see the types of variables that are defined for each host group.

Ansible merges all the variables present in files in directories under the `group_vars/` directory with the rest of the variables. Separating variables into files grouped by functionality makes the whole playbook easier to understand and maintain.

Using Special Inventory Variables

A number of variables are available that you can use to change how Ansible connects to a host listed in the inventory. Some of these are most useful as host-specific variables, but others might be relevant to all hosts in a group or in the inventory.

`ansible_connection`

The connection plug-in to use to access the managed host. By default, Ansible uses the `ssh` plug-in for all hosts except `localhost`, for which Ansible uses the `local` plug-in.

`ansible_host`

The actual IP address or fully qualified domain name to use when connecting to the managed host, instead of using the name from the inventory file. By default, this variable has the same value as the inventory hostname.

`ansible_port`

The port that Ansible uses to connect to the managed host. For the `ssh` connection plug-in, the default value is 22.

`ansible_user`

The user that Ansible uses to connect to the managed host. The default Ansible behavior is to connect to the managed host by using the same username as the user running the Ansible Playbook on the control node.

`ansible_become_user`

After Ansible has connected to the managed host, it switches to this user using `ansible_become_method`, which is `sudo` by default. You might need to provide authentication credentials in some way.

`ansible_python_interpreter`

The path to the Python executable that Ansible should use on the managed host. For Ansible 2.8 and later, this defaults to `auto_legacy`, which automatically selects a Python interpreter on the host running the playbook depending on what operating system it is running. Consequently, this setting is less likely to be required compared to earlier versions of Ansible.

Configuring Human Readable Inventory Hostnames

When Ansible executes a task on the remote host, the output displays the inventory hostname. Because you can specify alternative connection properties by using the preceding special inventory variables, you can assign arbitrary names to inventory hosts. When you assign inventory hosts with meaningful names, you are better able to understand playbook output and diagnose playbook errors.

Consider a playbook that uses the following YAML inventory file:

```
web_servers:  
  hosts:  
    server100.example.com:  
    server101.example.com:  
    server102.example.com:  
lb_servers:  
  hosts:  
    server103.example.com:
```

With the preceding inventory file, the Ansible output references these names. Consider the following hypothetical output from a playbook that uses this inventory file:

```
[user@host project]$ ansible-navigator run --mode stdout site.yml  
...output omitted...  
PLAY RECAP ****  
server100.example.com : ok=4    changed=0    unreachable=0    failed=0    ...  
server101.example.com : ok=4    changed=0    unreachable=0    failed=0    ...  
server102.example.com : ok=4    changed=0    unreachable=0    failed=0    ...  
server103.example.com : ok=3    changed=0    unreachable=0    failed=1    ...
```

From the preceding output, you cannot easily tell that the failed host is a load balancer.

To improve that output, use an inventory file that contains descriptive names and define the necessary special inventory variables:

```
web_servers:  
  hosts:  
    webserver_1:  
      ansible_host: server100.example.com  
    webserver_2:  
      ansible_host: server101.example.com  
    webserver_3:  
      ansible_host: server102.example.com  
lb_servers:  
  hosts:  
    loadbalancer:  
      ansible_host: server103.example.com
```

Now the output from the playbook provides descriptive names:

```
[user@host project]$ ansible-navigator run --mode stdout site.yml
...output omitted...
PLAY RECAP ****
loadbalancer      : ok=3    changed=0    unreachable=0    failed=1    ...
webserver_1       : ok=4    changed=0    unreachable=0    failed=0    ...
webserver_2       : ok=4    changed=0    unreachable=0    failed=0    ...
webserver_3       : ok=4    changed=0    unreachable=0    failed=0    ...
```

In some situations you might want to use an arbitrary hostname in your playbook that you map to a real IP address or hostname with the `ansible_host` directive. For example:

- You might want Ansible to connect to that host by using a specific IP address that is different from the one that resolves in DNS. For example, there might be a particular management address that is not public, or the machine might have multiple addresses in DNS but one is on the same network as the control node.
- You might be provisioning cloud systems that have arbitrary names, but you want to refer to those systems in your playbook with names that make sense based on the roles that they play. If you use a dynamic inventory, then your dynamic inventory source might assign host variables automatically based on the intended role of each system.
- You might be referring to the machine by a short name in the playbook, but you need to refer to it by a fully qualified domain name in the inventory to properly connect to it.

Identifying the Current Host by Using Variables

When a play is running, you can use a number of variables and facts to identify the name of the current managed host executing a task:

`inventory_hostname`

The name of the managed host currently being processed, taken from the inventory.

`ansible_host`

The actual IP address or hostname that was used to connect to the managed host, as previously discussed.

`ansible_facts['hostname']`

The short (unqualified) hostname collected as a fact from the managed host.

`ansible_facts['fqdn']`

The fully qualified domain name (FQDN) collected as a fact from the managed host.

One final variable that can be useful is `ansible_play_hosts`, which is a list of all the hosts that have not yet failed during the current play, and therefore are going to be used for the tasks remaining in the play.



References

[Ansible Documentation: Variable Precedence](#)

https://docs.ansible.com/ansible/6/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

[Ansible Documentation: How to Build Your Inventory](#)

https://docs.ansible.com/ansible/6/user_guide/intro_inventory.html

[Ansible Documentation: Special Variables](#)

https://docs.ansible.com/ansible/6/reference_appendices/special_variables.html

► Guided Exercise

Managing Inventory Variables

Set up directories that contain multiple host variable files for some of your managed hosts, and override the name used in the inventory file with a different name or IP address for at least one of your hosts.

Outcomes

- Split the location of host variable files across multiple directories to improve maintainability.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command ensures that Ansible is installed on workstation and also prepares the `/home/student/inventory-variables` directory for this exercise.

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

Instructions

- 1. Get familiar with the Ansible project and its current state.

- 1.1. As the student user on the workstation machine, change to the `/home/student/inventory-variables` directory.

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

- 1.2. List the files that the `lab` command has prepared for you.

```
[student@workstation inventory-variables]$ tree -L 1 -F
.
├── ansible.cfg
├── ansible-navigator.yml
├── deploy_apache.yml ①
├── deploy_haproxy.yml ②
├── deploy_webapp.yml ③
├── inventory.yml ④
└── roles/
    └── site.yml ⑤
```

- ① The `deploy_apache.yml` playbook installs the Apache HTTP Server and configures the firewall for the hosts in the `web_servers` group.
- ② The `deploy_haproxy.yml` playbook installs HAProxy on the `lb_servers` host group, which only includes the `servera` host. This playbook also configures

Chapter 5 | Managing Inventories

the firewall and declares the `serverb` and `serverc` hosts as back ends for HAProxy.

- ❸ The `deploy_webapp.yml` playbook deploys some initial web content on the hosts in the `web_servers` group.
- ❹ The `inventory.yml` file declares the `serverb` and `serverc` hosts as members of that host group.
- ❺ The global `site.yml` playbook invokes the three preceding playbooks.

1.3. Review the `inventory.yml` file.

```
[student@workstation inventory-variables]$ cat inventory.yml
lb_servers:
  hosts:
    servera.lab.example.com:

web_servers:
  hosts:
    server[b:c].lab.example.com:
```

- ▶ 2. The `deploy_haproxy.yml` playbook declares two variables: `haproxy_appservers`, which provides the HAProxy configuration; and `firewall_rules`, which defines the firewall configuration. Move those two variables into inventory variable files.

2.1. Review the `deploy_haproxy.yml` playbook.

```
[student@workstation inventory-variables]$ cat deploy_haproxy.yml
---
- name: Ensuring HAProxy is deployed
  hosts: lb_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure HAProxy is installed and configured
      ansible.builtin.include_role:
        name: haproxy
      vars:
        haproxy_appservers:
          - name: serverb.lab.example.com
            ip: 172.25.250.11
            port: 80
          - name: serverc.lab.example.com
            ip: 172.25.250.12
            port: 80

    - name: Ensure the firewall ports are opened
      ansible.builtin.include_role:
        name: firewall
      vars:
        firewall_rules:
          - port: 80/tcp
```

Chapter 5 | Managing Inventories

- 2.2. Create the `group_vars/lb_servers/` directory to hold the variable files for the `lb_servers` host group.

```
[student@workstation inventory-variables]$ mkdir -p group_vars/lb_servers
```

- 2.3. Create the `group_vars/lb_servers/haproxy.yml` variable file to store the HAProxy configuration. Copy and paste the `haproxy_appservers` variable definition from the `deploy_haproxy.yml` playbook.

```
[student@workstation inventory-variables]$ cat group_vars/lb_servers/haproxy.yml
---
haproxy_appservers:
  - name: serverb.lab.example.com
    ip: 172.25.250.11
    port: 80
  - name: serverc.lab.example.com
    ip: 172.25.250.12
    port: 80
```

- 2.4. Create the `group_vars/lb_servers/firewall.yml` variable file to store the firewall configuration. Copy and paste the `firewall_rules` variable definition from the `deploy_haproxy.yml` playbook.

```
[student@workstation inventory-variables]$ cat group_vars/lb_servers/firewall.yml
---
firewall_rules:
  - port: 80/tcp
```

- 2.5. Edit the `deploy_haproxy.yml` playbook and then remove the two variables. The resulting file should consist of the following content:

```
---
- name: Ensuring HAProxy is deployed
  hosts: lb_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure HAProxy is installed and configured
      ansible.builtin.include_role:
        name: haproxy

    - name: Ensure the firewall ports are opened
      ansible.builtin.include_role:
        name: firewall
```

- ▶ 3. The `deploy_apache.yml` playbook declares the `firewall_rules` variable that defines the firewall configuration for the web servers. Move those two variables into an inventory variable file.

- 3.1. Review the `deploy_apache.yml` playbook.

```
[student@workstation inventory-variables]$ cat deploy_apache.yml
---
- name: Ensuring Apache HTTP Server is deployed
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure Apache HTTP Server is installed and started
      ansible.builtin.include_role:
        name: apache

    - name: Ensure the firewall ports are opened
      ansible.builtin.include_role:
        name: firewall
      vars:
        firewall_rules:
          # Allow HTTP requests from the load_balancer
          - zone: internal
            service: http
          - zone: internal
            source: "172.25.250.10"
```

- 3.2. Create the `group_vars/web_servers/` directory to hold the variable files for the `web_servers` host group.

```
[student@workstation inventory-variables]$ mkdir -p group_vars/web_servers
```

- 3.3. Create the `group_vars/web_servers/firewall.yml` variable file to store the firewall configuration. Copy and paste the `firewall_rules` variable definition from the `deploy_apache.yml` playbook.

```
[student@workstation inventory-variables]$ cat group_vars/web_servers/firewall.yml
---
firewall_rules:
  # Allow HTTP requests from the load_balancer
  - zone: internal
    service: http
  - zone: internal
    source: "172.25.250.10"
```

- 3.4. Edit the `deploy_apache.yml` playbook and then remove the `firewall_rules` variable. The resulting file should consist of the following content:

```
---
- name: Ensuring Apache HTTP Server is deployed
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure Apache HTTP Server is installed and started
```

Chapter 5 | Managing Inventories

```
ansible.builtin.include_role:
  name: apache

  - name: Ensure the firewall ports are opened
  ansible.builtin.include_role:
    name: firewall
```

- 4. Edit the `inventory.yml` static inventory file so that Ansible displays the `servera.lab.example.com` host as `load_balancer` in its output. The resulting file should consist of the following content:

```
lb_servers:
  hosts:
    load_balancer:
      ansible_host: servera.lab.example.com

web_servers:
  hosts:
    server[b:c].lab.example.com:
```

- 5. Run the `site.yml` playbook and confirm that the web deployment was successful.

- 5.1. Use the `ansible-navigator run` command to run the `site.yml` playbook.

```
[student@workstation inventory-variables]$ ansible-navigator run \
> site.yml --mode stdout

PLAY [Ensuring HAProxy is deployed] ****
TASK [Ensure HAProxy is installed and configured] ****
TASK [haproxy : Ensure the HAProxy packages are installed] ****
changed: [load_balancer]

TASK [haproxy : Ensure HAProxy is started and enabled] ****
changed: [load_balancer]

TASK [haproxy : Ensure HAProxy configuration is set] ****
changed: [load_balancer]
...output omitted...
```

- 5.2. Use the `curl` command to confirm that the HAProxy server distributes the HTTP requests between the two Apache HTTP Servers.

```
[student@workstation inventory-variables]$ curl http://servera.lab.example.com
This is serverb.lab.example.com. (version v1.0)
[student@workstation inventory-variables]$ curl http://servera.lab.example.com
This is serverc.lab.example.com. (version v1.0)
```

Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Lab

Managing Inventories

Convert an inventory from INI format to YAML and apply best practices to variable structuring.

Outcomes

- Organize playbook variables in a directory structure.
- Write an inventory file in YAML format.
- Assign arbitrary hostnames for remote hosts in an inventory.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. The Git repository contains a `site.yml` playbook that configures a front-end load balancer and a pool of back-end web servers. The back-end server pool is partitioned into two groups, A and B. Each pool partition deploys an independent version of the web application. Use `Student@123` as the Git password when you push changes to this remote repository.

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

Instructions

- Clone the `https://git.lab.example.com/student/inventory-review.git` Git repository into the `/home/student/git-repos` directory and then create the `exercise` branch.

In the `~/git-repos/inventory-review/` directory, review the `inventory` file and then run the `site.yml` playbook using automation content navigator.

You can use the `curl` command to verify that the same version of the web application is deployed to both back-end web servers by making at least two web requests to `servera`.
- Configure the web servers in each host group to report different version numbers for the web application, as follows:

Create a directory for variable files for the `a_web_servers` host group. Create a variable file in that directory called `webapp.yml`. That file must set the value of the `webapp_version` variable to `v1.1a`.

Create another directory for the `b_web_servers` host group. It should also contain a variable file called `webapp.yml`. That second variable file must set the value of the `webapp_version` variable to `v1.1b`.
- Using automation content navigator, run the `deploy_webapp.yml` playbook to update the version information displayed by the web application.

Use the `curl` command to confirm that requests for web pages sent to `servera` produce two different versions of the web application.

Chapter 5 | Managing Inventories

- Commit the files that you have created to your local Git repository.
4. Create a YAML-formatted inventory file called `inventory.yml` using the data from the INI-formatted `inventory` file. If you generate the `inventory.yml` file using the `ansible-navigator` command, then **remove the variables that are added for each host**. Do not remove the original `inventory` file.
- Run the `site.yml` playbook using the new `inventory.yml` file.
5. You have decided to change your YAML inventory to use inventory hostnames that are mapped to their purposes, rather than using actual hostnames.
- Modify the `inventory.yml` file as follows:
- In the `lb_servers` host group, change `servera.lab.example.com` to `loadbalancer_1`. Assign `loadbalancer_1` an `ansible_host` variable that configures its real hostname as `servera.lab.example.com`.
 - In the `a_web_servers` host group, change `serverb.lab.example.com` to `backend_a1`. Assign `backend_a1` an `ansible_host` variable that configures its real hostname as `serverb.lab.example.com`.
 - In the `b_web_servers` host group, change `serverc.lab.example.com` to `backend_b1`. Assign `backend_b1` an `ansible_host` variable that configures its real hostname as `serverc.lab.example.com`.
6. Verify that the `site.yml` playbook runs without errors when you use the updated `inventory.yml` inventory file. You can use the `curl` command to test the load balancer again.

After the playbook runs without errors, commit the new `inventory.yml` file to your local Git repository. If prompted, then use `Student@123` as the Git password. After you have committed all project changes, push the local commits to the remote repository.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Managing Inventories

Convert an inventory from INI format to YAML and apply best practices to variable structuring.

Outcomes

- Organize playbook variables in a directory structure.
- Write an inventory file in YAML format.
- Assign arbitrary hostnames for remote hosts in an inventory.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. The Git repository contains a `site.yml` playbook that configures a front-end load balancer and a pool of back-end web servers. The back-end server pool is partitioned into two groups, A and B. Each pool partition deploys an independent version of the web application. Use `Student@123` as the Git password when you push changes to this remote repository.

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

Instructions

- Clone the `https://git.lab.example.com/student/inventory-review.git` Git repository into the `/home/student/git-repos` directory and then create the `exercise` branch.

In the `~/git-repos/inventory-review/` directory, review the `inventory` file and then run the `site.yml` playbook using automation content navigator.

You can use the `curl` command to verify that the same version of the web application is deployed to both back-end web servers by making at least two web requests to `serverA`.

- From a terminal, create the `/home/student/git-repos` directory if it does not exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- Clone the `https://git.lab.example.com/student/inventory-review.git` repository and then change directory to the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/inventory-review.git
Cloning into 'inventory-review'...
...output omitted...
[student@workstation git-repos]$ cd inventory-review
```

- 1.3. Create the `exercise` branch and check it out.

```
[student@workstation inventory-review]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 1.4. Review the `inventory` file.

```
[student@workstation inventory-review]$ cat inventory
[lb_servers]
servera.lab.example.com

[web_servers]

[web_servers:children]
a_web_servers
b_web_servers

# Group "A" of Web Servers
[a_web_servers]
serverb.lab.example.com

# Group "B" of Web Servers
[b_web_servers]
serverc.lab.example.com
```

In this project, the `web_servers` host group is composed of two other groups: `a_web_servers` and `b_web_servers`.

- 1.5. Using the `ee-supported-rhel8` execution environment, run the `site.yml` playbook.

```
[student@workstation inventory-review]$ ansible-navigator run site.yml

Play name          Ok   Changed ... Skipped ... Task count  Progress
0|Ensure HAProxy is deployed  7      6 ...      2 ...          9  Complete
1|Ensure Apache is deployed  14     12 ...     2 ...         16  Complete
2|Ensure Web App is deployed  4      2 ...      0 ...          4  Complete

^f/PgUp page up    ^b/PgDn page down    ↑↓ scroll    esc back ... Successful
```

Press ESC to exit from the `ansible-navigator` command.

- 1.6. Verify that the same version of the web application is deployed to all back-end web servers.

Chapter 5 | Managing Inventories

```
[student@workstation inventory-review]$ for x in 1 2; do curl servera; done
Hello from serverb.lab.example.com. (version v1.1)
Hello from serverc.lab.example.com. (version v1.1)
```

2. Configure the web servers in each host group to report different version numbers for the web application, as follows:

Create a directory for variable files for the `a_web_servers` host group. Create a variable file in that directory called `webapp.yml`. That file must set the value of the `webapp_version` variable to `v1.1a`.

Create another directory for the `b_web_servers` host group. It should also contain a variable file called `webapp.yml`. That second variable file must set the value of the `webapp_version` variable to `v1.1b`.

- 2.1. Create directories to hold group variable files for the `a_web_servers` and `b_web_servers` host groups.

```
[student@workstation inventory-review]$ mkdir -pv group_vars/{a,b}_web_servers
mkdir: created directory 'group_vars/a_web_servers'
mkdir: created directory 'group_vars/b_web_servers'
```

- 2.2. Create the `webapp.yml` variable file for the `a_web_servers` group. Use it to set the value of the `webapp_version` variable to `v1.1a`.

```
[student@workstation inventory-review]$ echo "webapp_version: v1.1a" > \
> group_vars/a_web_servers/webapp.yml
```

- 2.3. Create the `webapp.yml` variable file for the `b_web_servers` group. Use it to set the value of the `webapp_version` variable to `v1.1b`.

```
[student@workstation inventory-review]$ echo "webapp_version: v1.1b" > \
> group_vars/b_web_servers/webapp.yml
```

3. Using automation content navigator, run the `deploy_webapp.yml` playbook to update the version information displayed by the web application.

Use the `curl` command to confirm that requests for web pages sent to `servera` produce two different versions of the web application.

Commit the files that you have created to your local Git repository.

- 3.1. Run the `deploy_webapp.yml` playbook:

```
[student@workstation inventory-review]$ ansible-navigator run deploy_webapp.yml

Play name          Ok   Changed ... Failed ... Task count  Progress
0|Ensure Web App is deployed  4        2 ...      0 ...       4  Complete
```

Press ESC to exit from the `ansible-navigator` command.

- 3.2. Verify that requests to `servera` produce two different versions of the web application.

```
[student@workstation inventory-review]$ for x in 1 2; do curl servera; done
Hello from serverb.lab.example.com. (version v1.1a)
Hello from serverc.lab.example.com. (version v1.1b)
```

- 3.3. Commit these changes to your local Git repository. Optionally, if you decide to push the branch at this step, use `Student@123` as the Git password.

```
[student@workstation inventory-review]$ git status
On branch exercise
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    group_vars/a_web_servers/
    group_vars/b_web_servers/

nothing added to commit but untracked files present (use "git add" to track)
[student@workstation inventory-review]$ git add group_vars/{a,b}_web_servers
[student@workstation inventory-review]$ git status
On branch exercise
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file:   group_vars/a_web_servers/webapp.yml
  new file:   group_vars/b_web_servers/webapp.yml

[student@workstation inventory-review]$ git commit \
> -m "Created variable files for the A and B groups."
...output omitted...
```

4. Create a YAML-formatted inventory file called `inventory.yml` using the data from the INI-formatted `inventory` file. If you generate the `inventory.yml` file using the `ansible-navigator` command, then **remove the variables that are added for each host**. Do not remove the original `inventory` file.

Run the `site.yml` playbook using the new `inventory.yml` file.

- 4.1. Create a YAML-formatted inventory file called `inventory.yml` using the data from the INI-formatted `inventory` file:

```
[student@workstation inventory-review]$ ansible-navigator inventory \
> -i inventory --list --yaml -m stdout > inventory.yml
```

- 4.2. Ensure that the `inventory.yml` file does not contain host variables. Either of the following two examples for the `inventory.yml` file are acceptable. Trim the file to match one or the other example.

- Example 1: This example includes the `all` group and includes the optionally ungrouped group.

```
all:
  children:
    lb_servers:
      hosts:
        servera.lab.example.com:
```

Chapter 5 | Managing Inventories

```

ungrouped: {}
web_servers:
  children:
    a_web_servers:
      hosts:
        serverb.lab.example.com:
    b_web_servers:
      hosts:
        serverc.lab.example.com:

```

- Example 2: This example more closely matches the existing `inventory` file.

```

lb_servers:
  hosts:
    servera.lab.example.com:
web_servers:
  children:
    a_web_servers:
      hosts:
        serverb.lab.example.com:
    b_web_servers:
      hosts:
        serverc.lab.example.com:

```

4.3. Test the new `inventory.yml` inventory file.

```
[student@workstation inventory-review]$ ansible-navigator run site.yml \
> -i inventory.yml

Play name          Ok   Changed ... Skipped ... Task count  Progress
0|Ensure HAProxy is deployed  5       0 ...       2 ...           7  Complete
1|Ensure Apache is deployed   12      0 ...       2 ...          14  Complete
2|Ensure Web App is deployed  4       0 ...       0 ...           4  Complete

^f/PgUp page up     ^b/PgDn page down     ↑ scroll     esc back  ... Successful
```

Press ESC to exit from the `ansible-navigator` command.

5. You have decided to change your YAML inventory to use inventory hostnames that are mapped to their purposes, rather than using actual hostnames.

Modify the `inventory.yml` file as follows:

- In the `lb_servers` host group, change `servera.lab.example.com` to `loadbalancer_1`. Assign `loadbalancer_1` an `ansible_host` variable that configures its real hostname as `servera.lab.example.com`.
- In the `a_web_servers` host group, change `serverb.lab.example.com` to `backend_a1`. Assign `backend_a1` an `ansible_host` variable that configures its real hostname as `serverb.lab.example.com`.
- In the `b_web_servers` host group, change `serverc.lab.example.com` to `backend_b1`. Assign `backend_b1` an `ansible_host` variable that configures its real hostname as `serverc.lab.example.com`.

- 5.1. Modify the `inventory.yml` file so that each server follows its naming convention:

```

lb_servers:
  hosts:
    loadbalancer_1:
      ansible_host: servera.lab.example.com

web_servers:
  children:
    a_web_servers:
      hosts:
        backend_a1:
          ansible_host: serverb.lab.example.com
    b_web_servers:
      hosts:
        backend_b1:
          ansible_host: serverc.lab.example.com

```

6. Verify that the `site.yml` playbook runs without errors when you use the updated `inventory.yml` inventory file. You can use the `curl` command to test the load balancer again.

After the playbook runs without errors, commit the new `inventory.yml` file to your local Git repository. If prompted, then use `Student@123` as the Git password. After you have committed all project changes, push the local commits to the remote repository.

- 6.1. Verify that the `site.yml` playbook runs without errors, and that the playbook output contains inventory hostnames that follow the naming conventions.

```
[student@workstation inventory-review]$ ansible-navigator run site.yml \
> -i inventory.yml

Play name           Ok   Changed ... Skipped ... Task count  Progress
0|Ensure HAProxy is deployed  5      0 ...      2 ...          7  Complete
1|Ensure Apache is deployed  12     0 ...      2 ...         14  Complete
2|Ensure Web App is deployed 4      2 ...      0 ...          4  Complete

^f/PgUp page up      ^b/PgDn page down      ↑ scroll      esc back  ... Successful
```

Press ESC to exit from the `ansible-navigator` command.

- 6.2. Verify that requests to `servera` produce two different versions of the web application using `backend_a1` and `backend_b1`.

```
[student@workstation inventory-review]$ for x in 1 2; do curl servera; done
Hello from backend_a1. (version v1.1a)
Hello from backend_b1. (version v1.1b)
```

- 6.3. Commit the new `inventory.yml` file.

```
[student@workstation inventory-review]$ git add inventory.yml
[student@workstation inventory-review]$ git commit -m "Added YAML inventory"
...output omitted...
```

- 6.4. Verify that all project changes are committed, and push all local commits to the remote repository. If prompted, then use `Student@123` as the Git password.

```
[student@workstation inventory-review]$ git status
On branch exercise
Your branch is ahead of 'origin/exercise' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
[student@workstation inventory-review]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
```

Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the workstation machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- You can use inventory plug-ins provided by collections to dynamically obtain lists of hosts and groups from sources such as cloud, virtualization, and systems management platforms.
- Dynamic inventory scripts can generate host and group information from sources external to Ansible.
- You can use the `yaml` inventory plug-in to write static inventory files in YAML format.
- You can use the `ansible-navigator inventory` command to help you convert an inventory file in INI format into YAML format.
- A good practice is to keep most variables out of static inventory files.
- Take advantage of inventory groups and the `group_vars/` and `host_vars/` directories to improve project maintainability.
- Use the special inventory variables to control connections to hosts and use inventory hostnames to make playbook output more readable.

Chapter 6

Managing Task Execution

Goal

Control and optimize the execution of tasks by Ansible Playbooks.

Objectives

- Control automatic privilege escalation at the play, role, task, or block level.
- Configure tasks that can run before roles or after normal handlers, and simultaneously notify multiple handlers.
- Label tasks with tags, and run only tasks labeled with specific tags, or start playbook execution at a specific task.
- Optimize your playbook to run more efficiently, and use callback plug-ins to profile and analyze which tasks consume the most time.

Sections

- Controlling Privilege Escalation (and Guided Exercise)
- Controlling Task Execution (and Guided Exercise)
- Running Selected Tasks (and Guided Exercise)
- Optimizing Execution for Speed (and Guided Exercise)

Lab

- Managing Task Execution

Controlling Privilege Escalation

Objectives

- Control automatic privilege escalation at the play, role, task, or block level.

Privilege Escalation Strategies

Ansible Playbooks can achieve privilege escalation at many levels. Ansible uses directives or connection variables for the privilege escalation level you intend to control. For plays, roles, blocks, and tasks you need to use the privilege escalation directives. The directives are `become`, `become_user`, `become_method`, and `become_flags`.

Privilege Escalation by Configuration

If you have the `become` Boolean set to `true` (or `yes`) in the `privilege_escalation` section of your Ansible configuration file, then all plays in your playbook use privilege escalation by default. Those plays use the current value of `become_method` to change to the user specified by the `become_user` directive when running on the managed hosts.

You can also override the configuration file and specify privilege escalation settings when you run the playbook by using command-line options. The following table compares configuration directives and command-line options:

Privilege Escalation Directives and Options

Configuration directive	Command-line option
<code>become</code>	<code>--become</code> or <code>-b</code>
<code>become_method</code>	<code>--become-method=BECOME_METHOD</code>
<code>become_user</code>	<code>--become-user=BECOME_USER</code>
<code>become_password</code>	<code>--ask-become-pass</code> or <code>-K</code>



Important

If the Ansible configuration file specifies `become: false`, but the command line includes the `-b` option, then Ansible ignores the configuration file and uses privilege escalation by default.

You can get additional information about the precedence rules in *Controlling How Ansible Behaves: Precedence Rules* at https://docs.ansible.com/ansible/6/reference_appendices/general_precedence.html#general-precedence-rules.

Defining Privilege Escalation in Plays

When you write a playbook, you might need privilege escalation for some plays but not for all plays. You can specify which plays should use privilege escalation. If the play does not specify

to use privilege escalation, the default setting from the configuration file or the command line is used.

The following playbook contains three plays. The first play uses `become: true` to use privilege escalation no matter what the configuration file or command-line options specify. The second play uses `become: false` to not use privilege escalation, even if the configuration file or command-line options are specified. The third play does not have a `become` directive and uses privilege escalation based on the default settings from the Ansible configuration file or the command line. The `ansible_user_id` variable displays the name of the user on the managed host running the current play.

```
---
```

- name: Become the user "manager"
hosts: webservers
become: true
tasks:
 - name: Show the user used by this play
ansible.builtin.debug:
var: ansible_user_id
- name: Do not use privilege escalation
hosts: webservers
become: false
tasks:
 - name: Show the user used by this play
ansible.builtin.debug:
var: ansible_user_id
- name: Use privilege escalation based on defaults
hosts: webservers
tasks:
 - name: Show the user used by this play
ansible.builtin.debug:
var: ansible_user_id

To ensure that privilege escalation is used correctly by your play, specify the setting explicitly in the playbook. Specifying the escalation method or privileged user in configuration settings or inventory variables might be necessary depending on the task or the hosts in question.

Privilege Escalation in Tasks

You can enable or disable privilege escalation for specific tasks in a play. To do this, add the appropriate `become` directive to the task.

```
---
```

- name: Play with two tasks, one uses privilege escalation
hosts: all
become: false
tasks:
 - name: This task needs privileges
ansible.builtin.yum:
 - name: perl
 - state: installed**become: true**

```
- name: This task does not need privileges
  ansible.builtin.shell: perl -v
  register: perlcheck
  failed_when: perlcheck.rc != 0
```

In the preceding example, the play has privilege escalation disabled by default, but the first task uses privilege escalation.

Grouping Privilege Escalation Tasks with Blocks

If you have a subset of tasks in your play that require (or do not require) privilege escalation, you can set `become` on a block. All tasks in the block share the same privilege escalation setting, which overrides the setting made at the play level. The following examples illustrate things you can do with this mechanism:

- Enable privilege escalation for a subset of tasks in your play.
- Disable privilege escalation for a subset of tasks in your play.
- Add the `become_user` directive to use privilege escalation to perform a subset of tasks as some other regular user used by your application (such as a database user) instead of `root`.

To use this option, add the necessary directives to the `block` dictionary. In the following example, the playbook contains one play that has privilege escalation turned off by default. The play consists of a block (containing two tasks) and a separate task. The tasks in the block use privilege escalation, even though it is off by default for the play. The final task does not use privilege escalation because it is outside of the block.

```
---
- name: Deploy web services
  hosts: webservers
  become: false
  tasks:
    - name: Tasks that need privilege escalation
      block:
        - name: Ensure httpd is installed
          ansible.builtin.yum:
            name: httpd
            state: installed
        - name: Start and enable webserver
          ansible.builtin.service:
            name: httpd
            state: started
            enabled: true
      become: true
    - name: Test website from itself, do not become
      ansible.builtin.uri:
        url: http://{{ ansible_host }}
        return_content: true
      register: webpage
      failed_when: webpage.status != 200
```

Applying Privilege Escalation in Roles

You can use either of the following methods to perform privilege escalation:

- The first option is for the role itself to have privilege escalation variables set inside it or on its tasks. The documentation for the role might specify whether you have to set other variables, such as `become_method`, to use the role.
- You can also specify this information yourself in the Ansible configuration or playbook.

Alternatively, you can set the privilege escalation settings on the play that calls the role. You can also adjust these settings on individual roles, just like you can on individual tasks or blocks:

```
- name: Example play with one role
  hosts: localhost
  roles:
    - role: role-name
      become: true
```

Listing Privilege Escalation with Connection Variables

You can also use connection variables to configure privilege escalation. Apply these connection variables as inventory variables on groups or individual hosts. The following table compares playbook and configuration directives with connection variable names:

Privilege Escalation Directives and Connection Variables

Configuration or playbook directive	Connection variable
become	<code>ansible_become</code>
become_method	<code>ansible_become_method</code>
become_user	<code>ansible_become_user</code>
become_password	<code>ansible_become_pass</code>

You can use host-based or group-based connection variables such as `ansible_become`, `ansible_become_method`, `ansible_become_user`, and `ansible_become_pass` to override privilege escalation settings for individual hosts or groups that you originally set as configuration or playbook directives such as `become`, `become_method`, `become_user`, and `become_pass`.



Important

Connection variables override the `become` settings in the configuration file, as well as in plays, tasks, blocks, and roles.

Be cautious when using connection variables to configure privilege escalation because of their high precedence compared to the configuration and playbook directives.

The following example demonstrates configuring privilege escalation variables in a YAML inventory for a group:

Chapter 6 | Managing Task Execution

```
webservers:  
  hosts:  
    servera.lab.example.com:  
    serverb.lab.example.com:  
  vars:  
    ansible_become: true
```

You can set connection variables for privilege escalation at the host level. Doing this on a per-host basis is tedious, but can be helpful if one host in a larger group needs specific connection options. The following example demonstrates configuring privilege escalation by using connection variables at the host level:

```
webservers:  
  hosts:  
    servera.lab.example.com:  
      ansible_become: true  
    serverb.lab.example.com:
```

You can also set connection variables in the playbook; for example, on the play itself. This overrides the inventory variables (through normal variable precedence) and the setting of any become directives.

```
---  
- name: Example play using connection variables  
  hosts: webservers  
  vars:  
    ansible_become: true  
  tasks:  
    - name: Play will use privilege escalation even if inventory says no  
      ansible.builtin.yum:  
        name: perl  
        state: installed
```

Choosing Privilege Escalation Approaches

As you can see, Ansible provides many ways to control privilege escalation. Consider the following competing needs when choosing how to control privilege escalation:

- Run tasks with the least privilege when possible. This avoids inadvertent compromises and damage to managed hosts through playbook errors.
- Keep your playbook simple. This makes it easier for you to understand the playbook and whether it is running with elevated privileges.

Some users run their playbooks with privilege escalation always turned on. This approach is simple, and often the tasks that you are performing must run as `root`. Tasks that *do not* need to run as `root` also run with elevated privileges, increasing risk.

Some users connect to the `root` account to avoid privilege escalation. This is generally not a good practice. Anyone running the playbook must have `root` credentials on the managed hosts. This can also make it hard to determine which administrator performed which playbook run.

A good practice is to control which plays or tasks need privilege escalation selectively.

For example, if the apache user can start the httpd server, there is no need to run the task as the root user.

Ideally, you should configure privilege escalation in as simple a way as possible, and it should be clear to you if needed for a task. For example, you could use `become: true` to enable privilege escalation for a play and then use `become: false` to selectively disable privilege escalation for tasks that do not require it.

Alternatively, you could group tasks that need privilege escalation into one play and tasks that do not into another play (if compatible with the workflow). Setting privilege escalation in the configuration file might be necessary if a playbook requires privilege escalation, but you cannot edit that playbook for some reason.

Generally, if the playbook requires privilege escalation, it makes more sense to state that requirement. The biggest challenge is when different hosts that use a playbook require other privilege escalation methods to function correctly. In that case, it can make sense to set inventory variables such as `ansible_become_method` for those hosts or their group when enabling privilege escalation with `become` in the playbook.



References

Understanding Privilege Escalation – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/become.html

Blocks – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_blocks.html

► Guided Exercise

Controlling Privilege Escalation

Configure a playbook to escalate privileges only for specific plays, roles, tasks, or blocks that might need them to operate correctly.

Outcomes

- Select the appropriate escalation method and privilege isolation.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start task-escalation
```

Instructions

- 1. Clone the `https://git.lab.example.com/student/task-escalation.git` Git repository into the `/home/student/git-repos` directory and then create a new branch for this exercise.
- From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- Clone the `https://git.lab.example.com/student/task-escalation.git` repository and then change directory to the cloned repository:

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/task-escalation.git  
Cloning into 'task-escalation'...  
...output omitted...  
[student@workstation git-repos]$ cd task-escalation
```

- Create the `exercise` branch and check it out.

```
[student@workstation task-escalation]$ git checkout -b exercise  
Switched to a new branch 'exercise'
```

- 2. Review the global privilege escalation setting in the project `ansible.cfg` file. Notice that `become` is set to `false` so no tasks run with escalated privileges.

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

- 3. Examine the `intranet.yml` playbook to determine the tasks that require escalation.

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

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

    - name: firewalld enabled and running
      ansible.builtin.service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits http service
      ansible.posix.firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: true

    - name: httpd enabled and running
      ansible.builtin.service:
        name: httpd
        enabled: true
        state: started

    - name: Test intranet web server
      hosts: localhost
      tasks:
        - name: connect to intranet web server
```

Chapter 6 | Managing Task Execution

```
ansible.builtin.uri:  
  url: http://servera.lab.example.com  
  return_content: true  
  status_code: 200
```

The tasks from the first play need privilege escalation to run, but the one task from the second play does not. You can use any of the following methods to satisfy those requirements:

- Add `become: true` to the top of the first play. This setting causes every task in that play to run with privilege escalation unless otherwise specified.
- Wrap sequential tasks that need escalation with a `block` setting, and set privilege escalation for the whole block.

► 4. Add the privilege escalation to tasks in the first play by using a `block` statement.

```
---  
- name: Enable intranet services  
  hosts: servera.lab.example.com  
  tasks:  
    - name: Tasks that require privilege escalation  
      become: true  
      block:  
        - name: latest version of httpd and firewalld installed  
          ansible.builtin.yum:  
            name:  
              - httpd  
              - firewalld  
            state: latest  
  
        - name: test html page is installed  
          ansible.builtin.copy:  
            content: "Welcome to the example.com intranet!\n"  
            dest: /var/www/html/index.html  
  
        - name: firewalld enabled and running  
          ansible.builtin.service:  
            name: firewalld  
            enabled: true  
            state: started  
  
        - name: firewalld permits http service  
          ansible.posix.firewalld:  
            service: http  
            permanent: true  
            state: enabled  
            immediate: true  
  
        - name: httpd enabled and running  
          ansible.builtin.service:  
            name: httpd  
            enabled: true  
            state: started
```

Chapter 6 | Managing Task Execution

```
- name: Test intranet web server
hosts: localhost
tasks:
  - name: connect to intranet web server
    ansible.builtin.uri:
      url: http://servera.lab.example.com
      return_content: true
      status_code: 200
```

- 5. Use the `ansible-navigator` command to run the playbook:

```
[student@workstation task-escalation]$ ansible-navigator run intranet.yml

Play name          Ok  Changed ... Failed ... Task count  Progress
0|Enable intranet services   6      4 ...      0 ...           6  Complete
1|Test intranet web server   2      0 ...      0 ...           2  Complete

^f/PgUp page up     ^b/PgDn page down    ↑ scroll    esc back ... Successful
```

Press ESC to exit from the `ansible-navigator` command.

- 6. Test that the playbook ran correctly:

```
[student@workstation task-escalation]$ curl servera.lab.example.com
Welcome to the example.com intranet!
```

- 7. Remove the `block` statement from the first play and add the `become: true` directive. This statement enables privilege escalation for every task in the first play.

- 7.1. Instead of removing the `block` manually, use the `git` command to restore the original file.

```
[student@workstation task-escalation]$ git restore intranet.yml
```

- 7.2. Add privilege escalation to tasks in the first play by using a `become: true` statement. The second play does not require privilege escalation and inherits `become: false` from the settings defined in the `ansible.cfg` file.

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

  - name: test html page is installed
    ansible.builtin.copy:
      content: "Welcome to the example.com intranet!\n"
```

Chapter 6 | Managing Task Execution

```
dest: /var/www/html/index.html

- name: firewalld enabled and running
  ansible.builtin.service:
    name: firewalld
    enabled: true
    state: started

- name: firewalld permits http service
  ansible.posix.firewalld:
    service: http
    permanent: true
    state: enabled
    immediate: true

- name: httpd enabled and running
  ansible.builtin.service:
    name: httpd
    enabled: true
    state: started

- name: Test intranet web server
  hosts: localhost
  tasks:
    - name: connect to intranet web server
      ansible.builtin.uri:
        url: http://servera.lab.example.com
        return_content: true
        status_code: 200
```

► 8. Use the `ansible-navigator` command to run the playbook:

```
[student@workstation task-escalation]$ ansible-navigator run intranet.yml
```

Play name	Ok	Changed	...	Failed	...	Task count	Progress
0 Enable intranet services	6	0	...	0	...	6	Complete
1 Test intranet web server	2	0	...	0	...	2	Complete

^f/PgUp page up ^b/PgDn page down ↑↓ scroll esc back ... Successful

Press ESC to exit from the `ansible-navigator` command.

► 9. Test that the playbook ran correctly:

```
[student@workstation task-escalation]$ curl servera.lab.example.com
Welcome to the example.com intranet!
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish task-escalation
```

Controlling Task Execution

Objectives

- Configure tasks that can run before roles or after normal handlers, and simultaneously notify multiple handlers.

Controlling the Order of Execution

In a play, Ansible always runs the tasks from roles, called by the `roles` statement, before the tasks that you define under the `tasks` section. The following playbook contains both a `roles` and a `tasks` section.

```
---
- name: Ensure Apache is deployed
  hosts: www1.example.com
  gather_facts: false

  tasks:
    - name: Open the firewall
      ansible.posix.firewalld:
        service: http
        permanent: true
        state: enabled

  roles:
    - role: apache
```

When you run this playbook, Ansible runs the tasks from the role before the `Open the firewall` task, even though the `tasks` section is defined first:

```
[user@host ~]$ ansible-navigator run deploy_apache.yml -m stdout

PLAY [Ensure Apache is deployed] ****
TASK [apache : Ensure httpd packages are installed] ****
changed: [www1.example.com]

TASK [apache : Ensure httpd service is started and enabled] ****
changed: [www1.example.com]

TASK [Open the firewall] ****
changed: [www1.example.com]

PLAY RECAP ****
www1.example.com      : ok=3      changed=3      unreachable=0      ...
```

For readability, it is a good practice to write the `tasks` section after the `roles` section, so that the order of the playbook matches the order of execution.

**Note**

This order comes from the fact that each play listed in the playbook is a YAML dictionary of key-value pairs. The order of the top-level directives (`name`, `hosts`, `tasks`, `roles`, and so on) is arbitrary, but Ansible handles them in a standardized order when it parses and runs the play.

It is a good practice to write your plays in a consistent order starting with the name of the play, but Ansible does not require this. However, deviating from this convention can make it harder to read your playbook, and therefore changing the order is not recommended.

Importing or Including Roles as a Task

In recent versions of Ansible, you can include or import roles as a task instead of by using the `roles` section of the play. The advantage of this approach is that you can easily run a set of tasks, import or include a role, and then run more tasks. A potential disadvantage is that it might be less clear which roles your playbook uses without closely inspecting the playbook.

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 to include a role by using a task with the `include_role` module.

```
---
- name: Executing a role as a task
  hosts: remote.example.com

  tasks:
    - name: A normal task
      ansible.builtin.debug:
        msg: 'first task'

    - name: A task to include role2 here
      ansible.builtin.include_role:
        name: role2

    - name: Another normal task
      ansible.builtin.debug:
        msg: 'second task'
```

With `import_role`, the `ansible-navigator run` command starts by parsing and inserting the role in the play before starting the execution. Ansible detects and reports syntax errors immediately, and does not start executing the playbook if errors are detected.

With `include_role`, however, Ansible parses and inserts the role in the play when it reaches the `include_role` task, during the play execution. If Ansible detects syntax errors in the role, then execution of the playbook is aborted.

If you use the `when` directive with the `include_role` module, then Ansible does not parse the role if the condition in the `when` directive is false.

Defining Pre- and Post-tasks

You might want a play that runs some tasks, and the handlers they notify, before your roles. You might also want to run tasks in the play after your normal tasks and handlers run. There are two directives you can use instead of tasks to do this:

- pre_tasks is a tasks section that runs before the roles section.
- post_tasks is a tasks section that runs after the tasks section and any handlers notified by tasks.

The following playbook provides an example with pre_tasks, roles, tasks, post_tasks, and handlers sections. It is unusual for a play to contain all these sections.

```
---
- name: Deploying New Application Version
  hosts: webservers

  pre_tasks:
    # Stop monitoring the web server to avoid sending false alarms
    # while the service is updating.
    - name: Disabling Nagios for this host
      community.general.nagios:
        action: disable_alerts
        host: "{{ inventory_hostname }}"
        services: http
        delegate_to: nagios-srv

  roles:
    - role: deploy-content

  tasks:
    - name: Restarting memcached
      ansible.builtin.service:
        name: memcached
        status: restarted
      notify: Notifying the support team

    # Confirm that the application is fully operational
    # after the update.
    - name: Validating the new deployment
      ansible.builtin.uri:
        url: "http://{{ inventory_hostname }}/healthz"
        return_content: true
      register: result
      failed_when: "'OK' not in result.content"

  post_tasks:
    - name: Enabling Nagios for this host
      community.general.nagios:
        action: enable_alerts
        host: "{{ inventory_hostname }}"
        services: http
        delegate_to: nagios-srv

  handlers:
```

Chapter 6 | Managing Task Execution

```
# Send a message to the support team through Slack
# every time the memcached service is restarted.
- name: Notifying the support team
  community.general.slack:
    token: G922VJP25/D923DW937/3Ffe373sfhRE6y52Fg3rvf5G1K
    msg: 'Memcached on {{ inventory_hostname }} restarted'
    delegate_to: localhost
    become: false
```

Reviewing the Order of Execution

Ansible runs the play sections in the following order:

- `pre_tasks`
- Handlers that are notified in the `pre_tasks` section
- `roles`
- `tasks`
- Handlers that are notified in the `roles` and `tasks` sections
- `post_tasks`
- Handlers that are notified in the `post_tasks` section

The order of these sections in a play does not modify the order of execution, as given above. For example, if you write the `tasks` section before the `roles` section, then Ansible still runs the `roles` before the `tasks` in the `tasks` section. For readability, however, it is a good practice to organize your play in the order of execution: `pre_tasks`, `roles`, `tasks`, and `post_tasks`. You usually define the handlers at the end of the play.

Ansible runs and flushes notified handlers at several points during a run: after the `pre_tasks` section, after the `roles` and `tasks` sections, and after the `post_tasks` section. This means that a handler can run more than one time, at different times during play execution, if notified in multiple sections.

To immediately run any handlers that have been notified by a particular task in the play, add a task that uses the `meta` module with the `flush_handlers` parameter. This enables you to define specific points during task execution when all notified handlers are run.

In the following example, the play runs the `Restart api server` handler, if notified, after deploying a new configuration file and before using the application API. Without this call to the `meta` module, the play only calls the handler after running all of the tasks. If the handler has not run before the last task that uses the API, then that task might fail because the configuration file was updated, but the application has not yet reread the new configuration.

```
---
- name: Updating the application configuration and cache
  hosts: app_servers

  tasks:
    - name: Deploying the configuration file
      ansible.builtin.template:
        src: api-server.cfg.j2
        dest: /etc/api-server.cfg
        notify: Restart api server

    - name: Running all notified handlers
      ansible.builtin.meta: flush_handlers
```

```
- name: Asking the API server to rebuild its internal cache
  ansible.builtin.uri:
    url: "https://{{ inventory_hostname }}/rest/api/2/cache/"
    method: POST
    force_basic_auth: true
    user: admin
    password: redhat
    body_format: json
    body:
      type: data
      delay: 0
    status_code: 201

  handlers:
    - name: Restart api server
      ansible.builtin.service:
        name: api-server
        state: restarted
        enabled: true
```

Remember that in a play, handlers have global scope. A play can notify handlers defined in roles. One role can notify a handler defined by another role or by the play.

Ansible always runs handlers that have been notified in the order they are listed in the `handlers` section of the play, and not in the order in which they were notified.

Listening to Handlers

In addition to being notified by tasks, a handler can also *subscribe* to a specific notification, and run when that notification is triggered. This means that one notification can trigger multiple handlers.

By default, a handler runs when a notification string matches the handler name. However, because each handler must have a unique name, the only way to trigger multiple handlers at the same time is if each one subscribes to the same notification name.

The following example shows a task that notifies `My handlers` when it changes, which is whenever it runs, because it has `changed_when: true` set. The task notifies the `My handlers` handler and any handler that lists `My handlers` in a `listen` directive.

```
---
- name: Testing the listen directive
  hosts: localhost
  gather_facts: false
  become: false

  tasks:
    - name: Trigger handlers
      ansible.builtin.debug:
        msg: Trigerring the handlers
      notify: My handlers
      changed_when: true

  handlers:
```

```
# Listening to the "My handlers" event
- name: Listening to a notification
  ansible.builtin.debug:
    msg: First handler was notified
  listen: My handlers

# As an example, this handler is also triggered because
# its name matches the notification, but no two handlers
# can have the same name.
- name: My handlers
  ansible.builtin.debug:
    msg: Second handler was notified
```



Note

In the preceding playbook, it would be better if both handlers had unique names (not `My handlers`) and used `listen: My handlers`, because the playbook would be easier to read.

The `listen` directive is particularly helpful when used with roles. Roles use notifications to trigger their handlers. A role can document that it notifies a certain handler when an event occurs. Other roles or a play might be able to use this notification to run additional handlers defined outside the role.

For example, a role might notify one of its handlers when a service needs restarting. In your playbook, you can define a handler that listens to that notification and performs additional tasks when Ansible restarts the service, such as sending a message to a monitoring tool, or restarting a dependent service.

As another example, you can create a role that ensures the validity of a given SSL certificate and notifies a handler that renews the certificate if it has expired. In a playbook, you can call this role to verify your Apache HTTP Server certificate validity. You can add a handler that listens for that SSL certificate renewal notification and that restarts the `httpd` service.

Notifying Handlers

To summarize, a task can notify multiple handlers in at least two ways:

- It can notify a list of handlers individually by name.
- It can notify one name for which multiple handlers are configured to listen.

If the handlers are reused as a set by multiple tasks, then the easiest way to configure your playbook is to use the second approach with the `listen` directive. With this approach, you need only change the handlers to mark whether they are included in the set or not, and to ensure that they are listed in the correct order.

This approach is useful when a task needs to notify a lot of handlers. Instead of notifying each handler by name, the task only sends one notification. Ansible triggers all the handlers listening to that notification. This way, you can add or remove handlers without updating the task.

In the first approach, you must find and edit every affected task to add the handler to the list. You must also ensure that the handlers are listed in the correct order in the `handlers` section.

**Note**

It is an error to have a task send a notification with no handler matching that notification.

```
[user@host ~]$ cat no_handler_error.yml
---
- name: Testing notification with no handler
  hosts: localhost
  gather_facts: false
  become: false

  tasks:
    - name: Task that changes
      ansible.builtin.debug:
        msg: Trigerring a non existent handler
        changed_when: true
      notify: Restart service

[user@host ~]$ ansible-navigator run -m stdout no_handler_error.yml

PLAY [Testing notification with no handler] ****
TASK [Task that changes] ****
ERROR! The requested handler 'Restart service' was not found in either the
main handlers list nor in the listening handlers list
Please review the log for errors.
```

Controlling the Order of Host Execution

Ansible determines which hosts to manage for a play based on the `hosts` directive for the play. By default, Ansible runs the play against hosts in the order in which they are listed in the inventory. You can change that order on a play-by-play basis by using the `order` directive.

The following playbook alphabetically sorts the hosts in the `web_servers` group before running the task:

```
---
- name: Testing host order
  hosts: web_servers
  order: sorted

  tasks:
    - name: Creating a test file in /tmp
      ansible.builtin.copy:
        content: 'This is a sample file'
        dest: /tmp/test.out
```

The `order` directive accepts the following values:

`inventory`

The inventory order. This is the default value.

reverse_inventory

The reverse of the inventory order.

sorted

Sorts the hosts in alphabetical order. Numbers sort before letters.

reverse_sorted

Sorts the hosts in reverse alphabetical order.

shuffle

Randomizes the host list every time you run the play.



Note

Because Ansible normally runs each task in parallel on several hosts, the output of the `ansible-navigator` command might not reflect the expected order; the output shows the task completion order rather than the execution order.

For example, assume that the inventory is configured so that hosts are listed in alphabetical order, and plays are run on managed hosts in inventory order (the default behavior). You could still see output like this:

```
...output omitted...
TASK [Creating a test file in /tmp] ****
changed: [www2.example.com]
changed: [www3.example.com]
changed: [www1.example.com]
...output omitted...
```

Notice that the task on `www1.example.com` completed last.



References

Roles – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_reuse_roles.html

Handlers: Running Operations On Change – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_handlers.html

Re-using Ansible artifacts – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_reuse.html

► Guided Exercise

Controlling Task Execution

Use `pre_tasks` and `post_tasks` sections to control whether tasks run before or after roles, and `listen` directives to notify multiple handlers at the same time.

Outcomes

- Control the execution order of tasks.
- Trigger handlers using the `listen` directive.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/task-execution/` directory.

```
[student@workstation ~]$ lab start task-execution
```

Instructions

- 1. From a terminal, change to the `/home/student/task-execution/` directory and review the `deploy.yml` playbook.

```
[student@workstation ~]$ cd ~/task-execution
[student@workstation task-execution]$ cat deploy.yml
---
- name: Implementing Handlers
  hosts: web_servers

  pre_tasks:
    - name: Configuring Apache
      ansible.builtin.include_tasks: apache.yml

  roles:
    - role: firewall
```

The `deploy.yml` playbook consists of one play. This play uses a `pre_tasks` section to include tasks that configure an Apache HTTPD web server, and then a `roles` section to run a role that allows access through the firewall.

- 2. Edit the `deploy.yml` playbook to add tasks that notify handlers.

- 2.1. Use the `ansible.builtin.set_fact` module to assign the static values `80/tcp` for the `web_port` variable and `public` for the `web_zone` variable. Use `changed_when: true` to guarantee that the task always notifies the `display_variables` handler.

The content for the `pre_tasks` section should read as follows:

```
pre_tasks:
  - name: Configuring Apache
    ansible.builtin.include_tasks: apache.yml

  - name: Setting web port and zone for firewall
    ansible.builtin.set_fact:
      web_port: 80/tcp
      web_zone: public
      changed_when: true
    notify: display variables
```

- 2.2. After the `roles` section, add the `post_tasks` section with a task to copy the `index.html` file to the `/var/www/html` directory. Use the `ansible.builtin.copy` module to copy the file. Notify the `verify connectivity` handler if the task changes the managed host.

The content for the `post_tasks` section should read as follows:

```
roles:
  - role: firewall

post_tasks:
  - name: Ensure the web content is copied
    ansible.builtin.copy:
      src: index.html
      dest: /var/www/html/
    notify: verify connectivity
```



Note

You could define the `Ensure the web content is copied` task under a `tasks` section because Ansible also runs the `tasks` section after the `roles`. However, to mirror the `pre_tasks` section, define the task under the `post_tasks` section.

3. Add the handlers for the play. Two handlers must listen for the `display variables` notification in order to show the values for the `web_port` and the `web_zone` variables when they are defined. The same notification must trigger both handlers. In addition, add a handler named `verify connectivity` that verifies that the web content at `http://{{ ansible_facts['fqdn'] }}` can be retrieved at the end of the first play if the play changed the content. This confirms that the play worked.



Important

Make sure that you create this `handlers` section in the `deploy.yml` playbook at the end of the play.

- 3.1. Create the `handlers` section at the end of the play, and include two similar handlers using the `ansible.builtin.debug` module to print the value of the `web_port` and the `web_zone` variables. Give them an arbitrary string as a name, and use the `listen` directive for the `display variables` notification.

Chapter 6 | Managing Task Execution

The handlers section is highlighted and should consist of the following content:

```
post_tasks:  
  - name: Ensure the web content is copied  
    ansible.builtin.copy:  
      src: index.html  
      dest: /var/www/html/  
      notify: verify connectivity  
  
handlers:  
  - name: Showing the web port configured as pre_task  
    ansible.builtin.debug:  
      var: web_port  
    listen: display variables  
  
  - name: Showing the web zone configured as pre_task  
    ansible.builtin.debug:  
      var: web_zone  
    listen: display variables
```

- 3.2. Add the handler for the `verify connectivity` notification. Use the `ansible.builtin.uri` module to verify that the web content is available.

```
handlers:  
  - name: Showing the web port configured as pre_task  
    ansible.builtin.debug:  
      var: web_port  
    listen: display variables  
  
  - name: Showing the web zone configured as pre_task  
    ansible.builtin.debug:  
      var: web_zone  
    listen: display variables  
  
  - name: verify connectivity  
    ansible.builtin.uri:  
      url: http://{{ ansible_facts['fqdn'] }}  
      status_code: 200  
    become: false
```

The complete `deploy.yml` file should consist of the following content:

```
---  
- name: Implementing Handlers  
  hosts: web_servers  
  
  pre_tasks:  
    - name: Configuring Apache  
      ansible.builtin.include_tasks: apache.yml  
  
    - name: Setting web port and zone for firewall  
      ansible.builtin.set_fact:  
        web_port: 80/tcp
```

```

    web_zone: public
    changed_when: true
    notify: display variables

  roles:
    - role: firewall

  post_tasks:
    - name: Ensure the web content is copied
      ansible.builtin.copy:
        src: index.html
        dest: /var/www/html/
      notify: verify connectivity

  handlers:
    - name: Showing the web port configured as pre_task
      ansible.builtin.debug:
        var: web_port
      listen: display variables

    - name: Showing the web zone configured as pre_task
      ansible.builtin.debug:
        var: web_zone
      listen: display variables

    - name: verify connectivity
      ansible.builtin.uri:
        url: http://{{ ansible_facts['fqdn'] }}
        status_code: 200
      become: false

```

- ▶ 4. Run the `deploy.yml` Ansible Playbook to verify your work and confirm that Ansible triggers your handlers. Pay attention to the order of execution of the handlers. In particular, you should see that the handlers subscribed to `display variables` called in the `pre_tasks` section run before the `verify connectivity` handler called in the `post_tasks` section.

4.1. Run the `deploy.yml` playbook.

```

[student@workstation task-execution]$ ansible-navigator run \
> -m stdout deploy.yml

PLAY [Implementing Handlers] ****
TASK [Gathering Facts] ****
ok: [serverf.lab.example.com]

TASK [Configuring Apache] ****
included: /home/student/task-execution/apache.yml for serverf.lab.example.com

TASK [Installing apache] ****
changed: [serverf.lab.example.com]

TASK [Starting apache] ****

```

Chapter 6 | Managing Task Execution

```

changed: [serverf.lab.example.com]

TASK [Setting web port and zone for firewall] *****
changed: [serverf.lab.example.com]

RUNNING HANDLER [Showing the web port configured as pre_task] *****
ok: [serverf.lab.example.com] => {
    "web_port": "80/tcp"
}

RUNNING HANDLER [Showing the web zone configured as pre_task] *****
ok: [serverf.lab.example.com] => {
    "web_zone": "public"
}

TASK [firewall : Ensure Firewall Port Configuration] *****
changed: [serverf.lab.example.com] => (item={'port': '80/tcp', 'zone': 'public'})

RUNNING HANDLER [firewall : reload firewalld] *****
changed: [serverf.lab.example.com]

TASK [Ensure the web content is copied] *****
changed: [serverf.lab.example.com]

RUNNING HANDLER [verify connectivity]
*****
ok: [serverf.lab.example.com]

PLAY RECAP *****
serverf.lab.example.com      : ok=11    changed=6     unreachable=0    failed=0    ...

```

4.2. Use the `curl` command to verify that the playbook successfully updated the host:

```
[student@workstation task-execution]$ curl serverf.lab.example.com
This is the index file
```

- 5. Update two tasks in the `revert.yml` playbook to notify handlers and add a new handler to the playbook.

5.1. Add a `notify: package check` statement to the `Uninstall apache` task. Add a `notify: post` handler statement to the `Removing index.html file` task. After you modify the `revert.yml` file, the content for the playbook should read as follows:

```

...output omitted...
tasks:
  - name: Uninstall apache
    ansible.builtin.yum:
      name: httpd
      state: absent
      autoremove: true
      notify: package check

    post_tasks:

```

```
- name: Removing index.html file
ansible.builtin.file:
  path: /var/www/html/index.html
  state: absent
notify: post handler
...output omitted...
```

- 5.2. In the `revert.yml` playbook, create a handler called `post_handler` that uses the `ansible.builtin.debug` module to print The `index.html` file was deleted to stdout.

The complete `revert.yml` file should consist of the following content:

```
---
- name: Cleaning Firewall rules
hosts: web_servers

roles:
  - role: firewall
    vars:
      firewall_rules:
        - port: 80/tcp
          zone: public
          state: disabled

tasks:
  - name: Uninstall apache
    ansible.builtin.yum:
      name: httpd
      state: absent
      autoremove: true
    notify: package check

post_tasks:
  - name: Removing index.html file
    ansible.builtin.file:
      path: /var/www/html/index.html
      state: absent
    notify: post handler

handlers:
  - name: Gather package facts
    ansible.builtin.package_facts:
      manager: auto
      listen: package check

  - name: Check for the httpd package
    ansible.builtin.debug:
      msg: "httpd is not installed!"
    when: "'httpd' not in ansible_facts['packages']"
    listen: package check
```

Chapter 6 | Managing Task Execution

```
- name: post handler
  ansible.builtin.debug:
    msg: The index.html file was deleted
```

- 6. Run the `revert.yml` Ansible Playbook to verify your work and confirm that Ansible triggers your handlers. Pay attention to the order of execution of the handlers.

- 6.1. Run the `revert.yml` playbook.

```
[student@workstation task-execution]$ ansible-navigator run \
> -m stdout revert.yml

PLAY [Cleaning Firewall rules] ****
TASK [Gathering Facts] ****
ok: [serverf.lab.example.com]

TASK [firewall : Ensure Firewall Port Configuration] ****
changed: [serverf.lab.example.com] => (item={'port': '80/tcp', 'zone': 'public',
'state': 'disabled'})

TASK [Uninstall apache] ****
changed: [serverf.lab.example.com]

RUNNING HANDLER [firewall : reload firewalld] ****
changed: [serverf.lab.example.com]

RUNNING HANDLER [Gather package facts] ****
ok: [serverf.lab.example.com]

RUNNING HANDLER [Check for the httpd package] ****
ok: [serverf.lab.example.com] => {
  "msg": "httpd is not installed!"
}

TASK [Removing index.html file] ****
changed: [serverf.lab.example.com]

RUNNING HANDLER [post handler] ****
ok: [serverf.lab.example.com] => {
  "msg": "The index.html file was deleted"
}

PLAY RECAP ****
serverf.lab.example.com      : ok=9      changed=4      unreachable=0      failed=0    ...
```

- 6.2. Use the `curl` command to verify that the playbook successfully updated the host:

```
[student@workstation task-execution]$ curl serverf.lab.example.com
curl: (7) Failed to connect to serverf.lab.example.com port 80: No route to host
```

Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish task-execution
```

Running Selected Tasks

Objectives

- Label tasks with tags, run only tasks labeled with specific tags, or start playbook execution at a specific task.

Tagging Ansible Resources

When working with a large or complex playbook, you might consider running only a subset of its plays or tasks. You can apply *tags* to specific resources that you want to skip or run. A tag is a text label on an Ansible resource, such as a play or task. To tag a resource, use the `tags` keyword on the resource, followed by a list of the tags to apply.

When running a playbook with `ansible-navigator`, use the `--tags` option to filter the playbook and run only specific tagged plays or tasks. Tags are available for the following resources:

- Tag an entire play. Use the `tags` directive at the play level.

```
---
```

```
- name: Setup web services
  hosts: webservers
  tags:
    - setup
  ...output omitted...
```

- Tag each task. This is one of the most common ways that tags are used.

```
- name: Ensure that packages are installed
  ansible.builtin.yum:
    name: "{{ packages }}"
    state: installed
  tags:
    - install
```

- When you import a task file in a playbook, you can tag that task. Administrators can then set a global tag for the tasks loaded by the `import_tasks` directive.

```
- name: Import common web services tasks
  import_tasks: common.yml
  tags:
    - webproxy
    - webserver
```

- Tag a role in the `roles` section. All the tasks in the role are associated with this tag. In this example, the `databases` role has two tags, `production` and `staging`.

```
roles:  
  - { role: databases, tags: ['production', 'staging'] }  
  
• Tag a block of tasks. All the tasks in the block are associated with this tag. In this example, you group all httpd-related tasks under the webserver tag.  
  
- name: Install packages and start httpd service  
  block:  
    - name: Ensure httpd packages are installed  
      ansible.builtin.yum:  
        name:  
          - httpd  
          - php  
          - git  
          - php-mysqld  
        state: present  
  
    - name: Ensure SELinux allows httpd connections to a remote database  
      ansible.posix.seboolean:  
        name: httpd_can_network_connect_db  
        state: true  
        persistent: true  
  
    - name: Ensure httpd service is started and enabled  
      ansible.builtin.service:  
        name: httpd  
        state: started  
        enabled: true  
  
tags:  
  - webserver
```



Important

How Ansible applies tags to roles and task files depends on whether the role or task file was *imported* or *included*.

When you tag an imported resource, such as a role that uses the `roles` directive, or a task that uses the `ansible.builtin.import_role` or `ansible.builtin.import_tasks` modules, then Ansible applies the tag to all tasks in the imported role or task file.

When you tag an included resource, such as a task that uses the `ansible.builtin.include_role` or `ansible.builtin.include_tasks` modules, then Ansible applies the tag to the task itself and *only* runs tasks in the included role or task file if those tasks also use the same tag or the `always` tag.

If you plan to include a role or a task file, then you must also tag the tasks in the role or task file.

Managing Tagged Resources

Use the `ansible-navigator` command to run tasks with a specific tag, using the `--tags` option, or skip tasks with a specific tag, using the `--skip-tags` option.

Running Tasks with Specific Tags

The following playbook contains two tasks. The first task is tagged with the `webserver` tag. The second task does not have any associated tags.

```
---
- name: Example play using tagging
  hosts:
    - servera.lab.example.com
    - serverb.lab.example.com

  tasks:
    - name: httpd is installed
      ansible.builtin.yum:
        name: httpd
        state: latest
      tags: webserver

    - name: postfix is installed
      ansible.builtin.yum:
        name: postfix
        state: latest
```

To only run the first task, the `--tags` argument can be used:

```
[user@host ~]$ ansible-navigator run \
> -m stdout main.yml -i inventory --tags webserver

PLAY [Example play using tagging] ****

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

TASK [httpd is installed] ****
ok: [servera.lab.example.com]
ok: [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=0    unreachable=0    failed=0 ...
```

Because the `--tags` option was specified, the playbook ran only the task tagged with the `webserver` tag.

Skipping Tasks with Specific Tags

You can use the `--skip-tags` option to skip tasks with a specific tag and only run the tasks without that tag:

```
[user@host ~]$ ansible-navigator run \
> -m stdout main.yml -i inventory --skip-tags webserver

PLAY [Example play using tagging] ****

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

TASK [postfix is installed] ****
ok: [serverb.lab.example.com]
ok: [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 ...
```

Combining Tags to Run Multiple Tasks

You can associate multiple tags with the `--tags` and `--skip-tags` options using a comma-separated list:

```
[user@host ~]$ ansible-navigator run \
> -m stdout main.yml -i inventory --tags install,setup
```

Listing Tags in a Playbook

To list all tags that exist in a playbook, pass the `--list-tags` option to the `ansible-navigator` command. For example:

```
[user@host examples]$ ansible-navigator run \
> -m stdout playbook.yml -i inventory --list-tags

playbook: playbook.yml

play #1 (webservers): Setup web services TAGS: [setup]
  TASK TAGS: [setup]

play #2 (webservers): Teardown web services TAGS: [teardown]
  TASK TAGS: [teardown]
```

Assigning Special Tags

Ansible has a special tag that can be assigned in a playbook: `always`. A resource tagged `always` runs every time, even if it does not match the list of tags passed to the `--tags` option. The only exception is when it is explicitly skipped, using the `--skip-tags always` option.

A task that you tag with the `never` tag does not run, unless you run the playbook with the `--tags` option set to `never` or to one of the other tags associated with the task.

There are three additional special tags:

- The `tagged` tag runs any resource with an explicit tag.

- The untagged tag runs any resource that does not have an explicit tag, and excludes all tagged resources.
- The `all` tag includes all tasks in the play, whether they have a tag or not. This is the default behavior of Ansible.



References

Tags – Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_tags.html

► Guided Exercise

Running Selected Tasks

Use tags to run only specific tasks in a playbook.

Outcomes

- Tag specific tasks in a playbook to run or skip them.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start task-tagging
```

Instructions

- 1. Clone the `https://git.lab.example.com/student/task-tagging.git` repository to the `/home/student/git-repos` directory and then create a branch for this exercise.
- From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- Clone the `https://git.lab.example.com/student/task-tagging.git` repository and then change directory to the cloned repository:

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/task-tagging.git  
Cloning into 'task-tagging'...  
...output omitted...  
[student@workstation git-repos]$ cd task-tagging
```

- Create the `exercise` branch and check it out.

```
[student@workstation task-tagging]$ git checkout -b exercise  
Switched to a new branch 'exercise'
```

- 2. Create a new playbook to test the successful deployment of the web application.

- Open a new `test_webapp.yml` file for editing. Add a task that verifies that the content of the web application is accessible. Use the `uri` module to retrieve the contents. The file should consist of the following content:

```
---
- name: Web application smoke test
  hosts: web_servers
  gather_facts: false
  tasks:
    - name: Verify content of http://localhost
      ansible.builtin.uri:
        url: http://localhost
        return_content: true
      register: test_url
      failed_when: "'Hello from' not in test_url['content']"
      tags:
        - tests
```

- 2.2. Add the new playbook to the `site.yml` file. The file should consist of the following content:

```
- name: Deploy HAProxy
  ansible.builtin.import_playbook: deploy_haproxy.yml

- name: Deploy Web Server
  ansible.builtin.import_playbook: deploy_apache.yml

- name: Deploy Web App
  ansible.builtin.import_playbook: deploy_webapp.yml

- name: Test deployed Web App
  ansible.builtin.import_playbook: test_webapp.yml
```

- 3. Use the `site.yml` playbook to only test the web application deployment.

After the test fails, run the playbook to configure the environment, skipping all tests.

- 3.1. Use the `tests` tag to test the web application before it is set up.

```
[student@workstation task-tagging]$ ansible-navigator run \
> -m stdout site.yml --tags tests
...output omitted...
TASK [Verify content of http://localhost] *****
fatal: [serverc.lab.example.com]: FAILED! => {"ansible_facts": {
  "discovered_interpreter_python": "/usr/libexec/platform-python"}, "changed": false, "content": "", "elapsed": 0, "failed_when_result": true, "msg": "Status code was -1 and not [200]: Request failed: <urlopen error [Errno 111] Connection refused>", "redirected": false, "status": -1, "url": "http://localhost"}
fatal: [serverb.lab.example.com]: FAILED! => {"ansible_facts": {
  "discovered_interpreter_python": "/usr/libexec/platform-python"}, "changed": false, "content": "", "elapsed": 0, "failed_when_result": true, "msg": "Status code was -1 and not [200]: Request failed: <urlopen error [Errno 111] Connection refused>", "redirected": false, "status": -1, "url": "http://localhost"}
```

Chapter 6 | Managing Task Execution

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

This command ran only the task tagged with the `tests` tag; because the server was not set up, the test failed.

- 3.2. Run the playbook, skipping the tests to set up the web application.

```
[student@workstation task-tagging]$ ansible-navigator run \
> -m stdout site.yml --skip-tags tests
...output omitted...
PLAY RECAP ****
servera.lab.example.com    : ok=6      changed=6      unreachable=0      failed=0 ...
serverb.lab.example.com    : ok=6      changed=6      unreachable=0      failed=0 ...
serverc.lab.example.com    : ok=6      changed=6      unreachable=0      failed=0 ...
```

- 3.3. Use the `tests` tag again to test the web application.

```
[student@workstation task-tagging]$ ansible-navigator run \
> -m stdout site.yml --tags tests
...output omitted...
TASK [Verify content of http://localhost] ****
ok: [serverc.lab.example.com]
ok: [serverb.lab.example.com]

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

Now that the web application has been deployed, the test is successful.

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish task-tagging
```

Optimizing Execution for Speed

Objectives

- Optimize your playbook to run more efficiently, and use callback plug-ins to profile and analyze which tasks consume the most time.

Optimizing Playbook Execution

You can optimize your Ansible Playbooks in a number of ways. Writing efficient playbooks becomes increasingly important as the number of hosts you manage increases.

Optimizing the Infrastructure

Each release of automation execution environments adds enhancements and improvements. Running the latest version of Red Hat Ansible Automation Platform might help increase the speed of your playbooks as the core components of Ansible and especially the modules provided with it are optimized over time.

An architectural optimization that you can make is to keep your control node close to the managed nodes from a networking perspective. Ansible relies heavily on network communication and the transfer of data. High latency connections or low bandwidth between the control node and its managed hosts degrade the execution time of playbooks.

Disabling Fact Gathering

Each play has a hidden task which runs first, using the `ansible.builtin.setup` module to collect facts from each host. Those facts provide information about the nodes that plays can use through the `ansible_facts` variable.

Collecting the facts on each remote host takes time. If you do not use those facts in your play, skip the fact gathering task by setting the `gather_facts` directive to `false`.

The following play disables fact gathering:

```
---
- name: Demonstrate disabling the facts gathering
  hosts: web_servers
  gather_facts: false

  tasks:
    - ansible.builtin.debug:
        msg: "gather_facts is set to False"
```

The following example uses the Linux `time` command to compare the execution time of the previous playbook when fact gathering is enabled, and when it is disabled.

```
[user@host ~]$ time ansible-navigator run \
> -m stdout speed_facts.yml -i inventory
```

```

PLAY [Demonstrate activating the facts gathering] *****

TASK [Gathering Facts] *****
ok: [www1.example.com]
ok: [www2.example.com]
ok: [www3.example.com]

TASK [debug] *****
ok: [www1.example.com] => {
    "msg": "gather_facts is set to True"
}
ok: [www2.example.com] => {
    "msg": "gather_facts is set to True"
}
ok: [www3.example.com] => {
    "msg": "gather_facts is set to True"
}

PLAY RECAP *****
www1.example.com : ok=1    changed=0    unreachable=0    ...
www2.example.com : ok=1    changed=0    unreachable=0    ...
www3.example.com : ok=1    changed=0    unreachable=0    ...

real 0m6.171s
user 0m2.146s
sys 0m1.118s
[user@host ~]$ time ansible-navigator run \
> -m stdout speed_nofacts.yml -i inventory

PLAY [Demonstrate disabling the facts gathering] *****

TASK [debug] *****
ok: [www1.example.com] => {
    "msg": "gather_facts is set to False"
}
ok: [www2.example.com] => {
    "msg": "gather_facts is set to False"
}
ok: [www3.example.com] => {
    "msg": "gather_facts is set to False"
}

PLAY RECAP *****
www1.example.com : ok=1    changed=0    unreachable=0    ...
www2.example.com : ok=1    changed=0    unreachable=0    ...
www3.example.com : ok=1    changed=0    unreachable=0    ...

real 0m1.336s
user 0m1.116s
sys 0m0.246s

```

**Note**

To get the execution time of a playbook you can also use the Ansible `timer` callback plug-in, instead of using the `time` command. Callback plug-ins are discussed later in this section.

Playbooks often use the `ansible_facts['hostname']`, `ansible_hostname`, `ansible_facts['nodename']`, or `ansible_nodename` variables to refer to the host currently being processed. Those variables come from the fact gathering task, but you can usually replace them with the `inventory_hostname` and `inventory_hostname_short` magic variables.

Even if you disable fact gathering, you can choose to collect facts manually at any point in a play by running the `ansible.builtin.setup` module as a task, and the collected facts are then available for subsequent plays in the playbook.

Reusing Gathered Facts with Fact Caching

Ansible uses *cache plug-ins* to store gathered facts or inventory source data gathered by a play. You can take advantage of the fact cache to limit how many times you need to gather facts by reusing facts gathered earlier.

Fact caching is always enabled. You can only use one cache plug-in at a time. The `memory` cache plug-in is enabled by default if you do not change the `ansible-navigator` configuration. This plug-in caches facts gathered during the current Ansible run.

You can take advantage of this to improve performance for playbooks that contain multiple plays. The first play can gather facts for all the hosts for which you need facts in the playbook. Subsequent plays can then disable fact gathering and use the cached facts from the first play, improving performance.

The following playbook contains two simple plays and illustrates how this works.

```
- name: Gather facts for everyone
  hosts: all
  gather_facts: true

  # any tasks we might want for the first play
  # if you do not have tasks, "setup" will still run

- name: The next play, does not gather facts
  hosts: all
  gather_facts: false

  tasks:
    - name: Show that we still know the facts
      ansible.builtin.debug:
        var: ansible_facts
```

**Note**

Another way to use fact caching is to use `smart` gathering. You can set the following option for the `gathering` key in the `ansible.cfg` file to enable `smart` gathering:

```
[defaults]
gathering=smart
```

When enabled, `smart` gathering gathers facts on each new host in a playbook run, but if the same host is used across multiple plays, then the host is not contacted for fact gathering again in the run.

If you enable `smart` gathering, then removing both `gather_facts` lines in the previous playbook sample produces the same output. The first play gathers facts for all hosts, but the second play does not gather facts because they have already been gathered.

Fact caching also works on automation controller by default. In addition, when you edit a job template in automation controller, you can select the **Enable Fact Storage** checkbox. This changes the fact caching plug-in to one that stores facts gathered by jobs launched by that template, so that they can be reused between *multiple* playbook runs. (You need to periodically run a job that gathers facts to update the automation controller fact storage if you use this feature.)

If that checkbox is *not* selected, automation controller uses the default `memory` fact caching plug-in, which only caches facts during a particular job run.

**Important**

If facts might change from play to play in the same playbook, the disadvantage of relying on fact caching with the `memory` plug-in is that you do not get updated facts for the later plays. You can, of course, gather facts for any play that might be affected, or run the `ansible.builtin.setup` module manually as a task in a play that might be affected.

Limiting Fact Gathering

You can also selectively limit fact gathering if you disable automatic fact gathering and run the `ansible.builtin.setup` module as an explicit task with its `gather_subset` option. This is generally quicker than gathering all available facts.

The possible subsets include `all`, `min`, `hardware`, `network`, `virtual`, `ohai`, and `facter`. If you exclude the `all` subset, then you still get the `min` subset.

For example, if you only want to retrieve facts in the `network` subset, you can include it but exclude `all` and `min`:

```
- name: A play that gathers some facts
hosts: all
gather_facts: false

tasks:
  - name: Collect only network-related facts
```

Chapter 6 | Managing Task Execution

```
ansible.builtin.setup:  
  gather_subset:  
    - '!all'  
    - '!min'  
    - network
```

Increasing Parallelism

When Ansible is running a play, it runs the first task on every host in the current batch, and then runs the second task on every host in the current batch, and so on until the play completes. The `forks` parameter controls how many connections Ansible can have active at the same time. By default, this is set to 5, which means that even if there are 100 hosts to process on the current task, Ansible only communicates with them in groups of five. After it has communicated with all 100 hosts, Ansible moves to the next task.

By increasing the `forks` value, Ansible runs each task simultaneously on more hosts, and the playbook usually completes in less time. For example, if you set `forks` to 100, Ansible can attempt to open connections to all 100 hosts in the previous example simultaneously. This places more load on the control node, which still needs enough time to communicate with each of the hosts.

You can specify the number of forks to use in the Ansible configuration file, or you can use the `-f` option with the `ansible-navigator` command.

The following example shows the `forks` parameter set to 100 under the `[defaults]` section of the `ansible.cfg` configuration file.

```
[defaults]  
forks=100
```

Because the `forks` value specifies how many worker processes Ansible starts, a number that is too high might slow down your control node and your network. Try first with a conservative value, such as 20 or 50, and increase that number step by step, each time monitoring your system resources.

Avoiding Loops with the Package Manager Modules

Some modules accept a list of items to work on and do not require the use of a loop. This approach can increase efficiency, because the module is only called one time.

The modules for managing operating system packages work this way. The following example uses the `ansible.builtin.yum` module to install several packages in a single transaction, which is the most efficient way to install a group of packages.

```
---  
- name: Install the packages on the web servers  
  hosts: web_servers  
  become: true  
  gather_facts: false  
  
  tasks:  
    - name: Ensure the packages are installed  
      ansible.builtin.yum:  
        name:  
          - httpd
```

Chapter 6 | Managing Task Execution

```
- mod_ssl
- httpd-tools
- mariadb-server
- mariadb
- php
- php-mysqld
state: present
```

The preceding playbook would be equivalent to running the following command from the shell prompt:

```
[root@host ~]# yum install httpd mod_ssl httpd-tools \
> mariadb-server mariadb php php-mysqld
```

The following example is *not* efficient. It uses a loop to install the packages one at a time:

```
---
- name: Install the packages on the web servers
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure the packages are installed
      ansible.builtin.yum:
        name: "{{ item }}"
        state: present
      loop:
        - httpd
        - mod_ssl
        - httpd-tools
        - mariadb-server
        - mariadb
        - php
        - php-mysqld
```

The second example is equivalent to running multiple yum commands:

```
[root@host ~]# yum install httpd
[root@host ~]# yum install mod_ssl
[root@host ~]# yum install httpd-tools
[root@host ~]# yum install mariadb-server
[root@host ~]# yum install mariadb
[root@host ~]# yum install php
[root@host ~]# yum install php-mysqld
```

The second example is slower and less efficient because Ansible runs the `ansible.builtin.yum` module seven times, starting a process for the module seven times, and doing dependency resolution seven times.

Not all Ansible modules accept a list for the `name` parameter. For example, the `ansible.builtin.service` module only accepts a single value for its `name` parameter, and you need a loop to operate on multiple items:

Chapter 6 | Managing Task Execution

```
- name: Starting the services on the web servers
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure the services are started
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
        enabled: true
    loop:
      - httpd
      - mariadb
```

Use the `ansible-navigator doc` command to get information about what types of values different module arguments can accept:

```
[user@host ~]$ ansible-navigator doc ansible.builtin.yum -m stdout
...output omitted...
= name
  A package name or package specifier with version, like
  'name-1.0'.
  If a previous version is specified, the task also needs to
  turn `allow_downgrade` on. See the `allow_downgrade`
  documentation for caveats with downgrading packages.
  When using state=latest, this can be '*' which means run
  `yum -y update`.
  You can also pass a url or a local path to a rpm file (using
  state=present). To operate on several packages this can accept
  a comma separated string of packages or (as of 2.0) a list of
  packages.
  (Aliases: pkg)[Default: (null)]
...output omitted...
[user@host ~]$ ansible-navigator doc ansible.builtin.service -m stdout
...output omitted...
= name
  Name of the service.

  type: str
...output omitted...
```

Efficiently Copying Files to Managed Hosts

The `ansible.builtin.copy` module recursively copies files and directories to managed hosts. When the directory is large, with many files, the copy can take a long time. If you run the playbook multiple times, subsequent copies take less time because the module only copies the files that are different.

However, it is generally more efficient to use the `ansible.posix.synchronize` module to copy large numbers of files to managed hosts. This module uses `rsync` in the background and is usually faster than the `ansible.builtin.copy` module. By setting the `delete` option to `true`, the module can also remove files on the target that no longer exist on the source.

Chapter 6 | Managing Task Execution

The following playbook uses the `ansible.posix.synchronize` module to recursively copy the `web_content` directory to the web servers.

```
---
- name: Deploy the web content on the web servers
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure web content is updated
      ansible.posix.synchronize:
        src: web_content/
        dest: /var/www/html
```

Using Templates

The `ansible.builtin.lineinfile` module inserts or removes lines in a file, such as configuration directives in a configuration file. The following playbook updates the Apache HTTP Server configuration file by replacing several lines.

```
---
- name: Configure the Apache HTTP Server
  hosts: web_servers
  become: true
  gather_facts: false

  tasks:
    - name: Ensure proper configuration of the Apache HTTP Server
      ansible.builtin.lineinfile:
        dest: /etc/httpd/conf/httpd.conf
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
        state: present
      loop:
        - regexp: '^Listen 80$'
          line: 'Listen 8181'
        - regexp: '^ServerAdmin root@localhost'
          line: 'ServerAdmin support@example.com'
        - regexp: '^DocumentRoot "/var/www/html"'
          line: 'DocumentRoot "/var/www/web"'
        - regexp: '^<Directory "/var/www/html">'
          line: '<Directory "/var/www/web">'
```

When used with a loop, the `ansible.builtin.lineinfile` module is inefficient (and can be error-prone). In this situation, use either the `ansible.builtin.template` or the `ansible.builtin.copy` module instead.

```
---
- name: Configure the Apache HTTP Server
  hosts: web_servers
  become: true
```

Chapter 6 | Managing Task Execution

```
gather_facts: false

tasks:
  - name: Ensure proper configuration of the Apache HTTP Server
    ansible.builtin.template:
      src: httpd.conf.j2
      dest: /etc/httpd/conf/httpd.conf
```

The `httpd.conf.j2` template file in the previous example is the customized version of the `httpd.conf` file.

Enabling Pipelining

To run a task on a remote node, Ansible performs several SSH operations to copy the module and all its data to the remote node and to run the module. To increase the performance of your playbook, you can activate the pipelining feature. With pipelining, Ansible establishes fewer SSH connections.

To activate pipelining, set the `ANSIBLE_PIPELINING` environment variable to `true` in the `execution-environment` section of the `ansible-navigator.yml` configuration file.

```
---
ansible-navigator:
  ansible:
    config: ./ansible.cfg

  execution-environment:
    image: ee-supported-rhel8:latest
    pull-policy: missing
    environment-variables:
      set:
        ANSIBLE_PIPELINING: true
```

Ansible does not use pipelining by default because the feature requires that the `requiretty` sudo option on all the remote nodes be disabled. On Red Hat Enterprise Linux 8, that sudo option is disabled by default, but it might be active on other systems.

To disable the option, use the `visudo` command to edit the `/etc/sudoers` file on your managed nodes and disable the `requiretty` option:

```
[root@host ~]# visudo
...output omitted...
Defaults !requiretty
...output omitted...
```

Profiling Playbook Execution with Callback Plug-ins

Callback plug-ins extend Ansible by adjusting how it responds to various events. Some of these plug-ins modify the output of the command-line tools, such as the `ansible-navigator` command, to provide additional information. For example, the `timer` plug-in shows the playbook execution time in the output of the `ansible-navigator` command.

**Important**

Automation controller logs some information about jobs (playbook runs), which it extracts from the output of `ansible-navigator`. Because some callback plug-ins modify this output, you should use them with caution or avoid using them entirely, especially if you run the playbook with automation controller.

Ansible Automation Platform ships with a collection of callback plug-ins that you can enable in the `ansible.cfg` file by using the `callbacks_enabled` directive.

```
[defaults]
 callbacks_enabled=timer, profile_tasks, cgroup_perf_recap
```

Use the `ansible-navigator doc -t callback -l -m stdout` command to list the available callback plug-ins.

```
[user@host ~]$ ansible-navigator doc -t callback -l -m stdout
amazon.aws.aws_resource_actions summarizes all "resource:actions" completed
ansible.posix.cgroup_perf_recap Profiles system activity of tasks and full
execution using cgroups
ansible.posix.debug      formatted stdout/stderr display
ansible.posix.json       Ansible screen output as JSON
ansible.posix.profile_roles adds timing information to roles
ansible.posix.profile_tasks adds time information to tasks
ansible.posix.skippy     Ansible screen output that ignores skipped status
ansible.posix.timer      Adds time to play stats
awx_display              Playbook event dispatcher for ansible-runner
default                  default Ansible screen output
junit                    write playbook output to a JUnit file
minimal                 Ad hoc event dispatcher for ansible-runner
oneline                 oneline Ansible screen output
redhat.rhv.stdout        Output the log of ansible
redhat.satellite.foreman Sends events to Foreman
tree                     Save host events to files
```

Run the `ansible-navigator doc -t callback plug-in-name -m stdout` command to access the documentation for a specific plug-in.

```
[user@host ~]$ ansible-navigator doc -t callback cgroup_perf_recap -m stdout
> ANSIBLE.POSIX.CGROUP_PERF_RECAP (/usr/share/ansible/collections/
ansible_collections/ansible/posix/plugins/callback/cgroup_perf_recap.py)
```

This is an ansible callback plugin utilizes cgroups to profile system activity of ansible and individual tasks, and display a recap at the end of the playbook execution

OPTIONS (= is mandatory):

= control_group
Name of cgroups control group

set_via:

```
env:  
  - name: CGROUP_CONTROL_GROUP  
  
...output omitted...
```

Timing Tasks and Roles

You can use the `timer`, `profile_tasks`, and `profile_roles` callback plug-ins to help identify slow tasks and roles.

The `timer` plug-in displays the duration of playbook execution.

The `profile_tasks` plug-in displays the start time of each task, and the time spent on each task, sorted in descending order, at the end of the playbook execution.

The `profile_roles` plug-in displays the time spent on each role at the end of the output, sorted in descending order.

To activate these plug-ins, add or update the `callbacks_enabled` directive in the `ansible.cfg` file.

```
[defaults]  
callbacks_enabled=timer, profile_tasks, profile_roles
```

You do not have to enable all three plug-ins; select the ones that you need.

The following example shows the output of the `ansible-navigator run` command when you activate the three plug-ins.

```
[user@host ~]$ ansible-navigator run \  
> -m stdout deploy_webservers.yml  
...output omitted...  
PLAY RECAP *****  
www1.example.com : ok=9    changed=7    unreachable=0 ...  
www2.example.com : ok=10   changed=9    unreachable=0 ...  
www3.example.com : ok=10   changed=9    unreachable=0 ...  
  
Playbook run took 0 days, 0 hours, 0 minutes, 21 seconds  
Wednesday 08 September 2021 19:12:31 +0000 (0:00:00.858)      0:00:21.325 ***  
=====  
apache : Ensure httpd packages are installed ----- 7.14s  
haproxy : Ensure haproxy packages are present ----- 1.78s  
apache : Ensure SELinux allows httpd connections to a remote database --- 1.72s  
Gathering Facts ----- 3.89s  
haproxy : Ensure haproxy configuration is set ----- 1.12s  
haproxy : Ensure haproxy is started and enabled ----- 0.92s  
webapp : Ensure stub web content is deployed ----- 0.86s  
firewall : Ensure Firewall Sources Configuration ----- 0.85s  
firewall : Ensure Firewall Port Configuration ----- 0.84s  
firewall : Ensure Firewall Service Configuration ----- 0.79s  
apache : Ensure httpd service is started and enabled ----- 0.71s  
firewall : Firewall Port Configuration ----- 0.70s  
Wednesday 08 September 2021 19:12:31 +0000 (0:00:00.867)      0:00:21.333 ***  
=====
```

apache	-----	9.57s
gather_facts	-----	3.89s
haproxy	-----	3.81s
firewall	-----	3.18s
webapp	-----	0.86s
<hr/>		
total	-----	21.31s



References

ssh_config(5) and sudoers(5) man pages

Ansible Configuration Settings – Ansible Documentation

https://docs.ansible.com/ansible/6/reference_appendices/config.html

Callback Plugins – Ansible Documentation

<https://docs.ansible.com/ansible/6/plugins/callback.html>

► Guided Exercise

Optimizing Execution for Speed

Optimize a playbook for more efficient execution, analyzing the playbook's performance with callback plug-ins.

Outcomes

- Activate Ansible callback plug-ins in the configuration file.
- Profile a playbook execution time.
- Optimize a playbook.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/task-speed/` directory.

```
[student@workstation ~]$ lab start task-speed
```

Instructions

- 1. From a terminal, change to the `/home/student/task-speed/` directory and review the `deploy_webservers.yml` playbook.
- 1.1. Change to the `/home/student/task-speed/` directory.

```
[student@workstation ~]$ cd ~/task-speed  
[student@workstation task-speed]$
```

- 1.2. Review the contents of the `deploy_webservers.yml` playbook.

```
[student@workstation task-speed]$ cat deploy_webservers.yml  
---  
- name: Deploy the web servers  
  hosts: web_servers  
  become: true  
  
  tasks:  
    - name: Ensure required packages are installed  
      ansible.builtin.yum:  
        name: "{{ item }}"  
        state: present  
      loop:  
        - httpd  
        - mod_ssl  
        - httpd-tools
```

Chapter 6 | Managing Task Execution

```

- mariadb-server
- mariadb
- php
- php-mysqld

- name: Ensure the services are enabled
  ansible.builtin.service:
    name: "{{ item }}"
    state: started
    enabled: true
  loop:
    - httpd
    - mariadb

- name: Ensure the web content is installed
  ansible.builtin.copy:
    src: web_content/
    dest: /var/www/html

```

The playbook installs packages, starts services, and recursively copies a local directory to the managed nodes.

- ▶ 2. Activate the `timer` and the `profile_tasks` callback plug-ins, and then run the `deploy_webservers.yml` playbook.
 - 2.1. Edit the `ansible.cfg` file, and then add the `callbacks_enabled` directive with the two callback plug-ins that you need to activate.

```
[defaults]
inventory=inventory.yml
remote_user=devops
callbacks_enabled=timer, profile_tasks
```

- 2.2. Run the `deploy_webservers.yml` playbook, and take note of the total elapsed time.

```
[student@workstation task-speed]$ ansible-navigator run \
> -m stdout deploy_webservers.yml
...output omitted...
PLAY RECAP ****
serverb.lab.example.com    : ok=4      changed=3      unreachable=0      ...
serverc.lab.example.com    : ok=4      changed=3      unreachable=0      ...

Playbook run took 0 days, 0 hours, 1 minutes, 22 seconds
Wednesday 09 November 2022  20:27:38 +0000 (0:00:34.966) 0:01:22.396 ****
=====
Ensure required packages are installed ----- 38.29s
Ensure the web content is installed ----- 34.97s
Ensure the services are enabled ----- 7.14s
Gathering Facts ----- 1.96s
```

The playbook and task execution times are probably different on your system. However, web content deployment and package installations should be the most time-consuming tasks.

Chapter 6 | Managing Task Execution

- 3. Optimize the `deploy_webservers.yml` playbook. To do so, disable fact gathering, because the playbook does not use facts. Also remove the loop in the package installation task and replace the `ansible.builtin.copy` module with the `ansible.posix.synchronize` module.

If you make a mistake, you can restore the original file from the `deploy_webservers.yml.backup` file.

- 3.1. Edit the `deploy_webservers.yml` playbook. Add the `gather_facts` directive and set it to `false`. Do not close the file yet.

```
---
```

```
- name: Deploy the web servers
  hosts: web_servers
  become: true
  gather_facts: false

...output omitted...
```

- 3.2. Remove the loop from the package installation task. To do so, move the list of packages under the `name` parameter and remove the `loop` directive.

```
...output omitted...
tasks:
  - name: Ensure required packages are installed
    ansible.builtin.yum:
      name:
        - httpd
        - mod_ssl
        - httpd-tools
        - mariadb-server
        - mariadb
        - php
        - php-mysqlnd
      state: present

...output omitted...
```

The next task uses the `ansible.builtin.service` module to start the `httpd` and `mariadb` services. This task also uses a loop. However, in contrast to the `ansible.builtin.yum` module, the `ansible.builtin.service` module does not accept a list of services in its `name` parameter. Leave this task as it is.

- 3.3. In the last task, replace the `ansible.builtin.copy` module with the `ansible.posix.synchronize` module, which is more efficient when deploying large directories.

```
...output omitted...
- name: Ensure the web content is installed
  ansible.posix.synchronize:
    src: web_content/
    dest: /var/www/html
```

Save and close the file when done.

Chapter 6 | Managing Task Execution

- 4. Run the `clean.yml` playbook to clean up the managed hosts. When done, run the optimized `deploy_webservers.yml` playbook and compare the execution time with its first execution.

- 4.1. Run the `clean.yml` playbook.

```
[student@workstation task-speed]$ ansible-navigator run -m stdout clean.yml  
...output omitted...
```

- 4.2. Run the `deploy_webservers.yml` playbook.

```
[student@workstation task-speed]$ ansible-navigator run \  
> -m stdout deploy_webservers.yml  
...output omitted...  
PLAY RECAP *****  
serverb.lab.example.com : ok=3    changed=3    unreachable=0    ...  
serverc.lab.example.com : ok=3    changed=3    unreachable=0    ...  
  
Playbook run took 0 days, 0 hours, 0 minutes, 22 seconds  
Wednesday 09 November 2022  20:40:41 +0000 (0:00:01.524) 0:00:22.276 ***  
=====  
Ensure required packages are installed ----- 17.16s  
Ensure the services are enabled ----- 3.56s  
Ensure the web content is installed ----- 1.52s
```

Notice that the fact gathering task is absent. The playbook and task execution times should also be reduced.

- 5. Disable the `timer` and the `profile_tasks` callback plug-ins.

- 5.1. Disable the callback plug-ins by deleting the `callbacks_enabled` line in the `ansible.cfg` file.

```
[student@workstation task-speed]$ cat ansible.cfg  
[defaults]  
inventory=inventory.yml  
remote_user=devops
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish task-speed
```

▶ Lab

Managing Task Execution

Optimize the execution of plays and tasks in a playbook by selectively escalating privileges, configuring when tasks run, optimizing it for speed, and adding tags to specify that selected parts of it should run.

Outcomes

- Change privilege escalation to a more secure configuration.
- Add task hooks and handlers to alter the task behavior.
- Tag tasks to control their execution.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. When you are ready to push your changes to the remote repository, use `Student@123` as the Git password.

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

Instructions

1. Clone the `https://git.lab.example.com/student/task-review.git` repository to the `/home/student/git-repos` directory and then create the `exercise` branch.
2. Modify the `ansible.cfg` file so that privilege escalation is disabled by default. Because all the tasks in the `firewall`, `haproxy`, `apache`, and `webapp` roles require privilege escalation, enable privilege escalation at the play level in the `deploy_apache.yml`, `deploy_haproxy.yml`, and `deploy_webapp.yml` playbooks.
3. In the `deploy_haproxy.yml` playbook, add a copy task that runs before any other task in the play. That task must write the text `Playbook site.yml ready to start` to the `/tmp/site.ready` file on the servers in the `lb_servers` group.
4. Add a handler to the `haproxy` role that writes the message `Reloaded` to the `/tmp/haproxy.status` file. Notify the handler if the task that uses the `template` module in the `haproxy` role reports that the state of the system changed. Use `haproxy_filehandler` as the name of the handler.
5. Add the `apache_installer` tag to the `yum` task in the `apache` role.
6. Enable the `timer` and `profile_tasks` callback plug-ins for the playbook. The two plug-ins are part of the `ansible.posix` collection. If desired, then you can specify the FQCNs for the callback plug-ins.
7. Use automation content navigator to run the `site.yml` playbook in `stdout` mode. Analyze the output to find the task that uses the most time.

8. Refactor the most expensive task to make it more efficient. When done, run the `cleanup_apache.yml` playbook to clean up the web servers so that you can compare the results. To verify the changes, run the `site.yml` playbook with an appropriate tag so that you only run the modified task.
9. Commit and push your changes to the remote Git repository.

Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the `workstation` machine, change to the student user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

► Solution

Managing Task Execution

Optimize the execution of plays and tasks in a playbook by selectively escalating privileges, configuring when tasks run, optimizing it for speed, and adding tags to specify that selected parts of it should run.

Outcomes

- Change privilege escalation to a more secure configuration.
- Add task hooks and handlers to alter the task behavior.
- Tag tasks to control their execution.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. When you are ready to push your changes to the remote repository, use `Student@123` as the Git password.

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

Instructions

1. Clone the `https://git.lab.example.com/student/task-review.git` repository to the `/home/student/git-repos` directory and then create the `exercise` branch.
 - 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the repository and then change to the `/home/student/git-repos/task-review/` directory.

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/task-review.git  
Cloning into 'task-review'...  
...output omitted...  
[student@workstation git-repos]$ cd task-review/
```

- 1.3. Create the `exercise` branch and check it out.

```
[student@workstation task-review]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

2. Modify the `ansible.cfg` file so that privilege escalation is disabled by default.

Because all the tasks in the `firewall`, `haproxy`, `apache`, and `webapp` roles require privilege escalation, enable privilege escalation at the play level in the `deploy_apache.yml`, `deploy_haproxy.yml`, and `deploy_webapp.yml` playbooks.

- 2.1. Edit the `ansible.cfg` file to remove the `become=true` entry from the `privilege_escalation` block, use `become=false` in the `privilege_escalation` block, or remove the entire `[privilege_escalation]` block from the `ansible.cfg` file. If you choose the last option, then the file contains the following content:

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

- 2.2. Enable privilege escalation for the play in the `deploy_apache.yml` playbook. Add the `become: true` line to the play.

```
---
- name: Ensure Apache is deployed
  hosts: web_servers
  force_handlers: true
  gather_facts: false
  become: true

  roles:
    # The "apache" role has a dependency on the "firewall" role.
    # The "firewall" role requires the "firewall_rules" variable.
    - role: apache
```

- 2.3. Enable privilege escalation for the play in the `deploy_haproxy.yml` playbook. Add the line `become: true` to the play.

```
---
- name: Ensure HAProxy is deployed
  hosts: lb_servers
  force_handlers: true
  gather_facts: false
  become: true

  roles:
    # The "haproxy" role has a dependency on the "firewall" role.
    # The "firewall" role requires the "firewall_rules" variable.
    - role: haproxy
```

- 2.4. Enable privilege escalation for the play in the `deploy_webapp.yml` playbook. Add the line `become: true` to the play.

```

---
- name: Ensure Web App is deployed
  hosts: web_servers
  gather_facts: false
  become: true
  vars:
    - webapp_version: v1.0

  roles:
    - role: webapp

```

3. In the `deploy_haproxy.yml` playbook, add a copy task that runs before any other task in the play. That task must write the text `Playbook site.yml ready to start` to the `/tmp/site.ready` file on the servers in the `lb_servers` group.
- 3.1. Edit the `deploy_haproxy.yml` file. In the play in that file, add a `pre_tasks` block consisting of a copy task that creates a `/tmp/site.ready` file containing the text `Playbook site.yml ready to start`. If desired, then you can add a newline character to the end of the text string. The modified file contains the following content:

```

---
- name: Ensure HAProxy is deployed
  hosts: lb_servers
  force_handlers: true
  gather_facts: false
  become: true

  pre_tasks:
    - name: Setting the maintenance message
      ansible.builtin.copy:
        dest: /tmp/site.ready
        content: "Playbook site.yml ready to start\n"

  roles:
    # The "haproxy" role has a dependency on the "firewall" role.
    # The "firewall" role requires the "firewall_rules" variable.
    - role: haproxy

```

4. Add a handler to the `haproxy` role that writes the message `Reloaded` to the `/tmp/haproxy.status` file. Notify the handler if the task that uses the `template` module in the `haproxy` role reports that the state of the system changed. Use `haproxy_filehandler` as the name of the handler.
- 4.1. Edit the `roles/haproxy/handlers/main.yml` file and add the handler that copies the text to the file. If desired, then you can add a newline character to the end of the text string. The modified file contains the following content:

```

---
# handlers file for haproxy

- name: restart haproxy
  ansible.builtin.service:
    name: haproxy

```

```

state: restarted

- name: reload haproxy
  ansible.builtin.service:
    name: haproxy
    state: reloaded

- name: haproxy filehandler
  ansible.builtin.copy:
    dest: /tmp/haproxy.status
    content: "Reloaded\n"

```

- 4.2. Edit the `roles/haproxy/tasks/main.yml` file and modify the `Ensure haproxy configuration is set` task so that it notifies the `haproxy filehandler` handler if the task reports that it changed the current system. Because the modified task notifies two handlers, the existing `reload haproxy` handler must be moved to a separate line. The modified file contains the following content:

```

---
# tasks file for haproxy

- name: Ensure haproxy packages are present
  ansible.builtin.yum:
    name:
      - haproxy
      - socat
    state: present

- name: Ensure haproxy is started and enabled
  ansible.builtin.service:
    name: haproxy
    state: started
    enabled: true

- name: Ensure haproxy configuration is set
  ansible.builtin.template:
    src: haproxy.cfg.j2
    dest: /etc/haproxy/haproxy.cfg
    owner: root
    group: root
    mode: 0644
  notify:
    - reload haproxy
    - haproxy filehandler

```

5. Add the `apache_installer` tag to the `yum` task in the `apache` role.

- 5.1. Edit the `roles/apache/tasks/main.yml` file and add the `apache_installer` tag to the `yum` task. The modified file contains the following content:

```

---
# tasks file for apache

- name: Ensure httpd packages are installed

```

Chapter 6 | Managing Task Execution

```

ansible.builtin.yum:
  name: "{{ item }}"
  state: present
loop:
  - httpd
  - php
  - git
  - php-mysqld
tags: apache_installer

- name: Ensure SELinux allows httpd connections to a remote database
  ansible.posix.seboolean:
    name: httpd_can_network_connect_db
    state: true
    persistent: true

- name: Ensure httpd service is started and enabled
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true

```

- 6.** Enable the `timer` and `profile_tasks` callback plug-ins for the playbook. The two plug-ins are part of the `ansible.posix` collection. If desired, then you can specify the FQCNs for the callback plug-ins.

- 6.1. Edit the `ansible.cfg` configuration file and add the plug-ins to the `callbacks_enabled` directive. The modified file contains the following content:

```

[defaults]
inventory=inventory
remote_user=devops
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks

```

- 7.** Use automation content navigator to run the `site.yml` playbook in `stdout` mode. Analyze the output to find the task that uses the most time.

- 7.1. Run the `site.yml` playbook in `stdout` mode. Your output looks similar to the following.

```

[student@workstation task-review]$ ansible-navigator run site.yml -m stdout
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=8    changed=8    unreachable=0    failed=0 ...
serverb.lab.example.com : ok=5    changed=4    unreachable=0    failed=0 ...
serverc.lab.example.com : ok=5    changed=4    unreachable=0    failed=0 ...
Playbook run took 0 days, 0 hours, 0 minutes, 50 seconds
Monday 07 November 2022 21:27:18 +0000 (0:00:00.788)      0:00:50.513 ****
=====
apache : Ensure httpd packages are installed ----- 25.32s
haproxy : Ensure haproxy packages are present ----- 4.55s
haproxy : reload haproxy ----- 3.81s

```

```
haproxy : haproxy filehandler ----- 3.60s
firewall : reload firewalld ----- 2.25s
...output omitted...
```

- 7.2. The previous output sorted the tasks based on how long it took for each task to complete. The following task took the most time:

```
apache : Ensure httpd packages are installed ----- 25.32s
```

8. Refactor the most expensive task to make it more efficient. When done, run the `cleanup_apache.yml` playbook to clean up the web servers so that you can compare the results. To verify the changes, run the `site.yml` playbook with an appropriate tag so that you only run the modified task.
- 8.1. Identify the file containing the `Ensure httpd packages are installed` task. The `-R` option performs a recursive search and the `-l` option restricts the output to only show the file name. In addition to the `roles/apache/tasks/main.yml` file, your output displays one or more `site-artifact` files.

```
[student@workstation task-review]$ grep -Rl 'Ensure httpd packages are installed'
roles/apache/tasks/main.yml
...output omitted...
```

- 8.2. Edit the `roles/apache/tasks/main.yml` task file to remove the loop from the `yum` task and instead install all the packages in one transaction. The modified file contains the following content:

```
---
# tasks file for apache

- name: Ensure httpd packages are installed
  ansible.builtin.yum:
    name:
      - httpd
      - php
      - git
      - php-mysqld
    state: present
    tags: apache_installer

- name: Ensure SELinux allows httpd connections to a remote database
  ansible.posix.seboolean:
    name: httpd_can_network_connect_db
    state: true
    persistent: true

- name: Ensure httpd service is started and enabled
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true
```

- 8.3. Run the `cleanup_apache.yml` playbook to clean up the web servers.

Chapter 6 | Managing Task Execution

```
[student@workstation task-review]$ ansible-navigator run cleanup_apache.yml \
> -m stdout
...output omitted...
```

- 8.4. Use automation content navigator to run the `site.yml` playbook with the `apache_installer` tag in `stdout` mode. Verify that the `Ensure httpd` packages are installed task takes less time to complete.

```
[student@workstation task-review]$ ansible-navigator run site.yml \
> --tags apache_installer -m stdout
...output omitted...
PLAY RECAP ****
serverb.lab.example.com    : ok=1    changed=1    unreachable=0    failed=0 ...
serverc.lab.example.com    : ok=1    changed=1    unreachable=0    failed=0 ...
Playbook run took 0 days, 0 hours, 0 minutes, 15 seconds
Monday 07 November 2022 21:56:20 +0000 (0:00:14.971)      0:00:15.060 ****
=====
apache : Ensure httpd packages are installed ----- 14.97s
```

9. Commit and push your changes to the remote Git repository.

- 9.1. Add the changed files, commit the changes, and push them to the Git repository. If prompted, then use `Student@123` as the Git password.

```
[student@workstation task-review]$ git add .
[student@workstation task-review]$ git commit -m "Lab updates"
[exercise 0a55370] Lab updates
 7 files changed, 19 insertions(+), 10 deletions(-)
[student@workstation task-review]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
```

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

Summary

- You can configure privilege escalation at the play, role, block, or task level.
- Privilege escalation uses the `become`, `become_user`, `become_method`, and `become_flags` directives.
- Ansible runs the play sections in the following order: `pre_tasks`, `roles`, `tasks`, and `post_tasks`.
- Handlers run in the same order that they appear in the play.
- You can use tags to select or skip tasks during play execution.
- Disabling fact gathering speeds up the execution of playbooks.
- You can use the `timer`, `profile_tasks`, and `profile_roles` callback plug-ins to profile playbooks.

Chapter 7

Transforming Data with Filters and Plug-ins

Goal

Populate, manipulate, and manage data in variables using filters and plug-ins.

Objectives

- Format, parse, and define the values of variables using filters.
- Populate variables with data from external sources using lookup plug-ins.
- Implement loops using structures other than simple lists by using lookup plug-ins and filters.
- Use filters to inspect, validate, and manipulate variables containing networking information.

Sections

- Processing Variables Using Filters (and Guided Exercise)
- Templating External Data Using Lookups (and Guided Exercise)
- Implementing Advanced Loops (and Guided Exercise)
- Using Filters to Work with Network Addresses (and Guided Exercise)

Lab

- Transforming Data with Filters and Plug-ins

Processing Variables Using Filters

Objectives

- Format, parse, and define the values of variables using filters.

Ansible Filters

Ansible applies variable values to playbooks and templates by using Jinja2 expressions. For example, the following Jinja2 expression replaces the name of the variable enclosed in double braces with its value:

```
 {{ variable }}
```

Jinja2 expressions also support filters. Filters are used to modify or process the value from the variable that is placed in the playbook or template. Some filters are provided by the Jinja2 language; others are included with Red Hat Ansible Automation Platform as plug-ins. You can also create custom filters, but that is beyond the scope of this course. Filters can be extremely useful for preparing data for use in your playbook or template.

To use a filter in a Jinja2 expression, add a pipe (|) character after the variable name or value and then write the filter or filter expression. You can specify multiple filters in a pipeline, separating each filter or filter expression with a pipe character.

```
 {{ variable | filter }}
```

Providing Default Variable Values

You can use the `default` filter to ignore an undefined variable or to provide a value to an undefined variable. Both use cases prevent a playbook from generating an error. The following portion of an error message indicates that a task uses the `shell` variable, but that the variable is undefined:

```
FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'dict object' has no attribute 'shell'"}
```

You can use the `default` filter to prevent this error.

The following generic task uses the `ansible.builtin.user` module to manage users defined in the `user_list` variable. Each user in the list must define the `name` key. Additionally, each user in the list can optionally define the `groups`, `system`, `shell`, `state`, and `remove` keys.

Use the `default` filter with each optional key. Consult the module documentation to identify the keys required by the module and the default values for keys.

```
- name: Manage user
  ansible.builtin.user:
    name: "{{ item['name'] }}"
```

```

groups: "{{ item['groups'] | default(omit) }}" ①
system: "{{ item['system'] | default(false) }}" ②
shell: "{{ item['shell'] | default('/bin/bash') }}" ③
state: "{{ item['state'] | default('present') }}" ④
remove: "{{ item['remove'] | default(false) }}" ⑤
loop: "{{ user_list }}"

```

- ① If the `groups` key is not defined for the `item` variable, then do not generate an error and do not provide a value for the key.
- ② If the `system` key is not defined for the `item` variable, then use the `false` Boolean value for the key. Pass a value of `true` to create a system user.
- ③ If the `shell` key is not defined for the `item` variable, then use the `/bin/bash` value for the key. You might pass a value of `/sbin/nologin` if you create a system user.
- ④ If the `state` key is not defined for the `item` variable, then use the `present` value for the key. Pass a value of `absent` to ensure that the user does not exist.
- ⑤ If the `remove` key is not defined for the `item` variable, then use the `false` Boolean value for the key. Passing a value of `true` when removing a user is the equivalent of the `userdel -r` command.

Whenever you use a module, you must decide which module keys to use in the task. You can manually specify values for the keys or you can configure the keys to set their values based on variables passed to the task. Because undefined variables produce errors, you can use the `default` filter to either ignore an undefined variable or supply a value for the undefined variable.

Typically, the `default` filter only provides a value if a variable is not defined. In some situations, you might want to provide a value if the variable passed to the `default` filter is an empty string or evaluates to the `false` Boolean value.

You can do this by adding `true` to the `default` filter. Consider the following playbook:

```

---
- name: Default filter examples
  hosts: localhost
  tasks:
    - name: Default filter examples
      vars:
        pattern: "some text"
      ansible.builtin.debug:
        msg: "{{ item }}"
      loop:
        - "{{ pattern | regex_search('test') | default('MESSAGE') }}" ①
        - "{{ pattern | regex_search('test') | default('MESSAGE', true) }}" ②
        - "{{ pattern | bool | default('MESSAGE') }}" ③
        - "{{ pattern | bool | default('MESSAGE', true) }}" ④

```

- ① Because the regular expression is not found in the variable, the `regex_search` filter returns an empty string. The `default` filter is not used.
- ② Although the `regex_search` filter returns an empty string, the `default` filter is used because it includes `true`.
- ③ Because the string evaluates to the `false` Boolean, the `default` filter is not used.

Chapter 7 | Transforming Data with Filters and Plug-ins

- ④ Although the string evaluates to the `false` Boolean, the `default` filter is used because it includes `true`.

Where appropriate, you can use the `default` filter before passing a variable to additional filters.

Variable Types

To understand filters, you must first know more about how variable values are handled by Ansible.

Ansible stores runtime data in variables. The YAML structure or the content of the value defines the exact type of data. The following table lists some value types:

Type	Description
Strings	A sequence of characters.
Numbers	A numeric value.
Booleans	True or false values.
Dates	ISO-8601 calendar date.
Null	The variable becomes undefined.
Lists or Arrays	A sorted collection of values.
Dictionaries	A collection of key-value pairs.

Sometimes, you might first need to convert the value to an integer with the `int` filter, or to a float with the `float` filter. For example, the following Jinja2 expression increments the current hour value, which is collected as a fact and stored as a string, not an integer:

```
 {{ ( ansible_facts['date_time']['hour'] | int ) + 1 }}
```

Various filters are available that can perform mathematical operations on numbers, such as `log`, `pow`, `root`, `abs`, and `round`. The `root` filter, for example, takes the square root of the variable or value.

```
 {{ 1764 | root }}
```

Manipulating Lists

Many filters are available to analyze and manipulate lists.

If the list consists of numbers, you can use the `max`, `min`, or `sum` filters to find the largest number, the smallest number, or the sum of all list items.

```
 {{ [2, 4, 6, 8, 10, 12] | sum }}
```

Extracting List Elements

You can obtain information about the contents of lists, such as the first or last elements, or the length of a list:

```
- name: All three of these assertions are true
  ansible.builtin.assert:
    that:
      - "{{ [ 2, 4, 6, 8, 10, 12 ] | length }} is eq( 6 )"
      - "{{ [ 2, 4, 6, 8, 10, 12 ] | first }} is eq( 2 )"
      - "{{ [ 2, 4, 6, 8, 10, 12 ] | last }} is eq( 12 )"
```

The `random` filter returns a random element from the list:

```
{{ ['Douglas', 'Marvin', 'Arthur'] | random }}
```

Modifying the Order of List Elements

You can use any of the following methods to reorder a list. The `sort` filter returns a list that is sorted by the natural order of its elements. The `reverse` filter returns a list where the order is the opposite of the original order. The `shuffle` filter returns a list with the same elements, but in a random order.

```
- name: reversing and sorting lists
  ansible.builtin.assert:
    that:
      - "{{ [ 2, 4, 6, 8, 10 ] | reverse }} is eq( [ 10, 8, 6, 4, 2 ] )"
      - "{{ [ 4, 8, 10, 6, 2 ] | sort }} is eq( [ 2, 4, 6, 8, 10 ] )"
```

Merging Lists

Sometimes it is useful to merge several lists into a single list to simplify iteration. The `flatten` filter recursively takes any inner list in the input list value, and adds the inner values to the outer list.

```
- name: Flatten turns nested lists on the left to list on the right
  ansible.builtin.assert:
    that:
      - "{{ [ 2, [4, [6, 8]], 10 ] | flatten }} is eq( [ 2, 4, 6, 8, 10 ] )"
```

Use the `flatten` filter to merge list values that come from iterating a parent list.

Operating on Lists as Sets

Use the `unique` filter to ensure that a list has no duplicate elements. This filter is useful if you are operating on a list of facts that you have collected, such as usernames or hostnames that might have duplicate entries.

```
- name: The 'unique' filter leaves unique elements
  ansible.builtin.assert:
    that:
      - "{{ [ 1, 1, 2, 2, 2, 3, 4, 4 ] | unique }} is eq( [ 1, 2, 3, 4 ] )"
```

If two lists have no duplicate elements, then you can use set theory operations on them.

- The `union` filter returns a set with elements from both input sets.
- The `intersect` filter returns a set with elements common to both sets.

Chapter 7 | Transforming Data with Filters and Plug-ins

- The `difference` filter returns a set with elements from the first set that are not present in the second set.

```
- name: The 'difference' filter provides elements not in specified set
  ansible.builtin.assert:
    that:
      - "{{ [2, 4, 6, 8, 10] | difference([2, 4, 6, 16]) }} is eq( [8, 10] )"
```

Manipulating Dictionaries

Unlike lists, dictionaries are not ordered in any way, but rather are just a collection of key-value pairs. You can use filters to construct dictionaries and you can convert those dictionaries into lists, and vice versa.

Joining Dictionaries

Use the `combine` filter to join two dictionaries. Entries from the second dictionary have higher priority than entries from the first dictionary, as seen in the following task:

```
- name: The 'combine' filter combines two dictionaries into one
  vars:
    expected:
      A: 1
      B: 4
      C: 5
  ansible.builtin.assert:
    that:
      - "{{ {'A':1,'B':2} | combine({'B':4,'C':5}) }} is eq( expected )"
```

Converting Dictionaries

Use the `dict2items` filter to convert a dictionary to a list. Use the `items2dict` filter to convert a list to a dictionary.

```
- name: converting between dictionaries and lists
  vars:
    characters_dict:
      Douglas: Human
      Marvin: Robot
      Arthur: Human
    characters_items:
      - key: Douglas
        value: Human
      - key: Marvin
        value: Robot
      - key: Arthur
        value: Human
  ansible.builtin.assert:
    that:
      - "{{ characters_dict | dict2items }} is eq( characters_items )"
      - "{{ characters_items | items2dict }} is eq( characters_dict )"
```

Hashing, Encoding, and Manipulating Strings

Various filters are available to manipulate the text of a value. You can compute checksums, create password hashes, and convert text to and from Base64 encoding, as used by a number of applications.

Hashing Strings and Passwords

The hash filter returns the hash value of the input string, using the provided hashing algorithm:

```
- name: the string's SHA-1 hash
  vars:
    expected: '8bae3f7d0a461488ced07b3e10ab80d018eb1d8c'
  ansible.builtin.assert:
    that:
      - "'{{ 'Arthur' | hash('sha1') }}' is eq( expected )"
```

Use the password_hash filter to generate password hashes:

```
{{ 'secret_password' | password_hash('sha512') }}
```

Encoding Strings

Use the b64encode filter to translate binary data to Base64, or translate Base64 encoded data back to binary data with the b64decode filter:

```
- name: Base64 encoding and decoding of values
  ansible.builtin.assert:
    that:
      - "'{{ 'â€šiõú' | b64encode }}' is eq( 'w6LDic0vw7TDug==' )"
      - "'{{ 'w6LDic0vw7TDug==' | b64decode }}' is eq( 'â€šiõú' )"
```

Before sending strings to the underlying shell, and to avoid parsing or code injection issues, it is a good practice to sanitize the string by using the quote filter:

```
- name: Put quotes around 'my_string'
  shell: echo {{ my_string | quote }}
```

Formatting Text

Use the lower, upper, or capitalize filters to enforce the case of an input string:

```
- name: Change case of characters
  ansible.builtin.assert:
    that:
      - "'{{ 'Marvin' | lower }}' is eq( 'marvin' )"
      - "'{{ 'Marvin' | upper }}' is eq( 'MARVIN' )"
      - "'{{ 'marvin' | capitalize }}' is eq( 'Marvin' )"
```

Replacing Text

Use the replace filter to replace all occurrences of a substring inside the input string:

```
- name: Replace 'ar' with asterisks
ansible.builtin.assert:
  that:
    - "'{{ 'marvin, arthur' | replace('ar','**') }}' is eq( 'm**vin, **thur' )"
```

Use the `regex_search` and `regex_replace` filters to perform more complex searches and replacements by using regular expressions.

```
- name: Test results of regex search and search-and-replace
ansible.builtin.assert:
  that:
    - "'{{ 'marvin, arthur' | regex_search('ar\S*r') }}' is eq( 'arthur' )"
    - "'{{ 'arthur up' | regex_replace('ar(\S*)r','\1mb') }}' is eq( 'thumb
      up' )"
```

Manipulating Data Structures

Many data structures used by Ansible are in JSON format. JSON and YAML are closely related, and Ansible data structures can be processed as JSON. Likewise, many APIs that Ansible Playbooks might interact with consume or provide information in JSON format. Because this format is widely used, JSON filters are particularly useful.

Data Structure Queries

You can combine `selectattr` with the `map` filter to extract information from Ansible data structures.

Use the `selectattr` filter to select a sequence of objects based on attributes of the objects in the list. Use the `map` filter to turn a list of dictionaries into a simple list based on a given attribute.



Note

Although the `community.general` collection provides the `json_query` filter, you can usually achieve the same functionality using the `selectattr` and `map` filters. Red Hat does not support the `community.general` collection.

Consider the following playbook, which queries the `/api/v2/execution_environments` API endpoint and displays the ID for the Control Plane Execution Environment automation controller resource:

```
---
- name: Query automation controller execution environments
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Query EEs
      vars:
        username_password: "admin:redhat"
      ansible.builtin.uri:
        url: https://controller.lab.example.com/api/v2/execution_environments/
        method: GET
        headers:
```

Chapter 7 | Transforming Data with Filters and Plug-ins

```

    Authorization: Basic {{ username_password | string | b64encode }}
    validate_certs: false
    register: query_results

    - name: Show execution environment ID
      ansible.builtin.debug:
        msg: "{{ query_results['json']['results'] | selectattr('name', '==',
'Control Plane Execution Environment') | map(attribute='id') | first }}"

```

The `query_results['json']['results']` variable is a list containing entries for each execution environment resource. Because each entry defines a `name` key, you can select the entry for the `Control Plane Execution Environment` resource by using the `selectattr` filter. After selecting the correct entry, you can use the `map` filter to display the value of any key in that entry, such as the value of the `id` key. Because the `selectattr` filter combined with the `map` filter frequently creates a list consisting of one item, you can use the `first` filter to select that item.

The playbook might produce the following output to indicate that the ID for the `Control Plane Execution Environment` resource is 4:

```

PLAY [Query automation controller EEs] ****
TASK [Query projects] ****
ok: [localhost]

TASK [Show execution environment ID] ****
ok: [localhost] => {
    "msg": "4"
}

PLAY RECAP ****
localhost                  : ok=2      changed=0      unreachable=0      failed=0 ...

```

Similarly, you might list all the execution environment names by sending the `query_results['json']['results']` variable to the `map` filter:

```

    - name: Show execution environment names
      ansible.builtin.debug:
        msg: "{{ query_results['json']['results'] | map(attribute='name') }}"

```

Adding this task to the previous playbook generates this additional output:

```

TASK [Show execution environment names] ****
ok: [localhost] => {
    "msg": [
        "Control Plane Execution Environment",
        "Automation Hub Default execution environment",
        "Automation Hub Ansible Engine 2.9 execution environment",
        "Automation Hub Minimal execution environment"
    ]
}

```

You can use the `selectattr` and `map` filters with both JSON and YAML data structures.

Parsing and Encoding Data Structures

Transforming data structures to and from text is useful for debugging and communication. Data structures serialize to JSON or YAML format with the `to_json` and `to_yaml` filters. Use `to_nice_json` and `to_nice_yaml` filters to obtain a formatted, human-readable output.

```
- name: Convert between JSON and YAML format
  vars:
    hosts:
      - name: bastion
        ip:
          - 172.25.250.254
          - 172.25.252.1
    hosts_json: '[{"name": "bastion", "ip": ["172.25.250.254", "172.25.252.1"]}]'
  ansible.builtin.assert:
    that:
      - "'{{ hosts | to_json }}' is eq( hosts_json )"
```

Other filters are introduced in the following sections and chapters that are suited to specific scenarios. Review the official Ansible and Jinja2 documentation to discover more useful filters to meet your needs.



References

Filters

https://docs.ansible.com/ansible/6/user_guide/playbooks_filters.html

Template Designer Documentation: Filters

<https://jinja.palletsprojects.com/en/3.0.x/templates/#builtin-filters>

► Guided Exercise

Processing Variables Using Filters

Use filters to reformat the values of variables for use in subsequent tasks.

Outcomes

- Use filters to process and manipulate variables in a playbook or role.

Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates a Git repository at `https://git.lab.example.com/student/data-filters.git`, which contains a partially complete playbook project that you finish during this guided exercise.

```
[student@workstation ~]$ lab start data-filters
```

Instructions

In this exercise, you deploy an HAProxy load balancer on `servera` to distribute the incoming web requests between `serverb` and `serverc`. On those two back-end servers, you deploy Apache HTTP Server and some initial content. The playbooks you use to do this take advantage of filters to set default values and to process and reuse the data stored in existing variables.

- ▶ 1. Change to the `/home/student/git-repos` directory, clone the project repository, and then create a branch for this exercise. In this exercise, you do not change the `deploy_haproxy.yml` playbook or the `haproxy` role. Run the `deploy_haproxy.yml` playbook to deploy the load balancer.
 - 1.1. Create the `/home/student/git-repos` directory if it does not exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos  
[student@workstation ~]$ cd ~/git-repos
```

- 1.2. Clone the repository, and change to the project root directory.

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/data-filters.git  
Cloning into 'data-filters'...  
...output omitted...  
[student@workstation git-repos]$ cd data-filters
```

- 1.3. Create the exercise branch and check it out.

```
[student@workstation data-filters]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 1.4. Run the `deploy_haproxy.yml` playbook to deploy the load balancer. Because you have not deployed the web servers, requests to `servera` result in a 503 HTTP status code.

```
[student@workstation data-filters]$ ansible-navigator run \
> -m stdout deploy_haproxy.yml

PLAY [Ensure HAProxy is deployed] ****
...output omitted...
PLAY RECAP ****
loadbalancer_01 : ok=7    changed=6    unreachable=0    ...

[student@workstation data-filters]$ curl servera
<html><body><h1>503 Service Unavailable</h1>
...output omitted...
```

- ▶ 2. Carefully review the task list and variables for the `apache` role.

- 2.1. Review the `roles/apache/tasks/main.yml` task list file for the `apache` role.

```
[student@workstation data-filters]$ cat roles/apache/tasks/main.yml
...output omitted...
- name: Calculate the package list
  ansible.builtin.set_fact:
    # TODO: Combine the apache_base_packages and
    # apache_optional_packages variables into one list.
    apache_package_list: "{{ apache_base_packages }}"

- name: Ensure httpd packages are installed
  ansible.builtin.yum:
    name: "{{ apache_package_list }}"
    state: present
    # TODO: omit the 'enablerepo' directive
    # below if the apache_enablerepos_list is empty;
    # otherwise use the list as the value for the
    # 'enablerepo' directive.
    #enablerepo: "{{ apache_enablerepos_list }}"
...output omitted...
```

The `apache_package_list` variable must be a combined list of both the base and optional packages required to install the `httpd` service. You edit and correct the definition of this variable in a later step.

The variable of the `apache_base_packages` role is defined as a list that contains one package, `httpd`. To prevent host group variables from overriding this value, the variable is defined in the `roles/apache/vars/main.yml` file:

```
apache_base_packages:
  - httpd
```

Chapter 7 | Transforming Data with Filters and Plug-ins

The `apache_optional_packages` variable is defined in the role as an empty list in `roles/apache/defaults/main.yml` file:

```
apache_optional_packages: []
```

The `apache_enabledrepos_list` variable contains a list of Yum repository IDs. Any repository ID in this list is temporarily enabled to install any packages.

The default value is an empty list, as defined in the `roles/apache/defaults/main.yml` file:

```
apache_enablersrepos_list: []
```

- 3. Correct the Jinja2 expression that defines the `apache_package_list` variable in the first task of the `apache` role. Remove the `TODO` comment section and save the file.

Define the `apache_optional_packages` variable for the `web_servers` host group to contain the following values: `git`, `php`, and `php-mysqlnd`.

- 3.1. Edit the Jinja2 expression in the first task of the `apache` role that defines the `apache_package_list` variable. Add the `union` filter to create a single list from the lists stored in the `apache_base_packages` and `apache_optional_packages` variables.

When you finish, remove the comments from this task. The first task of the `roles/apache/tasks/main.yml` file should consist of the following content:

```
- name: Calculate the package list
  ansible.builtin.set_fact:
    apache_package_list: "{{ apache_base_packages |
      union(apache_optional_packages) }}"
```

**Warning**

The `apache_package_list` variable definition is a single line of text. Do not split Jinja2 filter expressions over multiple lines, or your playbook will fail.

Save the file.

- 3.2. Define the `apache_optional_packages` variable in a new file called `group_vars/web_servers/apache.yml`. The variable defines a list of three packages: `git`, `php`, and `php-mysqlnd`. This overrides the role's default value for this variable.

The `group_vars/web_servers/apache.yml` file should consist of the following content:

```
apache_optional_packages:
  - git
  - php
  - php-mysqlnd
```

Save the file.

Chapter 7 | Transforming Data with Filters and Plug-ins

- 4. Remove the comments from the `enablerepo` directive in the second task from the `apache` role. Edit the directive's Jinja2 expression to use the `default` filter to omit this directive if the variable evaluates to a Boolean value of `false`. Remove the `TODO` comment section for the second task, and save the task file. The second task should consist of the following content:

```
- name: Ensure httpd packages are installed
  ansible.builtin.yum:
    name: "{{ apache_package_list }}"
    state: present
    enablerepo: "{{ apache_enablerepos_list | default(omit, true) }}"
```

The completed `roles/apache/tasks/main.yml` file should consist of the following content:

```
---
# tasks file for apache

- name: Calculate the package list
  ansible.builtin.set_fact:
    apache_package_list: "{{ apache_base_packages |
union(apache_optional_packages) }}"

- name: Ensure httpd packages are installed
  ansible.builtin.yum:
    name: "{{ apache_package_list }}"
    state: present
    enablerepo: "{{ apache_enablerepos_list | default(omit, true) }}"

- name: Ensure SELinux allows httpd connections to a remote database
  seboolean:
    name: httpd_can_network_connect_db
    state: true
    persistent: true

- name: Ensure httpd service is started and enabled
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true
```

- 5. Run the `deploy_apache.yml` playbook with the `-v` option. Verify that optional packages are included in the `apache_package_list` fact.

```
[student@workstation data-filters]$ ansible-navigator run \
> -m stdout deploy_apache.yml -v
...output omitted...
TASK [apache : Calculate the package list] ****
ok: [webserver_01] => {"ansible_facts": {"apache_package_list": ["httpd", "git", "php", "php-mysqlnd"]}, "changed": false}
...output omitted...
```

- 6. Review the task list and variable definitions for the webapp role. The webapp role ensures that the correct web application content exists on each host. When correctly implemented, the role removes any content from the root web directory that does not belong to the web application.

Edit the three tasks in the webapp role that have a TODO comment. Use filters to implement the functionality indicated in each comment.

- 6.1. Review the tasks in the roles/webapp/tasks/main.yml file:

```
---
```

```
# tasks file for webapp
```

```
- name: Ensure stub web content is deployed
  ansible.builtin.copy:
    content: "{{ webapp_message }} (version {{ webapp_version }})\n"
    dest: "{{ webapp_content_root }}/index.html"
```

```
- name: Find deployed webapp files
  ansible.builtin.find:
    paths: "{{ webapp_content_root }}"
    recurse: true
  register: webapp_find_files
```

```
- name: Compute webapp file list
  ansible.builtin.set_fact:
    # TODO: Use the map filter to extract
    # the 'path' attribute of each entry
    # in the 'webapp_find_files'
    # variable 'files' list.
    webapp_deployed_files: []
```

```
- name: Compute relative webapp file list
  ansible.builtin.set_fact:
    # TODO: Use the 'map' filter, along with
    # the 'relpath' filter, to create the
    # 'webapp_rel_deployed_files' variable
    # from the 'webapp_deployed_files' variable.
    #
    # Files in the 'webapp_rel_deployed_files'
    # variable should have a path relative to
    # the 'webapp_content_root' variable.
    webapp_rel_deployed_files: []
```

```
- name: Remove Extraneous Files
  ansible.builtin.file:
    path: "{{ webapp_content_root }}/{{ item }}"
    state: absent
    # TODO: Loop over a list of files
    # that are in the 'webapp_rel_deployed_files'
    # list, but not in the 'webapp_file_list' list.
    # Use the difference filter.
    loop: []
```

The `webapp_file_list` variable is defined in the `roles/webapp/vars/main.yml` file. This variable defines a file manifest for the web application. For this version of the web application, the only file in the application is `index.html`.

The `webapp_content_root_dir` variable defines the directory location of web application content on each web server. The default value is `/var/www/html`.

- 6.2. Replace the empty list for the `webapp_deployed_files` variable in the third task of the `webapp` role with a Jinja2 expression. Start with the `webapp_find_files['files']` variable and apply the `map` filter, followed by the `list` filter. Provide the `map` filter with an argument of `attribute='path'` to retrieve the `path` attribute from each entry in the list.

After you remove the `TODO` comments, the third task should consist of the following content:

```
- name: Compute the webapp file list
  ansible.builtin.set_fact:
    webapp_deployed_files: "{{ webapp_find_files['files'] | map(attribute='path') | list }}"
```



Warning

The `webapp_deployed_files` variable definition is a single line of text. Do not split Jinja2 filter expressions over multiple lines, or your playbook will fail.

- 6.3. Replace the empty list for the `webapp_rel_deployed_files` variable in the fourth task of `webapp` role with a Jinja2 expression. Start with the `webapp_deployed_files` variable and apply the `map` filter, followed by the `list` filter.

The first argument to the `map` function is the `relpath` string, which executes the `relpath` function on each item of the `webapp_deployed_files` list. The second argument to the `map` function is the `webapp_content_root_dir` variable. This variable is passed as an argument to the `relpath` function.

After you remove the `TODO` comments, the fourth task should consist of the following content:

```
- name: Compute the relative webapp file list
  ansible.builtin.set_fact:
    webapp_rel_deployed_files: "{{ webapp_deployed_files | map('relpath', webapp_content_root_dir) | list }}"
```



Warning

The `webapp_rel_deployed_files` variable definition is a single line of text. Do not split Jinja2 filter expressions over multiple lines, or your playbook will fail.

- 6.4. Replace the empty loop list in the fifth task in the `webapp` role with a Jinja2 expression. Start with the `webapp_rel_deployed_files` variable and apply the `difference` filter. Provide the `webapp_file_list` variable as an argument to the `difference` filter.

After you remove the `TODO` comments, the fifth task should consist of the following content:

Chapter 7 | Transforming Data with Filters and Plug-ins

```
- name: Remove Extraneous Files
  ansible.builtin.file:
    path: "{{ webapp_content_root_dir }}/{{ item }}"
    state: absent
  loop: "{{ webapp_rel_deployed_files | difference(webapp_file_list) }}"
```

Save the changes to the `roles/webapp/tasks/main.yml` file.

The completed `roles/webapp/tasks/main.yml` file should consist of the following content:

```
---
# tasks file for webapp

- name: Ensure stub web content is deployed
  ansible.builtin.copy:
    content: "{{ webapp_message }} (version {{ webapp_version }})\n"
    dest: "{{ webapp_content_root_dir }}/index.html"

- name: Find deployed webapp files
  ansible.builtin.find:
    paths: "{{ webapp_content_root_dir }}"
    recurse: true
  register: webapp_find_files

- name: Compute the webapp file list
  ansible.builtin.set_fact:
    webapp_deployed_files: "{{ webapp_find_files['files'] | map(attribute='path') | list }}"

- name: Compute the relative webapp file list
  ansible.builtin.set_fact:
    webapp_rel_deployed_files: "{{ webapp_deployed_files | map('relpath', webapp_content_root_dir) | list }}"

- name: Remove Extraneous Files
  ansible.builtin.file:
    path: "{{ webapp_content_root_dir }}/{{ item }}"
    state: absent
  loop: "{{ webapp_rel_deployed_files | difference(webapp_file_list) }}"
```

- ▶ 7. Run the `deploy_webapp.yml` playbook with the `-v` option. Verify that the playbook identifies a non-application file on `webserver_01` and removes it.

```
[student@workstation data-filters]$ ansible-navigator run \
> -m stdout deploy_webapp.yml -v
...output omitted...
TASK [webapp : Compute the webapp file list] ****
ok: [webserver_01] => {"ansible_facts": {"webapp_deployed_files": ["/var/www/html/test.html", "/var/www/html/index.html"]}, "changed": false}
ok: [webserver_02] => {"ansible_facts": {"webapp_deployed_files": ["/var/www/html/index.html"]}, "changed": false}
```

```
TASK [webapp : Compute the relative webapp file list] ****
ok: [webserver_01] => {"ansible_facts": {"webapp_rel_deployed_files": ["test.html", "index.html"]}, "changed": false}
ok: [webserver_02] => {"ansible_facts": {"webapp_rel_deployed_files": ["index.html"]}, "changed": false}

TASK [webapp : Remove Extraneous Files] ****
changed: [webserver_01] => (item=test.html) => {"ansible_loop_var": "item",
 "changed": true, "item": "test.html", "path": "/var/www/html/test.html", "state": "absent"}

PLAY RECAP ****
webserver_01    : ok=5    changed=2   unreachable=0   failed=0      skipped=0 ...
webserver_02    : ok=4    changed=1   unreachable=0   failed=0      skipped=1 ...
```

The webserver_01 host initially has two files deployed in the /var/www/html directory: test.html and index.html. Someone with access to the host might have installed a temporary test page on the web server.

The playbook removes any web server files from the web content root directory that are not part of the actual web application.

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish data-filters
```

Templating External Data Using Lookups

Objectives

- Populate variables with data from external sources using lookup plug-ins.

Lookup Plug-ins

A lookup plug-in is an Ansible extension to the Jinja2 templating language. These plug-ins enable Ansible to use data from external sources, such as files and the shell environment.

Calling Lookup Plug-ins

You can call lookup plug-ins with one of two Jinja2 template functions, `lookup` or `query`. Both methods have a syntax that is similar to filters. Specify the name of the function, and in parentheses the name of the lookup plug-in that you want to call and any arguments that the plug-in needs.

For example, the following variable definition uses the `file` lookup plug-in to put the contents of the `/etc/hosts` file into the `hosts` Ansible variable:

```
vars:
  hosts: "{{ lookup('ansible.builtin.file', '/etc/hosts') }}"
```

You can include more than one file name to the `file` plug-in. When called with the `lookup` function, the contents of each file is separated by a comma in the resulting value.

Consider the following Jinja2 template expression:

```
vars:
  hosts: "{{ lookup('ansible.builtin.file', '/etc/hosts', '/etc/issue') }}"
```

This expression results in the following structure (line-wrapped):

```
hosts: "127.0.0.1  localhost localhost.localdomain localhost4
localhost4.localdomain4\n::1      localhost localhost.localdomain localhost6
localhost6.localdomain6\n\n,\n\\$\\nKernel \\\\r on an \\\\m (\\\\l)"
```

In Ansible 2.5 and later, the `query` function, or its abbreviation `q`, can be used instead of `lookup` to call lookup plug-ins. The difference between the two is that instead of the returned values being comma-separated, `query` always returns a list, which can be easier to parse and work with.

You can use the following to call the preceding example:

```
vars:
  hosts: "{{ query('ansible.builtin.file', '/etc/hosts', '/etc/issue') }}"
```

The `query` call returns the following data structure (line-wrapped):

```

hosts:
  - "127.0.0.1  localhost localhost.localdomain localhost4
localhost4.localdomain4\n::1          localhost localhost.localdomain localhost6
localhost6.localdomain6\n\n"
  - "\\\$\\nKernel \\r on an \\m (\\l)"

```

Selecting Lookup Plug-ins

The default Ansible installation provides numerous plug-ins. Use the `ansible-navigator doc --mode stdout -t lookup -l` command to obtain the complete list of available lookup plug-ins. For details about the purpose of specific plug-ins and how to use them, run the `ansible-navigator doc --mode stdout -t lookup PLUGIN_NAME` command.

Reading the Contents of Files

Ansible uses the `file` plug-in to load the contents of a local file into a variable. If you provide a relative path, the plug-in looks for files in the playbook's `files` directory.

The following example reads the contents of the user's public key file and uses the `ansible.posix.authorized_key` module to add the authorized key to the managed host. The example uses a loop and the `+` operator to append strings in the template to look up the `files/fred.key.pub` and `files/naoko.key.pub` files.

```

- name: Add authorized keys
  hosts: all
  vars:
    users:
      - fred
      - naoko
  tasks:
    - name: Add authorized keys
      ansible.posix.authorized_key:
        user: "{{ item }}"
        key: "{{ lookup('ansible.builtin.file', item + '.key.pub') }}"
      loop: "{{ users }}"

```



Note

In the preceding example, the public key files are included in the Ansible project's `files` subdirectory, so that they are available inside the execution environment.

The `file` plug-in can parse YAML and JSON files into properly structured data by using the `from_yaml` or `from_json` filters.

```
my_yaml: "{{ lookup('ansible.builtin.file', '/path/to/my.yaml') | from_yaml }}"
```

**Important**

The `file` plug-in reads files that are in the *execution environment*, not on the control node or managed host.

Execution environments read files that are *inside of the execution environment container*, not those on the host on which you ran the `ansible-navigator` command.

If you use the `ansible-playbook` command, or the `ansible-navigator` command with the `--ee false` option, then the execution environment is the same as the control node.

**Note**

If you need the content of a file that is not in the execution environment or your project directory, then you can use the `ansible.builtin.slurp` module instead of a lookup plug-in.

The following example sets the `pub_key` variable using the content of the `/home/user/.ssh/id_rsa.pub` file on the managed host:

```
- name: Collect file content
  ansible.builtin.slurp:
    src: /home/user/.ssh/id_rsa.pub
    register: slurped_file

- name: Print returned information
  vars:
    pub_key: "{{ slurped_file['content'] | b64decode }}"
  ansible.builtin.debug:
    var: pub_key
```

Applying Data with a Template

Like the `file` plug-in, the `template` plug-in returns the contents of files. The difference is that the `template` plug-in expects the file contents to be a Jinja2 template, and it evaluates that template before applying the contents. If you pass a relative path to the template file, the plug-in looks for it within the playbook `templates` directory.

The following example processes the `templates/my.template.j2` template.

```
{{ lookup('ansible.builtin.template', 'my.template.j2') }}
```

For example, assume that the template contains the following content:

```
Hello {{ name }}.
```

The following task displays the string "Hello class.".

```
- name: Print "Hello class."
  vars:
    name: class
  ansible.builtin.debug:
    msg: "{{ lookup('ansible.builtin.template', 'my.template.j2') }}"
```

The `template` plug-in also accepts some extra parameters, such as defining the start and end marking sequences. In case the output string is a YAML value, the `convert_data` option parses the string to provide structured data.

**Important**

Do not confuse the `template` lookup plug-in with the `template` module.

Reading Command Output in the Execution Environment

The `pipe` and `lines` plug-ins both run a command in the execution environment and return the output. The `pipe` plug-in returns the raw output generated by the command; the `lines` plug-in splits the output of that command into lines.

For example, consider the following Jinja2 expression:

```
{{ query('ansible.builtin.pipe', 'ls files') }}
```

This expression returns the raw output of the `ls` command as a string. If you use the `lines` plug-in, then the expression is as follows:

```
{{ query('ansible.builtin.lines', 'ls files') }}
```

This command results in a list with each line of output returned by `ls` as a list item.

You can use this functionality to retrieve the first line (or any specific line) of output from a set of commands:

```
- name: Prints the first line of some files
  ansible.builtin.debug:
    msg: "{{ item[0] }}"
  loop:
    - "{{ query('ansible.builtin.lines', 'cat files/my.file') }}"
    - "{{ query('ansible.builtin.lines', 'cat files/my.file.2') }}
```

Note that this example might not be the most efficient way to do this particular task, given the existence of the `head` command.

Getting Content from a URL

Similar to the way the `file` plug-in gets content from a file, the `url` plug-in gets content from a URL.

```
{{ lookup('ansible.builtin.url', 'https://my.site.com/my.file') }}
```

Many options are available for controlling authentication, selecting web proxies, or splitting the returned content into lines.

However, one advantage of using the `url` plug-in is that you can use the returned data as values in your variables, possibly processing it first with a filter.

Getting Information from the Kubernetes API

The `kubernetes.core.k8s` plug-in provides full access to the Kubernetes API through the `openshift` Python module. To fetch a Kubernetes object, you must provide the object type by using the `kind` option. Providing additional object details, such as the `namespace` or `label_selector` option, helps to filter the results:

```
{{ lookup('kubernetes.core.k8s', kind='Deployment', namespace='ns',  
         resource_name='my_res') }}
```



Note

Be aware that the `kubernetes.core.k8s` plug-in is a *lookup* plug-in. Its primary purpose is to extract information from the Kubernetes cluster, not to update it. Use the `kubernetes.core.k8s module` to manage a Kubernetes cluster.

Using Custom Lookup Plug-ins

Plug-ins are Python scripts. You can develop custom plug-ins and use them in your playbooks. For Ansible to find your lookup plug-ins, copy them into one of the following locations:

- In a custom Ansible Content Collection, copy the lookup plug-in scripts into the `./plugins/lookup` directory.
- In a custom role, copy the lookup plug-in scripts into the `./filter_plugins` directory.
- In an Ansible project, copy the lookup plug-in scripts into the `./lookup_plugins` directory in the same directory as the playbooks.

The Ansible team recommends using collections to distribute custom plug-ins. Plug-in development is beyond the scope of this course.

Handling Lookup Errors

Most Ansible plug-ins are designed to abort the Ansible Playbook in the event of a failure. However, the `lookup` function delegates execution to other plug-ins that might not need to abort the Ansible Playbook in such an event. For example, the `file` plug-in might not need to abort the Ansible Playbook if the file is not found, but playbook execution might need to continue after recovery.

To adapt to different plug-in needs, the `lookup` plug-in accepts the `errors` parameter.

```
{{ lookup('ansible.builtin.file', 'my.file', errors='warn') }}
```

The default value for the `errors` option is `strict`. This means that the `lookup` plug-in raises a fatal error if the underlying script fails. If the `errors` option has the value `warn`, then the `lookup` plug-in logs a warning when the underlying script fails and returns an empty string (or an empty list). If the `errors` option has the value `ignore`, then the `lookup` plug-in silently ignores the error and returns an empty string or list.



References

Lookup Plugins – Ansible Documentation

<https://docs.ansible.com/ansible/6/plugins/lookup.html>

Index of all Lookup Plugins – Ansible Documentation

https://docs.ansible.com/ansible/6/collections/index_lookup.html

► Guided Exercise

Templating External Data Using Lookups

Use the `lookup` and `query` functions to template data from external sources into playbooks and deployed template files.

Outcomes

- Identify different sources of information in Ansible Playbooks.
- Use existing lookup plug-ins to retrieve information.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the Git repository needed for the exercise.

```
[student@workstation ~]$ lab start data-lookups
```

Instructions

- 1. Clone the `https://git.lab.example.com/student/data-lookups.git` Git repository into the `/home/student/git-repos` directory and then create a branch for this exercise.
- 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/  
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/data-lookups.git` repository and then change directory to the cloned repository:

```
[student@workstation git-repos]$ git clone \  
> https://git.lab.example.com/student/data-lookups.git  
Cloning into 'data-lookups'...  
...output omitted...  
[student@workstation git-repos]$ cd data-lookups
```

- 1.3. Create the `exercise` branch.

```
[student@workstation data-lookups]$ git checkout -b exercise  
Switched to a new branch 'exercise'
```

- 2. Populate `site.yml` with tasks to create groups using YAML-formatted data from the `groups.yml` file.

2.1. Review the `groups.yml` file.

```
---  
- name: devs  
  members:  
    - user1  
    - user2  
- name: ops  
  members:  
    - user3
```

- 2.2. You can read the contents of a YAML-formatted file in several ways. In the "Load group information" task from the `site.yml` playbook, you use the `ansible.builtin.file` lookup plug-in to read the file, and the `from_yaml` filter to load its data into the `user_groups` fact as structured YAML so that Ansible can parse it later:

```
"{{ lookup('ansible.builtin.file', 'groups.yml') | from_yaml }}"
```

The completed task should consist of the following content:

```
- name: Load group information  
  ansible.builtin.set_fact:  
    user_groups: "{{ lookup('ansible.builtin.file', 'groups.yml') |  
from_yaml }}"
```

- 2.3. Complete the "Create groups" task to create each group specified by the YAML data in the `user_groups` fact. The task must use the `ansible.builtin.group` module, looping on the `user_groups` fact. Use `item.name` to specify the group name of the group to create for each iteration of the loop. The completed task should consist of the following content:

```
- name: Create groups  
  ansible.builtin.group:  
    name: "{{ item['name'] }}"  
    state: present  
  loop: "{{ user_groups }}"
```

▶ 3. Complete the "Create users" task.

- 3.1. Use the `lines` lookup plug-in to read user information from the `users.txt` file.

```
"{{ query('ansible.builtin.lines', 'cat users.txt') }}"
```

The `lines` plug-in reads the output of `cat users.txt` as individual lines. The `query` function ensures that the data is a list.

Configure the task to loop through the users in this list.

Use the `loop` keyword to configure the task to loop through the users in this list:

```
- name: Create users
ansible.builtin.debug:
  msg: "To be done"
loop: "{{ query('ansible.builtin.lines', 'cat users.txt') }}"
```

- 3.2. When you create the users, create a random password as well. The `password` lookup plug-in generates random passwords and optionally stores the passwords in a local file.

The previous step added a loop on the list of usernames from `users.txt`; you can use the same `item` variable to generate the associated file for each user.

```
"{{ lookup('ansible.builtin.password', 'credentials/' + item + ' length=9') }}"
```

Store the password into a task variable to make it easier to use. Ensure that you include the space after the first single quote in the '`length=9`' argument. The task should now consist of the following content:

```
- name: Create users
  vars:
    password_plain: "{{ lookup('ansible.builtin.password', 'credentials/' +
item + ' length=9') }}"
  ansible.builtin.debug:
    msg: "To be done"
  loop: "{{ query('ansible.builtin.lines', 'cat users.txt') }}"
```

- 3.3. Replace the `ansible.builtin.debug` module with the `ansible.builtin.user` module.

You need to hash the new password before you can use it. You also need to set `update_password: on_create` for the module.



Important

The `update_password: on_create` option only sets the password if the task creates a new user on the managed host. If the user already exists but the password specified by the playbook is different from the one already set for the user on the system, then Ansible does **not** change the user's password.

This helps ensure that Ansible does not change an existing user's password if that user has already changed it but you run the playbook a second time.

```
- name: Create users
  vars:
    password_plain: "{{ lookup('ansible.builtin.password', 'credentials/' +
item + ' length=9') }}"
  ansible.builtin.user:
    name: "{{ item }}"
    password: "{{ password_plain | password_hash('sha512') }}"
    update_password: on_create
    state: present
  loop: "{{ query('ansible.builtin.lines', 'cat users.txt') }}"
```

Chapter 7 | Transforming Data with Filters and Plug-ins

- 3.4. The last task uses the `subelements` filter to populate the members of each group. However, no lookup plug-ins are used. Confirm that the completed play consists of the following content:

```
- name: Populate users and groups
hosts: all
gather_facts: false
tasks:

  - name: Load group information
    ansible.builtin.set_fact:
      user_groups: "{{ lookup('ansible.builtin.file', 'groups.yml') | from_yaml }}"

  - name: Create groups
    ansible.builtin.group:
      name: "{{ item['name'] }}"
      state: present
    loop: "{{ user_groups }}"

  - name: Create users
    vars:
      password_plain: "{{ lookup('ansible.builtin.password', 'credentials/' + item + ' length=9') }}"
    ansible.builtin.user:
      name: "{{ item }}"
      password: "{{ password_plain | password_hash('sha512') }}"
      update_password: on_create
      state: present
    loop: "{{ query('ansible.builtin.lines', 'cat users.txt') }}"

  - name: Add users to groups
    ansible.builtin.user:
      name: '{{ item[1] }}'
      groups: "{{ item[0]['name'] }}"
      append: true
    loop: "{{ user_groups | subelements('members', 'skip_missing=true') }}"
```

- 4. Run the playbook and verify that the users, groups, and group members are configured correctly.

- 4.1. Use the `ansible-navigator run` command to run the playbook.

```
[student@workstation data-lookups]$ ansible-navigator run \
> -m stdout site.yml

PLAY [Populate users and groups] ****
TASK [Load group information] ****
ok: [serverf.lab.example.com]

TASK [Create groups] ****
changed: [serverf.lab.example.com] => (item={'name': 'devs', 'members': ['user1', 'user2']})
```

```
changed: [serverf.lab.example.com] => (item={'name': 'ops', 'members': ['user3']})  
  
TASK [Create users] ****  
changed: [serverf.lab.example.com] => (item=user1)  
changed: [serverf.lab.example.com] => (item=user2)  
changed: [serverf.lab.example.com] => (item=user3)  
  
TASK [Add users to groups] ****  
changed: [serverf.lab.example.com] => (item=[{'name': 'devs', 'members': ['user1', 'user2']}, 'user1'])  
changed: [serverf.lab.example.com] => (item=[{'name': 'devs', 'members': ['user1', 'user2']}, 'user2'])  
changed: [serverf.lab.example.com] => (item=[{'name': 'ops', 'members': ['user3']}, 'user3'])  
  
PLAY RECAP ****  
serverf.lab.example.com : ok=4    changed=3    unreachable=0    failed=0  
skipped=0    rescued=0    ignored=0
```

4.2. Verify that the users have been created on the `serverf` server.

```
[student@workstation data-lookups]$ ssh serverf "tail -n3 /etc/passwd"  
user1:x:1002:1004::/home/user1:/bin/bash  
user2:x:1003:1005::/home/user2:/bin/bash  
user3:x:1004:1006::/home/user3:/bin/bash
```

4.3. Verify that the groups exist and have the relevant members.

```
[student@workstation data-lookups]$ ssh serverf "tail -n5 /etc/group"  
devs:x:1002:user1,user2  
ops:x:1003:user3  
user1:x:1004:  
user2:x:1005:  
user3:x:1006:
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish data-lookups
```

Implementing Advanced Loops

Objectives

- Implement loops using structures other than simple lists using lookup plug-ins and filters.

Comparing Loops and Lookup Plug-ins

Using loops to iterate over tasks can help simplify your Ansible Playbooks. The `loop` keyword loops over a flat list of items. When used with lookup plug-ins, you can construct more complex data in your lists for your loops.

The `loop` keyword was introduced in Ansible 2.5. Before that, task iteration was implemented by using keywords that started with `with_` and ended with the name of a lookup plug-in. The equivalent to `loop` in this syntax is `with_list`, and is designed for iteration over a simple flat list. For simple lists, `loop` is the best syntax to use.

For example, the following three syntaxes have the same results. The first of these is preferred:

```
- name: using loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ mylist }}"

- name: using with_list
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_list: "{{ mylist }}"

- name: using lookup plugin
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ lookup('ansible.builtin.list', mylist) }}"
```

You can refactor a `with_*` iteration task to use the `loop` keyword by using an appropriate combination of lookup plug-ins and filters to match the functionality.

Using the `loop` keyword in place of `with_*` loops has the following benefits:

- No need to memorize or find a `with_*` style keyword to suit your iteration scenario. Instead, use plug-ins and filters to adapt a `loop` keyword task to your use case.
- Focus on learning the plug-ins and filters that are available in Ansible, which have broader applicability beyond iteration.
- Provides command-line access to the lookup plug-in documentation, with the `ansible-navigator doc --mode stdout -t lookup -l` command. This documentation helps you to discover lookup plug-ins and to design and use custom iteration scenarios.



Important

Guidance from upstream Ansible on when to use `loop` or `with_loops` has been evolving since Ansible 2.5. Red Hat recommends that you use the `loop` keyword instead of the `with_*` loops, but some use cases exist where the earlier syntax might be better. According to the documentation, the `with_*` syntax is not deprecated and it should be valid for the foreseeable future. The syntax of `loop` might continue to evolve in future releases of Ansible.

The upstream guidance for Ansible 6 includes the following tips:

- The `loop` keyword requires a list, and does not accept a string. If you are having problems, then remember the difference between `lookup` and `query`.
- Any use of `with_*` that is discussed in "Migrating from `with_X` to `Loop`" [https://docs.ansible.com/ansible/6/user_guide/playbooks_loops.html#migrating-to-loop] can be safely converted. These mostly use filters.
- If you need to use a `lookup` plug-in to convert a `with_*` construct to use `loop`, then it might be clearer to keep using the `with_*` syntax.

Example Iteration Scenarios

The following examples show some ways to construct more complex loops using Jinja2 expressions, filters, lookup plug-ins, and the `with_*` syntax.

Iterating over a List of Lists

The `with_items` keyword provides a way to iterate over complex lists. For example, consider a hypothetical play with the following task:

```
- name: Remove build files
  ansible.builtin.file:
    path: "{{ item }}"
    state: absent
  with_items:
    - "{{ app_a_tmp_files }}"
    - "{{ app_b_tmp_files }}"
    - "{{ app_c_tmp_files }}
```

The `app_a_tmp_files`, `app_b_tmp_files`, and `app_c_tmp_files` variables each contain a list of temporary files. The `with_items` keyword combines these three lists into a single list containing the entries from all three lists. It automatically performs one-level flattening of its list.

To refactor a `with_items` task to use the `loop` keyword, use the `flatten` filter. The `flatten` filter recursively searches for embedded lists, and creates a single list from discovered values.

The `flatten` filter accepts a `levels` argument, which specifies an integer number of levels to search for embedded lists. A `levels=1` argument specifies that values are obtained by only descending into one additional list for each item in the initial list. This is the same one-level flattening that `with_items` does implicitly.

To refactor a `with_items` task to use the `loop` keyword, you must also use the `flatten(levels=1)` filter:

```
- name: Remove build files
ansible.builtin.file:
  path: "{{ item }}"
  state: absent
loop: "{{ list_of_lists | flatten(levels=1) }}"
vars:
  list_of_lists:
    - "{{ app_a_tmp_files }}"
    - "{{ app_b_tmp_files }}"
    - "{{ app_c_tmp_files }}"
```



Important

Because the `loop` keyword does not perform this implicit one-level flattening, it is not exactly equivalent to `with_items`. However, provided that the list passed to the loop is a simple list, both methods behave identically. The distinction only matters if you have nested lists.

Iterating over Nested Lists

Data from variable files, Ansible facts, and external services are often a composition of simpler data structures, such as lists and dictionaries. Consider the following `users` variable:

```
users:
  - name: paul
    password: "{{ paul_pass }}"
    authorized:
      - keys/paul_key1.pub
      - keys/paul_key2.pub
    mysql:
      hosts:
        - "%"
        - "127.0.0.1"
        - "::1"
        - "localhost"
    groups:
      - wheel

  - name: john
    password: "{{ john_pass }}"
    authorized:
      - keys/john_key.pub
    mysql:
      password: other-mysql-password
      hosts:
        - "utility"
    groups:
      - wheel
      - devops
```

The `users` variable is a list. Each entry in the list is a dictionary with the `name`, `password`, `authorized`, `mysql`, and `groups` keys. The `name` and `password` keys define simple strings, and

Chapter 7 | Transforming Data with Filters and Plug-ins

the `authorized` and `groups` keys define lists. The `mysql` key references another dictionary that contains MySQL related metadata for each user.

Similar to the `flatten` filter, the `subelements` filter creates a single list from a list with nested lists. The filter processes a list of dictionaries, and each dictionary contains a key that refers to a list. To use the `subelements` filter, you must provide the name of a key on each dictionary that corresponds to a list.

To illustrate, consider again the previous `users` variable definition. The `subelements` filter enables iteration through all users and their authorized key files defined in the variable:

```
- name: Set authorized ssh key
  ansible.posix.authorized_key:
    user: "{{ item[0]['name'] }}"
    key: "{{ lookup('ansible.builtin.file', item[1]) }}"
  loop: "{{ users | subelements('authorized') }}"
```

The `subelements` filter creates a new list from the `users` variable data. Each item in the list is itself a two-element list. The first element contains a reference to each user. The second element contains a reference to a single entry from the `authorized` list for that user.

Iterating over a Dictionary

You often encounter data that is organized as a set of key-value pairs, commonly referred to in the Ansible community as a dictionary, instead of being organized as a list. For example, consider the following definition of a `users` variable:

```
users:
  demo1:
    name: Demo User 1
    mail: demo1@example.com
  demo2:
    name: Demo User 2
    mail: demo2@example.com
  ...output omitted...
  demo200:
    name: Demo User 200
    mail: demo200@example.com
```

Before Ansible 2.5, you had to use the `with_dict` keyword to iterate through the key-value pairs of this dictionary. For each iteration, the `item` variable has two attributes: `key` and `value`. The `key` attribute contains the value of one of the dictionary keys, and the `value` attribute contains the data associated with the dictionary key:

```
- name: Iterate over Users
  ansible.builtin.user:
    name: "{{ item['key'] }}"
    comment: "{{ item['value']['name'] }}"
    state: present
  with_dict: "{{ users }}"
```

Chapter 7 | Transforming Data with Filters and Plug-ins

Alternatively, you can use the `dict2items` filter to transform a dictionary into a list, and this is probably easier to understand. The items in this list are structured the same as items produced by the `with_dict` keyword:

```
- name: Iterate over Users
  ansible.builtin.user:
    name: "{{ item['key'] }}"
    comment: "{{ item['value']['name'] }}"
    state: present
  loop: "{{ users | dict2items }}"
```

Iterating over a File Globbing Pattern

You can construct a loop that iterates over a list of files that match a provided file globbing pattern with the `fileglob` lookup plug-in.

To illustrate, consider the following play:

```
- name: Test
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Test fileglob lookup plugin
      ansible.builtin.debug:
        msg: "{{ lookup('ansible.builtin.fileglob', '~/.bash*') }}"
```

The output from the `fileglob` lookup plug-in is a string of comma-separated files, indicated by the double quotation marks around the `msg` variable's data:

```
PLAY [Test] ****
TASK [Test fileglob lookup plugin] ****
ok: [localhost] => {
    "msg": "/home/student/.bash_logout,/home/student/.bash_profile,/home/
student/.bashrc,/home/student/.bash_history"
}

PLAY RECAP ****
localhost : ok=1    changed=0    unreachable=0    failed=0    ...
```

To force a lookup plug-in to return a list of values instead of a string of comma-separated values, use the `query` keyword instead of the `lookup` keyword. Consider the following modification of the previous playbook example:

```
- name: Test
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Test fileglob lookup plugin
      ansible.builtin.debug:
        msg: "{{ query('fileglob', '~/.bash*') }}"
```

The output of this modified playbook indicates that the `msg` keyword references a list of files, because the data is enclosed in brackets:

```
PLAY [Test] ****
TASK [Test fileglob lookup plugin] ****
ok: [localhost] => {
  "msg": [
    "/home/student/.bash_logout",
    "/home/student/.bash_profile",
    "/home/student/.bashrc",
    "/home/student/.bash_history"
  ]
}

PLAY RECAP ****
localhost : ok=1    changed=0    unreachable=0    failed=0    ...
```

To use the data from this lookup plug-in in a loop, ensure that the processed data is returned as a list. Both tasks in the following play iterate over the files matching the `~/.bash*` globbing pattern:

```
- name: Both tasks have the same result
hosts: localhost
gather_facts: false
tasks:

  - name: Iteration Option One
    ansible.builtin.debug:
      msg: "{{ item }}"
    loop: "{{ query('fileglob', '~/.bash*') }}"

  - name: Iteration Option Two
    ansible.builtin.debug:
      msg: "{{ item }}"
    with_fileglob: "~/.bash*"
```

The `with_fileglob` task is preferred in this case because it is cleaner.

Retrying a Task

Often you do not want a play to run until a specific condition is met. The `until` directive implements a special kind of loop to specify that kind of condition.

For example, during a blue-green deployment, you must wait until the status of the blue host is good before continuing the deployment on the green host.

```
- name: Perform smoke test
ansible.builtin.uri:
  url: "https://{{ blue }}/status"
  return_content: true
register: smoke_test
until: "'STATUS_OK' in smoke_test['content']"
retries: 12
delay: 10
```

The `blue` web server status page is queried every 10 seconds, and fails after 12 attempts if no queries are successful.



References

Loops - Ansible Documentation

https://docs.ansible.com/ansible/6/user_guide/playbooks_loops.html

► Guided Exercise

Implementing Advanced Loops

Write tasks that use advanced looping techniques that illustrate some important use cases for complex loops.

Outcomes

- Use filters and lookup plug-ins to iterate over complex data structures.

Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares an Ansible project in the `/home/student/data-loops/` directory on the `workstation` machine.

```
[student@workstation ~]$ lab start data-loops
```

Instructions

- 1. Use a loop with the `dict2items` filter to transform the `rgroups` variable into a list.
1. Review the structure of the `rgroups` variable section defined in the `~/data-loops/group_vars/all/my_vars.yml` file.
- ```
Group variables
rgroups:
 accounts: ①
 accounts: ②
 name: accounts ③
 gid: 7001
 development:
 name: development
 gid: 7002
...output omitted...
```
1. The `rgroups` variable is a dictionary.
  2. Each key in the dictionary represents a group name, and the value for each key is another dictionary containing metadata for that group.
  3. Each group metadata dictionary contains two keys: `name` and `gid`.
2. Review the `~/data-loops/add-groups.yml` playbook. This playbook adds a group to the targeted host.

```
- name: First task
 hosts: servera.lab.example.com
 gather_facts: false
```

**Chapter 7 |** Transforming Data with Filters and Plug-ins

```
become: true

tasks:
 - name: Add groups
 ansible.builtin.group:
 name: accounts
 gid: 7001
 state: present
```

13. Modify the playbook to use the `loop` keyword with the `dict2items` filter for your loop.

The `add-groups.yml` playbook should consist of the following content:

```

- name: First task
 hosts: servera.lab.example.com
 gather_facts: false
 become: true

 tasks:
 - name: Add groups
 ansible.builtin.group:
 name: "{{ item['key'] }}"
 gid: "{{ item['value']['gid'] }}"
 state: present
 loop: "{{ rgroups | dict2items }}"
```

14. Run the `add-groups.yml` playbook to add the groups.

```
[student@workstation ~]$ cd ~/data-loops/
[student@workstation data-loops]$ ansible-navigator run -m stdout add-groups.yml

PLAY [First task] ****
TASK [Add groups] ****
changed: [servera...]
 changed: [servera...] => (item={'key': 'accounts'...})
 changed: [servera...] => (item={'key': 'development'...})
...output omitted...

PLAY RECAP ****
servera...: ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

- 2. Use a nested loop with the `subelements` filter to transform the `my_users` and `my_groups` variables into a list.

- 2.1. Review the structure of the `my_users` variable section defined in the `~/data-loops/group_vars/all/my_vars.yml` file.

```
User variables
my_users:
 john:
 name: john
 my_groups:
```

1

2

3

```

- accounts
- development
jane:
 name: jane
 my_groups:
 - manufacturing
 - marketing
...output omitted...

```

- ➊ The `my_users` variable is a dictionary.
- ➋ Each key in the dictionary is itself a dictionary, with keys of `name` and `my_groups`.
- ➌ The `my_groups` key references a list of groups. For example, the user `john` becomes a member of the `accounts` and `development` groups.

2.2. Review the `~/data-loops/add-users.yml` playbook. This playbook adds a user and puts it in the correct group.

```

- name: Second task
hosts: servera.lab.example.com
gather_facts: false
become: true

tasks:
 - name: Add users and put them in the right groups
 ansible.builtin.user:
 name: john
 append: true
 groups: accounts
 state: present

```

2.3. Modify the playbook to use the `loop` keyword with the `subelements` filter for your loop.

The `add-users.yml` playbook should consist of the following content:

```

- name: Second task
hosts: servera.lab.example.com
gather_facts: false
become: true

tasks:
 - name: Add users and put them in the right groups
 ansible.builtin.user:
 name: "{{ item[0]['name'] }}"
 append: true
 groups: "{{ item[1] }}"
 state: present
 loop: "{{ my_users | subelements('my_groups') }}"

```

2.4. Run the `add-users.yml` playbook to add the users.

```
[student@workstation data-loops]$ ansible-navigator run -m stdout add-users.yml

PLAY [Second task] ****
TASK [Add users and put them in the right groups] ****
changed: [servera...] => (item=[{'name': 'john', 'my_groups': ['accounts', ...)])
ok: [servera...] => (item=[{'name': 'john', 'my_groups': ['accounts', ...)})
...output omitted...

PLAY RECAP ****
servera...: ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

- ▶ 3. Use a loop over the `public_keys_lists` variable defined in the `group_vars/all/my_vars.yml` file. Use the `map` filter, followed by the `flatten` filter, to generate a simple list of SSH public key files for all developers. Update the task's `key` keyword to contain the file content of each iteration item.

- 3.1. Review the structure of the `public_keys_lists` variable section defined in the `~/data-loops/group_vars/all/my_vars.yml` file.

```
Public key variables
public_key_lists:
 1
 - username: john
 2
 public_keys:
 3
 - pubkeys/john/id_rsa.pub
 - pubkeys/john/laptop_rsa.pub
 - username: jane
 public_keys:
 - pubkeys/jane/id_rsa.pub
...output omitted...
```

- ➊ The `public_key_lists` variable is a list.
  - ➋ Each entry in the list is a dictionary with the `public_keys` and `username` keys.
  - ➌ The `public_keys` key references a list of public key files for the given user. For example, the `joh`n user has two public key files: `pubkeys/john/id_rsa.pub` and `pubkeys/john/laptop_rsa.pub`. The file names are relative to the root directory of the playbook project.
- 3.2. Review the `~/data-loops/add-keys.yml` playbook. This playbook sets up authorized keys for the developer user.

```
- name: Third task
hosts: servera.lab.example.com
gather_facts: false
become: true

tasks:
 - name: Set up authorized keys
 ansible.posix.authorized_key:
```

```
user: developer
state: present
key: pubkeys/john/id_rsa.pub
```

- 3.3. Modify the playbook to perform the following tasks:

- Add a `map` filter to the end of the expression to extract the `public_keys` attribute. It creates a list of lists; each entry in the list is a list of SSH public key files for a particular user.
- Add a `flatten` filter after the `map` filter to create a single list of files.
- Use the `file lookup` plug-in to configure the `key` keyword with the file content of each iteration item.

The `add-keys.yml` playbook should consist of the following content:

```
- name: Third task
hosts: servera.lab.example.com
gather_facts: false
become: true

tasks:
 - name: Set up authorized keys
 ansible.posix.authorized_key:
 user: developer
 state: present
 key: "{{ lookup('ansible.builtin.file', item) }}"
 loop: "{{ public_key_lists | map(attribute='public_keys') | flatten }}"
```

- 4. Install the `ansible.posix` collection so that it is available to the execution environment. This collection is needed for the `ansible.posix.authorized_key` module.

- 4.1. On the `workstation` machine, open a browser and navigate to `hub.lab.example.com`.
- 4.2. Log in as the `student` user using `redhat123` as the password.
- 4.3. Navigate to **Collections > API token management** and then click **Load Token**.
- 4.4. Copy and paste the token in the `token` variable at the end of the `~/data-loops/ansible.cfg` file.
- 4.5. Install the collection:

```
[student@workstation data-loops]$ ansible-galaxy collection \
> install -r collections/requirements.yml -p collections/
...output omitted...
```

- 5. Run the `add-keys.yml` playbook to add the keys.

```
[student@workstation data-loops]$ ansible-navigator run -m stdout add-keys.yml
PLAY [Third task] ****
```

```
TASK [Set up authorized keys] *****
changed: [servera.lab.example.com] => (item=pubkeys/john/id_rsa.pub)
changed: [servera.lab.example.com] => (item=pubkeys/john/laptop_rsa.pub)
changed: [servera.lab.example.com] => (item=pubkeys/jane/id_rsa.pub)
...output omitted...

PLAY RECAP *****
servera...: ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

## Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish data-loops
```

# Using Filters to Work with Network Addresses

---

## Objectives

- Use filters to inspect, validate, and manipulate variables containing networking information.

## Gathering and Processing Networking Information

A number of filters and lookup plug-ins are available to collect and process network information for Ansible automation. These filters and plug-ins can be useful with fact gathering when you configure the network devices on your managed hosts.

The standard `ansible.builtin.setup` module, which automatically gathers facts at the start of many plays, collects a lot of network-related information from each managed host.

The `ansible_facts['interfaces']` fact is a list of all the network interface names on the system. You can use this list to examine the details of each of the network interfaces on the system. For example, if `enp11s0` is an interface on the system, then it has a fact called `ansible_facts['enp11s0']`, which is a dictionary containing information about the interface such as the MAC address, IPv4 and IPv6 addresses, kernel module, and so on.

The following facts are useful to remember:

### Selected Networking Facts

| Fact name                                        | Description                                                                                                  |
|--------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>ansible_facts['dns']['nameservers']</code> | The DNS name servers used for name resolution by the managed host (included in the <code>min</code> subset). |
| <code>ansible_facts['domain']</code>             | The domain for the managed host.                                                                             |
| <code>ansible_facts['all_ipv4_addresses']</code> | All the IPv4 addresses configured on the managed host.                                                       |
| <code>ansible_facts['all_ipv6_addresses']</code> | All the IPv6 addresses configured on the managed host.                                                       |
| <code>ansible_facts['fqdn']</code>               | The fully qualified domain name (DNS name) of the managed host.                                              |
| <code>ansible_facts['hostname']</code>           | The unqualified hostname; the string in the FQDN before the first period.                                    |

The following playbook shows an example of using network facts:

```

- name: Set up web servers
 hosts: web_servers
 become: true
```

```
tasks:
 - name: Configure web server status page
 ansible.builtin.copy:
 content: "Server: {{ ansible_facts['fqdn'] }} at
{{ ansible_facts['default_ipv4']['address'] }}"
 dest: /var/www/html/status.html
 owner: apache
 group: apache
 mode: '0444'

 - name: Ensure httpd is installed
 ansible.builtin.yum:
 name: httpd
 state: installed

 - name: Start and enable webserver
 ansible.builtin.service:
 name: httpd
 state: started
 enabled: true

 - name: Notify root of server provisioning
 community.general.mail:
 subject: >
 System {{ ansible_facts['hostname'] }} has
 been successfully provisioned.
```

## Network Information Filters

You can validate and manipulate networking-related data stored in variables and facts with the `ansible.utils.ipaddr` filter. The `ansible.utils.ipaddr` filter can validate the syntax of IP addresses, filter out bad data, convert from VLSM subnet masks to CIDR subnet prefix notation, perform subnet math, or find the next usable address in a network range.

Comprehensive documentation for this filter is available at "`ipaddr filter`" [[https://docs.ansible.com/ansible/6/collections/ansible/utils/docssite/filters\\_ipaddr.html](https://docs.ansible.com/ansible/6/collections/ansible/utils/docssite/filters_ipaddr.html)], but this section explores some of its uses.



### Note

The `ansible.utils.ipaddr` filter needs the `netaddr` Python module, provided by the `python3-netaddr` package in Red Hat Enterprise Linux 8.

This Python module is installed by default in the `ansible-automation-platform-22/ee-supported-rhel8` execution environment. Ensure that the Python module is installed on your control node if you are using the `ansible-playbook` command, or if you are using the `ansible-navigator` command with the execution environment disabled.

## Testing IP Addresses

In its simplest form, the `ansible.utils.ipaddr` filter with no arguments accepts a single value. If the value is an IP address, then the filter returns the IP address. If the value is not an IP address, then the filter returns `false`.

If the value is a list, then the filter returns the valid IP addresses but not the invalid ones. If all the items are invalid, then the filter returns an empty list.

```
{% my_hosts_list | ansible.utils.ipaddr %}
```

You can also pass arguments to the `ansible.utils.ipaddr` filter. These arguments can make the filter return different information based on the data input.

## Filtering Data

The following subsections explore some options that you can use with the `ansible.utils.ipaddr` filter. To better understand how they work, each of the following examples shows the output that results when that filter is used with a variation on the following task:

```
- name: Filter example
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr }}"
```

The `ipaddr` filter in the `msg` line is replaced with a version that uses the appropriate option. For example, the following version of the line retrieves the network masks of the given addresses.

```
msg: "{{ listips | ansible.utils.ipaddr('netmask') }}"
```

The `listips` variable used to generate the examples contains the following list:

```
listips:
 - 192.168.2.1
 - 10.0.0.128/25
 - 172.24.10.0/255.255.255.0
 - 172.24.10.0/255.255.255.255
 - ff02::1
 - ::1
 - 2001::1/64
 - 2001::/64
 - www.redhat.com
```

### host

Outputs only valid individual IP addresses, in the correct CIDR prefix format for host addresses.

## Chapter 7 | Transforming Data with Filters and Plug-ins

```
"msg": [
 "192.168.2.1/32",
 "172.24.11.0/32",
 "ff02::1/128",
 "::1/128",
 "2001::1/64"
]
```

### net

Filters out values that are not valid network specifications, converting the netmask into a CIDR prefix if necessary.

```
"msg": [
 "10.0.0.128/25",
 "172.24.10.0/24",
 "2001::/64"
]
```

### private

Displays input IP addresses or network ranges that are in ranges reserved by IANA to be private. This includes RFC 1918 address spaces and IPv6 site-local addresses at a minimum.

```
"msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/255.255.255.0",
 "172.24.10.0/255.255.255.255"
]
```

### public

Outputs addresses and networks in public (globally routable) address spaces, according to IANA. (This example has no IPv4 addresses in the public space.)

```
"msg": [
 "2001::1/64",
 "2001::/64"
]
```

The `ansible.utils.ipwrap` filter puts brackets around the address part of items that appear to be IPv6 addresses. The filter passes all other values unchanged. This is useful in templates for certain configuration files that expect this syntax for IPv6 addresses.

```
"msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/255.255.255.0",
 "172.24.10.0/255.255.255.255",
 "[ff02::1]",
 "[::1]",
 "[2001::1]/64",
]
```

```
[
 "[2001::]/64",
 "www.redhat.com"
]
```

**Note**

Most of the filters in the preceding examples remove the `www.redhat.com` list item, which is not an IP address. The `ansible.utils.ipwrap` filter does not do this, but you could use the `ansible.utils.ipaddr` filter first to handle this scenario:

```
"{{ listips | ansible.utils.ipaddr | ansible.utils.ipwrap }}"
```

## Manipulating IP Addresses

You can use a number of the `ansible.utils.ipaddr` options to format networking data from an IPv4 or IPv6 address.

For example:

- To return the address part, `192.0.2.1`:

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('address') }}"
```

- To return the variable-length subnet mask, `255.255.255.0`:

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('netmask') }}"
```

- To return the CIDR prefix, `24`:

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('prefix') }}"
```

- To return the network's broadcast address, `192.0.2.255`:

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('broadcast') }}"
```

- To return the network's network address, `192.0.2.0`:

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('network') }}"
```

- To return the IP address in DNS PTR record format, `1.2.0.192.in-addr.arpa.:`

```
"{{ '192.0.2.1/24' | ansible.utils.ipaddr('revdns') }}"
```

This correctly converts IPv6 addresses as well.

If you pass a list of items to `ansible.utils.ipaddr`, it attempts to convert the items that appear to be valid addresses and filter out items that are not.

 **Important**

The broadcast option should not be used with an IPv6 address/prefix because that protocol does not assign a broadcast address. Other options should work with IPv6 addresses as expected.

## Reformatting or Calculating Network Information

You can display an IP address on a given network by passing an integer to the `ansible.utils.ipaddr` filter. The following returns the fifth address (192.0.2.5/24) on the 192.0.2.0/24 network.

```
"{{ '192.0.2.0/24' | ansible.utils.ipaddr(5) }}"
```

You can convert the network specification to a range of usable addresses for hosts with the `range_usable` option. The following example returns the 192.0.2.1-192.0.2.254 address range:

```
"{{ '192.0.2.0/24' | ansible.utils.ipaddr('range_usable') }}"
```

If you pass a network specification to the `ansible.utils.network_in_usable` filter, you can specify a particular address as an option to determine if that address is usable for hosts on that network (meaning, the address is not reserved for broadcasts or other uses). The following example returns the Boolean value `true`:

```
"{{ '192.0.2.0/24' | ansible.utils.network_in_usable('192.0.2.5') }}"
```

If you replace 192.0.2.5 with 192.0.2.255 (the reserved broadcast address) or 192.0.3.2 (not on the network), it returns the Boolean value `false`.

You can also find the next host on a network. The following example returns the 192.0.2.6/24 address:

```
"{{ '192.0.2.5/24' | ansible.utils.ipaddr('next_usable') }}"
```

When the end of the network is reached, it returns an empty value.

Network administrators can use the `ansible.utils.cidr_merge` filter to aggregate a list of subnets and individual hosts into the minimal representation. For example, consider the following list:

```
mynetworks:
 - 10.0.1.0/24
 - 10.0.2.0/24
 - 10.0.3.0/24
 - 10.0.100.0/24
 - 2001:db8::/64
 - 2001:db8:0:1::/64
```

You can use the "`{{ mynetworks | ansible.utils.cidr_merge }}`" expression to return the following list:

```
"msg": [
 "10.0.1.0/24",
 "10.0.2.0/23",
 "10.0.100.0/24",
 "2001:db8::/63"
]
```

More filters, as well as more options for the preceding filters, are available in the `ansible.utils` collection. Find more information about using these filters in the References note at the end of this section.



## References

### **ipaddr filter – Ansible Documentation**

[https://docs.ansible.com/ansible/6/collections/ansible/utils/docsite/filters\\_ipaddr.html](https://docs.ansible.com/ansible/6/collections/ansible/utils/docsite/filters_ipaddr.html)

## ► Guided Exercise

# Using Filters to Work with Network Addresses

Use filters and plug-ins to manipulate and validate data stored in variables that contain networking-related information.

### Outcomes

- Use the `ansible.utils.ipaddr` filter to derive networking information from IP address data retrieved from gathered facts.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command prepares an Ansible project in the `/home/student/data-netfilters/` directory on the `workstation` machine.

```
[student@workstation ~]$ lab start data-netfilters
```

### Instructions

- 1. The `lab` command provides the `/home/student/data-netfilters/default_ipv4.yml` playbook. Change to the `~/data-netfilters` directory and review the `default_ipv4.yml` playbook.

```
[student@workstation ~]$ cd ~/data-netfilters/
[student@workstation data-netfilters]$ cat default_ipv4.yml
- name: Display the default ipv4 address facts
 hosts: servera.lab.example.com
 become: false

 tasks:
 - name: Print the default ipv4 address facts
 ansible.builtin.debug:
 msg: "{{ ansible_default_ipv4 }}"
```

- 1.1. Run the `~/data-netfilters/default_ipv4.yml` playbook to list the `ansible_default_ipv4` facts.

```
[student@workstation data-netfilters]$ ansible-navigator run \
> -m stdout default_ipv4.yml

PLAY [Display the default ipv4 address facts] ****
TASK [Gathering Facts] ****
```

```

ok: [servera.lab.example.com]

TASK [Print the default ipv4 address facts] ****
ok: [servera.lab.example.com] => {
 "msg": {
 "address": "172.25.250.10",
 "alias": "eth0",
 "broadcast": "172.25.250.255",
 "gateway": "172.25.250.254",
 "interface": "eth0",
 "macaddress": "52:54:00:00:fa:0a",
 "mtu": 1500,
 "netmask": "255.255.255.0",
 "network": "172.25.250.0",
 "prefix": "24",
 "type": "ether"
 }
}

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

```

- ▶ 2. The `~/data-netfilters/filtering_data.yml` file contains incomplete tasks that use the `ansible.utils.ipaddr` filter with various options. Edit the `~/data-netfilters/filtering_data.yml` file, replacing any occurrence of `#FIXME#` in a fact definition with the expected value, as shown in the following substeps.

The `listips` variable that is used to generate the examples contains the following list:

```

listips:
 - 192.168.2.1
 - 10.0.0.128/25
 - 172.24.10.0/255.255.255.0
 - 172.24.10.0/255.255.255.255
 - ff02::1
 - ::1
 - 2001::1/64
 - 2001::/64
 - www.redhat.com

```

- 2.1. Review the first example, which uses `ansible.utils.ipaddr` without any options to output the items in the list that are valid IP addresses or networks. You do not need to change anything in that task.

```

- name: 1) Display all addresses
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr }}"

```

- 2.2. Use the `host` option with the `ansible.utils.ipaddr` filter to output only valid individual IP addresses, in the correct CIDR prefix format for host addresses.

**Chapter 7 |** Transforming Data with Filters and Plug-ins

```
- name: 2) Use host
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('host') }}"
```

- 2.3. Use the `net` option to output only values that are valid network specifications, converting the netmask to CIDR prefix format if necessary.

```
- name: 3) Use net
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('net') }}"
```

- 2.4. Use the `private` option to output only IP addresses or network ranges that are reserved by IANA to be private.

```
- name: 4) Use private
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('private') }}"
```

- 2.5. Use the `public` option to output only IP addresses and networks in globally routable public address spaces, according to IANA.

```
- name: 5) Use public
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('public') }}"
```

- 2.6. Use the `ansible.utils.ipwrap` filter to put brackets around the address part of IPv6 addresses.

```
- name: 6) Use ipwrap
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipwrap }}"
```

- 2.7. Most of the filters in the preceding examples removed the `www.redhat.com` list item, which is not an IP address, except for the `ansible.utils.ipwrap` filter. The following example uses the `ansible.utils.ipaddr` filter, followed by the `ansible.utils.ipwrap` filter, to exclude the `www.redhat.com` list item.

```
- name: 7) Use ipaddr and ipwrap
ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr | ansible.utils.ipwrap }}"
```

- 2.8. Verify that the completed `filtering_data.yml` playbook should consist of the following content:

```
Complete each task by setting the fact as the expected value.
Replace FIXME by the appropriate filter usage.

- name: First play for netfilter exercise
 hosts: servera.lab.example.com
 become: false
```

```

vars:
 listips:
 - 192.168.2.1
 - 10.0.0.128/25
 - 172.24.10.0/255.255.255.0
 - 172.24.10.0/255.255.255.255
 - ff02::1
 - ::1
 - 2001::1/64
 - 2001::/64
 - www.redhat.com

tasks:
 - name: 1) Display all addresses
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr }}"

 - name: 2) Use host
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('host') }}"

 - name: 3) Use net
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('net') }}"

 - name: 4) Use private
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('private') }}"

 - name: 5) Use public
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr('public') }}"

 - name: 6) Use ipwrap
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipwrap }}"

 - name: 7) Use ipaddr and ipwrap
 ansible.builtin.debug:
 msg: "{{ listips | ansible.utils.ipaddr | ansible.utils.ipwrap }}"

```

2.9. Run the `filtering_data.yml` playbook and verify the output of each task.

```

[student@workstation data-netfilters]$ ansible-navigator run \
> -m stdout filtering_data.yml

PLAY [First play for netfilter exercise] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [1) Display all addresses] ****
ok: [servera.lab.example.com] => {
 "msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/255.255.255.0",
 "172.24.10.0/255.255.255.255",
 "ff02::1",
 "::1",
 "2001::1/64",
 "2001::/64",
 "www.redhat.com"
]
}

```

```
"192.168.2.1",
"10.0.0.128/25",
"172.24.10.0/24",
"172.24.10.0/32",
"ff02::1",
":1",
"2001::1/64",
"2001::/64"
]
}

TASK [2) Use host] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "192.168.2.1/32",
 "172.24.10.0/32",
 "ff02::1/128",
 ":1/128",
 "2001::1/64"
]
}

TASK [3) Use net] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "10.0.0.128/25",
 "172.24.10.0/24",
 "2001::/64"
]
}

TASK [4) Use private] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/255.255.255.0",
 "172.24.10.0/255.255.255.255"
]
}

TASK [5) Use public] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "2001::1/64",
 "2001::/64"
]
}

TASK [6) Use ipwrap] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/255.255.255.0",
 "172.24.10.0/255.255.255.255"
]
}
```

**Chapter 7 |** Transforming Data with Filters and Plug-ins

```

 "172.24.10.0/255.255.255.255",
 "[ff02::1]",
 "[::1]",
 "[2001::1]/64",
 "[2001::]/64",
 "www.redhat.com"
]
}

TASK [7] Use ipaddr and ipwrap] *****
ok: [servera.lab.example.com] => {
 "msg": [
 "192.168.2.1",
 "10.0.0.128/25",
 "172.24.10.0/24",
 "172.24.10.0/32",
 "[ff02::1]",
 "[::1]",
 "[2001::1]/64",
 "[2001::]/64"
]
}

PLAY RECAP *****
servera.lab.example.com : ok=8 changed=0 unreachable=0 failed=0 ...

```

- ▶ 3. Alternatively, you can run the `filtering_data.yml` playbook with `ansible-navigator` in interactive mode to view the result of these tasks individually.

```
[student@workstation data-netfilters]$ ansible-navigator run filtering_data.yml
```

- 3.1. Press 0 to display details about the First play for netfilter exercise play.

| Play Name                  | Ok | Changed | Unreachable | ...Task count | Progress   |
|----------------------------|----|---------|-------------|---------------|------------|
| 0 First play for netfilter | 8  | 0       | 0           | ...           | 8 Complete |

`^f/PgUp page up`    `^b/PgDn page down`    `↑↓ scroll ...`    `help`    `Successful`

- 3.2. Press 2 to display details of the Use host task.

| Result | Host                           | Number   | Changed      | Task                     |
|--------|--------------------------------|----------|--------------|--------------------------|
| 0 ok   | servera.lab.example.com        | 0        | False        | Gathering Facts          |
| 1 ok   | servera.lab.example.com        | 1        | False        | 1) Display all addresses |
| 2 ok   | <b>servera.lab.example.com</b> | <b>2</b> | <b>False</b> | <b>2) Use host</b>       |
| 3 ok   | servera.lab.example.com        | 3        | False        | 3) Use net               |

*...output omitted...*

`^f/PgUp page up`    `^b/PgDn page down`    `↑↓ scroll ...`    `help`    `Successful`

- 3.3. Review the details to see the filtered IP addresses.

```
OK: [servera.lab.example.com] ['192.168.2.1/32', ... '2001::1/64']
...output omitted...
14| - 192.168.2.1/32
15| - 172.24.10.0/32
16| - ff02::1/128
17| - ::1/128
18| - 2001::1/64
19| start: '2022-11-15T17:26:19.910457'

...output omitted...
```

Press ESC three times to exit from the `ansible-navigator` command.

- ▶ **4.** Open the `~/data-netfilters/site.yml` playbook. The `tasks` section of the play in that playbook has two parts. The first set of tasks consists of `set_fact` tasks for you to complete with the `ansible.utils.ipaddr` filter. Replace any occurrence of `#FIXME#` in a fact definition with the value specified by the following substeps. The second set of tasks consists of `debug` tasks that print the information contained in the facts that you set.
- 4.1. Use `ansible.utils.ipaddr` filter to verify that the `ipv4_addr` variable contains a properly formatted IP address.

```
- name: Set facts derived from ipv4_addr and ipv4_subnet
 ansible.builtin.set_fact:
 server_address: "{{ ipv4_addr | ansible.utils.ipaddr }}"
```

- 4.2. Use the `revdns` option with `ansible.utils.ipaddr` filter to retrieve the IP address in DNS PTR record format.

```
- name: Set facts derived from ipv4_addr and ipv4_subnet
 ansible.builtin.set_fact:
 server_address: "{{ ipv4_addr | ansible.utils.ipaddr }}"
 ptr_record: "{{ ipv4_addr | ansible.utils.ipaddr('revdns') }}"
```

- 4.3. Use the `network/prefix` option to retrieve the network address and CIDR prefix.

```
- name: Set facts derived from net_mask
 ansible.builtin.set_fact:
 cidr: "{{ net_mask | ansible.utils.ipaddr('network/prefix') }}"
```

- 4.4. Use the `broadcast` option to retrieve the broadcast address.

```
- name: Set facts derived from net_mask
 ansible.builtin.set_fact:
 cidr: "{{ net_mask | ansible.utils.ipaddr('network/prefix') }}"
 broadcast: "{{ net_mask | ansible.utils.ipaddr('broadcast') }}"
```

- 4.5. Use the `cidr` variable, which you set in an earlier task, to provide the network and its CIDR prefix as an option to the `ansible.utils.ipaddr` filter so that the list it returns only includes IP addresses that are on that network.

```

- name: Print addresses on {{ cidr }} from this task's list
 ansible.builtin.debug:
 msg: "{{ item | ansible.utils.ipaddr(cidr) }}"
 loop:

```

**Important**

In the above example, `cidr` must *not* be in quotes because it is a variable defined earlier in the play that must be expanded, not a literal option for the filter.

4.6. Verify that the completed `site.yml` playbook consists of the following content:

```

Complete each task by setting the fact as the expected value.
Replace FIXME by the appropriate filter usage.
Tasks make use of the gathered fact 'default_ipv4', and its keys 'address', and
'netmask'.

- name: Second play for netfilter exercise
 hosts: servera.lab.example.com
 become: false

 tasks:
 - name: Set address and network facts
 ansible.builtin.set_fact:
 ipv4_addr: "{{ ansible_facts['default_ipv4']['address'] }}"
 ipv4_subnet: "{{ ansible_facts['default_ipv4']['netmask'] }}"

 - name: Set facts derived from ipv4_addr and ipv4_subnet
 ansible.builtin.set_fact:
 server_address: "{{ ipv4_addr | ansible.utils.ipaddr }}"
 ptr_record: "{{ ipv4_addr | ansible.utils.ipaddr('revdns') }}"
 net_mask: "{{ ipv4_addr }}/{{ ipv4_subnet }}"

 - name: Set facts derived from net_mask
 ansible.builtin.set_fact:
 cidr: "{{ net_mask | ansible.utils.ipaddr('network/prefix') }}"
 broadcast: "{{ net_mask | ansible.utils.ipaddr('broadcast') }}"

 - name: Display server_address
 ansible.builtin.debug:
 msg: server_address is {{ server_address }}

 - name: Display PTR record form for server_address
 ansible.builtin.debug:
 msg: ptr_record is {{ ptr_record }}

 - name: Display address and netmask in VLSM notation
 ansible.builtin.debug:
 msg: net_mask is {{ net_mask }}

 - name: Display address and prefix in CIDR notation

```

```

ansible.builtin.debug:
 msg: cidr is {{ cidr }}

- name: Display broadcast address for {{ cidr }}
 ansible.builtin.debug:
 msg: broadcast is {{ broadcast }}

- name: Print addresses on {{ cidr }} from this task's list
 ansible.builtin.debug:
 msg: "{{ item | ansible.utils.ipaddr(cidr) }}"
 loop:
 - 172.16.0.1
 - 172.25.250.4
 - 192.168.122.20
 - 192.0.2.55

```

4.7. Run the `site.yml` playbook and verify the output of each task.

```

[student@workstation data-netfilters]$ ansible-navigator run \
> -m stdout site.yml

PLAY [Second play for netfilter exercise] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Set address and network facts] ****
ok: [servera.lab.example.com]

TASK [Set facts derived from ipv4_addr and ipv4_subnet] ****
ok: [servera.lab.example.com]

TASK [Set facts derived from net_mask] ****
ok: [servera.lab.example.com]

TASK [Display server_address] ****
ok: [servera.lab.example.com] => {
 "msg": "server_address is 172.25.250.10"
}

TASK [Display PTR record form for server_address] ****
ok: [servera.lab.example.com] => {
 "msg": "ptr_record is 10.250.25.172.in-addr.arpa."
}

TASK [Display address and netmask in VLSM notation] ****
ok: [servera.lab.example.com] => {
 "msg": "net_mask is 172.25.250.10/255.255.255.0"
}

TASK [Display address and prefix in CIDR notation] ****
ok: [servera.lab.example.com] => {
 "msg": "cidr is 172.25.250.0/24"
}

```

```
TASK [Display broadcast address for 172.25.250.0/24] ****
ok: [servera.lab.example.com] => {
 "msg": "broadcast is 172.25.250.255"
}

TASK [Print addresses on 172.25.250.0/24 from this task's list] ****
ok: [servera.lab.example.com] => (item=172.16.0.1) => {
 "msg": ""
}
ok: [servera.lab.example.com] => (item=172.25.250.4) => {
 "msg": "172.25.250.4"
}
ok: [servera.lab.example.com] => (item=192.168.122.20) => {
 "msg": ""
}
ok: [servera.lab.example.com] => (item=192.0.2.55) => {
 "msg": ""
}

PLAY RECAP ****
servera.lab.example.com : ok=10 changed=0 unreachable=0 failed=0 ...
```

- 4.8. Run the `site.yml` playbook with the `ansible-navigator` command in interactive mode to view the result of these tasks one at a time.

```
[student@workstation data-netfilters]$ ansible-navigator run site.yml
```

You can investigate each result in the same way that you did in step 3.

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish data-netfilters
```

## ▶ Lab

# Transforming Data with Filters and Plug-ins

Use filters and plug-ins to transform data in variables and facts in order to use it for other tasks in the same play.

## Outcomes

- Use filters and lookup plug-ins to manipulate variables in playbook and role tasks.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the remote Git repository `https://git.lab.example.com/student/data-review.git` is initialized. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository using Student@123 as the Git password.

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

## Instructions

1. Clone the `https://git.lab.example.com/student/data-review.git` repository to the `/home/student/git-repos` directory and then create the `exercise` branch.  
The tasks file, `roles/firewall/tasks/main.yml`, contains three tasks that use the `firewalld` module to configure firewall rules defined by the `firewall_rules` variable.  
Edit the tasks file so that it uses filters to set default values for variables in each of the three tasks if they are not set, as follows:
  - For the `state` option, if `item['state']` is not set, then set it to `enabled` by default.
  - For the `zone` option, if `item['zone']` is not set, then omit the `zone` option.
  - In the "Ensure Firewall Port Configuration" task, in the `port` option to the `firewalld` module, if `item['protocol']` is not set then set it to `tcp`. Also use the `lower` filter to ensure that the value of the `item['protocol']` option is in lowercase.
2. Examine the `firewall` role in the `~/git-repos/data-review` project directory.  
If you completed the preceding step successfully, the playbook runs without errors.
3. Test the changes you made to the `roles/firewall/tasks/main.yml` file in the preceding step by running the `test_firewall_role.yml` playbook.  
If you completed the preceding step successfully, the playbook runs without errors.
4. Examine the `deploy_apache.yml` playbook. It calls the `apache` role to ensure that Apache HTTP Server is deployed, which itself calls the `firewall` role. The playbook also defines the `firewall_rules` variable to make sure that the `http` service for `firewalld` and the IPv4

address of the load balancer (172.25.250.10) are both enabled in the `internal` zone for `firewalld`.

Edit the playbook. You need to change the value of the `firewall_rules` variable in the `deploy_apache.yml` playbook to a Jinja2 expression that uses the `template` lookup plug-in to dynamically generate the variable's setting from the `apache_firewall_rules.yml.j2` Jinja2 template. You must use the `from_yaml` filter in the Jinja2 expression to convert the resulting value from a text string into a YAML data structure that can be interpreted by Ansible.

5. Examine the `templates/apache_firewall_rules.yml.j2` template that your playbook now uses to set the `firewall_rules` variable.

For the first rule, the `apache_port` variable is set by files in the `group_vars` directory.

The template contains a Jinja2 `for` loop that creates a rule for each host in the `lb_servers` group, setting the source IP address from the value of the `load_balancer_ip_addr` variable.

This is not the best solution. It requires you to set `load_balancer_ip_addr` as a host variable for each load balancer in the `lb_servers` group. To remove this manual maintenance, use gathered facts from these hosts to set that value instead.

Edit the `templates/apache_firewall_rules.yml.j2` template to replace the `load_balancer_ip_addr` host variable in the Jinja2 `for` loop with the `hostvars[server]['ansible_facts']['default_ipv4']['address']` fact.

6. Run the `site.yml` playbook to test your changes. If you successfully completed the preceding steps, then the playbook runs without errors.
7. Verify that web browser requests from `workstation` to the load balancer on `servera` succeed, and that direct requests from `workstation` to either TCP port 80 or TCP port 8008 on the back-end web servers, `serverb` and `serverc`, are denied.
8. Save your work, and then use Git to commit your changes and push them to the remote Git repository. If prompted, use `Student@123` as the Git password.

## Evaluation

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful. You *must* add, commit, and push any additional changes to the remote Git repository before grading.

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

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

## ► Solution

# Transforming Data with Filters and Plug-ins

Use filters and plug-ins to transform data in variables and facts in order to use it for other tasks in the same play.

### Outcomes

- Use filters and lookup plug-ins to manipulate variables in playbook and role tasks.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the remote Git repository `https://git.lab.example.com/student/data-review.git` is initialized. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository using Student@123 as the Git password.

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

### Instructions

1. Clone the `https://git.lab.example.com/student/data-review.git` repository to the `/home/student/git-repos` directory and then create the `exercise` branch.
  - 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/data-review.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/data-review.git
Cloning into 'data-review'...
...output omitted...
[student@workstation git-repos]$ cd data-review
```

- 1.3. Create the `exercise` branch and check it out.

```
[student@workstation data-review]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

2. Examine the `firewall` role in the `~/git-repos/data-review` project directory.

The tasks file, `roles/firewall/tasks/main.yml`, contains three tasks that use the `firewalld` module to configure firewall rules defined by the `firewall_rules` variable.

Edit the tasks file so that it uses filters to set default values for variables in each of the three tasks if they are not set, as follows:

- For the `state` option, if `item['state']` is not set, then set it to `enabled` by default.
- For the `zone` option, if `item['zone']` is not set, then omit the `zone` option.
- In the "Ensure Firewall Port Configuration" task, in the `port` option to the `firewalld` module, if `item['protocol']` is not set then set it to `tcp`. Also use the `lower` filter to ensure that the value of the `item['protocol']` option is in lowercase.

2.1. Edit the `roles/firewall/tasks/main.yml` file. For the `state` option of all three `firewalld` tasks, add the `default('enabled')` filter to the `item['state']` Jinja2 expression.

The resulting file contains the following content:

```
- name: Ensure Firewall Sources Configuration
ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['source'] is defined

- name: Ensure Firewall Service Configuration
ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['service'] is defined

- name: Ensure Firewall Port Configuration
ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] }}"
 zone: "{{ item['zone'] }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['port'] is defined
```

2.2. Continue editing the tasks file. For the `zone` option of all three `firewalld` tasks, add the `default(omit)` filter to the `item['zone']` Jinja2 expression.

**Note**

If desired, you can use the `default(omit, true)` filter instead. Adding `true` to the `default` filter means that the filter is also used if the variable is either an empty string or is set to the `false` Boolean value.

The `default` filter is always used if the variable is undefined.

After this step, the task file contains the following content:

```
- name: Ensure Firewall Sources Configuration
ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['source'] is defined

- name: Ensure Firewall Service Configuration
ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['service'] is defined

- name: Ensure Firewall Port Configuration
ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['port'] is defined
```

- 2.3. Edit the task file again. In the third `firewalld` task, "Ensure Firewall Port Configuration", replace `{{ item['protocol'] }}` with `{{ item['protocol'] | default('tcp') | lower }}`. Save your work.

The completed task file contains the following content:

```
- name: Ensure Firewall Sources Configuration
ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
```

```

loop: "{{ firewall_rules }}"
when: item['source'] is defined

- name: Ensure Firewall Service Configuration
 ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['service'] is defined

- name: Ensure Firewall Port Configuration
 ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] | default('tcp') | lower }}"
 zone: "{{ item['zone'] | default(omit) }}"
 immediate: true
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['port'] is defined

```

3. Test the changes you made to the `roles/firewall/tasks/main.yml` file in the preceding step by running the `test_firewall_role.yml` playbook.

If you completed the preceding step successfully, the playbook runs without errors.

Run the `test_firewall_role.yml` playbook to test your changes:

```

[student@workstation data-review]$ ansible-navigator run \
> -m stdout test_firewall_role.yml

PLAY [Test Firewall Role] *****
...output omitted...

PLAY RECAP *****
serverb.lab.example.com : ok=4 changed=2 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=4 changed=2 unreachable=0 failed=0 ...

```

4. Examine the `deploy_apache.yml` playbook. It calls the `apache` role to ensure that Apache HTTP Server is deployed, which itself calls the `firewall` role. The playbook also defines the `firewall_rules` variable to make sure that the `http` service for `firewalld` and the IPv4 address of the load balancer (`172.25.250.10`) are both enabled in the `internal` zone for `firewalld`.

Edit the playbook. You need to change the value of the `firewall_rules` variable in the `deploy_apache.yml` playbook to a Jinja2 expression that uses the `template` lookup plug-in to dynamically generate the variable's setting from the `apache_firewall_rules.yml.j2` Jinja2 template. You must use the `from_yaml` filter in the Jinja2 expression to convert the resulting value from a text string into a YAML data structure that can be interpreted by Ansible.

One correct solution results in the following contents in the `deploy_apache.yml` Ansible Playbook:

```
- name: Ensure Apache is deployed
 hosts: web_servers
 force_handlers: true
 gather_facts: false

 roles:
 # Use the apache_firewall_rules.yml.j2 template to
 # generate the firewall rules.
 - role: apache
 firewall_rules: "{{ lookup('ansible.builtin.template',
'apache_firewall_rules.yml.j2') | from_yaml }}"
```



### Note

In the preceding example, ensure you enter the expression for the `firewall_rules` variable on a single line.

If you use linting rules that indicate that lines should not exceed 80 characters, then you might define the Jinja2 expression over multiple lines by defining the `firewall_rules` variable using a greater than sign. For example:

```
firewall_rules: >
 {{ lookup('ansible.builtin.template', 'apache_firewall_rules.yml.j2')
 | from_yaml }}
```

5. Examine the `templates/apache_firewall_rules.yml.j2` template that your playbook now uses to set the `firewall_rules` variable.

For the first rule, the `apache_port` variable is set by files in the `group_vars` directory.

The template contains a Jinja2 `for` loop that creates a rule for each host in the `lb_servers` group, setting the source IP address from the value of the `load_balancer_ip_addr` variable.

This is not the best solution. It requires you to set `load_balancer_ip_addr` as a host variable for each load balancer in the `lb_servers` group. To remove this manual maintenance, use gathered facts from these hosts to set that value instead.

Edit the `templates/apache_firewall_rules.yml.j2` template to replace the `load_balancer_ip_addr` host variable in the Jinja2 `for` loop with the `hostvars[server]['ansible_facts']['default_ipv4']['address']` fact.

The completed `templates/apache_firewall_rules.yml.j2` template should contain the following content:

```
- port: {{ apache_port }}
 protocol: TCP
 zone: internal
{% for server in groups['lb_servers'] %}
- zone: internal
 source: {{ hostvars[server]['ansible_facts']['default_ipv4']['address'] }}
{% endfor %}
```

Save the template.

**Chapter 7 |** Transforming Data with Filters and Plug-ins

6. Run the `site.yml` playbook to test your changes. If you successfully completed the preceding steps, then the playbook runs without errors.

```
[student@workstation data-review]$ ansible-navigator run site.yml -m stdout
...output omitted...
[student@workstation data-review]$ echo $?
0
```

7. Verify that web browser requests from `workstation` to the load balancer on `servera` succeed, and that direct requests from `workstation` to either TCP port 80 or TCP port 8008 on the back-end web servers, `serverb` and `serverc`, are denied.

```
[student@workstation data-review]$ curl servera
This is serverb. (version v1.0)
[student@workstation data-review]$ curl servera
This is serverc. (version v1.0)
[student@workstation data-review]$ curl serverb; curl serverb:8008
curl: (7) Failed to connect to serverb port 80: No route to host
curl: (7) Failed to connect to serverb port 8008: No route to host
[student@workstation data-review]$ curl serverc; curl serverc:8008
curl: (7) Failed to connect to serverc port 80: No route to host
curl: (7) Failed to connect to serverc port 8008: No route to host
```

8. Save your work, and then use Git to commit your changes and push them to the remote Git repository. If prompted, use `Student@123` as the Git password.

```
[student@workstation data-review]$ git status
On branch exercise
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git restore <file>..." to discard changes in working directory)
 modified: deploy_apache.yml
 modified: roles/firewall/tasks/main.yml
 modified: templates/apache_firewall_rules.yml.j2

no changes added to commit (use "git add" and/or "git commit -a")
[student@workstation data-review]$ git add deploy_apache.yml \
> roles/firewall/tasks/main.yml templates/apache_firewall_rules.yml.j2
[student@workstation data-review]$ git commit \
> -m "Added Filters and Plug-ins"
...output omitted...
[student@workstation data-review]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
```

## Evaluation

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful. You *must* add, commit, and push any additional changes to the remote Git repository before grading.

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

## Finish

On the workstation machine, change to the student user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

# Summary

---

- You can use filters in Jinja2 expressions and templates to format, transform, and parse data.
- Some filters are native to Jinja2 and some are provided by Ansible or by Ansible Content Collections.
- You can apply multiple filters sequentially in the same Jinja2 expression.
- Lookup plug-ins get data from external sources for plays.
- Lookup plug-ins and filters run in the execution environment, not on the managed host.
- You can combine filters and lookup plug-ins to implement sophisticated loops.
- Ansible provides many supplementary filters for processing networking-related data, such as `ansible.utils.ipaddr`.



## Chapter 8

# Coordinating Rolling Updates

### Goal

Use advanced features of Ansible to manage rolling updates in order to minimize downtime, and to ensure the maintainability and simplicity of Ansible Playbooks.

### Objectives

- Run a task for a managed host on a different host, and control whether facts gathered by that task are delegated to the managed host or to the other host.
- Tune the number of simultaneous connections that Ansible opens to managed hosts, and how Ansible processes groups of managed hosts through the play's tasks.
- Tune the behavior of the `serial` directive when batching hosts for execution, abort the play if it fails for too many hosts, and create tasks that run only once for each batch or for all hosts in the inventory.

### Sections

- Delegating Tasks and Facts (and Guided Exercise)
- Configuring Parallelism (and Guided Exercise)
- Managing Rolling Updates (and Guided Exercise)

### Lab

- Coordinating Rolling Updates

# Delegating Tasks and Facts

## Objectives

- Run a task for a managed host on a different host, and control whether facts gathered by that task are delegated to the managed host or to the other host.

## Delegating Tasks

Sometimes, when Ansible is running a play to ensure the correct configuration of a system, it might need to perform one or more tasks on another host on behalf of the managed host. For example, you might need to log in to a network device to change a DHCP configuration, make sure that certain groups exist in an Active Directory domain, or communicate with the API of a service by using tools that are not available on the managed host.

In a play, you can *delegate* a task to run on another host instead of the current managed host.

A task delegates the action to a host by using the `delegate_to` directive. This directive points Ansible to the host that executes the task in place of the corresponding target. The host that you delegate the task to does not need to be listed in the Ansible inventory.

The following example runs the `uname -a` command on each host in the play, and then runs the `uname -a` command on `host.lab.example.com` on behalf of each host in the play.

```

- name: Delegation Example
 hosts: demo.lab.example.com
 become: false

 tasks:
 - name: Get system information
 ansible.builtin.command: uname -a
 register: managed_host

 - name: Display demo system information
 ansible.builtin.debug:
 var: managed_host

 - name: Get system information
 ansible.builtin.command: uname -a
 delegate_to: host.lab.example.com
 register: delegated

 - name: Display localhost system information
 ansible.builtin.debug:
 var: delegated
```

In the following example, the first task is delegated to each of the HAProxy load balancers in the `lbServers` Ansible group in turn, removing the managed host from all the load balancers. The

second task, which is not delegated, then stops the web server on the managed host. Both tasks run for each host in the play.

```
- name: Remove the server from HAProxy
 community.general.haproxy:
 state: disabled
 host: "{{ ansible_facts['fqdn'] }}"
 socket: /var/lib/haproxy/stats
 delegate_to: "{{ item }}"
 loop: "{{ groups['lbervers'] }}"

- name: Make sure Apache HTTPD is stopped
 ansible.builtin.service:
 name: httpd
 state: stopped
```

You might also delegate tasks to another host when you want to verify access to a service on the managed host currently being processed by Ansible. For example, you might want to ensure that communication from a test client can pass through a host-based firewall running on that host.

```
- name: Access the web service
 uri:
 url: http://{{ ansible_facts['fqdn'] }}
 timeout: 5
 delegate_to: test-client.example.com
```

## Delegating to the Execution Environment

One of the most common places you might delegate a task is to the execution environment, or `localhost` environment. For example, you need to communicate with the API for a controller that you cannot reach from the managed host, but that you can reach from the execution environment, so you delegate the task.

```
- name: Get information about controller instance
 ansible.builtin.uri:
 url: https://{{ ansible_facts['fqdn'] }}/api/v2/ping/
 method: GET
 validate_certs: no
 return_content: yes
 delegate_to: localhost
 register: controller_ping
```

**Important**

The introduction of automation execution environments in Red Hat Ansible Automation Platform 2 changes how delegation to `localhost` works. With the `ansible-navigator` command, using `delegate_to: localhost` causes the module to run *within* the automation execution environment container. In earlier versions that used the `ansible-playbook` command, the module ran directly on the control node, with various consequences.

- Privilege escalation might fail if the `sudo` command is not present. The supported automation execution environments provided by Red Hat require the `sudo` command.
- If you are reusing a playbook that assumes it can access resources or commands on the control node, those resources might not be available inside the automation execution environment container.

## Delegating Facts

In an earlier example, the `ansible_facts['fqdn']` fact was used in a task that was delegated to each of the load balancers. In that task, the fully qualified domain name (FQDN) for the managed host is used, not that of the current load balancer.

When you delegate a task, Ansible uses the host variables and facts for the managed host (the current `inventory_hostname`) for which the task is running. For example, if a task is running for `demo`, but has been delegated to `localhost`, then the variables and facts for `demo` are used. In most cases, it is correct to use the variables and facts for the managed host, even when the task is delegated to another host.

However, sometimes you might want to assign the facts collected by a delegated task to the host to which the task was delegated. To change this setting, set the `delegate_facts` directive to `true`.

```
- name: Delegate Fact Example
hosts: localhost
gather_facts: false
tasks:
 - name: Set a fact in delegated task on demo
 ansible.builtin.set_fact:
 myfact: Where am I set?
 delegate_to: demo.lab.example.com
 delegate_facts: true

 - name: Display the facts from demo.lab.example.com
 ansible.builtin.debug:
 msg: "{{ hostvars['demo.lab.example.com']['myfact'] }}"
```

The preceding play runs for `localhost`, but the first task is delegated to `demo.lab.example.com`. Using the `delegate_facts` directive on that task instructs Ansible to gather facts into the `hostvars['demo.lab.example.com']` namespace for the delegated host, instead of the default `hostvars['localhost']` namespace for the current managed host.



## References

### Controlling where tasks run: delegation and local actions – Ansible Documentation

[https://docs.ansible.com/ansible/6/user\\_guide/playbooks\\_delegation.html](https://docs.ansible.com/ansible/6/user_guide/playbooks_delegation.html)

## ► Guided Exercise

# Delegating Tasks and Facts

Run a playbook that includes a task that is delegated to another host.

### Outcomes

- Delegate a task to run on another host.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start update-delegation
```

### Instructions

- 1. Clone the `https://git.lab.example.com/student/update-delegation.git` Git repository to the `/home/student/git-repos` directory and then create a branch for this exercise.
- From a terminal, create the `/home/student/git-repos` directory if it does not already exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- Clone the `https://git.lab.example.com/student/update-delegation.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/update-delegation.git
Cloning into 'update-delegation'...
...output omitted...
[student@workstation git-repos]$ cd update-delegation
```

- Create the `exercise` branch.

```
[student@workstation update-delegation]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 2. Review the contents of the `inventory.yml` inventory file. Verify that the `web_servers` host group contains servers A to F.

```
web_servers:
 hosts:
 server[a:f].lab.example.com:
```

- 3. Add a task to the `query_times.yml` playbook that perform the following actions:
- Query the time on each server in the `web_servers` host group.
  - Store the results in the `/tmp/times.txt` file on the `workstation` machine.
- 3.1. Add a task that adds two lines of text for each server to the `/tmp/times.txt` file on the `workstation` machine. The first line in that file contains the name of a web server. The second line contains the time on the managed host. For both lines, the task uses the facts gathered from the `web_servers` host group. If the file does not exist, it is created.
- Ensure that each line of text appends to the end of the `/tmp/times.txt` file on the `workstation` machine by delegating this task for each managed host to `workstation`.

The task should consist of the following content:

```
- name: Save server times to workstation
 ansible.builtin.lineinfile:
 path: /tmp/times.txt
 state: present
 mode: '0644'
 create: true
 insertafter: EOF
 line: |
 {{ ansible_facts['fqdn'] }}
 {{ '%c' | strftime(ansible_facts['date_time']['epoch']) }}
```

`delegate_to: workstation`



### Note

You can use the `strftime` filter to display time in a particular format by passing a format to the filter. Consult the `date(1)` man page for available formats.

- 4. Run the playbook by using the `ansible-navigator run` command to test the delegation of tasks:

```
[student@workstation update-delegation]$ ansible-navigator run \
> -m stdout query_times.yml

PLAY [Query server times and store them locally] ****...
TASK [Gathering Facts] ****...
ok: [servera.lab.example.com]
...output omitted...
ok: [serverf.lab.example.com]

TASK [Save server times to workstation] ****...
changed: [servera.lab.example.com -> workstation]
```

```
...output omitted...
changed: [serverf.lab.example.com -> workstation]

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 ...
serverc.lab.example.com : ok=2 changed=1 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=2 changed=1 unreachable=0 failed=0 ...
servere.lab.example.com : ok=2 changed=1 unreachable=0 failed=0 ...
serverf.lab.example.com : ok=2 changed=1 unreachable=0 failed=0 ...
```

- 5. Verify that the playbook has gathered the times correctly and that Ansible stored the information in the /tmp/times.txt file on the workstation machine.

```
[student@workstation update-delegation]$ cat /tmp/times.txt
serverb.lab.example.com
Thu Dec 1 14:49:24 2022

servera.lab.example.com
Thu Dec 1 14:49:24 2022

serverd.lab.example.com
Thu Dec 1 14:49:24 2022

serverc.lab.example.com
Thu Dec 1 14:49:24 2022

servere.lab.example.com
Thu Dec 1 14:49:24 2022

serverf.lab.example.com
Thu Dec 1 14:49:25 2022
```

The order of hosts in your file might differ. There might be multiple entries for each host if you run the playbook multiple times.

## Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish update-delegation
```

# Configuring Parallelism

## Objectives

- Tune the number of simultaneous connections that Ansible opens to managed hosts and how Ansible processes groups of managed hosts through the play's tasks.

## Configure Parallelism in Ansible Using Forks

When Ansible processes a playbook, it runs each play in order. After determining the list of hosts for the play, Ansible runs through each task in order. Normally, all hosts must successfully complete a task before any host starts the next task in the play.

In theory, Ansible could connect to all hosts in the play simultaneously for each task. This approach works fine for small lists of hosts, but if the play targets hundreds of hosts, it can put a heavy load on the control node and the automation execution environment.

The maximum number of simultaneous connections that Ansible makes is controlled by the `forks` parameter in the Ansible configuration file. This parameter is set to 5 by default, which you can verify by using one of the following methods:

```
[user@host demo]$ ansible-navigator config dump -m stdout
...output omitted...
DEFAULT_FORCE_HANDLERS(default) = False
DEFAULT_FORKS(default) = 5
DEFAULT_GATHERING(default) = implicit
...output omitted...
```

```
[user@host demo]$ ansible-navigator config list -m stdout
...output omitted...
DEFAULT_FORKS:
 default: 5
 description: Maximum number of forks Ansible will use to execute tasks on target
 hosts.
 env:
 - name: ANSIBLE_FORKS
 ini:
 - key: forks
 section: defaults
 name: Number of task forks
 type: integer
...output omitted...
```



### Note

You can search the output of the previous two commands. To search for the default `forks` parameter type /`DEFAULT_FORKS` and press Enter.

## Chapter 8 | Coordinating Rolling Updates

For example, assume that Ansible is configured with the default value of five forks and that the play has ten managed hosts. Ansible runs the first task in the play on the first five managed hosts, followed by a second round of execution of the first task on the other five managed hosts. After running the first task on all the managed hosts, Ansible runs the next task across all the managed hosts in groups of five hosts at a time. Ansible does this with each task in turn until the play ends.

The default setting for `forks` is conservative. If a play that is running in your execution environment manages Linux hosts, most tasks run on the managed hosts, and the execution environment has less load. In this case, you can usually set the `forks` parameter to a much higher value, possibly closer to 100, and realize performance improvements.

If your plays run a lot of code in the execution environment, then you should raise the fork limit judiciously. If you use Ansible to manage network routers and switches, then most of those modules run in the execution environment and not on the network device. Because of the increased load this places on the execution environment, the number of forks should be significantly lower compared to a play running in an execution environment that manages only Linux hosts.

You can override the default setting for the `forks` option from the command line by modifying the Ansible configuration file. The `ansible-navigator run` command offers the `-f` or `--forks` option to specify the number of forks.

## Running Batches of Hosts Through the Entire Play

Normally, when Ansible runs a play, it ensures that all managed hosts complete each task before it starts the next task. After all managed hosts have completed all tasks, then any notified handlers are run.

However, running all tasks on all hosts can lead to undesirable behavior. For example, if a play updates a cluster of load-balanced web servers, it might need to take each web server out of service during the update. If all the servers are updated in the same play, they could all be out of service simultaneously.

One way to avoid this problem is to use the `serial` keyword to run the hosts through the play in batches. The play processes each batch of hosts in sequence.

In the example below, Ansible runs the play on two managed hosts simultaneously until all managed hosts have been updated. Ansible first runs the tasks in the play on the first two managed hosts. If any of those hosts notifies a handler, then Ansible runs the handler. When the play is complete on those two managed hosts, Ansible repeats the process on the next two managed hosts. Ansible continues to run the play in this way until all managed hosts are updated.

```

- name: Rolling update
 hosts: webservers
 serial: 2
 tasks:
 - name: Latest apache httpd package is installed
 ansible.builtin.yum:
 name: httpd
 state: latest
 notify: restart apache

 handlers:
 - name: Restart apache
```

```
ansible.builtin.service:
 name: httpd
 state: restarted
```

Suppose that the `webservers` group in the previous example contains five web servers behind a load balancer. With the `serial` parameter set to 2, the play runs on up to two web servers simultaneously. Thus, a majority of the five web servers is always available.

In contrast, in the absence of the `serial` keyword, the play and resulting handler would run across all five web servers at the same time. This approach would probably lead to a service outage because web services on all the web servers would be restarted at the same time.



### Important

For certain purposes, each batch of hosts counts as if it was a full play running on a subset of hosts. This means that if an entire batch fails, then the play fails, which causes the entire playbook run to fail.

In the previous scenario with the `serial` parameter set to 2, if something is wrong and the play fails for the first two processed hosts, then the playbook aborts and the play does not run on the remaining three hosts. This is a useful feature because only a subset of the servers would be unavailable, leaving the service degraded rather than down.

The `serial` keyword can also be specified as a percentage. This percentage is applied to the total number of hosts in the play to determine the rolling update batch size. Regardless of the percentage, the number of hosts per pass is always one or greater.



### References

**Setting the batch size with serial – Controlling playbook execution: strategies and more – Ansible Documentation**

[https://docs.ansible.com/ansible/6/user\\_guide/playbooks\\_strategies.html#setting-the-batch-size-with-serial](https://docs.ansible.com/ansible/6/user_guide/playbooks_strategies.html#setting-the-batch-size-with-serial)

**Ansible Performance Tuning (For Fun and Profit)**

<https://www.ansible.com/blog/ansible-performance-tuning>

## ► Guided Exercise

# Configuring Parallelism

Explore the effects of different `serial` and `forks` directives on how Ansible processes a play.

### Outcomes

- Tune parallel and serial executions of a playbook across multiple managed hosts.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the environment is ready for this exercise.

```
[student@workstation ~]$ lab start update-parallelism
```

### Instructions

- 1. Clone the `https://git.lab.example.com/student/update-parallelism.git` Git repository to the `/home/student/git-repos` directory and then create a branch for this exercise.
- 1.1. Create the `/home/student/git-repos` directory if it does not exist, and then change to this directory.

```
[student@workstation ~]$ mkdir -p ~/git-repos
[student@workstation ~]$ cd ~/git-repos
```

- 1.2. Clone the repository `https://git.lab.example.com/student/update-parallelism.git`, and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/update-parallelism.git
Cloning into 'update-parallelism'...
...output omitted...
[student@workstation git-repos]$ cd update-parallelism
```

- 1.3. Create the `exercise` branch.

```
[student@workstation update-parallelism]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 2. Examine the contents of the project directory to become familiar with the project files.

- 2.1. Examine the contents of the `ansible.cfg` file. Note that the inventory file is set to `inventory`, the `forks` parameter is set to 4, and the `ansible.posix.timer` and `ansible.posix.profile_tasks` callback plug-ins are enabled.

```
[defaults]
inventory=inventory
remote_user=devops
forks=4
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks
```

- 2.2. Examine the contents of the `inventory` file. Note that it contains a host group, `webservers`, that contains four hosts.

```
[webservers]
servera.lab.example.com
serverb.lab.example.com
serverc.lab.example.com
serverd.lab.example.com
```

- 2.3. Examine the contents of the `update_httpd.yml` playbook. This playbook runs against the `webservers` host group, ensuring that the latest `httpd` package is installed and that the `httpd` service is enabled and started.

```

- name: Update the web server
 hosts: webservers
 become: true
 gather_facts: false

 tasks:
 - name: Latest httpd package installed
 ansible.builtin.yum:
 name: httpd
 state: latest
 notify:
 - Restart httpd

 handlers:
 - name: Restart httpd
 ansible.builtin.service:
 name: httpd
 enabled: true
 state: restarted
```

- 2.4. Examine the contents of the `remove_httpd.yml` playbook. This playbook runs against the `webservers` host group, ensuring that the `httpd` service is disabled and stopped and that the `httpd` package is not installed.

```

- name: Remove the web server
 hosts: webservers
 become: true
 gather_facts: false
```

```
tasks:
 - name: Query service facts
 ansible.builtin.service_facts:

 - name: Disable httpd service
 ansible.builtin.service:
 name: httpd
 enabled: false
 state: stopped
 when: ansible_facts['services']['httpd.service'] is defined

 - name: Remove httpd package
 ansible.builtin.yum:
 name: httpd
 state: absent
 autoremove: true
```

- 3. Use the `ansible-navigator run` command to run the `update_httpd.yml` playbook.

- 3.1. Watch the playbook as it runs and note how Ansible performs each task on all four hosts at the same time. Also, note the output from the callback plug-ins to determine how long it takes the `ansible-navigator` command to run the playbook.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout update_httpd.yml

PLAY [Update the web server] ****
...output omitted...

PLAY RECAP ****
servera.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverb.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
Playbook run took 0 days, 0 hours, 0 minutes, 12 seconds
Monday 05 December 2022 20:21:04 +0000 (0:00:01.638) 0:00:12.687 ****
=====
Latest httpd package installed ----- 11.02s
Restart httpd ----- 1.64s
```

- 3.2. Run the `remove_httpd.yml` playbook to stop and disable the `httpd` service and remove the `httpd` package.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout remove_httpd.yml
...output omitted...
```

- 4. Change the value of the `forks` parameter to `1` in the `ansible.cfg` file, and then time the execution of the `update_httpd.yml` playbook again.

- 4.1. Change the value of the `forks` parameter to `1` in the `ansible.cfg` file.

```
[defaults]
inventory=inventory
remote_user=devops
forks=1
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks
```

- 4.2. Run the `update_httpd.yml` playbook again. Watch the playbook as it runs. This time Ansible performs each task on one host at a time. Notice how decreasing the number of forks causes the playbook execution to take longer.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout update_httpd.yml

PLAY [Update the web server] ****
...output omitted...

PLAY RECAP ****
servera.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverb.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=2 changed=2 unreachable=0 failed=0 ...
Playbook run took 0 days, 0 hours, 0 minutes, 23 seconds
Monday 05 December 2022 20:28:31 +0000 (0:00:05.212) 0:00:23.075 ****
=====
Latest httpd package installed ----- 17.84s
Restart httpd ----- 5.21s
```

- 4.3. Run the `remove_httpd.yml` playbook to stop and disable the `httpd` service and remove the `httpd` package.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout remove_httpd.yml
...output omitted...
```

- 5. Add the `serial` parameter to the play in the `update_httpd.yml` playbooks so that the play only runs on two hosts simultaneously. The beginning of the playbook should consist of the following content:

```

- name: Update the web server
hosts: webservers
become: true
gather_facts: false
serial: 2
```

- 6. Set the value of the `forks` parameter to 2 in the `ansible.cfg` file. The `forks` parameter must be at least equal to the value for of the `serial` parameter; otherwise, it restricts execution speed.

**Chapter 8 |** Coordinating Rolling Updates

```
[defaults]
inventory=inventory
remote_user=devops
forks=2
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks
```

- 7. Run the `update_httpd.yml` playbook again.

- 7.1. Run the `update_httpd.yml` playbook again, and watch as it runs. Note that Ansible runs through the entire play on two hosts before rerunning the play on the two remaining hosts.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout update_httpd.yml

PLAY [Update the web server] ****

TASK [Latest httpd package installed] ****
Monday 05 December 2022 20:32:49 +0000 (0:00:00.024) 0:00:00.024 ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

...output omitted...

PLAY [Update the web server] ****

TASK [Latest httpd package installed] ****
Monday 05 December 2022 20:32:56 +0000 (0:00:01.633) 0:00:06.508 ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

...output omitted...
```

- 7.2. Run the `remove_httpd.yml` playbook to stop and disable the `httpd` service and remove the `httpd` package.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout remove_httpd.yml
...output omitted...
```

- 8. Set the value of the `forks` parameter back to 4 in the `ansible.cfg` file.

```
[defaults]
inventory=inventory
remote_user=devops
forks=4
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks
```

- 9. Set the `serial` parameter in the `update_httpd.yml` playbook to 3. The beginning of the playbook should consist of the following content:

```

- name: Update the web server
hosts: webservers
become: true
gather_facts: false
serial: 3
```

- ▶ 10. Run the `update_httpd.yml` playbook again, and watch as it runs. Note that Ansible runs through the entire play on three hosts, and then runs the play on the one remaining host.

```
[student@workstation update-parallelism]$ ansible-navigator run \
> -m stdout update_httpd.yml

PLAY [Update the web server] *****

TASK [Latest httpd package installed] *****
Monday 05 December 2022 20:36:44 +0000 (0:00:00.026) 0:00:00.026 ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
changed: [serverc.lab.example.com]

...output omitted...

PLAY [Update the web server] *****

TASK [Latest httpd package installed] *****
Monday 05 December 2022 20:36:51 +0000 (0:00:01.666) 0:00:06.753 ****
changed: [serverd.lab.example.com]

...output omitted...
```

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish update-parallelism
```

# Managing Rolling Updates

## Objectives

- Tune the behavior of the `serial` directive when batching hosts for execution, abort the play if it fails for too many hosts, and create tasks that run only once for each batch or for all hosts in the inventory.

## Overview

Ansible has several features that enable *rolling updates*, a strategy that staggers deployments to batches of servers. With this strategy, infrastructure deployment can complete with zero downtime.

Ansible can halt the deployment and limit the errors to servers in a particular batch when an unforeseen problem occurs. With tests and monitoring in place, you can configure playbook tasks to perform the following actions:

- Roll back configuration for hosts in the affected batch.
- Quarantine affected hosts to enable analysis of the failed deployment.
- Send notifications of the deployment to stakeholders.

## Controlling Batch Size

By default, Ansible runs a play by running one task for all hosts before running the next task. Imagine that your play has a task that cannot succeed due to some error. If your play runs on all hosts in parallel, they all fail and abort the play when Ansible reaches that task, and no host completes the play. As a result, it is likely that none of the hosts are working correctly, which could lead to an outage.

To avoid this, you could process some of the hosts through the play before starting the next batch of hosts. If too many hosts fail, then you could have Ansible abort the entire play before all of your hosts attempt to run it.

The following sections describe how to configure your play to abort if too many hosts in a batch fail.

## Setting a Fixed Batch Size

To process hosts through a play in batches, use the `serial` directive in your play. The `serial` directive specifies how many hosts should be in each batch. Ansible processes each batch of hosts through the play before starting the next batch. If all hosts in the current batch fail, the entire play is aborted, and Ansible does not start the next batch.

Consider the beginning portion of this play:

```

- name: Update Webservers
 hosts: web_servers
 serial: 2
```

## Chapter 8 | Coordinating Rolling Updates

In this example, the `serial` directive instructs Ansible to process hosts in the `web_servers` host group in batches of two hosts. The play is repeated with a new batch if the play runs without error.

This process continues until all the hosts in the play are processed. If the total number of hosts in the play is not divisible by the batch size, then the last batch might contain fewer hosts than the indicated value of the `serial` directive. In the previous example, the last batch contains one host if the total number of web servers is an odd value.

Remember, if you use an integer with the `serial` directive, then as the number of hosts in the play increases, the number of batches needed to complete the play also increases. With a `serial` value of 2, a host group with 200 hosts requires 100 batches to complete, but a host group with 20 hosts requires ten batches.

## Setting Batch Size as a Percentage

You can also specify a percentage for the value of the `serial` directive:

```

- name: Update Webservers
 hosts: web_servers
 serial: 25%
```

The play runs on the percentage of the hosts that you specify.

Ansible applies the percentage to the total number of hosts in the host group. If the resulting value is not an integer number of hosts, then the value is truncated (rounded down) to the nearest integer. The batch size cannot be zero hosts. If the truncated value is zero, Ansible changes the batch size to one host.

The following table illustrates the use of a percentage and the resulting size and number of batches:

| Description           | Host group 1 | Host group 2 | Host group 3 |
|-----------------------|--------------|--------------|--------------|
| Total number of hosts | 3            | 13           | 19           |
| serial value          | 25%          | 25%          | 25%          |
| Exact number of hosts | 0.75         | 3.25         | 4.75         |
| Rounded down          | 0            | 3            | 4            |
| Resulting batch size  | 1            | 3            | 4            |
| Number of batches     | 3            | 5            | 5            |
| Size of last batch    | 1            | 1            | 3            |

## Setting Batch Sizes to Change During the Play

You can change the batch size as the play runs. For example, you could test a play on a batch of one host; if that host fails, the entire play aborts. However, if the play is successful on one host, you could increase the batch size to 10 percent of your hosts, then 50 percent of the managed hosts, and then the remainder.

## Chapter 8 | Coordinating Rolling Updates

You can gradually change the batch size by setting the `serial` directive to a list of values. This list can contain any combination of integers and percentages and resets the size of each batch in sequence. If the value is a percentage, then Ansible computes the batch size based on the total size of the host group and not the size of the group of unprocessed hosts.

Consider the following example:

```

```

```
- name: Update Webservers
 hosts: web_servers
 serial:
 - 1
 - 10%
 - 100%
```

The first batch contains a single host.

The second batch contains 10% of the total hosts in the `web_servers` host group. Ansible computes the actual value according to the previously discussed rules.

The third batch contains all the remaining unprocessed hosts in the play. This setting enables Ansible to process all the remaining hosts efficiently.

If unprocessed hosts remain after the last batch corresponding to the previous `serial` directive entry, the last batch repeats until all hosts are processed. Consider the following play, which runs against a `web_servers` host group with 100 hosts:

```

```

```
- name: Update Webservers
 hosts: web_servers
 serial:
 - 1
 - 10%
 - 25%
```

The first batch contains one host and the second batch contains ten hosts (10% of 100). The third batch processes 25 hosts (25% of 100), leaving 64 unprocessed hosts ( $1 + 10 + 25$  processed). Ansible continues executing the play in batch sizes of 25 hosts (25% of 100) until fewer than 25 unprocessed hosts remain. In this example, the final batch processes the remaining 14 hosts ( $1 + 10 + 25 + 25 + 25 + 14 = 100$ ).

## Aborting the Play

By default, Ansible tries to get as many hosts to complete a play as possible. If a task fails for a host, the host is dropped from the play, but Ansible continues to run the remaining tasks for other hosts. The play only stops if all hosts fail.

However, if you use the `serial` directive to organize hosts into batches, then if all hosts in the current batch fail, Ansible stops the play for *all remaining hosts*, not just the remaining hosts in the current batch. If all hosts in a batch fail, the play aborts and the next batch does not start.

Ansible keeps a list of the active servers for each batch in the `ansible_play_batch` variable. Ansible removes a host that fails a task from the `ansible_play_batch` list, and updates this list after every task.

## Chapter 8 | Coordinating Rolling Updates

Consider the following hypothetical playbook, which runs against a `web_servers` host group with 100 hosts:

```

- name: Update Webservers
 hosts: web_servers
 tasks:
 - name: Step One
 ansible.builtin.shell: /usr/bin/some_command
 - name: Step Two
 ansible.builtin.shell: /usr/bin/some_other_command
```

If 99 web servers fail the first task, but one host succeeds, Ansible continues to the second task. When Ansible runs the second task, Ansible only runs that task for the one host that previously succeeded.

If you use the `serial` directive, playbook execution continues, provided that hosts remain in the current batch with no failures. Consider this modification to the hypothetical playbook:

```

- name: Update Webservers
 hosts: web_servers
 serial: 2
 tasks:
 - name: Step One
 ansible.builtin.shell: /usr/bin/some_command
 - name: Step Two
 ansible.builtin.shell: /usr/bin/some_other_command
```

If the first batch of two contains a host that succeeds and a host that fails, then the batch completes, and Ansible moves on to the second batch of two. If both hosts in the second batch fail on a task in the play, Ansible aborts the entire play and does not start any more batches.

In this scenario, running the playbook can produce the following states:

- One host completes the play.
- Three hosts are in an error state.
- The rest of the hosts remain unaltered.

## Specifying Failure Tolerance

By default, Ansible only halts play execution when all hosts in a batch fail. However, you might want a play to abort if more than a certain percentage of hosts in the inventory fail, even if no entire batch fails. It is also possible to "fail fast" and abort the play if any tasks fail.

You can add the `max_fail_percentage` directive to a play to alter the default Ansible failure behavior. When the number of failed hosts in a batch exceeds this percentage, Ansible halts playbook execution.

Consider the following hypothetical playbook that runs against the `web_servers` host group, which contains 100 hosts:

```

- name: Update Webservers
 hosts: web_servers
 max_fail_percentage: 30%
 serial:
 - 2
 - 10%
 - 100%
 tasks:
 - name: Step One
 ansible.builtin.shell: /usr/bin/some_command
 - name: Step Two
 ansible.builtin.shell: /usr/bin/some_other_command
```

The first batch contains two hosts. Because 30% of 2 is 0.6, a single host failure causes execution to stop.

If both hosts in the first batch succeed, then Ansible continues with the second batch of 10 hosts. Because 30% of 10 is 3, more than three host failures must occur for Ansible to stop playbook execution. If three or fewer hosts experience errors in the second batch, Ansible continues with the third batch.

To implement a "fail fast" strategy, set the `max_fail_percentage` to zero.



### Important

To summarize the Ansible failure behavior:

- If the `serial` directive and the `max_fail_percentage` values are not defined, all hosts are run through the play in one batch. If all hosts fail, then the play fails.
- If the `serial` directive is defined, then hosts are run through the play in multiple batches, and the play fails if all hosts in any one batch fail.
- If the `max_fail_percentage` directive is defined, the play fails if more than that percentage of hosts in a batch fail.

If a play fails, Ansible aborts all remaining plays in the playbook.

## Running a Task Once

In certain scenarios, you might only need to run a task once for an entire batch of hosts rather than once for each host in the batch. To accommodate this scenario, add the `run_once` directive to the task with a Boolean `true` (or `yes`) value.

Consider the following hypothetical task:

```
- name: Reactivate Hosts
 ansible.builtin.shell: /sbin/activate.sh {{ active_hosts_string }}
 run_once: true
 delegate_to: monitor.example.com
 vars:
 active_hosts_string: "{{ ansible_play_batch | join(' ')}}"
```

This task runs once and runs on the `monitor.example.com` host. The task uses the `active_hosts_string` variable to pass a list of active hosts as command-line arguments to an activation script. The variable contains only those hosts in the current batch that have succeeded for all previous tasks.



### Important

Setting the `run_once: true` directive causes a task to run once for each batch.

If you only need to run a task once for all hosts in a play, and the play has multiple batches, then you can add the following conditional statement to the task:

```
when: inventory_hostname == ansible_play_hosts[0]
```

This conditional statement runs the task only for the first host in the play.



### References

**Controlling playbook execution: strategies and more – Ansible Documentation**

[https://docs.ansible.com/ansible/6/user\\_guide/playbooks\\_strategies.html](https://docs.ansible.com/ansible/6/user_guide/playbooks_strategies.html)

## ► Guided Exercise

# Managing Rolling Updates

Run a playbook that uses unequal batch sizes with the `serial` directive, aborts if it fails for too many hosts, and runs a specific task once per batch.

## Outcomes

- Control the update process of an existing HAProxy cluster by controlling the update with the `serial` directive, which determines the size of the batch to use.

## Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab.

```
[student@workstation ~]$ lab start update-management
```

## Instructions

- 1. Clone the `https://git.lab.example.com/student/update-management.git` Git repository into the `/home/student/git-repos` directory and then create a new branch for this exercise.
- 1.1. Create the `/home/student/git-repos` directory from a terminal if it does not already exist, and then change to it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/update-management.git` repository and then change to the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/update-management.git
Cloning into 'update-management'...
...output omitted...
[student@workstation git-repos]$ cd update-management
```

- 1.3. Create the `exercise` branch.

```
[student@workstation update-management]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 2. Install the content collections specified in the `collections/requirements.yml` file into the `collections/` directory.

**Chapter 8 |** Coordinating Rolling Updates

- 2.1. Log in to the private automation hub at `https://hub.lab.example.com` with the student username and the `redhat123` password.
- 2.2. Navigate to **Collections > API token management**, and then click **Load token**. Copy the API token.
- 2.3. Update both token lines in the `ansible.cfg` file by using the copied token. Your token might be different from the one displayed in this example.

```
...output omitted...

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified
token=f41f07130d6eb6ef2ded63a574c161b509c647dd

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=f41f07130d6eb6ef2ded63a574c161b509c647dd
```

- 2.4. Use the `ansible-galaxy` command to install the `community.general` and `redhat.rhel_system_roles` content collections into the `collections/` directory. Later in the exercise, the `update_webapp.yml` playbook uses the `community.general.haproxy` module to interact with the HAProxy server, and the `apache` role requires the `redhat.rhel_system_roles.selinux` module.

```
[student@workstation update-management]$ ansible-galaxy collection install \
> -r collections/requirements.yml -p collections/
Starting galaxy collection install process
...output omitted...
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
rh-certified/collections/artifacts/redhat-rhel_system_roles-1.20.0.tar.gz
to /home/student/.ansible/tmp/ansible-local-22938l4moi2mi/tmprv5lya26/redhat-
rhel_system_roles-1.20.0-smd0f1c_
Installing 'redhat.rhel_system_roles:1.20.0' to '/home/student/git-repos/update-
management/collections/ansible_collections/redhat/rhel_system_roles'
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
community/collections/artifacts/community-general-6.1.0.tar.gz to /home/
student/.ansible/tmp/ansible-local-22938l4moi2mi/tmprv5lya26/community-
general-6.1.0_ty0hzx6
redhat.rhel_system_roles:1.20.0 was installed successfully
Installing 'community.general:6.1.0' to '/home/student/git-repos/update-
management/collections/ansible_collections/community/general'
community.general:6.1.0 was installed successfully
```

- 3. Using the `ee-supported-rhel8:latest` automation execution environment, run the `site.yml` playbook. The playbook deploys a front-end load balancer and a set of back-end web servers.

```
[student@workstation update-management]$ ansible-navigator run site.yml
```

| Play name                    | Ok | Changed | ... | Failed | ... | Task count | Progress |
|------------------------------|----|---------|-----|--------|-----|------------|----------|
| 0 Gather web_server facts    | 5  | 0       | ... | 0      | ... | 5          | Complete |
| 1 Ensure HAProxy is deployed | 6  | 5       | ... | 0      | ... | 6          | Complete |

**Chapter 8 |** Coordinating Rolling Updates

```

2|Set Load Balancer facts 1 0 ... 0 ... 1 Complete
3|Ensure Apache is deployed 35 25 ... 0 ... 35 Complete
4|Ensure Web App is deployed 15 5 ... 0 ... 15 Complete

^f/PgUp page up ^b/PgDn page down ↑ scroll esc back ... Successful

```

Press Esc to exit from the `ansible-navigator` command.

- 4. Use the `curl` command to send five requests to the load balancer.

```
[student@workstation update-management]$ for x in {1..5}; do curl servera; done
serverb: /var/www/html/index.html
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
```

The load balancer distributes requests to all five back-end web servers that serve content from the `/var/www/html/` directory.

**Note**

The server's order might be different from the order displayed throughout this example.

- 5. The `update_webapp.yml` playbook performs a rolling update of the web application hosted on the back-end web servers. Review the `update_webapp.yml` playbook and the use of the `serial` keyword.

- 5.1. Review the top section of the playbook:

```

- name: Update web servers to use a new document root
 hosts: web_servers
 become: true
 force_handlers: true
 vars:
 webapp_content_dir: /srv/web/app/html

 serial:
 - 1
 - 25%
 - 100%
...output omitted...

```

The `update_webapp.yml` playbook acts on all hosts in the `web_servers` host group. The `webapp_content_dir` variable specifies the root directory for Apache web documents. If not specified, the `webapp_content_dir` variable defaults to the `/var/www/html/` directory.

The `serial` keyword specifies that tasks in the play are processed in three batches.

- The first batch contains one host. After this batch finishes, four hosts still require processing.

**Chapter 8 |** Coordinating Rolling Updates

- The second batch contains one host because 25% of the host group size is 1.25 and that truncates to one.
- The third batch contains three hosts because 100% of the hosts group size is five but only three hosts remain unprocessed.

5.2. Review the `pre_tasks` section of the play:

```
pre_tasks:
 - name: Remove web server from service during the update
 community.general.haproxy:
 state: disabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ item }}"
 with_items: "{{ groups['lb_servers'] }}"
```

Before the playbook updates the web application on each server, the `haproxy` module disables the server in all load balancers. This task protects external clients from errors discovered after application deployment.

5.3. Review the `roles` section of the play:

```
roles:
 - role: apache
 - role: webapp
```

The `apache` role modifies the `/etc/httpd/conf/httpd.conf` configuration file to use the directory specified by the `webapp_content_dir` variable. The `webapp` role deploys content to the directory that is specified by the `webapp_content_dir` variable.

5.4. Review the `post_tasks` section of the play:

```
post_tasks:
 # Firewall rules dictate that requests to backend web
 # servers must originate from a load balancer.
 - name: Smoke Test - Ensure HTTP 200 OK
 ansible.builtin.uri:
 url: "http://{{ inventory_hostname }}:{{ apache_port }}"
 status_code: 200
 delegate_to: "{{ groups['lb_servers'][0] }}"
 become: false

 # If the test fails, servers are not re-enabled
 # in the load balancers, and the update process halts.
 - name: Enable healthy server in load balancers
 community.general.haproxy:
 state: enabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ item }}"
 with_items: "{{ groups['lb_servers'] }}"
```

**Chapter 8 |** Coordinating Rolling Updates

After the playbook deploys the web application, a smoke test ensures that each back-end web server responds with a 200 HTTP status code. The firewall rules on each web server only enable web requests from a load balancer, so all smoke tests are delegated to a load balancer.

If the smoke test fails for a server, then further processing of that server halts, and the web server is not re-enabled in the load balancer.

The second task enables the server in the load balancer when the smoke test passes.

- ▶ 6. Use the `update_webapp.yml` playbook to deploy the web application to the `/srv/web/app/html/` directory.

Use the `curl` command to send another five requests to the load balancer. Notice how the failed back-end web server is not included in the load-balancing pool.

- 6.1. Using the `ee-supported-rhel8:latest` automation execution environment, run the `update_webapp.yml` playbook. The playbook fails. Leave this terminal running in interactive mode so that you can troubleshoot the problem.

```
[student@workstation update-management]$ ansible-navigator run update_webapp.yml

Play name Ok Changed ... Failed ... Task count Progress
0|Update web servers ... 10 5 ... 1 ... 11 Complete

^f/PgUp page up ^b/PgDn page down ↑ scroll esc back ... Failed
```

- 6.2. In a separate terminal tab or window, send another five requests to the load balancer. Notice how the load balancer only redirects requests to four of the original five web servers.

```
[student@workstation update-management]$ for x in {1..5}; do curl servera; done
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
serverc: /var/www/html/index.html
```

The `update_webapp.yml` playbook first removed one of the servers from the load-balancing pool (`serverb` in this example). After applying changes to the web server, the playbook ran a task to validate that the web server could still respond successfully to web requests. Because the web server did not respond successfully, the playbook did not add the server back to the load-balancing pool.

In this example, each web server displays a custom message. A real-world example would deploy the same content to each web server. Users accessing the load balancer would be unaware that one or more web servers were removed from the load-balancing pool.

- 6.3. Return to the terminal tab or window that contains the interactive session for the `ansible-navigator` command. Press 0 to view details about the `Update web servers to use a new document root` play.

```
...output omitted...
0|Update web servers ... 10 5 ... 1 ... 11 Complete
...output omitted...
```

**Chapter 8 |** Coordinating Rolling Updates

The output shows that task 10 (Smoke Test - Ensure HTTP 200 OK) failed for `serverb.lab.example.com`.

```
Result Host ... Task ...
0|Ok serverb.lab.example.com ... Gathering Facts ...
1|Ok serverb.lab.example.com ... Remove web server from service during ...
2|Ok serverb.lab.example.com ... Apache Port Check ...
...output omitted...
10|Failed serverb.lab.example.com ... Smoke Test - Ensure HTTP 200 OK ...
^f/PgUp page up ^b/PgDn page down ↑ scroll esc back ... Failed
```

- 6.4. Continue to use interactive mode to discover more details about the problem. Because the item number exceeds nine, you must enter a colon before the item number. Press :10 to view details about the `Smoke Test - Ensure HTTP 200 OK` task.

```
...output omitted...
10|Failed serverb.lab.example.com ... Smoke Test - Ensure HTTP 200 OK ...
...output omitted...
```

The details page indicates that `serverb.lab.example.com` returned a status code of 403 instead of the expected status code of 200. This same information is also displayed on the `msg` line in the output.

```
Play name: Update web servers to use a new document root:10
Task name: Smoke Test - Ensure HTTP 200 OK
Failed: serverb.lab.example.com Status code was 403 and not [200]: HTTP Error 403:
Forbidden
...output omitted...
```

Press :q to exit from the `ansible-navigator` command.

- 7. Web servers often return a status code of 403 because of either a regular file permission problem or an SELinux access problem. Identify the problem and a potential solution.
- 7.1. Use SSH to connect to `serverb.lab.example.com` as the `student` user.

```
[student@workstation update-management]$ ssh student@serverb.lab.example.com
...output omitted...
[student@serverb ~]$
```

- 7.2. Use the `ls` command to display the permissions and ownership for the `/srv/web/app/html/index.html` file. Regular file permissions indicate that the `index.html` file is readable by everyone, so the file permissions are not the problem.

```
[student@serverb ~]$ ls -l /srv/web/app/html/index.html
-rw-r--r--. 1 root root ... /srv/web/app/html/index.html
```

- 7.3. Use the `ls` command to display SELinux context information for the `/srv/web/app/html/index.html` file. The `var_t` SELinux context type is not appropriate for web content. SELinux prevents the `httpd` process from accessing files with an incorrect context type.

## Chapter 8 | Coordinating Rolling Updates

```
[student@serverb ~]$ ls -Z /srv/web/app/html/index.html
system_u:object_r:var_t:s0 /srv/web/app/html/index.html
```

- 7.4. By default, the `/var/www/html/` directory uses a context type allowed by SELinux. Identify the context type for the `/var/www/html/` directory.

```
[student@serverb ~]$ ls -Zd /var/www/html
system_u:object_r:httpd_sys_content_t:s0 /var/www/html
```

Although SELinux provides additional context types for web content, this exercise uses the `httpd_sys_content_t` context type.



### Important

The `ansible.builtin.file` module can use the `setype` option to set the SELinux context type for a directory. However, files and directories created within the directory do not inherit the specified context type.

- 7.5. Exit the SSH session on `serverb.lab.example.com` and return to `workstation.lab.example.com`.

```
[student@serverb ~]$ logout
Connection to serverb.lab.example.com closed.
[student@workstation update-management]$
```

- ▶ 8. Update the `apache` role to resolve the issue. Modify the `roles/apache/tasks/main.yml` task file to add a code block after the `Start` and `enable httpd` task. Move the existing `Customize Apache HTTPD Configuration` task and the `Ensure that {{ webapp_content_dir }} exists` task into the block. The resulting file should consist of the following content:

```

tasks file for apache

- name: Apache Port Check
 ansible.builtin.assert:
 that:
 - apache_port in apache_standard_ports_list
 fail_msg: "{{ tmp_msg}}: {{ apache_standard_ports_list }}"
 success_msg: The specified apache port ({{ apache_port }}) is allowed.
 vars:
 tmp_msg: "'apache_port' value ({{ apache_port }}) is not in the list"

- name: Install httpd
 ansible.builtin.yum:
 name:
 - httpd
 state: present

- name: Start and enable httpd
 ansible.builtin.service:
 name: httpd
```

```

state: started
enabled: true

- name: Customize SELinux for web_content_dir
 block:
 - name: Set webapp_base fact
 ansible.builtin.set_fact:
 webapp_base: "{{ webapp_content_dir | split('/') }}"
 ①

 - name: Web directory is a subdirectory of /srv ②
 ansible.builtin.assert:
 that:
 - webapp_base[0] == ''
 - webapp_base[1] == 'srv'
 - webapp_base[2] is defined
 fail_msg: "'{{ webapp_content_dir }}' is not a subdirectory of /srv."
 success_msg: "'{{ webapp_content_dir }}' is a subdirectory of /srv.

 - name: Customize Apache HTTPD Configuration ③
 ansible.builtin.template:
 src: templates/httpd.conf.j2
 dest: /etc/httpd/conf/httpd.conf
 notify: restart httpd

 - name: Ensure that {{ webapp_content_dir }} exists
 ansible.builtin.file:
 path: "{{ webapp_content_dir }}"
 state: directory
 owner: root
 group: root
 mode: '0755'

 - name: Create SELinux file context for the directory ④
 ansible.builtin.include_role:
 name: redhat.rhel_system_roles.selinux
 vars:
 selinux_fcontexts:
 - target: "/{{ webapp_base[1] }}/{{ webapp_base[2] }}(/.*)?"
 setype: "httpd_sys_content_t"
 state: present
 selinux_restore_dirs:
 - /{{ webapp_base[1] }}/{{ webapp_base[2] }}
when: "webapp_content_dir != '/var/www/html'" ⑤

```

- ① The first task splits the `webapp_content_dir` variable into an array using a forward slash as a delimiter.
- ② The second task in the block checks that the directory specified by the `webapp_content_dir` variable is a subdirectory of the `/srv/` directory. This task checks that the first item of the (`webapp_base[0]`) array is empty because this is the value to the left of the first forward slash. The second item of the (`webapp_base[1]`)

**Chapter 8 |** Coordinating Rolling Updates

array must match the value `srv`. The third item of the (`webapp_base[2]`) array must be defined. The playbook exits with an error message if the task fails.

- ③ The third and fourth tasks are from the existing task file and have been moved into the block. The directory specified by the `webapp_content_dir` variable must exist before running the `restorecon` command.
- ④ The fifth task in the block creates a new file context rule that uses the first two directories specified by the `webapp_content_dir` variable and then applies the new rule.
- ⑤ The block of code only applies if the `webapp_content_dir` variable does not match the default value of `/var/www/html`.

**Note**

The `~/git-repos/update-management/solutions/main.yml` file contains the correct configuration and can be used for comparison.

- ▶ 9. Run the `update_webapp.yml` playbook again and notice that the playbook succeeds.

As the playbook runs, monitor the output of the `issue_requests.sh` script in another terminal tab or window. The output shows the back-end servers gradually switching over to using the new web document root directory.

- 9.1. Run the `issue_requests.sh` script in a separate terminal tab or window and monitor the output during the execution of the `update_webapp.yml` playbook. The script sends a web request to the `servera.lab.example.com` load balancer every 2 seconds. Output is displayed in the terminal and logged to the `curl_output.log` file.

```
[student@workstation update-management]$./issue_requests.sh
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
```

Initially, responses cycle through the four hosts remaining in the load-balancing pool. Leave the `issue_requests.sh` script running and return to the previous terminal tab or window.

- 9.2. Using the `ee-supported-rhel8:latest` automation execution environment, run the `update_webapp.yml` playbook. While the playbook runs, switch back to the terminal tab or window that is running the `issue_requests.sh` script.

```
[student@workstation update-management]$ ansible-navigator run update_webapp.yml
...output omitted...
```

- 9.3. Monitor the output of the `issue_requests.sh` script while running the `update_webapp.yml` playbook.

Eventually, the smoke test passes for the new application, and the `serverb` server is returned to service with an updated web document root directory:

The playbook removes the `serverc` server from service:

```
...output omitted...
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
```

The playbook processes the next batch, which only contains the **serverc** server.

The playbook removes **serverc** from service:

```
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /var/www/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
serverb: /srv/web/app/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
```

Eventually, the smoke test passes for the **serverc** server and the server is put back into service:

```
...output omitted...
serverb: /srv/web/app/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
serverd: /var/www/html/index.html
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
```

The last batch processes all remaining web servers. The playbook first disables all three of these servers, leaving only the **serverb** and **serverc** servers to handle requests:

```
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
serverd: /var/www/html/index.html
```

**Chapter 8 |** Coordinating Rolling Updates

```
servere: /var/www/html/index.html
serverf: /var/www/html/index.html
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
...output omitted...
```

Eventually, each server passes the smoke test and is put back into service:

```
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
serverd: /srv/web/app/html/index.html
servere: /srv/web/app/html/index.html
serverf: /srv/web/app/html/index.html
...output omitted...
```

9.4. Press **Ctrl+C** to stop the `issue_requests.sh` script.

```
...output omitted...
serverb: /srv/web/app/html/index.html
serverc: /srv/web/app/html/index.html
serverd: /srv/web/app/html/index.html
servere: /srv/web/app/html/index.html
serverf: /srv/web/app/html/index.html
^C
[student@workstation update-management]$
```

- 10. Return to the terminal tab or window that is running the `ansible-navigator` command. The `update_webapp.yml` playbook is completed successfully.

| Play name            | Ok  | Changed | ... | Failed | Skipped | ... | Task count | Progress     |
|----------------------|-----|---------|-----|--------|---------|-----|------------|--------------|
| 0 Update web servers | ... | 22      | 4   | ...    | 0       | 16  | ...        | 38 Complete  |
| 1 Update web servers | ... | 23      | 9   | ...    | 0       | 16  | ...        | 39 Complete  |
| 2 Update web servers | ... | 71      | 27  | ...    | 0       | 46  | ...        | 117 Complete |

**^f/PgUp** page up    **^b/PgDn** page down    **↑↓ scroll** scroll    **esc back** back    ... Successful

Your numbers might be slightly different than the numbers displayed in this output. Press **ESC** to exit from the `ansible-navigator` command.

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish update-management
```

## ▶ Lab

# Coordinating Rolling Updates

Modify a playbook to implement rolling updates.

## Outcomes

- Delegate tasks to other hosts.
- Implement deployment batches with the `serial` keyword.
- Limit failures with the `max_fail_percentage` keyword.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository at `https://git.lab.example.com/student/update-review.git`. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository by using `Student@123` as the Git password.

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

## Instructions

1. Clone the `https://git.lab.example.com/student/update-review.git` repository into the `/home/student/git-repos/update-review` directory, and then create the `exercise` branch for this exercise. Run the `site.yml` playbook to deploy the web application infrastructure.
2. The `update_webapp.yml` playbook requires collections that are defined in the `collections/requirements.yml` file.

Obtain a token from `https://hub.lab.example.com` by logging in as the `student` user with the `redhat123` password. Use this token to update the `token` option in the `ansible.cfg` file. Use the `ansible-galaxy` command to install the collections. The collections must be available to the execution environment, so they must be installed in the `/home/student/git-repos/update-review/collections/` directory.

3. Add a `pre_tasks` section to the play in the `update_webapp.yml` playbook. Add a task to the section that disables the web servers on the HAProxy load balancer. Without this task, external clients might experience problems with the web application due to unforeseen deployment issues.

Configure the new task as follows:

- Use the `community.general.haproxy` module.
- Disable the host in the app back end by using the `inventory_hostname` variable.
- Delegate each action to the load balancer. The `{{ groups['lb_servers'][0] }}` Jinja2 expression provides the name of this load balancer.

## Chapter 8 | Coordinating Rolling Updates

4. In the `update_webapp.yml` playbook, the first task in the `post_tasks` section is a "smoke test". That task verifies that the web application on each web server responds to requests that originate from that same server. This test is not realistic because requests to the web server normally originate from the load balancer, but it verifies that the web servers are operating.  
However, if you only test a web server's response from the web server itself, you do not test any network-related functions.  
Using delegation, modify the smoke test task so that each request to a web server originates from the load balancer. Use the `inventory_hostname` variable to connect to each web server.
5. Add a task to the `post_tasks` section after the smoke test to re-enable each web server on the HAProxy load balancer. This task is similar in structure to the `community.general.haproxy` task in the `pre_tasks` section, except that it sets the state to `enabled` instead of `disabled`.
6. Configure the play in the `update_webapp.yml` playbook to run through its tasks in batches to mitigate the effects of unforeseen deployment errors. Ensure that the playbook uses no more than three batches to complete the upgrade of all web server hosts. Set the first batch to consist of 5% of the hosts, the second batch 35% of the hosts, and the last batch to consist of all remaining hosts.  
Add an appropriate keyword to the play to ensure that playbook execution stops if any host fails a task during the upgrade.
7. Run the `update_webapp.yml` playbook to perform the rolling update.
8. Commit and push your changes to the remote Git repository.

## Evaluation

As the student user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.



### Note

Make sure to commit and push your changes to the Git repository before rerunning the script.

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

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

## ► Solution

# Coordinating Rolling Updates

Modify a playbook to implement rolling updates.

## Outcomes

- Delegate tasks to other hosts.
- Implement deployment batches with the `serial` keyword.
- Limit failures with the `max_fail_percentage` keyword.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository at `https://git.lab.example.com/student/update-review.git`. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository by using `Student@123` as the Git password.

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

## Instructions

1. Clone the `https://git.lab.example.com/student/update-review.git` repository into the `/home/student/git-repos/update-review` directory, and then create the `exercise` branch for this exercise. Run the `site.yml` playbook to deploy the web application infrastructure.
  - 1.1. From a terminal, create the directory `/home/student/git-repos` if it does not exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the repository and then change into the cloned repository.

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/update-review.git
Cloning into 'update-review'...
...output omitted...
[student@workstation git-repos]$ cd update-review
```

- 1.3. Create the `exercise` branch.

**Chapter 8 |** Coordinating Rolling Updates

```
[student@workstation update-review]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

- 1.4. Use the `ansible-navigator run` command to run the `site.yml` playbook.

```
[student@workstation update-review]$ ansible-navigator run \
> -m stdout site.yml

PLAY [Gather web_server facts] ****

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

PLAY [Ensure HAProxy is deployed] ****

...output omitted...

PLAY RECAP ****
servera.lab.example.com : ok=7 changed=6 unreachable=0 failed=0 ...
serverb.lab.example.com : ok=12 changed=8 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=12 changed=8 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=12 changed=8 unreachable=0 failed=0 ...
servere.lab.example.com : ok=12 changed=8 unreachable=0 failed=0 ...
serverf.lab.example.com : ok=12 changed=8 unreachable=0 failed=0 ...
```

2. The `update_webapp.yml` playbook requires collections that are defined in the `collections/requirements.yml` file.

Obtain a token from <https://hub.lab.example.com> by logging in as the student user with the `redhat123` password. Use this token to update the `token` option in the `ansible.cfg` file. Use the `ansible-galaxy` command to install the collections. The collections must be available to the execution environment, so they must be installed in the `/home/student/git-repos/update-review/collections/` directory.

- 2.1. Log in to the private automation hub at <https://hub.lab.example.com> as the student user with `redhat123` as the password.
- 2.2. Navigate to **Collections > API token management**, and then click **Load token**. Copy the API token.
- 2.3. Update both `token` lines in the `ansible.cfg` file by using the copied token. Your token is probably different from the one that is displayed in this example.

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

```
[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified
token=f41f07130d6eb6ef2ded63a574c161b509c647dd
```

```
[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=f41f07130d6eb6ef2ded63a574c161b509c647dd
```

- 2.4. Use the `ansible-galaxy` command to install the `community.general` content collection into the `collections/` directory.

```
[student@workstation update-review]$ ansible-galaxy collection install \
> -r collections/requirements.yml -p collections/
Starting galaxy collection install process
...output omitted...
Installing 'community.general:6.1.0' to '/home/student/git-repos/update-review/
collections/ansible_collections/community/general'
community.general:6.1.0 was installed successfully
```

3. Add a `pre_tasks` section to the play in the `update_webapp.yml` playbook. Add a task to the section that disables the web servers on the HAProxy load balancer. Without this task, external clients might experience problems with the web application due to unforeseen deployment issues.

Configure the new task as follows:

- Use the `community.general.haproxy` module.
- Disable the host in the app back end by using the `inventory_hostname` variable.
- Delegate each action to the load balancer. The `{{ groups['lb_servers'][0] }}` Jinja2 expression provides the name of this load balancer.

- 3.1. Edit the play in the `update_webapp.yml` playbook by adding a `pre_tasks` section containing a task that uses the `community.general.haproxy` module to disable the web server on the load balancer.

```
pre_tasks:
 - name: Remove web server from service during the update
 community.general.haproxy:
 state: disabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ groups['lb_servers'][0] }}"
```

4. In the `update_webapp.yml` playbook, the first task in the `post_tasks` section is a "smoke test". That task verifies that the web application on each web server responds to requests that originate from that same server. This test is not realistic because requests to the web server normally originate from the load balancer, but it verifies that the web servers are operating.

However, if you only test a web server's response from the web server itself, you do not test any network-related functions.

Using delegation, modify the smoke test task so that each request to a web server originates from the load balancer. Use the `inventory_hostname` variable to connect to each web server.

- 4.1. Add a `delegate_to` directive to the smoke test task that runs on the load balancer. Change the testing URL to point to the current value of the `inventory_hostname` variable:

## Chapter 8 | Coordinating Rolling Updates

```
- name: Smoke Test - Ensure HTTP 200 OK
 ansible.builtin.uri:
 url: "http://{{ inventory_hostname }}:{{ apache_port }}"
 status_code: 200
 become: false
 delegate_to: "{{ groups['lb_servers'][0] }}"
```

5. Add a task to the `post_tasks` section after the smoke test to re-enable each web server on the HAProxy load balancer. This task is similar in structure to the `community.general.haproxy` task in the `pre_tasks` section, except that it sets the state to `enabled` instead of `disabled`.

```
- name: Enable healthy server in load balancers
 community.general.haproxy:
 state: enabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ groups['lb_servers'][0] }}"
```

6. Configure the play in the `update_webapp.yml` playbook to run through its tasks in batches to mitigate the effects of unforeseen deployment errors. Ensure that the playbook uses no more than three batches to complete the upgrade of all web server hosts. Set the first batch to consist of 5% of the hosts, the second batch 35% of the hosts, and the last batch to consist of all remaining hosts.

Add an appropriate keyword to the play to ensure that playbook execution stops if any host fails a task during the upgrade.

- 6.1. Add the `max_fail_percentage` directive and set the value to 0 to stop execution of the play at any failure. Add the `serial` directive and set the value to a list of three elements: 5%, 35%, and 100% to ensure that all servers are updated in the last batch. The beginning of the play should appear as shown in the following example:

```
- name: Upgrade Web Application
 hosts: web_servers
 become: true
 vars:
 webapp_version: v1.1
 max_fail_percentage: 0
 serial:
 - "5%"
 - "35%"
 - "100%"
```

- 6.2. The completed `update_webapp.yml` playbook should appear as shown in the following example:

```

- name: Upgrade Web Application
 hosts: web_servers
 become: true
 vars:
 webapp_version: v1.1
```

**Chapter 8 |** Coordinating Rolling Updates

```

max_fail_percentage: 0
serial:
 - "5%"
 - "35%"
 - "100%"

pre_tasks:
 - name: Remove web server from service during the update
 community.general.haproxy:
 state: disabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ groups['lb_servers'][0] }}"

roles:
 - role: webapp

post_tasks:
 - name: Smoke Test - Ensure HTTP 200 OK
 ansible.builtin.uri:
 url: "http://{{ inventory_hostname }}:{{ apache_port }}"
 status_code: 200
 become: false
 delegate_to: "{{ groups['lb_servers'][0] }}"

 - name: Enable healthy server in load balancers
 community.general.haproxy:
 state: enabled
 backend: app
 host: "{{ inventory_hostname }}"
 delegate_to: "{{ groups['lb_servers'][0] }}"

```

7. Run the `update_webapp.yml` playbook to perform the rolling update.

- 7.1. Use the `ansible-navigator` command to run the `update_webapp.yml` playbook.

```

[student@workstation update-review]$ ansible-navigator run \
> -m stdout update_webapp.yml
...output omitted...

PLAY RECAP ****
serverb.lab.example.com : ok=6 changed=3 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=6 changed=3 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=6 changed=3 unreachable=0 failed=0 ...
servere.lab.example.com : ok=6 changed=3 unreachable=0 failed=0 ...
serverf.lab.example.com : ok=6 changed=3 unreachable=0 failed=0 ...

```

8. Commit and push your changes to the remote Git repository.

- 8.1. Add the changed files, commit the changes, and push them to the Git repository. If prompted, use `Student@123` as the password.

```
[student@workstation update-review]$ git add .
[student@workstation update-review]$ git commit -m "Rolling updates"
[exercise 8d38803] Rolling updates
 1 files changed, 26 insertions(+), 12 deletions(-)
[student@workstation update-review]$ git push -u origin exercise
Password for 'https://student@git.lab.example.com': Student@123
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 931 bytes | 931.00 KiB/s, done.
Total 7 (delta 3), reused 0 (delta 0)
To http://git.lab.example.com:8081/git/update-review.git
 a36da15..5feb08e exercise -> exercise
```

## Evaluation

As the student user on the workstation machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.



### Note

Make sure to commit and push your changes to the Git repository before rerunning the script.

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

## Finish

On the workstation machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

# Summary

---

- The `delegate_to` keyword can delegate a task to run on a different host.
- The `forks` parameter in the Ansible configuration file specifies the maximum number of parallel connections to managed hosts.
- The `serial` keyword configures Ansible to run hosts through a play in multiple batches.
- The `max_fail_percentage` keyword configures Ansible to abort the play if more than a certain percentage of hosts in the current batch fail.
- The `run_once` keyword specifies that a task runs once for a batch rather than once for each host in the batch.



## Chapter 9

# Creating Content Collections and Execution Environments

### Goal

Write your own Ansible Content Collections, publish them, embed them in a custom automation execution environment, and run them in playbooks by using automation controller.

### Objectives

- Create content collections and distribute them for reuse.
- Build a custom automation execution environment image by using the `ansible-builder` command.
- Validate that a custom automation execution environment works as expected by testing it with the `ansible-navigator` command, and then distribute the automation execution environment for reuse.
- Run a playbook in automation controller that uses a content collection in the project or content provided by a specific automation execution environment.

### Sections

- Writing Ansible Content Collections (and Guided Exercise)
- Building a Custom Automation Execution Environment (and Guided Exercise)
- Validating a Custom Execution Environment (and Guided Exercise)
- Using Custom Content Collections and Execution Environments in Automation Controller (and Guided Exercise)

### Lab

- Creating Content Collections and Execution Environments

# Writing Ansible Content Collections

## Objectives

- Create content collections and distribute them for reuse.

## Developing Ansible Content Collections

Red Hat, its partners, and the Ansible community provide Ansible Content Collections for many software products and technologies. However, customers often have automation needs that no existing collection covers. For example, you might have to pilot an application for which there is no available collection, or your company might develop internal tools that you need to automate. In all those situations, you should consider developing collections.



### Note

Ansible 2.8 and earlier do not support collections. With those versions, you develop and distribute individual roles and provide modules and plug-ins inside those roles or alongside your playbooks.

Red Hat still supports that architecture but recommends moving your roles, modules, plug-ins, and filters inside collections. Automation hub and private automation hub do not support individual roles.

## Selecting a Namespace for Collections

Remember that Ansible organizes collections in namespaces. The namespace is the first part of a collection name. For example, the namespace of the `amazon.aws` collection is `amazon`.

Carefully choose a namespace for grouping your collections:

- If you plan to publish your collections to Ansible Galaxy, then use your Ansible Galaxy username as the namespace.
- If you publish your collections to private automation hub, then ask the platform administrators to create a namespace for you. Note that you might be able to create a namespace yourself if your user account has the required permissions.
- If you are a Red Hat partner and publish your collections to automation hub, then use the namespace that Red Hat has assigned for your company.

## Creating the Collection Directory Structure

Use the `ansible-galaxy collection init` command to create the directory structure for your new collection. Specify the name of the collection, including the namespace, as the argument to the command:

```
[user@host ~]$ ansible-galaxy collection init mynamespace.mycollection
- Collection mynamespace.mycollection was created successfully
```

This command creates the collection and organizes content in the following directory and file structure:

```
[user@host ~]$ tree mynamespace/
mynamespace/
└── mycollection
 ├── docs ①
 ├── galaxy.yml ②
 ├── plugins ③
 │ └── README.md
 ├── README.md ④
 └── roles ⑤

4 directories, 3 files
```

- ① The `docs/` directory stores additional documentation. You can use it to provide detailed documentation about your modules and plug-ins.
- ② The `galaxy.yml` file contains the necessary information to build and publish your collection.
- ③ The `plugins/` directory stores the modules, plug-ins, and filters that the collection provides. You develop those artifacts in Python.
- ④ The `README.md` file describes your collection. It usually provides some installation instructions and lists the required dependencies. Distribution platforms such as Ansible Galaxy or private automation hub use that file as the front page for your collection.
- ⑤ The `roles/` directory stores your roles.

**Note**

You can remove the files and directories that you are not using. For example, if there are no roles in your collection, then you can remove the `roles/` directory.

## Adding Content to a Collection

Organize the modules, plug-ins, and filters that you develop in subdirectories under the `plugins/` directory. For modules, create the `plugins/modules/` subdirectory and then copy the Python scripts for your modules into that subdirectory. For inventory plug-ins, create and then use the `plugins/inventory/` subdirectory. Similarly, the other plug-in types have dedicated subdirectories. Developing modules, plug-ins, and filters is beyond the scope of this course.

If you have existing roles, then you can move their directory structure under the `roles/` directory. To create a role for the collection, use the `ansible-galaxy role init` command from inside the `roles/` directory:

```
[user@host ~]$ cd mynamespace/mycollection/roles/
[user@host roles]$ ansible-galaxy role init myrole
- Role myrole was created successfully
[user@host roles]$ ls myrole/
defaults files handlers meta README.md tasks templates tests vars
```

## Updating Collection Metadata

The `galaxy.yml` YAML file at the root of the collection directory provides information for Ansible to build and publish the collection. The file includes comments that describe each parameter. The following `galaxy.yml` file is a complete example:

```

namespace: mynamespace
name: mycollection
version: 1.0.0
readme: README.md
authors:
 - your name <example@domain.com>
description: Ansible modules to manage my company's custom software
license:
 - GPL-3.0-or-later

repository: https://git.example.com/training/my-collection

The URL to any online docs
documentation: https://git.example.com/training/my-collection/tree/main/docs

The URL to the homepage of the collection/project
homepage: https://git.example.com/training/my-collection

The URL to the collection issue tracker
issues: https://git.example.com/training/my-collection/issues

dependencies:
 community.general: '>=1.0.0'
 ansible.posix: '>=1.0.0'
```

## Declaring Collection Dependencies

Some collections depend on other collections to work correctly. For example, a role in your new collection might call modules from other collections. Declare all those required collections by using the `dependencies` parameter in your `galaxy.yml` file.

That parameter is a dictionary of collections. For each entry, the key is the collection FQCN, and the value is the collection version (use `>=1.0.0` when you do not need a specific version). The following example lists the `community.general` and `ansible.posix` collections as requirements:

```
...output omitted...
dependencies:
 community.general: '>=1.0.0'
 ansible.posix: '>=1.0.0'
...output omitted...
```

You do not need to declare the `ansible.builtin` collection.

More complex collections might provide modules, plug-ins, and filters that depend on additional Python libraries. Users must install the corresponding Python packages in the execution

environment before they can use the collection. For example, the `amazon.aws` collection uses the Amazon Web Services Software Development Kit (SDK) that the `boto` Python package provides.

**Note**

If you are using the `ansible-playbook` command, or the `ansible-navigator` command with the `--ee false` option, the execution environment is the same as the control node and these Python components need to be installed on the control node. Otherwise, these resources need to be installed in the execution environment that you are using with that collection.

The supported execution environments provide all the Python dependencies and other tools needed for the content collections that they include by default.

For those Python dependencies, create a `requirements.txt` file at the root of your collection. Each line of the file declares a Python package. The following example shows the content of the `requirements.txt` file for the `amazon.aws` collection. At the end of the package name, the version part is optional if you do not require a specific version.

```
botocore>=1.18.0
boto3>=1.15.0
boto>=2.49.0
```

Some collections also require system-level packages in the execution environment. For example, the `ansible.posix` collection requires the `rsync` RPM package. At the root of your collection, create the `bindep.txt` file and list the RPM packages, one per line. The following example shows the content of the `bindep.txt` file for the `ansible.posix` collection.

```
rsync [platform:centos-8 platform:rhel-8]
```

The `bindep` tool, which processes the `bindep.txt` file, can manage packages for several Linux distributions. Different distributions might have other package names for the same software. In the preceding file, the `[platform:centos-8 platform:rhel-8]` directive provides the name of the Linux distributions. Another standard directive is `[platform:rpm]`, which targets all the Linux distributions that use the RPM packaging system.

**Note**

When a user or the automation controller runs the `ansible-galaxy collection install` command to install your collection, the command automatically installs the additional collections that you have declared in the `galaxy.yml` file.

However, the command does not process the `requirements.txt` file for Python packages or the `bindep.txt` file for system packages.

On the other hand, the `ansible-builder` command that you run to create automation execution environments uses all those files. Another section of this course discusses the execution environment builder in more detail.

You can define additional metadata for the collection in the `meta/runtime.yml` file. If your collection requires a specific version of Ansible, then add the `requires_ansible` parameter to the file. The following example specifies that the collection works with Ansible 2.10 or later.

```

requires_ansible: ">=2.10"
```

Because the `ansible-galaxy collection init` command does not create the `meta/` directory, you must create it yourself, as well as the `runtime.yml` file.



### Important

The `meta/runtime.yml` file and the `requires_ansible` parameter are mandatory if you plan to publish your collection to Ansible Galaxy, automation hub, or private automation hub. Otherwise, the validation process that those platforms run rejects the collection.

## Building Collections

From inside the collection directory, run the `ansible-galaxy collection build` command to prepare the collection:

```
[user@host mycollection]$ ansible-galaxy collection build
Created collection for mynamespace.mycollection at /home/user/mynamespace/
mycollection/mynamespace-mycollection-1.0.0.tar.gz
```

You can use the resulting `.tar.gz` file to install and test the collection locally, to share the collection with team members, or to publish it to a distribution platform such as Ansible Galaxy or private automation hub.

## Validating and Testing Collections

When you publish your collection to Ansible Galaxy, automation hub, or private automation hub, the platforms automatically run some tests.

For example, private automation hub uses the `ansible-lint` tool to verify that your collection conforms to the standards and good practices that the Ansible team establishes. The distribution platform publishes the resulting report alongside the collection. The following output shows an import log for the `mynamespace.mycollection` collection.

```
Importing with galaxy-importer 0.4.4
Getting doc strings via ansible-doc
Finding content inside collection
Loading role myrole
Linting role myrole via ansible-lint...
roles/myrole/tasks/main.yml:3: fqcn-builtins Use FQCN for builtin actions.
roles/myrole/tasks/main.yml:3: risky-file-permissions File permissions unset or
incorrect.
CHANGELOG.rst file not found at top level of collection.
Collection loading complete

Done
```

If the import log displays the messages `Collection loading complete` and `Done`, then the collection uploaded successfully. Notice that the log produces warnings about `fqcn-builtins`

and `risky-file-permissions`. Although not required, you might update the collection to fix any reported problems.

If your collection does not include a `CHANGELOG.rst` file, then the import log displays the message: `CHANGELOG.rst` file not found at top level of collection. Because this file is not required, you can safely ignore this message. You can find information about generating change logs in the References section.

For collections that Red Hat partners develop, automation hub tests more thoroughly using the `ansible-test` tool.

The Ansible team recommends that you install and then use those tools before publishing your collections. In addition, some community projects also run those tools and reject contributions that do not pass the tests.

At a minimum, install your collection on a test system from the `.tar.gz` file, and then develop a playbook that uses the collection and demonstrates that it works as expected.



### Note

To test your collection in playbooks, you cannot directly use the collection from your development directory by pointing the `collections_paths` directive in the `ansible.cfg` configuration file to that working directory. You first need to build and then install the collection from the `.tar.gz` file.

## Publishing Collections

Use the distribution platform web UI to publish your collection from the `.tar.gz` file.

For private automation hub, navigate to **Collections > Namespaces**, select your namespace, click **Upload collection**, and then provide the `.tar.gz` file. Before users can use your collection, an administrator must review and approve it. If your user account has that permission, then navigate to **Collections > Approval**. You can view the import log, which displays the result of the `ansible-lint` validation, and then approve or reject the collection.

| Namespace   | Collection   | Version | Date created | Status                    |
|-------------|--------------|---------|--------------|---------------------------|
| mynamespace | mycollection | 1.0.0   | an hour ago  | <span>Needs Review</span> |

Figure 9.1: Approving Ansible Content Collections

## Chapter 9 | Creating Content Collections and Execution Environments

Whenever you modify your collection, edit the `galaxy.yml` file and update the `version` parameter. The distribution platform rejects your collection otherwise. Create a `.tar.gz` archive and then publish the new version.

```
[user@host mycollection]$ cat galaxy.yml

namespace: mynamespace
name: mycollection
version: 2.0.0
readme: README.md
authors:
 - your name <example@domain.com>
...output omitted...
[user@host mycollection]$ ansible-galaxy collection build
Created collection for mynamespace.mycollection at /home/user/mynamespace/
mycollection/mynamespace-mycollection-2.0.0.tar.gz
```

As an alternative to uploading a collection archive file using the private automation hub web UI, the `ansible-galaxy` command also supports publishing collections.

Update your Ansible configuration file to contain settings similar to the following. Your `server_list` line might contain additional servers.

```
[galaxy]
server_list = inbound_mynamespace

[galaxy_server.inbound_mynamespace]
url=https://hub.lab.example.com/api/galaxy/content/inbound-mynamespace/
token=<put your token here>
```

The private automation hub web UI displays the inbound URL for your namespace. Navigate to **Collections > Namespaces**, click the **View collections** link for the desired namespace, and then click the **CLI configuration** tab. Retrieve your API token from the private automation hub web UI. Navigate to **Collections > API token management** and click **Load token**. Update the `token` line with your authentication token.

Run the `ansible-galaxy collection publish` command to publish your collection to Ansible Galaxy or private automation hub. Notice that the command output displays the same messages as the private automation hub web UI import log.

```
[user@host mycollection]$ ansible-galaxy collection \
> publish mynamespace-mycollection-2.0.0.tar.gz
...output omitted...
[WARNINg]: Galaxy import warning message: roles/myrole/tasks/main.yml:3: fqcn-
builtins Use FQCN for builtin actions.
[WARNINg]: Galaxy import warning message: roles/myrole/tasks/main.yml:3: risky-
file-permissions File permissions unset or
incorrect.
[ERROR]: Galaxy import error message: CHANGELOG.rst file not found at top level
of collection.
Collection has been successfully published and imported to the Galaxy server
inbound_mynamespace https://hub.lab.example.com/api/galaxy/content/inbound-
mynamespace/
```



## References

### **Developing Collections – Ansible Documentation**

[https://docs.ansible.com/ansible/6/dev\\_guide/developing\\_collections.html](https://docs.ansible.com/ansible/6/dev_guide/developing_collections.html)

### **Collection Galaxy Metadata Structure – Ansible Documentation**

[https://docs.ansible.com/ansible/6/dev\\_guide/collections\\_galaxy\\_meta.html](https://docs.ansible.com/ansible/6/dev_guide/collections_galaxy_meta.html)

### **Python Packages Requirements File Format**

[https://pip.pypa.io/en/stable/cli/pip\\_install/#requirements-file-format](https://pip.pypa.io/en/stable/cli/pip_install/#requirements-file-format)

### **Introduction to binep**

<https://docs.opendev.org/opendev/binep/latest/readme.html>

### **Generating Changelogs and Porting Guide Entries in a Collection**

[https://docs.ansible.com/ansible/6/dev\\_guide/developing\\_collections\\_changelogs.html](https://docs.ansible.com/ansible/6/dev_guide/developing_collections_changelogs.html)

## ► Guided Exercise

# Writing Ansible Content Collections

Create a content collection and publish it to a private automation hub.

## Outcomes

- Create a collection directory structure.
- Add content to the collection.
- Build the collection.
- Publish the collection.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command deploys the Ansible environment. It also prepares an Ansible project in the `/home/student/create-writing/` directory on the workstation machine.

```
[student@workstation ~]$ lab start create-writing
```

## Instructions

- 1. Change to the `/home/student/create-writing/` directory, and run the following command to create a collection:

```
[student@workstation ~]$ cd ~/create-writing/
[student@workstation create-writing]$ ansible-galaxy collection \
> init student.testcollection
- Collection student.testcollection was created successfully
```

- 2. Review the new collection.

- 2.1. Use the `tree` command to review the directory structure of the new collection.

```
[student@workstation create-writing]$ tree student/testcollection/
student/testcollection/
├── docs
├── galaxy.yml
├── plugins
│ └── README.md
└── README.md
 └── roles

3 directories, 3 files
```

- 2.2. Review the collection metadata, such as namespace, name, and version in the `galaxy.yml` file. The following `grep` command displays any line that does not start with the `#` character and is not a blank line.

```
[student@workstation create-writing]$ cd student/testcollection/
[student@workstation testcollection]$ grep -Ev "^#|^$" galaxy.yml | head
namespace: student
name: testcollection
version: 1.0.0
readme: README.md
authors:
- your name <example@domain.com>
description: your collection description
license:
- GPL-2.0-or-later
license_file: ''
```

- 3. Create the `meta/runtime.yml` file to specify that the collection requires Ansible 2.9.10 or later to work.

- 3.1. Create the `~/create-writing/student/testcollection/meta/` directory.

```
[student@workstation testcollection]$ mkdir meta
```

- 3.2. Create the `~/create-writing/student/testcollection/meta/runtime.yml` file with the following content:

```
...
requires_ansible: '>=2.9.10'
```

- 4. Place a module in this collection. Copy the `myping.py` module provided in the `/home/student/create-writing/` directory.

```
[student@workstation testcollection]$ ls plugins/
README.md
[student@workstation testcollection]$ mkdir plugins/modules
[student@workstation testcollection]$ cp ~/create-writing/myping.py \
> plugins/modules/
[student@workstation testcollection]$ ls plugins/modules/
myping.py
```

- 5. Create a role inside the collection.

- 5.1. Use the `ansible-galaxy` command to create the `mymotd` role, and then change to the `mymotd` directory.

```
[student@workstation testcollection]$ cd roles/
[student@workstation roles]$ ansible-galaxy role init mymotd
- Role mymotd was created successfully
[student@workstation roles]$ cd mymotd/
```

**Chapter 9 |** Creating Content Collections and Execution Environments

- 5.2. Create the `templates/motd.j2` template in the role with the following content. This example shows setting the `_date` variable with Jinja2 so that a date such as September 26, 2022 displays as 26-Sep-2022. The `date(1)` man page contains information about formatting options.

```
{% set the_date = '%d-%b-%Y' | strftime(ansible_facts['date_time']['epoch']) %}
Connected to: {{ ansible_facts['fqdn'] }}
Ansible last updated the /etc/motd file on {{ the_date }}.
```

- 5.3. Create the `tasks/main.yml` task file in the role with following content.

```

tasks file for mymotd
- name: update the /etc/motd file
 ansible.builtin.template:
 src: motd.j2
 dest: /etc/motd
 owner: root
 group: root
 mode: '0444'
```

- ▶ 6. Use the `tree` command to review the new directory structure of the collection.

```
[student@workstation mymotd]$ cd ~/create-writing/student/testcollection/
[student@workstation testcollection]$ tree
.
├── docs
├── galaxy.yml
├── meta
│ └── runtime.yml
└── plugins
 ├── modules
 │ └── myping.py
 └── README.md
├── README.md
└── roles
 └── mymotd
 ├── defaults
 │ └── main.yml
 ├── files
 ├── handlers
 │ └── main.yml
 ├── meta
 │ └── main.yml
 ├── README.md
 ├── tasks
 │ └── main.yml
 ├── templates
 │ └── motd.j2
 ├── tests
 │ └── inventory
 │ └── test.yml
 └── vars
```

```
└── main.yml
14 directories, 14 files
```

- 6.1. Delete the unneeded files and directories created by the `ansible-galaxy collection init` and the `ansible-galaxy role init` commands. Red Hat recommends that you delete unneeded files and directories.

```
[student@workstation testcollection]$ rm -vr docs \
> roles/mymotd/{defaults,files,handlers,tests,vars}
removed directory 'docs'
removed 'roles/mymotd/defaults/main.yml'
removed directory 'roles/mymotd/defaults/'
removed directory 'roles/mymotd/files'
removed 'roles/mymotd/handlers/main.yml'
removed directory 'roles/mymotd/handlers'
removed 'roles/mymotd/tests/inventory'
removed 'roles/mymotd/tests/test.yml'
removed directory 'roles/mymotd/tests'
removed 'roles/mymotd/vars/main.yml'
removed directory 'roles/mymotd/vars'
[student@workstation testcollection]$ tree
. .
├── galaxy.yml
├── meta
│ └── runtime.yml
├── plugins
│ ├── modules
│ │ └── myping.py
│ └── README.md
├── README.md
└── roles
 └── mymotd
 ├── meta
 │ └── main.yml
 ├── README.md
 ├── tasks
 │ └── main.yml
 └── templates
 └── motd.j2

8 directories, 9 files
```

- 7. Use the `ansible-galaxy` command to build the collection.

```
[student@workstation testcollection]$ ansible-galaxy collection build
Created collection for student.testcollection at /home/student/create-writing/
student/testcollection/student-testcollection-1.0.0.tar.gz
[student@workstation testcollection]$ ls
galaxy.yml meta plugins README.md roles student-testcollection-1.0.0.tar.gz
```

- 8. Create the `student` namespace in private automation hub and then upload and approve the new collection.

**Chapter 9 |** Creating Content Collections and Execution Environments

- 8.1. Navigate to <https://hub.lab.example.com> and log in as the student user with **redhat123** as the password.
- 8.2. From the private automation hub web UI, navigate to **Collections > Namespaces** and then click **Create**.
- 8.3. On the **Create new namespace** page, complete the details as follows and then click **Create** to create the namespace.

| Field            | Value              |
|------------------|--------------------|
| Name             | student            |
| Namespace owners | Content Developers |

**Important**

The Content Developers group must be a namespace owner for group members, such as the student user, to upload to the namespace.

- 8.4. Click **Upload collection**.
  - 8.5. Click **Select file**. Select the archive located at `/home/student/create-writing/student/testcollection/student-testcollection-1.0.0.tar.gz` and then click **Upload**.  
Even though the private automation hub web UI displays a message about a missing `CHANGELOG.rst` file, the collection uploads successfully.
  - 8.6. Click **Collections > Approval** and then click **Approve** to approve the `student.testcollection` collection.
  - 8.7. Navigate to **Collections > Collections** and verify that the private automation hub server displays the `testcollection` automation content collection.
- 9. Use the `ansible-galaxy` command to install the `student.testcollection` collection from private automation hub.
- 9.1. Change to the `/home/student/create-writing/` directory.

```
[student@workstation testcollection]$ cd ~/create-writing/
```

- 9.2. From the private automation hub web UI, navigate to **Collections > Repository Management**. Copy the CLI configuration for the published repository.
- 9.3. Update the `/home/student/create-writing/ansible.cfg` file to include the published repository. Paste the configuration copied in the previous step.

```
[defaults]
inventory = ./inventory
remote_user = devops
collections_paths = ./collections:/usr/share/ansible/collections

[galaxy]
```

```
server_list = published_repo

[galaxy_server.published_repo]
url=https://hub.lab.example.com/api/galaxy/content/published/
token=<put your token here>
```

- 9.4. From the private automation hub web UI, navigate to **Collections > API token management** and then click **Load token**. Click the **Copy to clipboard** icon.
- 9.5. Update the `/home/student/create-writing/ansible.cfg` file. Modify the `token` line to use the API token copied in the previous step. Your token is different from the one displayed in this example.

```
[defaults]
inventory = ./inventory
remote_user = devops
collections_paths = ./collections:/usr/share/ansible/collections

[galaxy]
server_list = published_repo

[galaxy_server.published_repo]
url=https://hub.lab.example.com/api/galaxy/content/published/
token=adc9fea87c50206c439ca3bb39a9404c6c600cf3
```

- 9.6. The project includes a `~/create-writing/collections/requirements.yml` file that references the `student.testcollection` collection. Review the file to confirm that the contents are as follows:

```

collections:
 - name: student.testcollection
```

- 9.7. Ensure that you are in the `/home/student/create-writing` directory, and install the collection using the `ansible-galaxy` command. The collection is installed into the `./collections` directory so that it is loaded into the automation execution environment.

```
[student@workstation create-writing]$ ansible-galaxy collection \
> install -r collections/requirements.yml -p collections/
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://hub.lab.example.com/api/galaxy/v3/plugin/ansible/content/
published/collections/artifacts/student-testcollection-1.0.0.tar.gz to /
home/student/.ansible/tmp/ansible-local-247766f2gkx05/tmptvok5tde/student-
testcollection-1.0.0-nb_d_xxi
Installing 'student.testcollection:1.0.0' to '/home/student/create-writing/
collections/ansible_collections/student/testcollection'
student.testcollection:1.0.0 was installed successfully
```

- 10. Review and run the `site.yml` playbook to test the `student.testcollection` collection.

10.1. Review the `site.yml` playbook.

```

- name: Test custom collection
 hosts: serverd.lab.example.com
 become: true

 roles:
 - role: student.testcollection.mymotd
```

10.2. Use the `ansible-navigator run` command to run the `site.yml` playbook.

```
[student@workstation create-writing]$ ansible-navigator run site.yml \
> -m stdout

PLAY [Test custom collection] ****
TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]

TASK [student.testcollection.mymotd : update the /etc/motd file] ****
changed: [serverd.lab.example.com]

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

10.3. Log in to `serverd` to verify that the play was successful. The date displayed on your server is different from this example.

```
[student@workstation create-writing]$ ssh serverd
Connected to: serverd.lab.example.com
Ansible last updated the /etc/motd file on 16-Jan-2023.
...output omitted...
[student@serverd ~]$ logout
Connection to serverd closed.
[student@workstation create-writing]$
```

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish create-writing
```

# Building a Custom Automation Execution Environment

## Objectives

- Build a custom automation execution environment image by using the `ansible-builder` command.

## Deciding When to Create a Custom Automation Execution Environment

Red Hat provides several Ansible automation execution environments that suit the needs of many users. These automation execution environments include the most common Ansible Content Collections.

You can use the `ansible-navigator images` command to inspect a container image and view the collections, Python packages, and operating system packages it includes.

| Image: ee-supported-rhel8:latest  | Description                                   |
|-----------------------------------|-----------------------------------------------|
| 0 Image information               | Information collected from image inspection   |
| 1 General information             | OS and python version information             |
| 2 Ansible version and collections | Information about ansible and ansible collect |
| 3 Python packages                 | Information about python and python packages  |
| 4 Operating system packages       | Information about operating system packages   |
| 5 Everything                      | All image information                         |

The automation execution environments that Red Hat provides might be sufficient for your playbooks. In that case, you do not need to create new automation execution environments.

Sometimes, you might need an Ansible Content Collection that is not included in one of the existing automation execution environments. Often, you can install these extra collections in your Ansible project and then use an existing automation execution environment without having to build a new one.

On the other hand, you should consider creating a custom automation execution environment in the following situations:

- You frequently reuse specific collections that are not included in the existing automation execution environments. It is more efficient to embed those collections in a custom automation execution environment than to install them from the project `requirements.yml` file every time you run a playbook in that project. This is especially true if you have many projects that use that collection.
- The collection you want to use requires Python packages or software that are not included in an existing automation execution environment.
- You need to use a collection that conflicts with another collection in your existing automation execution environments.

# Preparing for a New Automation Execution Environment

You use the `ansible-builder` command to create the container images used for automation execution environments. The `ansible-builder` RPM package provides that command.

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

Create a working directory to prepare the files that you need to build the automation execution environment container image. The `ansible-builder` command searches this working directory for its `execution-environment.yml` configuration file, which it uses to determine how to build the container image.

The following is an example `execution-environment.yml` file:

```

version: 1

build_arg_defaults:
 EE_BASE_IMAGE: registry.redhat.io/ansible-automation-platform-22/ee-minimal-
 rhel8:latest ①
 EE_BUILDER_IMAGE: registry.redhat.io/ansible-automation-platform-22/ansible-
 builder-rhel8:latest ②

ansible_config: ansible.cfg ③

dependencies:
 galaxy: requirements.yml ④
 python: requirements.txt ⑤
 system: bindep.txt ⑥
```

- ① The `EE_BASE_IMAGE` parameter specifies the automation execution environment container image to use as the starting point. By default, the `ansible-builder` command uses the Red Hat minimal image `registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest`.



## Note

You are not allowed to publicly distribute container images that you build on top of the automation execution environment images that Red Hat provides with Red Hat Ansible Automation Platform.

To distribute your images on a public platform, such as Quay.io, use the public `quay.io/ansible/ansible-runner:stable-2.12-latest` base image instead. Alternatively, you can publish your `execution-environment.yml` file and let users build the container image themselves.

Red Hat only supports container images that you build on top of the Red Hat images.

- ② The optional `EE_BUILDER_IMAGE` parameter points to a container image that includes the tools that the build process uses to construct automation execution environment container images. You should not need to use an image different from the default one.

- ③ The optional `ansible_config` parameter specifies an Ansible configuration file. The build process uses that file to pull collections from a location that requires authentication. For example, you must authenticate using an authentication URL and a token to install collections from an Ansible automation hub. If you only install collections from a public distribution platform, such as Ansible Galaxy, then you do not need to provide that file.
- ④ The `galaxy` parameter points to the file that lists the collections to install inside the automation execution environment. Ansible users usually set the file name to `requirements.yml`.
- ⑤ The `python` parameter points to the file that lists the Python packages to install. Some collections include a `requirements.txt` file. If you install such a collection, then you do not need to list its Python dependencies. Python developers usually set the file name to `requirements.txt`.
- ⑥ The `system` parameter points to the file that lists the RPM packages to install. Some collections include a `bindep.txt` file. If you install such a collection, then you do not need to list its RPM dependencies. Ansible users usually set the file name to `bindep.txt`, which is also the name used by the `bindep` tool that processes the file.

## Declaring the Ansible Content Collections to Install

In the `requirements.yml` file, list the content collections that `ansible-builder` should install inside the automation execution environment.

The following example is a straightforward `requirements.yml` file that instructs the `ansible-builder` command to install the `community.aws` and `community.general` collections from Ansible Galaxy.

```

collections:
 - community.aws
 - community.general
```

The following example is a more complex `requirements.yml` file.

```

collections:
 - name: redhat.insights ①
 version: 1.0.5 ②
 source: https://console.redhat.com/api/automation-hub/ ③
 - name: ansible.posix
 source: https://hub.example.com/api/galaxy/content/rh-certified/ ④
```

- ① The name of the collection.
- ② A specific version of the collection. If not specified, then the `ansible-builder` command selects the latest version.
- ③ The distribution platform that provides the collection. This example uses the Ansible automation hub address. You must create an `ansible.cfg` file and populate it with the token used to access that server. In the `execution-environment.yml` configuration file, you reference that `ansible.cfg` file by setting the `ansible_config` parameter.

## Chapter 9 | Creating Content Collections and Execution Environments

- ④ A collection from private automation hub. You also need to provide your token in an `ansible.cfg` file.

The following example `ansible.cfg` file provides the parameters that the `ansible-builder` command needs to authenticate to Ansible automation hub and to private automation hub (at `hub.example.com` in this example).

```
[galaxy]
server_list = ansible_automation_hub, my_hub, galaxy

[galaxy_server.ansible_automation_hub]
url=https://console.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token
token=eyJh...0Nwk

[galaxy_server.my_hub]
url=https://hub.example.com/api/galaxy/content/rh-certified/
token=e8e4...b0c2

[galaxy_server.galaxy]
url=https://galaxy.ansible.com/
```

## Declaring Python Packages

In the `requirements.txt` file, list the Python packages that `ansible-builder` must install inside the automation execution environment.

The following example `requirements.txt` file lists some required Python packages. For each package, you can provide a specific version to select.

```
sh==1.13.1
jsonschema>=3.2.0,<4.0.1
textfsm
ttp
xmldict
dnspython
```

Some collections include a `requirements.txt` file. In that case, you do not have to duplicate that file in your `requirements.txt` file.

## Declaring RPM Packages

In the `bindep.txt` file, list the RPM packages that `ansible-builder` must install inside the automation execution environment.

The following `bindep.txt` file lists the RPM packages required by the preceding examples.

```
rsync [platform:rpm]
kubernetes-client [platform:rpm]
```

The `bindep` tool, which processes the `bindep.txt` file, can manage packages for several Linux distributions. Different distributions might have other package names for the same software. In

the preceding file, the [platform:rpm] directive targets all the Linux distributions that use the RPM packaging system.

Some collections include a `bindep.txt` file. In that case, you do not have to duplicate that file in your `bindep.txt` file.

If the system on which you run the build process has a valid subscription, then the `ansible-builder` command has access to the same RPM packages as that system. However, if you do not register your system with Red Hat, then the build process has only access to the Red Hat Universal Base Images (UBI) repositories. These repositories are publicly available but only provide a limited set of RPM packages. The packages you define in the `bindep.txt` file might not be available in these repositories.

## Building a New Automation Execution Environment

The execution environment builder automatically pulls the base and the builder images if they are not already available locally. If you use images from a container registry that requires authentication, then you must authenticate before starting the build process.

The following example shows how to log in to the Red Hat container registry that hosts the default images, `ee-minimal-rhel8` and `ansible-builder-rhel8`:

```
[user@host demo]$ podman login registry.redhat.io
```

After you have prepared your configuration files and authenticated to the container registry, use the `ansible-builder build` command to create your automation execution environment container image.

Use the `--tag` (or `-t`) option to provide a name for the container image. The name has two parts: a name and an optional tag. For example, use the `--tag ee-demo:v1.0` option to name the container `demo` and give it the `v1.0` tag. The tag defaults to `latest` if you do not specify it. For example, using the `--tag demo` option is the equivalent of using `--tag demo:latest`.

A successful build creates a new container image.

```
[user@host demo]$ ansible-builder build --tag ee-demo:v1.0
Running command:
podman build -f context/Containerfile -t ee-demo:v1.0 context
Complete! The build context can be found at: /home/user/demo/context
```

Use the `podman images` command to display local container images.

```
[user@host demo]$ podman images
REPOSITORY TAG IMAGE ID CREATED SIZE
localhost/ee-demo v1.0 9e1e281437a2 2 hours ago 744 MB
...output omitted...
```

## Interacting with the Build Process

For more advanced configurations, you might have to customize the build process. For example, suppose your private automation hub uses a TLS certificate signed by a corporate certificate authority (CA). In that case, you must provide that CA certificate to the build process so that it can download collections from the distribution platform.

The execution environment builder operates in two stages:

1. The command creates the `context` / directory in the current directory. In that directory, it creates the `Containerfile` file, the equivalent of a `Dockerfile` file, which contains instructions for the `podman build` command. It also creates an `_build` / subdirectory and then copies your `requirements.yml`, `ansible.cfg`, `requirements.txt`, and `bindep.txt` files into it so that the build process can access them.
2. The command then runs the `podman build` command, which constructs the resulting automation execution environment container image.

To prepare for customizing the build process, use the `ansible-builder create` command to perform the first stage:

```
[user@host demo]$ ls
ansible.cfg bindep.txt execution-environment.yml requirements.txt requirements.yml
[user@host demo]$ ansible-builder create
Complete! The build context can be found at: /home/user/demo/context
[user@host demo]$ tree
.
├── ansible.cfg
├── bindep.txt
└── context
 ├── _build
 │ ├── ansible.cfg
 │ ├── bindep.txt
 │ ├── requirements.txt
 │ └── requirements.yml
 └── Containerfile
└── execution-environment.yml
└── requirements.txt
└── requirements.yml

2 directories, 10 files
```



### Important

The `ansible-builder create` command always overwrites the `context` / `Containerfile` file. Any changes you make to the `context` / `Containerfile` file are lost.

## Adjusting the Build Arguments and Base Image

Default values for build arguments can be specified in the `execution-environment.yml` configuration file, defining the `build_arg_defaults` variable as a dictionary of arguments and their values. You can also specify these settings from the `ansible-builder` command line with the `--build-arg` option. If you also have an `execution-environment.yml` file, the values specified by the `--build-arg` option always take precedence.

Build arguments that are used by `ansible-builder` include:

- `ANSIBLE_GALAXY_CLI_COLLECTION_OPTS` passes additional command-line options to the `ansible-galaxy` command that the build process runs. For example, you can add the `--pre` option if you want to install prerelease versions of collections.

## Chapter 9 | Creating Content Collections and Execution Environments

- `EE_BASE_IMAGE` specifies the parent image for the execution environment.
- `EE_BUILDER_IMAGE` specifies the image used for compiling tasks.

```
build_arg_defaults:
 EE_BASE_IMAGE: registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest
 EE_BUILDER_IMAGE: registry.redhat.io/ansible-automation-platform-22/ansible-builder-rhel8:latest
```

The current build arguments are hard coded into the `Containerfile` file when you use the `ansible-builder create` command to generate that file.

## Modifying the Containerfile File

After running the `ansible-builder create` command, you can navigate to the context/directory and modify the `Containerfile` file to adapt it to your needs.

For example, you could add the following two directives in the `Containerfile` file to inject a private CA's certificate into the image:

```
ARG EE_BASE_IMAGE=registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest
ARG EE_BUILDER_IMAGE=registry.redhat.io/ansible-automation-platform-22/ansible-builder-rhel8:latest

FROM $EE_BASE_IMAGE as galaxy
ARG ANSIBLE_GALAXY_CLI_COLLECTION_OPTS=
USER root

COPY my-company-ca.pem /etc/pki/ca-trust/source/anchors
RUN update-ca-trust

ADD _build/ansible.cfg ~/.ansible.cfg
...output omitted...
```

For this example, you would place the `my-company-ca.pem` certificate file in the context/directory.

## Building the Execution Environment Image

When ready to build the container image, you manually execute the second stage by running the `podman build` command.

```
[user@host demo]$ podman build -f context/Containerfile -t ee-demo:v2.0 context
STEP 1: FROM registry.redhat.io/ansible-automation-platform-22/ee-minimal-rhel8:latest AS galaxy
STEP 2: ARG ANSIBLE_GALAXY_CLI_COLLECTION_OPTS=
--> 5894c40a543
STEP 3: USER root
--> 913568f77bb
STEP 4: COPY my-company-ca.pem /etc/pki/ca-trust/source/anchors
--> b9c4d7eaad0
STEP 5: RUN update-ca-trust
```

```
--> be65558ebb8
STEP 6: ADD _build/ansible.cfg ~/.ansible.cfg
--> 4a5410ba1c2
...output omitted...
STEP 22: COMMIT ee-demo:v2.0
--> 8be47093daa
8be47093daaae18160468f3db6e0335d55d0d11c838c7e285d74037e4a31d8a9
[user@host demo]$ podman images
REPOSITORY TAG IMAGE ID CREATED SIZE
localhost/ee-demo v2.0 ba77b0d0a59d 2 minutes ago 308 MB
...output omitted...
```



## References

**Knowledgebase: What's new with Ansible Automation Platform 2.0: Developing with ansible-builder and Automation execution environments.**

<https://access.redhat.com/articles/6177982>

**Knowledgebase: Universal Base Images (UBI): Images, Repositories, Packages, and Source Code**

<https://access.redhat.com/articles/4238681>

**The Anatomy of Automation Execution Environments**

<https://www.ansible.com/blog/the-anatomy-of-automation-execution-environments>

**Introduction to Ansible Builder**

<https://www.ansible.com/blog/introduction-to-ansible-builder>

**Introduction to bindep**

<https://docs.opendev.org/opendev/bindep/latest/readme.html>

For more information, refer to the *Using Ansible Builder* chapter in the *Ansible Builder Guide* at

[https://access.redhat.com/documentation/en-us/red\\_hat\\_ansible\\_automation\\_platform/2.1/html-single/ansible\\_builder\\_guide/index#assembly-using-builder](https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.1/html-single/ansible_builder_guide/index#assembly-using-builder)

## ► Guided Exercise

# Building a Custom Automation Execution Environment

Create a custom automation execution environment that provides a specific Ansible Content Collection and its supporting software.

### Outcomes

- Create the configuration files needed for a custom automation execution environment.
- Build a new automation execution environment.
- Verify that the new automation execution environment contains the specified Ansible Content Collection.

### Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates an Ansible project in the `/home/student/builder-custom/` directory.

```
[student@workstation ~]$ lab start builder-custom
```

### Instructions

- 1. Review some of the files in the `exercise.motd` custom Ansible Content Collection. This collection contains a role that updates the Message of the Day file that is displayed when users log in.
- 1.1. Change to the `~/builder-custom/exercise.motd/` directory and extract the `~/builder-custom/exercise.motd.tar.gz` archive.

```
[student@workstation ~]$ cd ~/builder-custom/exercise.motd/
[student@workstation exercise.motd]$ tar -xzf ../exercise.motd.tar.gz
```

- 1.2. Use the `tree` command to list the files in the working directory.

```
[student@workstation exercise.motd]$ tree
. . .
└── roles
 └── banner
 └── defaults
```

```
| └── main.yml
|── meta
| └── main.yml
|── README.md
|── tasks
| └── main.yml
└── templates
 └── motd.j2
```

6 directories, 9 files

13. Display the content of the `motd.j2` template file.

```
[student@workstation exercise.motd]$ cat roles/banner/templates/motd.j2
```

```
=====
=====
== ==
{{ formatted_msg_1 }}
{{ formatted_msg_2 }}
== ==
=====
```

The banner role creates the `/etc/motd.d/banner` file using this template.

14. Display the content of the main task file in the banner role.

```
[student@workstation exercise.motd]$ cat roles/banner/tasks/main.yml

tasks file for banner
- name: Ensure /etc/motd.d/ exists
 ansible.builtin.file:
 path: /etc/motd.d
 state: directory
 owner: root
 group: root
 mode: '0755'

- name: Create custom motd banner
 vars:
 left_pad_1: "{{ (width - (message_1 | length)) // 2 }}"
 left_pad_2: "{{ (width - (message_2 | length)) // 2 }}"
 right_pad_1: "{{ left_pad_1|int + ((width - (message_1 | length)) % 2) }}"
 right_pad_2: "{{ left_pad_2|int + ((width - (message_2 | length)) % 2) }}"
 formatted_msg_1: "={{% for x in range((left_pad_1|int) - 1) %} {% endfor %}}"
 {{ message_1 }}{{% for y in range((right_pad_1|int) - 1) %} {% endfor %}}"
 formatted_msg_2: "={{% for x in range((left_pad_2|int) - 1) %} {% endfor %}}"
 {{ message_2 }}{{% for y in range((right_pad_2|int) - 1) %} {% endfor %}}"
 ansible.builtin.template:
 src: motd.j2
 dest: /etc/motd.d/banner
```

The banner role uses the `width`, `message_1`, and `message_2` variables to align two custom messages in the center of the `/etc/motd.d/banner` file. When using

the role, users might override the default value of the `message_1` and `message_2` variables.

► 2. Create the configuration files for the new automation execution environment.

2.1. Create and change into the `~/builder-custom/ee-motd/` directory.

```
[student@workstation exercise.motd]$ mkdir ~/builder-custom/ee-motd/
[student@workstation exercise.motd]$ cd ~/builder-custom/ee-motd/
```

2.2. Create the `execution-environment.yml` file with the following content:

```

version: 1
build_arg_defaults:
 EE_BASE_IMAGE: 'hub.lab.example.com/ee-minimal-rhel8:latest'
 EE_BUILDER_IMAGE: 'hub.lab.example.com/ansible-builder-rhel8:latest'
dependencies:
 galaxy: requirements.yml
 python: requirements.txt
 system: bindep.txt
```

The `lab` command downloaded the `ee-minimal-rhel8` container image to the `workstation` machine. This container image can be used as a base image for additional customizations.



**Note**

You can use the `~/builder-custom/solutions/execution-environment.yml` file for comparison.

2.3. Create the `requirements.yml` file with the following content:

```

collections:
 - name: /build/exercise.motd.tar.gz
 type: file
```



**Note**

The path to the `exercise.motd.tar.gz` archive file is the absolute path within a container used during the build process.

You can use the `~/builder-custom/solutions/requirements.yml` file for comparison.

2.4. Create the `requirements.txt` file for the Python package requirements with the following content:

```
Python dependencies
funmoto
```

**Note**

You can specify the version number for the Python package. For example:

```
funmotd==0.3
```

Use the `~/builder-custom/solutions/requirements.txt` file for comparison.

2.5. Create the `bindep.txt` file with the following content:

```
System-level dependencies
hostname
```

Use `bindep.txt` file to specify the system packages to install. Add one package per line.

**Note**

As with the Python packages, you can specify the version to install.

Use the `~/builder-custom/solutions/bindep.txt` file for comparison.

▶ 3. Verify that `ansible-builder` is installed.

```
[student@workstation ee-motd]$ rpm -qa | grep ansible
python39-ansible-runner-2.2.1-1.el8ap.noarch
ansible-runner-2.2.1-1.el8ap.noarch
ansible-navigator-2.1.0-1.el8ap.noarch
ansible-builder-1.1.0-3.el8ap.noarch
ansible-core-2.13.4-2.el8ap.x86_64
```

**Note**

If `ansible-builder` is not installed, install the `ansible-builder` package. The password for the `student` user is `student`.

```
[student@workstation ee-motd]$ sudo yum install ansible-builder
```

▶ 4. Create the automation execution environment.

4.1. Use the `ansible-builder create` command to create additional files used during the build process.

```
[student@workstation ee-motd]$ ansible-builder create
Complete! The build context can be found at: /home/student/builder-custom/ee-motd/
context
```

**Note**

Usually, running the `ansible-builder create` command is optional. Running the `ansible-builder build` command creates the `context` directory and associated files.

In this exercise, you add an additional file to the `context/_build/` directory.

- 4.2. Use the `tree` command to list the files in the current directory. Notice how the `ansible-builder` command created a `context` directory with a `Containerfile` file and an `_build` subdirectory.

```
[student@workstation ee-motd]$ tree
.
├── bindep.txt
└── context
 ├── _build
 │ └── bindep.txt
 │ └── requirements.txt
 │ └── requirements.yml
 └── Containerfile
 ├── execution-environment.yml
 └── requirements.txt
 └── requirements.yml

2 directories, 8 files
```

- 4.3. Copy the `~/builder-custom/exercise.motd.tar.gz` archive to the `context/_build/` directory.

```
[student@workstation ee-motd]$ cp ./exercise.motd.tar.gz context/_build/
```

The build process mounts the contents of the `context/_build/` directory into the container at the `/build/` mount point. Consequently, the `context/_build/exercise.motd.tar.gz` archive is accessible to the container at the location specified in the `requirements.yml` file: `/build/exercise.motd.tar.gz`.

- 4.4. Use the `ansible-builder build` command to build the custom automation execution environment. Label the resulting container image with the `ee-motd-minimal:1.0` tag. The `ansible-builder build` command might take a few minutes to complete.

```
[student@workstation ee-motd]$ ansible-builder build \
> -t ee-motd-minimal:1.0
Running command:
podman build -f context/Containerfile -t ee-motd-minimal:1.0 context
Complete! The build context can be found at: /home/student/builder-custom/ee-motd/
context
```

**Note**

If you have not authenticated to hub.lab.example.com, then you receive an authorization error. Use the podman login hub.lab.example.com command to authenticate the session. When prompted, enter student for the username and redhat123 for the password.

**Important**

The ansible-builder build command displays an associated podman build command. Use the podman build command if you make customizations to the context/Containerfile file that cannot be made by modifying the execution-environment.yml file.

- 4.5. Run the podman images command to list local container images. The image ID for your container might be different from the one displayed.

```
[student@workstation ee-motd]$ podman images localhost/ee-motd-minimal
REPOSITORY TAG IMAGE ID CREATED SIZE
localhost/ee-motd-minimal 1.0 122452abb43b 18 seconds ago 409 MB
```

- ▶ 5. Use the ansible-navigator images command to examine the ee-motd-minimal container image.

- 5.1. Verify that ansible-navigator is installed.

```
[student@workstation ee-motd]$ rpm -qa | grep ansible
python39-ansible-runner-2.2.1-1.el8ap.noarch
ansible-runner-2.2.1-1.el8ap.noarch
ansible-navigator-2.1.0-1.el8ap.noarch
ansible-builder-1.1.0-3.el8ap.noarch
ansible-core-2.13.4-2.el8ap.x86_64
```

**Note**

If the ansible-navigator package is not installed, install it.

```
[student@workstation ee-motd]$ sudo yum install ansible-navigator
```

- 5.2. Use ansible-navigator command to verify that the custom automation execution environment exists locally. Specify the name of the automation execution environment image as localhost/ee-motd-minimal with the tag 1.0, and use never as the image pull policy.

```
[student@workstation ee-motd]$ ansible-navigator images --pp never \
> --eei localhost/ee-motd-minimal:1.0
...output omitted...
```

**Chapter 9 |** Creating Content Collections and Execution Environments

- 5.3. Locate the ee-motd-minimal image and type the number associated with that name. If the number of the container image is greater than nine, then type :NUMBER and press Enter. This is similar to navigating to a specific line number using the Vi and Vim editors.

| Image                    | Tag    | Execution environment | ... |
|--------------------------|--------|-----------------------|-----|
| 0 ansible-builder-rhel8  | latest | False                 | ... |
| 1 ee-minimal-rhel8       | latest | True                  | ... |
| <b>2 ee-motd-minimal</b> | 1.0    | True                  | ... |

- 5.4. Type the number for the Ansible version and collections line.

| Image: ee-motd-minimal:1.0               | Description                                  |
|------------------------------------------|----------------------------------------------|
| 0 Image information                      | Information collected from image inspection  |
| 1 General information                    | OS and python version information            |
| <b>2 Ansible version and collections</b> | Information about ansible and ansible ...    |
| 3 Python packages                        | Information about python and python packages |
| 4 Operating system packages              | Information about operating system packages  |
| 5 Everything                             | All image information                        |

- 5.5. The resulting output indicates that the automation execution environment contains the `exercise.motd` content collection.

```
Image: ee-motd-minimal:1.0 (primary) (Information about ansible and ...)
0|---
1|ansible:
2| collections:
3| details:
4| exercise.motd: 1.0.0
5| version:
6| details: ansible [core 2.13.4]
```

- 5.6. Use the Esc key to go back one level, then type the number for the Python packages line.

| Image: ee-motd-minimal:1.0 (pr...)       | Description                                  |
|------------------------------------------|----------------------------------------------|
| 0 Image information                      | Information collected from image inspection  |
| 1 General information                    | OS and python version information            |
| <b>2 Ansible version and collections</b> | Information about ansible and ansible ...    |
| <b>3 Python packages</b>                 | Information about python and python packages |
| 4 Operating system packages              | Information about operating system packages  |
| 5 Everything                             | All image information                        |

- 5.7. Verify that the `funmotd` package is in the list of installed Python packages.

| Name             | Version | Summary                                |
|------------------|---------|----------------------------------------|
| 0 ansible-compat | 2.0.2   | Ansible compatibility goodies          |
| 1 ansible-core   | 2.13.4  | Radically simple IT automation         |
| 2 ansible-lint   | 6.0.2   | Checks playbooks for practices and ... |
| 3 ansible-runner | 2.2.1   | "Consistent Ansible Python API and     |
| 4 bcrypt         | 3.2.0   | Modern password hashing for your s.... |
| 5 braceexpander  | 2.2.1   | Bash style brace expander.             |

**Chapter 9 |** Creating Content Collections and Execution Environments

```
6|cffi 1.14.3 Foreign Function Interface for Pyt...
7|chardet 3.0.4 Universal encoding detector for Pyt...
8|commonmark 0.9.1 Python parser for the CommonMark Ma...
9|cryptography 3.3.1 cryptography is a package which pro...
10|decorator 5.0.7 Decorators for Humans
11|docutils 0.16 Docutils -- Python Documentation Ut...
12|enrich 1.2.7 enrich
13|funmotd 1.1 TV Show and Movie Quotes MOTD for...
...output omitted...
```

- 5.8. Use the Esc key to go back one level, and then type the number for the **Operating system packages** line.

| Image: ee-motd-minimal:1.0 (pr...  | Description                                  |
|------------------------------------|----------------------------------------------|
| 0 Image information                | Information collected from image inspection  |
| 1 General information              | OS and python version information            |
| 2 Ansible version and collections  | Information about ansible and ansible ...    |
| 3 Python packages                  | Information about python and python packages |
| <b>4 Operating system packages</b> | Information about operating system packages  |
| 5 Everything                       | All image information                        |

- 5.9. Verify that the hostname RPM package you specified as a dependency in `bindep.txt` is included in the container image.

```
...output omitted...
37|grep 3.1 Pattern matching utilities
38|gzip 1.9 The GNU data compression program
39|hostname 3.20 Utility to set/show the host n...
40|info 6.5 A stand-alone TTY-based reader...
41|json-c 0.13.1 JSON implementation in C
...output omitted...
```

Type :q and press Enter to exit the `ansible-navigator` command when finished.

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish builder-custom
```

# Validating a Custom Execution Environment

## Objectives

- Validate that a custom automation execution environment works as expected by testing it with the `ansible-navigator` command, and then distribute the automation execution environment for reuse.

## Testing Automation Execution Environments Locally

After the `ansible-builder` command successfully creates a new automation execution environment, it is important to validate that it works correctly.

To ensure that the automation execution environment uses its internal collections, you can remove locally installed collections from your system. Use the `ansible-galaxy` command to list installed collections.

```
[user@host ~]$ ansible-galaxy collection list

/home/user/.ansible/collections/ansible_collections
Collection Version

ansible.controller 4.2.0
community.crypto 2.7.0
community.general 5.6.0
```

In addition to showing the installed collections, the output displays where the collections are installed. Recursively removing the `ansible_collections` directory removes the locally installed collections.

```
[user@host ~]$ rm -rf /home/user/.ansible/collections/ansible_collections
```

## Running a Test Playbook

If your new automation execution environment uses custom collections, then create a playbook that tests one or more of those collections. The test might use a simple playbook to test a role or a module included in the collection.

```

- name: Testing a custom Ansible Content Collection
 hosts: all

 roles:
 - role: mynamespace.mycollection.my_role

 tasks:
 - name: Ensure my custom module works
 mynamespace.mycollection.my_module:
```

 **Important**

When using automation execution environments, `localhost` is the container itself rather than the machine from which you run the `ansible-navigator` command. Playbooks originally written for the `ansible-playbook` command that target `localhost` might not work as intended.

If you want plays or tasks to target your local machine, then define your local machine in the inventory file and modify the plays and tasks to point to that machine name.

For playbooks that you intend to run locally as well as on automation controller, you can use `hosts: all` to target all inventory hosts. Then you can create an inventory that only contains the desired hosts.

Use the `ansible-navigator` command to run your test playbook using the new automation execution environment.

- Use the `--inventory` (or `-i`) option to specify an inventory file or directory.
- Use the `--execution-environment-image` (or `--eei`) option to specify the automation execution environment container image.
- If the container image exists locally, then use the `--pull-policy never` (or `--pp never`) option. Without that option, and if the image tag is `latest`, then the `ansible-navigator` command attempts to pull the container image from a container registry, such as the Red Hat Ecosystem Catalog or Docker Hub.
- If the playbook requires privilege escalation and does not use `become: true`, then add the `--become` (or `-b`) option to enable privilege escalation.
- Optionally add the `--mode stdout` (or `-m stdout`) option to redirect the playbook output to your terminal window. Alternatively, run the playbook in the default interactive mode.

```
[user@host ~]$ ansible-navigator run test_playbook.yml -i inventory \
> --eei localhost/ee-demo:v2.0 --pp never -b -m stdout
```

If the playbook succeeds, then you can share the automation execution environment. If the playbook does not succeed, then investigate any reported issues.

## Providing Authentication Credentials

When Ansible runs a play on a managed host, it generally must provide authentication credentials to that host to log in and make changes.

For Linux-based managed hosts, one common solution is to install the user account specified by the Ansible `remote_user` directive with the public SSH key that matches a private SSH key available to the automation execution environment.

The `ansible-playbook` command (or `ansible-navigator` with the `--ee false` option) uses the same machine as the control node and automation execution environment. In this mode, Ansible can read the private SSH key directly from your `~/.ssh` directory. If you use automation execution environment containers, however, then you need to provide the private SSH key to the container.

## Automation Controller Machine Credentials

Automation controller can store your private SSH key in a *machine credential*, a special resource that can be updated but not read from the web UI. If you use password-based SSH authentication, it can instead store an SSH username and password. You can provide the machine credential in the playbook's job template so that the credential is automatically passed to the automation execution environment when automation controller runs the playbook. The machine credential can also store a password used for privilege escalation.

## Automation Content Navigator Authentication

Automation content navigator can get your private SSH key from ssh-agent. If you use the default graphical GNOME desktop environment, then ssh-agent starts automatically when you log in. If you log in through a text-based terminal or SSH, you can start ssh-agent with the eval \$(ssh-agent) command. When ssh-agent is running, you can run the ssh-add command to add your private SSH keys to ssh-agent. Automation content navigator provides keys stored by ssh-agent to the automation execution environment.



### Warning

It is possible to put a private SSH key inside the Ansible project and to use the `private_key_file` directive to point Ansible at that private SSH key.

This is not a recommended practice. If you do this, it is too easy to accidentally commit the private SSH key to your version control system. This can be particularly bad if your SSH key is not passphrase protected and the contents of your Git repository or other version control system are publicly visible.

By default, `ansible-navigator` does not prompt you for passwords or other interactive input. This means that normally you cannot use options that prompt you for such things as a connection password (`--ask-pass`), a privilege escalation password (`--ask-become-pass`), or vault passwords (`--ask-vault-pass`). (This design decision is intentional, because automation controller does not support interactive prompting in this manner either.)

To enable the `ansible-navigator` command to support interactive prompts for passwords, you can either use the `--pae false` option, or you can edit your `ansible-navigator.yml` configuration file and add the following section:

```

ansible-navigator:
 playbook-artifact:
 enable: false
```

In addition, you need to use the `--mode stdout` (or `-m stdout`) option with the `ansible-navigator run` command.

## Sharing an Automation Execution Environment from Private Automation Hub

The `ansible-builder` command creates automation execution environment container images on your local system. For automation controller and your users to access these images, push them to a container registry, such as Quay.io or private automation hub.

**Important**

You are not allowed to publicly distribute container images that you build on top of the automation execution environment images that Red Hat provides.

Push a container image to private automation hub by using a procedure such as the following. In this example, private automation hub is accessible at `hub.example.com`.

- List your local container images.

```
[user@host ~]$ podman images
REPOSITORY TAG IMAGE ID CREATED SIZE
localhost/ee-demo v2.0 ba77b0d0a59d 2 minutes ago 308 MB
...output omitted...
```

- Use the `podman tag` command to tag the local container image for your private automation hub. The image name on private automation hub does not have to be the same as the local image. Also, private automation hub organizes the images in namespaces. You can use those namespaces to group your images per team, for example. The `podman push` command that you use in a following step creates the namespace if it does not exist.

```
[user@host ~]$ podman tag localhost/ee-demo:v2.0 \
> hub.example.com/mynamespace/ee-demo:v2.0
```

**Note**

If you do not specify a tag, then the `latest` tag is used. A tag is arbitrary text and can specify a version, a date, an environment, and so on. Examples include `v1.0`, `20220923`, and `devel`.

A container image can have multiple tags, such as `latest`, `v1.0`, and `best`. Add each tag individually.

An individual tag, such as `latest`, can only apply to one container image at a time. Adding a tag name to an image removes that tag name from any other container image with the same name.

Using a tag other than `latest` makes it easier for automation controller to use a specific version of the automation execution environment.

- Use the `podman login` command to log in to private automation hub.

```
[user@host ~]$ podman login hub.example.com
```

- Use the `podman push` command to push the container image to private automation hub.

```
[user@host ~]$ podman push hub.example.com/mynamespace/ee-demo:v2.0
```



## References

`podman-login(1)`, `podman-tag(1)`, and `podman-push(1)` man pages

## Ansible Navigator Documentation

<https://ansible.readthedocs.io/projects/navigator/>

## ► Guided Exercise

# Validating a Custom Execution Environment

Validate that a custom automation execution environment works as expected and then publish it to private automation hub.

### Outcomes

- Use the `ansible-navigator` command to run a playbook using a local execution environment.
- Upload an execution environment to private automation hub.

### Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command installs Ansible Automation Platform and downloads the local execution environment that you need for this lab.

```
[student@workstation ~]$ lab start builder-validate
```

### Instructions

- 1. The `lab start builder-validate` command set up a test playbook, `~/builder-validate/motd-test.yml`, which uses the `exercise.motd.banner` role in the `exercise.motd` Ansible Content Collection. Review the test playbook and the inventory file provided by the `lab` command.
- 1.1. Change into the `~/builder-validate/` directory.

```
[student@workstation ~]$ cd ~/builder-validate/
[student@workstation builder-validate]$
```

- 1.2. Review the contents of the `motd-test.yml` playbook.

```

- name: Test banner role from exercise.motd collection
 hosts: all
 collections:
 - exercise.motd
 roles:
 - name: exercise.motd.banner
```

The playbook calls the `banner` role from the `exercise.motd` content collection. Notice that the single play targets all inventory hosts.

- 1.3. Review the `inventory` file.

```
[servers]
serverc
serverd
```

- 2. Display the existing messages of the day for the `serverc` machine.

- 2.1. Use the `ssh` command to connect to the `serverc` machine as the `student` user.

```
[student@workstation builder-validate]$ ssh student@serverc
Activate the web console with: systemctl enable --now cockpit.socket ①
```

This system is not registered to Red Hat Insights. See <https://cloud.redhat.com/>  
To register this system, run: insights-client --register ②

- ① The `/etc/motd.d/cockpit` file contains the message to activate the web console.
- ② The `/etc/motd.d/insights-client` file contains the Red Hat Insights message.

- 2.2. Use the `logout` command to return to the `workstation` machine.

```
[student@serverc ~]$ logout
Connection to serverc closed.
[student@workstation builder-validate]$
```

- 3. Run the `~/builder-validate/motd-test.yml` playbook. Try the default automation execution environment selected by `ansible-navigator` first, and when that fails, test the custom `ee-motd-minimal` automation execution environment that includes the `exercise.motd` collection. (This custom automation execution environment is the same as the one you created in the preceding guided exercise, but has been prepared by the `lab` command.)

- 3.1. Attempt to run the `motd-test.yml` playbook using the `ansible-navigator` command. The attempt fails because the `exercise.motd.banner` role from the custom `exercise.motd` collection is not installed on the `workstation` machine.

```
[student@workstation builder-validate]$ ansible-navigator run motd-test.yml \
> -m stdout
ERROR! the role 'exercise.motd.banner' was not found in
 exercise.motd:ansible.legacy:/home/student/builder-validate/roles:/home/
 runner/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles:/home/student/
 builder-validate

The error appears to be in '/home/student/builder-validate/motd-test.yml': line 7,
 column 7, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:
```

```
roles:
 - name: exercise.motd.banner
 ^ here
```

- 3.2. List the local container images.

```
[student@workstation builder-validate]$ podman images
REPOSITORY TAG IMAGE ID ...
localhost/ee-motd-minimal latest 77c649cc9312 ...
...output omitted...
```



### Note

Your machine might display additional images, and the image ID for the `localhost/ee-motd-minimal` container image might be different.

- 3.3. Use the `ansible-navigator` command with the following options to run the `motd-test.yml` playbook.

| Option                                       | Purpose                                                                           |
|----------------------------------------------|-----------------------------------------------------------------------------------|
| <code>--eei localhost/ee-motd-minimal</code> | Select the custom container image for the execution environment.                  |
| <code>--pp never</code>                      | Do not attempt to download the container image from a container image repository. |

```
[student@workstation builder-validate]$ ansible-navigator run motd-test.yml \
> --eei localhost/ee-motd-minimal --pp never -m stdout

PLAY [Test banner role from exercise.motd collection] ****
TASK [Gathering Facts] ****
ok: [serverc]
ok: [serverd]

TASK [exercise.motd.banner : Ensure /etc/motd.d/ exists] ****
ok: [serverc]
ok: [serverd]

TASK [exercise.motd.banner : Create custom motd banner] ****
changed: [serverc]
changed: [serverd]

PLAY RECAP ****
serverc : ok=3 changed=1 unreachable=0 failed=0 ...
serverd : ok=3 changed=1 unreachable=0 failed=0 ...
```

- 4. Verify that the playbook run configured the `serverc` machine to include an additional message of the day (MOTD) message.

- 4.1. Use the `ssh` command to connect to the `serverc` machine as the `student` user.

```
[student@workstation builder-validate]$ ssh student@serverc
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
=====
=====
== This system is managed by Ansible. ==
=====
=====
...output omitted...
```

In addition to the previous web console and Red Hat Insights messages, there is a new message indicating that the system is managed by Ansible. This new message comes from the /etc/motd.d/banner file.

- 4.2. Use the `logout` command to return to the workstation machine.

```
[student@serverc ~]$ logout
Connection to serverc closed.
[student@workstation builder-validate]$
```

- ▶ 5. Publish the tested `localhost/ee-motd-minimal` execution environment to the container registry on the classroom private automation hub.
  - 5.1. Use the `podman tag` command to add the `hub.lab.example.com/ee-motd-minimal` label to the `localhost/ee-motd-minimal` image. The command does not produce any output.

```
[student@workstation builder-validate]$ podman tag localhost/ee-motd-minimal \
> hub.lab.example.com/ee-motd-minimal
```

- 5.2. Use the `podman login` command to log in to private automation hub at `hub.lab.example.com`:

```
[student@workstation builder-validate]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 5.3. Push the `hub.lab.example.com/ee-motd-minimal` container image to private automation hub.

```
[student@workstation builder-validate]$ podman push \
> hub.lab.example.com/ee-motd-minimal
...output omitted...
Writing manifest to image destination
Storing signatures
```

**Note**

In some cases, the entire container image is not pushed to private automation hub. If the podman push command produces an error, then try rerunning the command.

- 5.4. Open a web browser and navigate to <https://hub.lab.example.com>. Log in with student as the username and redhat123 as the password.
- 5.5. From the private automation hub web UI, navigate to **Execution Environments > Execution Environments**. The **Execution Environments** page lists the ee-motd-minimal container repository.
- ▶ 6. Verify that the image can be retrieved from the container registry on the classroom private automation hub and that it works.

- 6.1. Display the existing MOTD messages for the servere machine.

```
[student@workstation builder-validate]$ ssh student@servere
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
```

- 6.2. Use the logout command to return to the workstation machine.

```
[student@servere ~]$ logout
Connection to servere closed.
[student@workstation builder-validate]$
```

- 6.3. Add servere to the existing inventory file.

```
[student@workstation builder-validate]$ echo "servere" >> inventory
```

- 6.4. Rerun the motd-test.yml playbook using the ansible-navigator run command and use the following options.

| Option                                    | Purpose                                                                  |
|-------------------------------------------|--------------------------------------------------------------------------|
| --eei hub.lab.example.com/ee-motd-minimal | Select the custom container image for the execution environment.         |
| --pp always                               | Always download the container image from the container image repository. |

```
[student@workstation builder-validate]$ ansible-navigator run motd-test.yml \
> --eei hub.lab.example.com/ee-motd-minimal --pp always -m stdout

Execution environment image and pull policy overview

Execution environment image name: hub.lab.example.com/ee-motd-minimal:latest
```

```
Execution environment image tag: latest
Execution environment pull arguments: None
Execution environment pull policy: always
Execution environment pull needed: True

Updating the execution environment

Running the command: podman pull hub.lab.example.com/ee-motd-minimal:latest
Trying to pull hub.lab.example.com/ee-motd-minimal:latest...
Getting image source signatures
Copying blob c64247c247c3 skipped: already exists
...output omitted...
Writing manifest to image destination
Storing signatures
778e5596d50c79db2d2e67e7534b7c37ba48a8c8afa6bc7a375bb6b700f91379
...output omitted...
```

- 6.5. Verify that the **servere** machine now contains the additional MOTD message.

```
[student@workstation builder-validate]$ ssh student@servere
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
=====
=====
== ==
== This system is managed by Ansible. ==
== ==
=====
=====
...output omitted...
```

- 6.6. Use the **logout** command to return to the **workstation** machine.

```
[student@servere ~]$ logout
Connection to servere closed.
[student@workstation builder-validate]$
```

## Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish builder-validate
```

# Using Custom Content Collections and Execution Environments in Automation Controller

## Objectives

- Run a playbook in automation controller that uses an Ansible Content Collection in the Ansible project or provided by a specific automation execution environment.

## Using Custom Collections with Existing Execution Environments

You might want to run an Ansible Playbook that requires an Ansible Content Collection that is not included in an existing automation execution environment. In some cases, one of your automation execution environments might already provide all the resources needed by the collection, other than the collection itself.

If so, then you can configure automation controller to automatically download the collection into the automation execution environment when it runs your playbook.

To configure this, you need to do two things:

- Set up a `collections/requirements.yml` file in your Ansible project, which specifies the additional collections that your playbook needs.
- Configure automation controller with the authentication credentials it needs to access the Ansible automation hub or private automation hub that stores those collections.

## Preparing Ansible Projects for Automation Controller

When you use the `ansible-navigator run` command to run a playbook that requires Ansible Content Collections that are not in the automation execution environment, you must manually install the collections. Usually, the project provides a `collections/requirements.yml` file that you can process with the `ansible-galaxy collection install -r collections/requirements.yml` command to do this.

Automation controller does not require you to manually run this command. Instead, it automatically runs the following commands to install collections and roles when you run a playbook:

- `ansible-galaxy collection install -r collections/requirements.yml`
- `ansible-galaxy role install -r roles/requirements.yml`

The `collections/requirements.yml` file that you provide in your Ansible project might point to collections available on Ansible automation hub or private automation hub.

For example, the following `collections/requirements.yml` file lists two collections. Ansible automation hub hosts the `redhat.insights` collection, and private automation hub at `hub.example.com` hosts the `ansible.posix` collection.

```

collections:
 - name: redhat.insights
 source: https://console.redhat.com/api/automation-hub/

 - name: ansible.posix
 source: https://hub.example.com/api/galaxy/content/rh-certified/

```

## Storing Authentication Credentials for Collections

Ansible automation hub and private automation hubs require login credentials. You need to configure automation controller with a credential that provides the automation hub API token and URLs for Ansible automation hub and for any private automation hubs that your playbook uses to download collections.

To perform that configuration, follow these steps:

- Log in to the automation controller web UI.
- Navigate to **Resources > Credentials** and click **Add**.
- Select the **Ansible Galaxy/Automation Hub API Token** credential type and then enter the Ansible automation hub URL and your token. You can obtain the Ansible automation hub URL and your token at <https://console.redhat.com/ansible/automation-hub/token>.

The screenshot shows the 'Create New Credential' interface in the Red Hat Ansible Automation Platform. The 'Credential Type' dropdown is highlighted with a red box. The form includes fields for Name (Ansible automation hub), Description (Certified Ansible Content Collections), Organization (Default), and Type Details (Galaxy Server URL: https://console.redhat.com/api/automation-hub, Auth Server URL: edhat-external/protocol/openid-connect/token, API Token: masked), along with Save and Cancel buttons.

**Figure 9.2: Creating a credential for accessing Ansible automation hub**

- Click **Save**.

## Chapter 9 | Creating Content Collections and Execution Environments

- Associate the credential with all the organizations in automation controller that need access to Ansible automation hub. To do so, navigate to Access > Organizations and select an organization.
- Click Edit and then add the credential in the Galaxy Credentials field, in the order you want to search the platforms for collections.

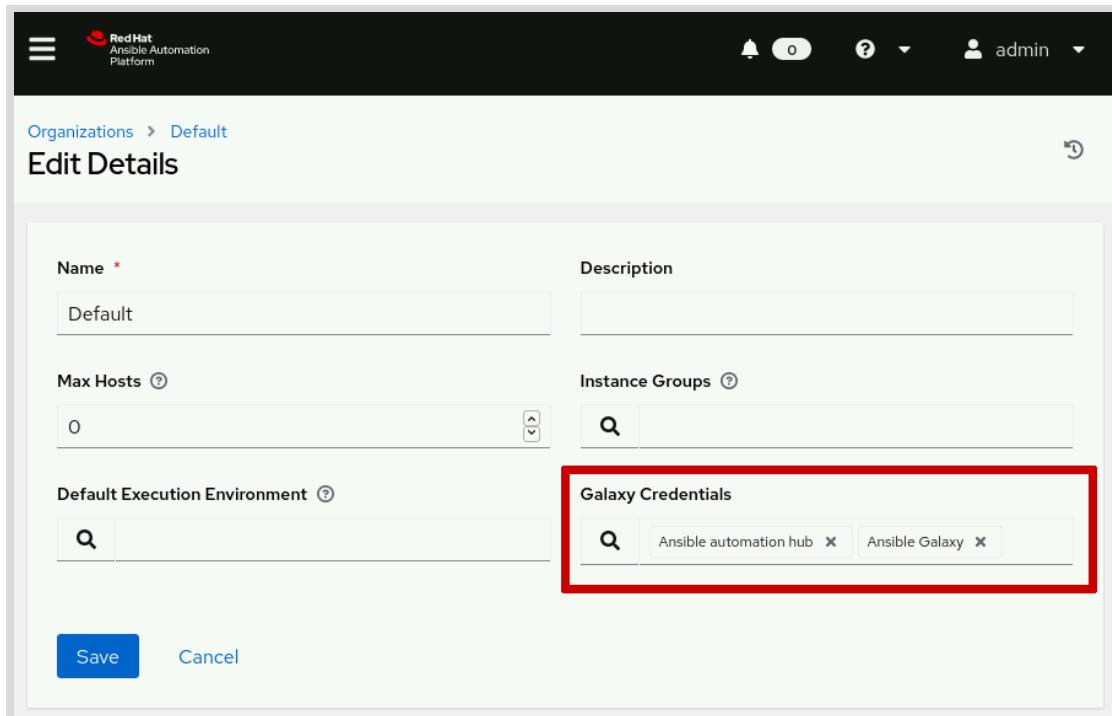


Figure 9.3: Associating Ansible automation hub credentials with organizations

- Click Save.
- You might need to repeat the process for your private automation hub.



### Important

If you installed both automation controller and private automation hub at the same time, then the installation script creates the following credential resources:

- Automation Hub Community Repository
- Automation Hub Published Repository
- Automation Hub RH Certified Repository

These credentials use the API token for the private automation hub admin user. Because loading a new token in the private automation hub web UI deletes the previous token, if you load a new API token for the private automation hub admin user, then you must update these automation controller credentials with the new API token.

## Using Custom Automation Execution Environments with Automation Controller

Automation controller uses an automation execution environment to run playbooks. By default, automation controller already includes the supported automation execution environments from Red Hat.

You can use your own custom automation execution environments as well. You can configure automation controller to download the container images for these execution environments automatically.

To implement this, perform the following:

- Configure automation controller with a credential that it can use to authenticate to the container registry that contains your execution environment's container image.
- Configure automation controller with information about what container images to pull from where, and how often to pull them.
- *(Optional)* Specify a default execution environment for automation controller to use when running playbooks from a particular project.
- *(Optional)* Specify an execution environment to use when running a playbook from a particular template, if you do not want the template to use the project's default execution environment.

## Storing Container Registry Credentials

Automation controller automatically downloads the container images for its automation execution environments from a container registry. This could be a container registry built into automation hub, or one provided by a service such as Quay.

Many container registries require authentication for access. You need to configure automation controller with a credential to store this authentication information.

The following steps show how to create a credential for a container registry:

- Log in to the automation controller web UI.
- Navigate to Resources > **Credentials** and then click **Add**.
- Select the **Container Registry** credential type and then enter the URL of the container registry, your username, and your password. The following screen capture shows how to configure a credential for the container registry of private automation hub at `hub.example.com`.

The screenshot shows the 'Create New Credential' page in the Red Hat Ansible Automation Platform. The 'Name' field contains 'privhub'. The 'Description' field contains 'My private automation hub'. The 'Organization' field has a search icon. The 'Credential Type' field is highlighted with a red box and contains 'Container Registry'. In the 'Type Details' section, the 'Authentication URL' is 'https://hub.example.com', the 'Username' is 'starango', and the 'Password or Token' field shows '\*\*\*\*\*'. Under 'Options', the 'Verify SSL' checkbox is checked. At the bottom are 'Save' and 'Cancel' buttons.

**Figure 9.4: Creating a credential for accessing a container registry**

- Click Save.



### Important

If you installed both automation controller and private automation hub at the same time, then the installation script creates the automation controller **Automation Hub Container Registry** credential. This credential uses the private automation hub **admin** user and the password for that user specified during installation.

If you update the password for the private automation hub **admin** user, then you must update the automation controller **Automation Hub Container Registry** credential as well.

## Configuring Automation Execution Environments

You can configure your automation execution environment images as follows:

- Navigate to **Administration > Execution Environments** and then click **Add**.
- Complete the form.

| Field               | Description                                                                              |
|---------------------|------------------------------------------------------------------------------------------|
| Name                | Specify a name to use in automation controller for the automation execution environment. |
| Image               | Specify the container image for the automation execution environment.                    |
| Pull                | Select how automation controller should pull the container image.                        |
| Registry credential | Select an existing credential for the container registry.                                |

The screenshot shows the 'Edit details' page for an execution environment named 'Demo EE'. The 'Name' field contains 'Demo EE'. The 'Image' field contains 'hub.example.com/mynamespace/ee-demo:v2.0.0'. The 'Pull' dropdown is set to 'Only pull the image if not present before running.' The 'Description' field contains 'Test automation execution environment container i ...'. The 'Organization' section is empty. The 'Registry credential' field contains 'privhub' and is highlighted with a red box. At the bottom, there are 'Save' and 'Cancel' buttons.

Figure 9.5: Configuring an automation execution environment

- Click Save.



### Important

If you do not specify a tag for the image, then the tag defaults to `latest`.

Using the `latest` tag might be problematic. Suppose, for example, that a developer creates a new version of an execution environment but forgets to assign a tag before pushing it to the container registry. The `latest` tag is then automatically moved to that new, untested image. All your jobs in automation controller start by using that image, which can have severe consequences if the image does not work correctly.

## Configuring the Default Automation Execution Environment for a Project

When you set up your Ansible project in automation controller, you can specify a default automation execution environment for playbooks from that project to use.

Define the default execution environment for your project as follows:

- Navigate to **Resources > Projects** and then click **Add**.
- Complete the form to configure your project. Select the automation execution environment to use by default in the **Execution Environment** field.

The screenshot shows the 'Create New Project' page in the Red Hat Ansible Automation Platform. The 'Execution Environment' field is highlighted with a red box. The 'Name' field contains 'My project', the 'Description' field contains 'Testing my new automation execution environment', the 'Organization' field contains 'Default', and the 'Source Control Type' field contains 'Git'. In the 'Type Details' section, the 'Source Control URL' field contains 'git@git.lab.example.com:dev/new-ee-test.git', the 'Source Control Branch/Tag/Commit' field is empty, the 'Source Control Refspec' field is empty, and the 'Source Control Credential' field contains 'mygit'.

**Figure 9.6: Selecting an automation execution environment for a project**

- Click **Save**.

## Specifying an Automation Execution Environment in a Template

A template specifies all the parameters that automation controller needs to run a playbook, including the project containing the playbook, the inventory to use, and machine credentials needed to access the managed nodes in its plays.

When you set up a job template to run a playbook, you can also select a specific execution environment to use when running the playbook. This overrides the default execution environment specified by the playbook's project.

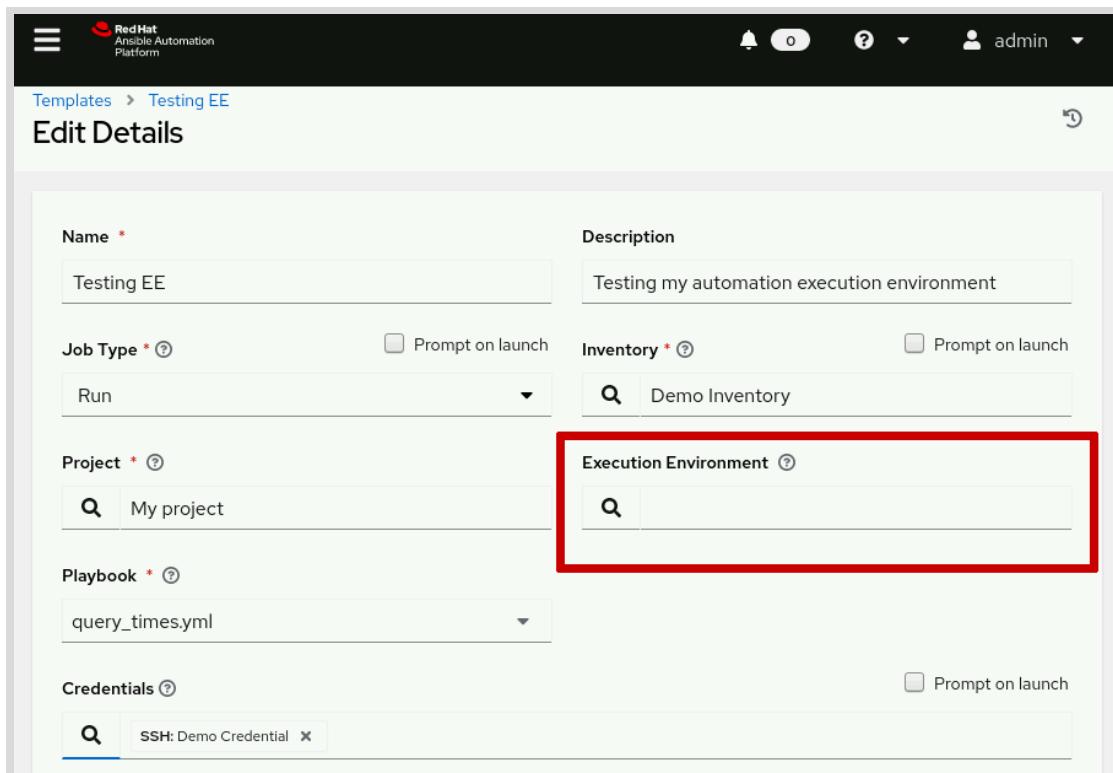


Figure 9.7: Selecting an automation execution environment in a template

If you leave the Execution Environment field empty, then automation controller uses the default automation execution environment from the playbook's project.

After creating the template, click **Launch** to start a job (run the playbook). The first time you launch a job that uses a new automation execution environment, the job might take longer than later runs. Automation controller must download the automation execution environment's container image if it is not present.



## References

### What's New in Ansible Automation Platform 2.2

<https://www.ansible.com/blog/whats-new-in-ansible-automation-platform-2.2>

### Automation Controller User Guide

<https://docs.ansible.com/automation-controller/4.2.1/html/userguide/index.html>

## ► Guided Exercise

# Using Custom Content Collections and Execution Environments in Automation Controller

Run a playbook in automation controller that retrieves an Ansible Content Collection by using a requirements file and that uses content in a custom Ansible Content Collection.

## Outcomes

- Run a playbook on automation controller that uses a customized collection stored on a private automation hub.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/builder-use` directory and configures automation controller with the following resources:

- The `Exercise Credential` machine credential.
- The `Git Project Credential` source control credential.
- The `Exercise Inventory` resource.

The command also requests a new API token for the private automation hub `admin` user and updates the relevant automation controller credentials with the new token.

```
[student@workstation ~]$ lab start builder-use
```

## Instructions

- 1. Publish the provided Ansible Content Collection on private automation hub.
- 1.1. Navigate to `https://hub.lab.example.com` and log in as the student user with `redhat123` as the password.
  - 1.2. From the private automation hub web UI, navigate to `Collections > Namespaces` and then click `Create`.
  - 1.3. On the `Create new namespace` page, complete the details as follows and click `Create` to create the namespace.

| Field            | Value              |
|------------------|--------------------|
| Name             | exercise           |
| Namespace owners | Content Developers |

**Important**

The Content Developers group must be a namespace owner for group members, such as the student user, to upload to the namespace.

- 1.4. Click **Upload collection** and then **Select file**.
- 1.5. In the UI, browse to the /home/student/builder-use directory, select the `exercise-motd-1.0.0.tar.gz` file, and click **Upload**.

The upload produces the following messages:

```
Importing with galaxy-importer 0.4.4
Getting doc strings via ansible-doc
Finding content inside collection
Loading role banner
Linting role banner via ansible-lint...
CHANGELOG.rst file not found at top level of collection.
Collection loading complete
```

Done

**Note**

A collection can include an optional `CHANGELOG.rst` file. It is safe to ignore the message about not being able to find the `CHANGELOG.rst` file.

- 1.6. After the upload completes, navigate to **Collections > Approval** and then click **Approve**.
- ▶ 2. Create a project from the automation controller web UI.
  - 2.1. Navigate to `https://controller.lab.example.com` and log in as the `admin` user with `redhat` as the password.
  - 2.2. Navigate to **Resources > Projects** and then click **Add**.
  - 2.3. Add a new project that installs the `exercise.motd` collection using a requirements file located in the project. Use the following information for the project and then click **Save**.

| Field                     | Value                                                           |
|---------------------------|-----------------------------------------------------------------|
| Name                      | Install Collection                                              |
| Organization              | Default                                                         |
| Source Control Type       | Git                                                             |
| Source Control URL        | <code>git@git.lab.example.com:student/use-collection.git</code> |
| Source Control Credential | Git Project Credential                                          |

2.4. Wait for the synchronization to complete.



### Note

If the synchronization fails, then ensure that you entered the correct source control URL and that the uploaded `motd` collection has been approved from the private automation hub web UI. Attempt the project synchronization again.

2.5. From the **Details** page, click **Successful** to display details about the project synchronization. Notice that the job installed collections found in the `collections/requirements.yml` file.

```
...output omitted...
TASK [fetch galaxy collections from collections/requirements.(yml/yaml)] *****
changed: [localhost] => (item=/var/lib/awx/projects/_8__install_collection/
collections/requirements.yml)
PLAY RECAP *****
localhost : ok=4 changed=2 ... failed=0 skipped=2 ...
```

2.6. *(Optional)* Locate the installed `exercise.motd` collection on the automation controller server. Open a terminal and use the `ssh` command to connect to the `controller` machine as the `awx` user. Display the contents of the `/var/lib/awx/projects/.__awx_cache` directory.

```
[student@workstation ~]$ ssh awx@controller \
> 'tree /var/lib/awx/projects/.__awx_cache'
/var/lib/awx/projects/.__awx_cache
└── _8__install_collection ①
 └── 3 ②
 ├── requirements_collections
 │ └── ansible_collections ③
 │ ├── exercise ④
 │ │ └── motd ⑤
 │ │ ├── FILES.json
 │ │ └── MANIFEST.json
...output omitted...
```

- ① Each automation controller project has a unique identification number associated with it. In this example, the `Install Collection` project has project ID 8. Your project ID might be different.
- ② This numbered directory represents the job ID of the automation controller job that performed the project synchronization. Your job ID might be different.
- ③ The `ansible_collections` directory contains any collections installed during the project synchronization.
- ④ The `exercise` director contains installed collections from the `exercise` namespace.
- ⑤ The `motd` directory contains files and directories from the `exercise.motd` collection.

**Note**

This step is for information purposes only. Most developers do not have access to view the `/var/lib/awx/projects/.__awx_cache` directory on the automation controller server.

- ▶ 3. Display the existing messages of the day for the `servera` and `serverb` machines. The machines do not display custom messages of the day.

- 3.1. Use the `ssh` command to connect to the `servera` machine as the `student` user.

```
[student@workstation ~]$ ssh student@servera
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
```

- 3.2. Use the `logout` command to return to the `workstation` machine.

```
[student@servera ~]$ logout
Connection to servera closed.
[student@workstation ~]$
```

- 3.3. Use the `ssh` command to connect to the `serverb` machine as the `student` user.

```
[student@workstation ~]$ ssh student@serverb
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
```

- 3.4. Use the `logout` command to return to the `workstation` machine.

```
[student@serverb ~]$ logout
Connection to serverb closed.
[student@workstation ~]$
```

- ▶ 4. Create a job template to run the `motd-test.yml` playbook from the `Install Collection` project.

- 4.1. From the automation controller web UI, navigate to **Resources > Templates**.

- 4.2. On the **Templates** page, click **Add > Add job template**. Create a job template using the following information and then click **Save**.

| Field                 | Value                                        |
|-----------------------|----------------------------------------------|
| Name                  | MOTD Test 1                                  |
| Description           | Test using installed collection              |
| Inventory             | Exercise Inventory                           |
| Project               | Install Collection                           |
| Execution Environment | Automation Hub Minimal execution environment |
| Playbook              | motd-test.yml                                |
| Credentials           | Exercise Credential                          |
| Limit                 | servera.lab.example.com                      |

- 4.3. Click Launch to launch the MOTD Test 1 job template. The job succeeds and the output indicates that servera.lab.example.com was changed.

```
...output omitted...
TASK [exercise.motd.banner : Create custom motd banner] *****
changed: [servera.lab.example.com]
...output omitted...
```

- ▶ 5. Verify that the servera machine contains an additional message of the day.

- 5.1. Use the ssh command to connect to the servera machine as the student user. The system displays a message indicating that it is managed by Ansible.

```
[student@workstation ~]$ ssh student@servera
...output omitted...
=====
=====
== ==
== This system is managed by Ansible. ==
== ==
=====
=====
...output omitted...
```

- 5.2. Use the logout command to return to the workstation machine.

```
[student@servera ~]$ logout
Connection to servera closed.
[student@workstation ~]$
```

- ▶ 6. Add a new execution environment resource using the automation controller web UI.

- 6.1. Navigate to Administration > Execution Environments and then click Add.

- 6.2. Create an execution environment resource using the following information, and then click **Save**:

| Field               | Value                                              |
|---------------------|----------------------------------------------------|
| Name                | MOTD execution environment                         |
| Image               | hub.lab.example.com/ee-motd-minimal                |
| Pull                | Only pull the image if not present before running. |
| Organization        | Default                                            |
| Registry credential | Automation Hub Container Registry                  |

- 7. Add a new automation controller project that uses the `exercise.motd` collection installed within a custom automation execution environment.

- 7.1. Navigate to **Resources > Projects** and then click **Add**. Create a project resource using the following information, and then click **Save**:

| Field                     | Value                                      |
|---------------------------|--------------------------------------------|
| Name                      | Use Custom EE                              |
| Organization              | Default                                    |
| Source Control Type       | Git                                        |
| Source Control URL        | git@git.lab.example.com:student/use-ee.git |
| Source Control Credential | Git Project Credential                     |

- 7.2. Wait for the synchronization to complete.



#### Note

If the synchronization fails, then ensure that you entered the correct source control URL and attempt the project synchronization again.

- 8. Create a job template to run the `motd-test.yml` playbook from the `Use Custom EE` project.

- 8.1. Navigate to **Resources > Templates** and then click **Add > Add job template**. Create the job template using the following information:

| Field                 | Value                      |
|-----------------------|----------------------------|
| Name                  | MOTD Test 2                |
| Description           | Test using custom EE       |
| Inventory             | Exercise Inventory         |
| Project               | Use Custom EE              |
| Execution Environment | MOTD execution environment |
| Playbook              | motd-test.yml              |
| Credentials           | Exercise Credential        |
| Variables             | message: "My test message" |
| Limit                 | serverb.lab.example.com    |

**Note**

The job template selects a specific automation execution environment and only uses one host from the **Exercise Inventory** resource. The job template overrides the value of the **message** variable set in the **exercise.motd.banner** role.

8.2. Click **Save**.

8.3. Click **Launch** to launch the MOTD Test 2 job template. The job succeeds and the output indicates that **serverb.lab.example.com** was changed.

```
...output omitted...
TASK [exercise.motd.banner : Create custom motd banner] ****
changed: [serverb.lab.example.com]
...output omitted...
```

► 9. Verify that the **serverb** machine contains an additional message of the day.

9.1. Use the **ssh** command to connect to the **serverb** machine as the **student** user. The system displays a message using the value of the **message** variable configured in the MOTD Test 2 job template.

```
[student@workstation ~]$ ssh student@serverb
...output omitted...
=====
=====
== ==
== My test message
== ==
=====
=====
...output omitted...
```

- 9.2. Use the `logout` command to return to the `workstation` machine.

```
[student@serverb ~]$ logout
Connection to serverb closed.
[student@workstation ~]$
```

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish builder-use
```

## ▶ Lab

# Creating Content Collections and Execution Environments

Create an Ansible Content Collection, upload it to private automation hub, create an automation execution environment that includes the collection, and then use that automation execution environment to run a playbook from automation controller.

## Outcomes

- Create an Ansible Content Collection that includes roles and plug-ins.
- Create a custom automation execution environment.
- Upload an Ansible Content Collection and an automation execution environment to private automation hub.
- Configure automation controller to use a custom automation execution environment from private automation hub.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/create-review` directory, prepares automation controller and private automation hub for the exercise, and creates a Git repository that includes a playbook to test your work. The command also provides the `lab-resources.txt` file that lists all the resources that you configure during the exercise. You can use that file to copy and then paste those resources.

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

## Instructions

You can access the private automation hub web UI at <https://hub.lab.example.com> using `student` as the username and `redhat123` as the password. You can access the automation controller web UI at <https://controller.lab.example.com> using `admin` as the username and `redhat` as the password.

1. On the workstation machine, in the `/home/student/create-review` directory, create an Ansible Content Collection named `training.web` that includes two roles and a lookup plug-in.

The two roles are available in the `/home/student/create-review/roles/` directory. You have to copy them to the correct location in the collection after you create the skeleton directories for the collection.

The `firewall` role depends on the `ansible.posix` collection (version `>=1.0.0`). Configure your collection to specify this dependency.

The lookup plug-in creates a QR (Quick Response) code in PNG (Portable Network Graphics) format. It is available in the `/home/student/create-review/lookup.tgz`

file. Extract that archive into the correct location in your collection. It also requires the `pillow` and `qrcode` Python packages. Configure your collection to specify those two Python dependencies.

The collection requires Ansible version 2.9.10 or later.

Build the collection and then copy the resulting `.tar.gz` file to the `/home/student/create-review/` directory.

**2.** Publish the `training.web` collection to your private automation hub.

Use the private automation hub web UI to create a collection namespace named `training` and configure the `Content Developers` group to be namespace owners of the collection.

Upload the `training.web` collection to that namespace, and approve the collection after you upload it.

**3.** Install the `ansible-builder` package on the `workstation` machine. The `student` user can run commands with escalated privileges. Use `student` as the `sudo` password.

**4.** Prepare the configuration files to build a custom automation execution environment so that the `training.web` collection is included in it.

- On `workstation`, create the `~/create-review/ee-build` directory to store the configuration files.
- Create an `ee-build/execution-environment.yml` file to control the build process. An example file is available in the `/home/student/create-review/` directory. If you use that example file, then you must remove the parameters that you do not need.
  - Set the `hub.lab.example.com/ee-minimal-rhel8:latest` container image as the base image.
  - Set the `hub.lab.example.com/ansible-builder-rhel8:latest` container image as the builder image.
- Copy the `~/create-review/ansible.cfg` file to the `~/create-review/ee-build/` directory. That file configures access to the private automation hub at `https://hub.lab.example.com` so that the build process can retrieve the `training.web` and the `ansible.posix` collections.
- Get a token from the private automation hub web UI, and edit that `ansible.cfg` file to update both `token` parameters. Use the same token from private automation hub for both parameters.
- Create the `ee-build/requirements.yml` file and configure it to install the `training.web` collection into the automation execution environment.

**5.** Configure the container image to include your private automation hub's TLS CA certificate.

In the `~/create-review/ee-build/` directory, use the `ansible-builder` command to create the `ee-build/context/` directory by performing the first stage of the build process.

Add the lab environment's TLS CA certificate to the container image. This enables the build process to retrieve collections from the lab environment's private automation hub. You have been provided with two files to help you do this, which should be used as follows:

- Copy `~/create-review/Containerfile` to `~/create-review/ee-build/context/Containerfile`.
- Copy `/etc/pki/tls/certs/classroom-ca.pem` to `~/create-review/ee-build/context/classroom-ca.pem`.

6. Build the automation execution environment container image and use `hub.lab.example.com/review/ee-training-rhel8:v1.0` as the tag. Push the container image to the `hub.lab.example.com` container registry.

**Important**

You must log in to the `hub.lab.example.com` container registry to download and push container images. Log in as student using `redhat123` as the password.

7. Use the automation controller web UI to create an execution environment resource that uses the following settings:

| Field               | Value                                                          |
|---------------------|----------------------------------------------------------------|
| Name                | Training Review                                                |
| Image               | <code>hub.lab.example.com/review/ee-training-rhel8:v1.0</code> |
| Pull                | Always pull the container before running.                      |
| Organization        | Default                                                        |
| Registry credential | Automation Hub Container Registry                              |

8. Create an automation controller job template that uses the following settings:

| Field                 | Value                 |
|-----------------------|-----------------------|
| Name                  | Deploy QR Codes       |
| Inventory             | Development           |
| Project               | QR Codes              |
| Execution environment | Training Review       |
| Playbook              | <code>test.yml</code> |
| Credentials           | Developers            |

**Note**

The `lab` command created the `Development` inventory, the `QR Codes` project, and the `Developers` machine credential.

9. Start a job that uses the `Deploy QR Codes` job template and verify that the job completes successfully. If successful, then navigating to `http://serverf.lab.example.com` displays a QR code image.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

## Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

## ► Solution

# Creating Content Collections and Execution Environments

Create an Ansible Content Collection, upload it to private automation hub, create an automation execution environment that includes the collection, and then use that automation execution environment to run a playbook from automation controller.

## Outcomes

- Create an Ansible Content Collection that includes roles and plug-ins.
- Create a custom automation execution environment.
- Upload an Ansible Content Collection and an automation execution environment to private automation hub.
- Configure automation controller to use a custom automation execution environment from private automation hub.

## Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/create-review` directory, prepares automation controller and private automation hub for the exercise, and creates a Git repository that includes a playbook to test your work. The command also provides the `lab-resources.txt` file that lists all the resources that you configure during the exercise. You can use that file to copy and then paste those resources.

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

## Instructions

You can access the private automation hub web UI at <https://hub.lab.example.com> using `student` as the username and `redhat123` as the password. You can access the automation controller web UI at <https://controller.lab.example.com> using `admin` as the username and `redhat` as the password.

1. On the workstation machine, in the `/home/student/create-review` directory, create an Ansible Content Collection named `training.web` that includes two roles and a lookup plug-in.

The two roles are available in the `/home/student/create-review/roles/` directory. You have to copy them to the correct location in the collection after you create the skeleton directories for the collection.

The `firewall` role depends on the `ansible.posix` collection (version `>=1.0.0`). Configure your collection to specify this dependency.

The lookup plug-in creates a QR (Quick Response) code in PNG (Portable Network Graphics) format. It is available in the `/home/student/create-review/lookup.tgz`

file. Extract that archive into the correct location in your collection. It also requires the `pillow` and `qrcode` Python packages. Configure your collection to specify those two Python dependencies.

The collection requires Ansible version 2.9.10 or later.

Build the collection and then copy the resulting `.tar.gz` file to the `/home/student/create-review/` directory.

- 1.1. Change to the `/home/student/create-review/` directory and then run the `ansible-galaxy collection init` command to create the collection:

```
[student@workstation ~]$ cd ~/create-review/
[student@workstation create-review]$ ansible-galaxy collection init training.web
- Collection training.web was created successfully
```

- 1.2. Copy the two roles in the `~/create-review/roles/` directory to the `~/create-review/training/web/roles/` directory:

```
[student@workstation create-review]$ ls roles/
apache firewall
[student@workstation create-review]$ cp -r roles/* training/web/roles/
```

- 1.3. Edit the `training/web/galaxy.yml` file to declare the `ansible.posix` collection as a required dependency of your new collection:

```
...output omitted...
Collections that this collection requires to be installed for it to be usable.
The key of the dict is the
collection label 'namespace.name'. The value is a version range
L(specifiers,https://python-semanticversion.readthedocs.io/en/latest/
#requirement-specification). Multiple version
range specifiers can be set and are separated by ','
dependencies:
 ansible.posix: '>=1.0.0'
...output omitted...
```

- 1.4. In the `~/create-review` directory, extract the `lookup.tgz` file containing the lookup plug-in into the `training/web/plugins/` directory:

```
[student@workstation create-review]$ tar xf lookup.tgz -C training/web/plugins/
```

- 1.5. To declare the Python package dependencies, create the `training/web/requirements.txt` file with the following content:

```
pillow
qrcode
```

- 1.6. Create the `~/create-review/training/web/meta/` directory.

```
[student@workstation create-review]$ mkdir training/web/meta
```

**Chapter 9 |** Creating Content Collections and Execution Environments

- 1.7. Create the `~/create-review/training/web/meta/runtime.yml` file with the following content:

```

requires_ansible: '>=2.9.10'
```

- 1.8. In the `~/create-review/training/web` directory, build the collection.

```
[student@workstation create-review]$ cd training/web
[student@workstation web]$ ansible-galaxy collection build
Created collection for training.web at /home/student/create-review/training/web/
training-web-1.0.0.tar.gz
```

- 1.9. Copy the resulting `training-web-1.0.0.tar.gz` file from the `~/create-review/training/web` directory to the `/home/student/create-review` directory.

```
[student@workstation web]$ cp training-web-1.0.0.tar.gz ~/create-review/
```

2. Publish the `training.web` collection to your private automation hub.

Use the private automation hub web UI to create a collection namespace named `training` and configure the `Content Developers` group to be namespace owners of the collection.

Upload the `training.web` collection to that namespace, and approve the collection after you upload it.

- 2.1. Navigate to `https://hub.lab.example.com` and log in as `student` using `redhat123` as the password.

- 2.2. Navigate to **Collections > Namespaces** and click **Create**.

Create the namespace using the following information and then click **Create**.

| Field            | Value              |
|------------------|--------------------|
| Name             | training           |
| Namespace owners | Content Developers |

- 2.3. Upload the collection. Click **Upload collection**, select the `/home/student/create-review/training-web-1.0.0.tar.gz` file, and then click **Upload**. Wait for the upload to complete.

- 2.4. Approve the collection. Navigate to **Collections > Approval** and then click **Approve**.

- 2.5. Confirm that the collection is published. Navigate to **Collections > Collections** and then select **Published in Filter by repository**. The web UI displays the web collection.

Click `web` and then click the `Contents` tab. Confirm that the page lists the two roles, `apache` and `firewall`, and the `qr code` lookup plug-in.

3. Install the `ansible-builder` package on the `workstation` machine. The `student` user can run commands with escalated privileges. Use `student` as the `sudo` password.

```
[student@workstation web]$ sudo yum install ansible-builder
[sudo] password for student: student
...output omitted...
Is this ok [y/N]: y
...output omitted...
```

4. Prepare the configuration files to build a custom automation execution environment so that the `training.web` collection is included in it.
- On `workstation`, create the `~/create-review/ee-build` directory to store the configuration files.
  - Create an `ee-build/execution-environment.yml` file to control the build process. An example file is available in the `/home/student/create-review/` directory. If you use that example file, then you must remove the parameters that you do not need.
    - Set the `hub.lab.example.com/ee-minimal-rhel8:latest` container image as the base image.
    - Set the `hub.lab.example.com/ansible-builder-rhel8:latest` container image as the builder image.
  - Copy the `~/create-review/ansible.cfg` file to the `~/create-review/ee-build/` directory. That file configures access to the private automation hub at `https://hub.lab.example.com` so that the build process can retrieve the `training.web` and the `ansible.posix` collections.
  - Get a token from the private automation hub web UI, and edit that `ansible.cfg` file to update both `token` parameters. Use the same token from private automation hub for both parameters.
  - Create the `ee-build/requirements.yml` file and configure it to install the `training.web` collection into the automation execution environment.

#### 4.1. Change to the `~/create-review` directory and create the `ee-build/` directory.

```
[student@workstation web]$ cd ~/create-review/
[student@workstation create-review]$ mkdir ee-build
```

#### 4.2. Copy the `execution-environment.yml` example file into the `ee-build/` directory.

```
[student@workstation create-review]$ cp execution-environment.yml ee-build/
```

#### 4.3. Edit the `ee-build/execution-environment.yml` file.

Set the `EE_BASE_IMAGE` parameter to `hub.lab.example.com/ee-minimal-rhel8:latest` and the `EE_BUILDER_IMAGE` parameter to `hub.lab.example.com/ansible-builder-rhel8:latest`.

Remove the `python` and `system` parameters from the `dependencies` section. The `training.web` collection already provides the list of required Python packages and it does not depend on any RPM packages.

The completed `ee-build/execution-environment.yml` file should consist of the following content:

```

version: 1

build_arg_defaults:
 EE_BASE_IMAGE: hub.lab.example.com/ee-minimal-rhel8:latest
 EE_BUILDER_IMAGE: hub.lab.example.com/ansible-builder-rhel8:latest

ansible_config: ansible.cfg

dependencies:
 galaxy: requirements.yml
```

- 4.4. Copy the `~/create-review/ansible.cfg` file into the `ee-build/` directory.

```
[student@workstation create-review]$ cd ee-build
[student@workstation ee-build]$ cp ~/create-review/ansible.cfg .
```

You update the `token` parameters in that file in the next step.

- 4.5. Retrieve the API token from the private automation hub web UI at <https://hub.lab.example.com>.

Navigate to **Collections > API token management**, click **Load token**, and then click the **Copy to clipboard** icon.

Edit the `~/create-review/ee-build/ansible.cfg` file to update the two `token` parameter lines by replacing each `#FIXME#` string with the copied token. Your token is probably different from the one that is displayed in the following example.

```
[galaxy]
server_list = published_repo, rh-certified_repo

Published collections (for training.web)
[galaxy_server.published_repo]
url=https://hub.lab.example.com/api/galaxy/content/published/
token=c6aec560d9d0a8006dc6d8f258092e09a53fd7bd

Certified collections (for ansible.posix)
[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=c6aec560d9d0a8006dc6d8f258092e09a53fd7bd
```

- 4.6. In the `~/create-review/ee-build/` directory, create the `requirements.yml` file to list the `training.web` collection that you want in the new automation execution environment. The completed `ee-build/requirements.yml` file should consist of the following content:

```

collections:
 - name: training.web
```

5. Configure the container image to include your private automation hub's TLS CA certificate.

In the `~/create-review/ee-build/` directory, use the `ansible-builder` command to create the `ee-build/context/` directory by performing the first stage of the build process.

Add the lab environment's TLS CA certificate to the container image. This enables the build process to retrieve collections from the lab environment's private automation hub. You have been provided with two files to help you do this, which should be used as follows:

- Copy `~/create-review/Containerfile` to `~/create-review/ee-build/context/Containerfile`.
- Copy `/etc/pki/tls/certs/classroom-ca.pem` to `~/create-review/ee-build/context/classroom-ca.pem`.

5.1. Run the `ansible-builder create` command from the `ee-build/` directory:

```
[student@workstation ee-build]$ ansible-builder create
Complete! The build context can be found at: /home/student/create-review/ee-build/
context
```

5.2. Copy the `/home/student/create-review/Containerfile` and `/etc/pki/tls/certs/classroom-ca.pem` files into the `context/` directory.

```
[student@workstation ee-build]$ cp ~/create-review/Containerfile context/
[student@workstation ee-build]$ cp /etc/pki/tls/certs/classroom-ca.pem context/
```

6. Build the automation execution environment container image and use `hub.lab.example.com/review/ee-training-rhel8:v1.0` as the tag. Push the container image to the `hub.lab.example.com` container registry.



### Important

You must log in to the `hub.lab.example.com` container registry to download and push container images. Log in as `student` using `redhat123` as the password.

6.1. Log in to the `hub.lab.example.com` container registry:

```
[student@workstation ee-build]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

6.2. Run the `podman build` command from the `ee-build/` directory. Add the `-t` option to specify the `hub.lab.example.com/review/ee-training-rhel8:v1.0` tag.

```
[student@workstation ee-build]$ podman build -f context/Containerfile \
> -t hub.lab.example.com/review/ee-training-rhel8:v1.0 context
...output omitted...
```

6.3. Confirm that the new container image is available locally. Your container image ID is different from the one displayed in the following output.

```
[student@workstation ee-build]$ podman images
REPOSITORY TAG IMAGE ID ...
hub.lab.example.com/review/ee-training-rhel8 v1.0 c418503ae3ac ...
...output omitted...
```

6.4. Push the container image to your private automation hub:

```
[student@workstation ee-build]$ podman push \
> hub.lab.example.com/review/ee-training-rhel8:v1.0
...output omitted...
```

7. Use the automation controller web UI to create an execution environment resource that uses the following settings:

| <b>Field</b>        | <b>Value</b>                                      |
|---------------------|---------------------------------------------------|
| Name                | Training Review                                   |
| Image               | hub.lab.example.com/review/ee-training-rhel8:v1.0 |
| Pull                | Always pull the container before running.         |
| Organization        | Default                                           |
| Registry credential | Automation Hub Container Registry                 |

- 7.1. Navigate to <https://controller.lab.example.com> and log in as **admin** using **redhat** as the password.
- 7.2. Navigate to **Administration > Execution Environments** and click **Add**.
- 7.3. Create the automation execution environment with the settings in the preceding table. When finished, click **Save**.
8. Create an automation controller job template that uses the following settings:

| <b>Field</b>          | <b>Value</b>    |
|-----------------------|-----------------|
| Name                  | Deploy QR Codes |
| Inventory             | Development     |
| Project               | QR Codes        |
| Execution environment | Training Review |
| Playbook              | test.yml        |
| Credentials           | Developers      |

**Note**

The `lab` command created the `Development` inventory, the `QR Codes` project, and the `Developers` machine credential.

- 8.1. Navigate to **Resources > Templates** and click **Add > Add job template**.
- 8.2. Create the job template with the settings in the preceding table. When finished, click **Save**.
9. Start a job that uses the `Deploy QR Codes` job template and verify that the job completes successfully. If successful, then navigating to `http://serverf.lab.example.com` displays a QR code image.
  - 9.1. Navigate to **Resources > Templates** and click the **Launch Template** icon for the `Deploy QR Codes` job template. Wait for the job to complete.
  - 9.2. Click the **Details** tab for the job. The job displays the **Successful** status.
  - 9.3. Confirm the successful execution of the job by opening a web browser and navigating to `http://serverf.lab.example.com`. If the job completed correctly, then the web page displays a QR code image.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

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

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

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

# Summary

---

- The `ansible-galaxy collection init` command creates the directory structure for new Ansible Content Collections.
- Dependencies for a collection are specified in three files: other collections under the `dependencies` parameter of the `galaxy.yml` file, Python packages in the `requirements.txt` file, and RPM packages in the `bindep.txt` file.
- The `ansible-galaxy collection publish` command is used to publish collections on private automation hub.
- The `ansible-builder` command creates custom automation execution environments.
- The `execution-environment.yml` file is the configuration file for the `ansible-builder` command. It specifies the collections, Python packages, and RPM packages to include in the new automation execution environment.
- The `podman images`, `podman tag`, and `podman push` commands list, tag, and push automation execution environment images to a container registry.
- Automation controller can automatically pull collections and roles specified in an Ansible project's `requirements.yml` file, but you need to set up stored credentials so that it can authenticate to automation hub.
- Automation controller can use and automatically pull custom automation execution environments, but you need to configure credentials for the container registry and specify the location of the execution environment first.

## Chapter 10

# Comprehensive Review

### Goal

Review tasks from *Developing Advanced Automation with Red Hat Ansible Automation Platform*.

### Sections

- Comprehensive Review

### Lab

- Managing Inventory Variables to Use with Automation Content Navigator
- Optimizing a Playbook for Large-scale Use
- Creating and Using Ansible Content Collections and Automation Execution Environments

# Comprehensive Review

---

## Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills that you learned in *Developing Advanced Automation with Red Hat Ansible Automation Platform*.

## Reviewing Developing Advanced Automation with Red Hat Ansible Automation Platform

Before beginning the comprehensive review for this course, you should be comfortable with the topics covered in each chapter. Do not hesitate to ask the instructor for extra guidance or clarification on these topics.

### **Chapter 1, Developing Playbooks with Ansible Automation Platform 2**

Develop Ansible Playbooks with Red Hat Ansible Automation Platform 2 following recommended practices.

- Describe the architecture of Red Hat Ansible Automation Platform 2 and how its new features help with Ansible automation development.
- Install automation content navigator and use it to run an existing playbook with a supported execution environment.
- Create and manage Ansible Playbooks in a Git repository, following recommended practices.
- Demonstrate and describe common recommended practices for developing and maintaining effective Ansible automation solutions.

### **Chapter 2, Managing Content Collections and Execution Environments**

Run playbooks that use content collections not included in `ansible-core`, either from an existing execution environment or by downloading them from automation hub.

- Describe how Ansible Content Collections are used to distribute modules and plug-ins, and create plays that use content from them.
- Search automation hub for Ansible Content Collections, and install them from the command line by name or by using a `requirements.yml` file.
- Identify the automation execution environments provided by Red Hat and select the correct one for your use case.

### **Chapter 3, Running Playbooks with Automation Controller**

Explain what automation controller is and demonstrate how to use it to run playbooks that you developed with automation content navigator.

- Describe the architecture and use cases of the automation controller component of Red Hat Ansible Automation Platform.

**Chapter 10 |** Comprehensive Review

- Navigate and describe the automation controller web UI, and successfully launch a job using a job template, project, credential, and inventory.

## **Chapter 4, Working with Ansible Configuration Settings**

Examine and adjust the configuration of Ansible and automation content navigator to simplify development and to troubleshoot issues.

- Browse the current Ansible configuration by using automation content navigator.
- Change configuration settings for automation content navigator with its configuration file, and determine where the configuration file is located.

## **Chapter 5, Managing Inventories**

Manage inventories by using advanced features of Ansible.

- Describe what dynamic inventories are, and install and use an existing script or plug-in as an Ansible dynamic inventory source.
- Write static inventory files in YAML format.
- Structure host and group variables by using multiple files per host or group, and use special variables to override the host, port, or remote user that Ansible uses for a specific host.

## **Chapter 6, Managing Task Execution**

Control and optimize the execution of tasks by Ansible Playbooks.

- Control automatic privilege escalation at the play, role, task, or block level.
- Configure tasks that can run before roles or after normal handlers, and simultaneously notify multiple handlers.
- Label tasks with tags, and run only tasks labeled with specific tags, or start playbook execution at a specific task.
- Optimize your playbook to run more efficiently, and use callback plug-ins to profile and analyze which tasks consume the most time.

## **Chapter 7, Transforming Data with Filters and Plug-ins**

Populate, manipulate, and manage data in variables using filters and plug-ins.

- Format, parse, and define the values of variables using filters.
- Populate variables with data from external sources using lookup plug-ins.
- Implement loops using structures other than simple lists by using lookup plug-ins and filters.
- Use filters to inspect, validate, and manipulate variables containing networking information.

## **Chapter 8, Coordinating Rolling Updates**

Use advanced features of Ansible to manage rolling updates in order to minimize downtime, and to ensure the maintainability and simplicity of Ansible Playbooks.

- Run a task for a managed host on a different host, and control whether facts gathered by that task are delegated to the managed host or to the other host.

- Tune the number of simultaneous connections that Ansible opens to managed hosts, and how Ansible processes groups of managed hosts through the play's tasks.
- Tune the behavior of the `serial` directive when batching hosts for execution, abort the play if it fails for too many hosts, and create tasks that run only once for each batch or for all hosts in the inventory.

## **Chapter 9, Creating Content Collections and Execution Environments**

Write your own Ansible Content Collections, publish them, embed them in a custom automation execution environment, and run them in playbooks by using automation controller.

- Create content collections and distribute them for reuse.
- Build a custom automation execution environment image by using the `ansible-builder` command.
- Validate that a custom automation execution environment works as expected by testing it with the `ansible-navigator` command, and then distribute the automation execution environment for reuse.
- Run a playbook in automation controller that uses a content collection in the project or content provided by a specific automation execution environment.

## ▶ Lab

# Managing Inventory Variables to Use with Automation Content Navigator

Manage group and inventory variables, configure automation content navigator settings, and push your changes to a new branch using Git.

## Outcomes

- Use Git to clone a repository, create a branch, ignore files, and push changes to a branch.
- Configure basic automation content navigator settings.
- Create Ansible group variables.
- Create Ansible inventory variables in the YAML format.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. When you are ready to push your changes to the remote repository, use `Student@123` as the Git password.

```
[student@workstation ~]$ lab start review-cr1
```

All the computer systems have a `student` user account with `student` as the password. The password for the `root` user account is `redhat`.

## Specifications

- Change an existing project that is stored in a Git repository:
  - The repository is stored at `https://git.lab.example.com/student/review-cr1.git`.
  - The repository must be cloned into the `/home/student/git-repos/review-cr1` directory.
  - All your work must take place within the `student/inventory-edits` branch.
  - Git must be configured to not track or commit files with the `.log` file extension, or files in the project's `.ssh/` and `collections/ansible_collections/` directories.
  - Obtain a token from `https://hub.lab.example.com`, logging in as the `student` user with the password `redhat123`. Set the token in `ansible.cfg` configuration file provided.
- Configure Ansible and automation content navigator as follows:
  - Ansible must be configured to use the `inventory/` directory by default.

**Chapter 10 |** Comprehensive Review

- Automation content navigator must use the `ee-supported-rhel8:latest` image as the default automation execution environment and must not generate any playbook artifact files.
- Configure group variables:
  - The `/home/student/git-repos/review-cr1/haproxy.yml` playbook contains the `lb_service` and `firewall_services` defined variables. To make this playbook more flexible, move these variables out of the playbook, and into the `group_vars` directory in the `review-cr1` directory.
- Configure inventory variables:
  - The `/home/student/git-repos/review-cr1/webapp.yml` playbook requires that the `web_service` and `firewall_services` variables be defined. Create a `group_vars/web/vars.yml` file to define these variables for the `web` group with the following values:

| Variable                       | Value                                    |
|--------------------------------|------------------------------------------|
| <code>web_service</code>       | <code>httpd</code>                       |
| <code>firewall_services</code> | <code>http</code> and <code>https</code> |

- The `/home/student/git-repos/review-cr1/db.yml` playbook runs plays against the `db` group. Add the `db` group to the existing inventory with the single host `serverd.lab.example.com` as a member. This playbook requires that the `db_service` and `db_port` variables be defined. Create a `group_vars/db/vars.yml` file to define these variables for the `db` group with the following values:

| Variable                | Value                 |
|-------------------------|-----------------------|
| <code>db_service</code> | <code>mariadb</code>  |
| <code>db_port</code>    | <code>3306/tcp</code> |

- After all the variables are defined, use automation content navigator to run the following playbooks:
  - `/home/student/git-repos/review-cr1/haproxy.yml`
  - `/home/student/git-repos/review-cr1/webapp.yml`
  - `/home/student/git-repos/review-cr1/db.yml`
- Navigate to `http://servera.lab.example.com` to test the web application. The web application displays `This is the server server[bc] and the database is up.`
- After you confirm that the web application is running, commit and push your changes to the `student/inventory-edits` remote branch.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade review-cr1
```

## Finish

On the **workstation** machine, change to the **student** user home directory and use the **lab** command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr1
```

## ► Solution

# Managing Inventory Variables to Use with Automation Content Navigator

Manage group and inventory variables, configure automation content navigator settings, and push your changes to a new branch using Git.

### Outcomes

- Use Git to clone a repository, create a branch, ignore files, and push changes to a branch.
- Configure basic automation content navigator settings.
- Create Ansible group variables.
- Create Ansible inventory variables in the YAML format.

### Before You Begin

As the student user on the workstation machine, use the `lab` command to prepare your system for this exercise.

This command initializes the remote Git repository that you need for this lab. When you are ready to push your changes to the remote repository, use `Student@123` as the Git password.

```
[student@workstation ~]$ lab start review-cr1
```

All the computer systems have a `student` user account with `student` as the password. The password for the `root` user account is `redhat`.

1. On the workstation machine, clone the `https://git.lab.example.com/student/review-cr1.git` repository into the `/home/student/git-repos/review-cr1/` directory, and create a branch named `student/inventory-edits`.

- 1.1. Clone the repository to the `/home/student/git-repos/review-cr1/` directory:

```
[student@workstation ~]$ mkdir -p git-repos/review-cr1
[student@workstation ~]$ git clone \
> https://git.lab.example.com/student/review-cr1.git \
> /home/student/git-repos/review-cr1/
```

- 1.2. Change to the `/home/student/git-repos/review-cr1/` directory. Create a branch named `student/inventory-edits` and switch to it:

```
[student@workstation ~]$ cd /home/student/git-repos/review-cr1/
[student@workstation review-cr1]$ git checkout -b student/inventory-edits
[student@workstation review-cr1]$ git status
On branch student/inventory-edits
nothing to commit, working tree clean
```

2. Configure Git to ignore any files with the .log file extension, and the .ssh/ and collections/ansible\_collections/ directories. Git should not track or commit these files or directories.
  - 2.1. Create a /home/student/git-repos/review-cr1/.gitignore file with the following content:

```
* .log
.ssh/
collections/ansible_collections
```

3. Retrieve a token from https://hub.lab.example.com, logging in as the student user with the password redhat123, and then use it to update the token option in the ansible.cfg file. Use the ansible-galaxy command to install the collections. The collections must be available to the execution environment, so they must be installed in the /home/student/git-repos/review-cr1/collections/ directory.
  - 3.1. Log in to the private automation hub at https://hub.lab.example.com as the student user and using redhat123 as the password.
  - 3.2. Navigate to Collections > API token management and then click Load token. Copy the API token.
  - 3.3. Using the copied token, update the token line in the ansible.cfg file. Your token is probably different from the one that is displayed in this example.

```
...output omitted...

[galaxy_server.community_repo]
url=https://hub.lab.example.com/api/galaxy/content/community/
token=f41f07130d6eb6ef2ded63a574c161b509c647dd
```

- 3.4. Use the ansible-galaxy command to install the community.mysql content collection into the collections/ directory.

```
[student@workstation review-cr1]$ ansible-galaxy collection install \
> -r collections/requirements.yml
Starting galaxy collection install process
...output omitted...
Installing 'community.mysql:3.5.1' to '/home/student/git-repos/review-cr1/
collections/ansible_collections/community/mysql'
community.mysql:3.5.1 was installed successfully
```

- 3.5. Automation content navigator should use the ee-supported-rhel8:latest image as the default automation execution environment and should not generate playbook

**Chapter 10 |** Comprehensive Review

artifacts. Create the /home/student/git-repos/review-cr1/ansible-navigator.yml file with the following content:

```

ansible-navigator:
 execution-environment:
 image: ee-supported-rhel8:latest
 playbook-artifact:
 enable: false
```

4. Remove the variables from the /home/student/git-repos/review-cr1/haproxy.yml playbook to make it more flexible.
  - 4.1. Edit the /home/student/git-repos/review-cr1/haproxy.yml playbook and remove the vars: section. The completed playbook should contain the following content:

```

- name: Deploy haproxy
 hosts: haproxy
 become: true

 tasks:
...output omitted...
```

- 4.2. Create the /home/student/git-repos/review-cr1/group\_vars/{haproxy,web,db} directories:

```
[student@workstation review-cr1]$ mkdir -p \
> /home/student/git-repos/review-cr1/group_vars/{haproxy,web,db}
```

- 4.3. Create the /home/student/git-repos/review-cr1/group\_vars/haproxy/vars.yml file and add the lb\_service and firewall\_services variables into it:

```

lb_service: haproxy
firewall_services:
 - http
 - https
```

- 4.4. Create the /home/student/git-repos/review-cr1/group\_vars/web/vars.yml file and add the web\_service and firewall\_services variables to it:

```

web_service: httpd
firewall_services:
 - http
 - https
```

- 4.5. Create the /home/student/git-repos/review-cr1/group\_vars/db/vars.yml file and add the db\_service and db\_port variables to it:

```

db_service: mariadb
db_port: 3306/tcp
```

- 4.6. Add the db group to the /home/student/git-repos/review-cr1/inventory/inventory-static.yml inventory file with the single host serverd.lab.example.com:

```
web:
 hosts:
 serverb.lab.example.com:
 serverc.lab.example.com:

db:
 hosts:
 serverd.lab.example.com:
```

5. Use automation content navigator to run the playbooks.

- 5.1. Verify that automation content navigator is installed, and if not, install it.

```
[student@workstation review-cr1]$ rpm -q ansible-navigator
ansible-navigator-2.1.0-1.el8ap.noarch
```



### Note

You might need to install automation content navigator:

```
[student@workstation review-cr1]$ sudo yum install ansible-navigator
[sudo] password for student: student
...output omitted...
```

- 5.2. The /home/student/git-repos/review-cr1/haproxy.yml playbook runs against the haproxy group that is defined in the /home/student/git-repos/review-cr1/inventory/inventory-dynamic.py dynamic inventory file. This inventory file must have execute permissions before it can be used:

```
[student@workstation review-cr1]$ chmod 755 inventory/inventory-dynamic.py
```

- 5.3. Use the podman login command to log in to the private automation hub at hub.lab.example.com.

```
[student@workstation review-cr1]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 5.4. Run the /home/student/git-repos/review-cr1/haproxy.yml playbook:

**Chapter 10 |** Comprehensive Review

```
[student@workstation review-cr1]$ ansible-navigator run -m stdout haproxy.yml
...output omitted...
```

5.5. Run the /home/student/git-repos/review-cr1/webapp.yml playbook:

```
[student@workstation review-cr1]$ ansible-navigator run -m stdout webapp.yml
...output omitted...
```

5.6. Run the /home/student/git-repos/review-cr1/db.yml playbooks:

```
[student@workstation review-cr1]$ ansible-navigator run -m stdout db.yml
...output omitted...
```

6. Verify that the web application is up:

```
[student@workstation review-cr1]$ curl http://servera.lab.example.com
This is the server serverc and the database is up
[student@workstation review-cr1]$ curl http://servera.lab.example.com
This is the server serverb and the database is up
```

7. Push your changes to the student/inventory-edits branch.

7.1. List all modified and untracked files:

```
[student@workstation review-cr1]$ git status
On branch student/inventory-edits
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git restore <file>..." to discard changes in working directory)
 modified: ansible.cfg
 modified: haproxy.yml
 modified: inventory/inventory-dynamic.py
 modified: inventory/inventory-static.yml

Untracked files:
 (use "git add <file>..." to include in what will be committed)
 .gitignore
 ansible-navigator.yml
 group_vars/

no changes added to commit (use "git add" and/or "git commit -a")
```

7.2. Add the files you created and edited to the staging area:

```
[student@workstation review-cr1]$ git add --all
```

7.3. Commit the staged files to the local repository:

```
[student@workstation review-cr1]$ git commit -m 'updating inventory'
...output omitted...
```

- 7.4. Push the changes from the local repository to the `student/inventory-edits` branch on the remote repository. If prompted, use `Student@123` as the Git password.

```
[student@workstation review-cr1]$ git push -u origin student/inventory-edits
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
```

- 7.5. Verify that your changes were pushed to the `student/inventory-edits` branch:

```
[student@workstation review-cr1]$ git status
On branch student/inventory-edits
Your branch is up to date with 'origin/student/inventory-edits'.

nothing to commit, working tree clean
```

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade review-cr1
```

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr1
```

## ▶ Lab

# Optimizing a Playbook for Large-scale Use

Use filters and lookup plug-ins to get and format values for variables, modify a play to implement rolling updates, and optimize the execution of plays and tasks in a playbook.

## Outcomes

- Use filters and lookup plug-ins to manipulate variables in play and role tasks.
- Delegate tasks to other hosts, run hosts through plays in batches with the `serial` keyword, and limit failures with the `max_fail_percentage` keyword.
- Set privilege escalation per play or per task.
- Add tags to target specific tasks.

## Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the remote `https://git.lab.example.com/student/review-cr2.git` Git repository is initialized. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository using `Student@123` as the Git password.

```
[student@workstation ~]$ lab start review-cr2
```

## Specifications

- Create the `/home/student/git-repos` directory if it does not exist. Clone the `https://git.lab.example.com/student/review-cr2.git` Git repository into the `/home/student/git-repos` directory. Create the `exercise` branch for this exercise to store your changes.
- Configure the project's `ansible.cfg` file so that privilege escalation is not performed by default. Edit the plays in the `deploy_apache.yml`, `deploy_haproxy.yml`, and `deploy_webapp.yml` playbooks so that privilege escalation is enabled at the play level. Privilege escalation is not required for fact gathering tasks.
- Edit the `roles/firewall/tasks/main.yml` tasks file so that it uses filters to set default values for variables in each of the three tasks if they are not set, as follows:
  - For the `state` option, if `item['state']` is not set, then set it to `enabled` by default.
  - For the `zone` option, if `item['zone']` is not set, then omit the `zone` option.
  - In the `Ensure Firewall Port Configuration` task, in the `port` option to the `firewalld` module, if `item['protocol']` is not set, then set it to `tcp`. Use the `lower` filter to ensure that its value is in lowercase.

- You can test your changes to the `roles/firewall/tasks/main.yml` file by running the `test_firewall_role.yml` playbook using automation content navigator and the `ee-supported-rhel8` automation execution environment.
- Edit the `deploy_apache.yml` playbook. Change the value of the `firewall_rules` variable to a Jinja2 expression that uses the `template` lookup plug-in to dynamically generate the variable's setting from the `apache_firewall_rules.yml.j2` Jinja2 template. Use the `from_yaml` filter in the Jinja2 expression to convert the resulting value from a text string into a YAML data structure that Ansible can interpret.
- Edit the `templates/apache_firewall_rules.yml.j2` template to replace the `load_balancer_ip_addr` host variable in the Jinja2 for loop with the `hostvars[server]['ansible_facts']['default_ipv4']['address']` fact.
- Refactor the most expensive task in the `apache` role to make it more efficient.
  - Enable the `timer` and `profile_tasks` callback plug-ins for the project.
  - Using the `ee-supported-rhel8:latest` automation execution environment, run the `site.yml` playbook and analyze the output to find the most time-expensive task.
  - Refactor that time-expensive task.
  - Add the `apache_installer` tag to the refactored task.
  - To verify your work, rerun the `site.yml` playbook but limit the execution to the task with the `apache_installer` tag.
- The `update_webapp.yml` playbook performs a rolling update of the web content on the back-end web servers. It is not yet functional. Edit the `update_webapp.yml` playbook as follows:
  - Add a `pre_tasks` section to the play. In that section, add a task that uses the `community.general.haproxy` module to disable the web servers on the HAProxy load balancer. Disable the host by using the `inventory_hostname` variable in the app back end. Delegate the task to the load balancer. The `{{ groups['lb_servers'][0] }}` Jinja2 expression provides the name of this load balancer.
  - Add a task at the end of the `post_tasks` section to re-enable the web servers.
  - Configure the play in the playbook to run in batches. The first batch must contain 5% of the hosts, the second batch 35% of the hosts, and the final batch all remaining hosts in the play.
  - Set `max_fail_percentage` on the play with a value that ensures that playbook execution stops if any host fails a task during the execution of the play.
  - To verify your work, run the `update_webapp.yml` playbook using automation content navigator and the `ee-supported-rhel8` automation execution environment.
- To verify the correct deployment of the load balancer and the web servers, run the `curl servera` command several times from the `workstation` machine. The HAProxy server that you installed on the `servera` machine dispatches the requests between the back-end web servers that are installed on the `serverb` to `serverf` machines. The command must return the name of a different back-end web server each time you run it.

```
[student@workstation review-cr2]$ curl servera
This is serverd. (version v1.0)
[student@workstation review-cr2]$ curl servera
This is servere. (version v1.0)
[student@workstation review-cr2]$ curl servera
This is serverf. (version v1.0)
```

- Commit and push your changes to the `exercise` branch. If prompted, use `Student@123` as the password.

## Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.



### Note

Make sure you commit and push your changes to the Git repository before grading your work.

```
[student@workstation ~]$ lab grade review-cr2
```

## Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr2
```

## ► Solution

# Optimizing a Playbook for Large-scale Use

Use filters and lookup plug-ins to get and format values for variables, modify a play to implement rolling updates, and optimize the execution of plays and tasks in a playbook.

## Outcomes

- Use filters and lookup plug-ins to manipulate variables in play and role tasks.
- Delegate tasks to other hosts, run hosts through plays in batches with the `serial` keyword, and limit failures with the `max_fail_percentage` keyword.
- Set privilege escalation per play or per task.
- Add tags to target specific tasks.

## Before You Begin

As the student user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the remote `https://git.lab.example.com/student/review-cr2.git` Git repository is initialized. The Git repository contains playbooks that configure a front-end load balancer and a pool of back-end web servers. You can push changes to this repository using `Student@123` as the Git password.

```
[student@workstation ~]$ lab start review-cr2
```

1. Clone the `https://git.lab.example.com/student/review-cr2.git` Git repository into the `/home/student/git-repos` directory and then create a branch for this exercise.
  - 1.1. From a terminal, create the `/home/student/git-repos` directory if it does not exist, and then change into it.

```
[student@workstation ~]$ mkdir -p ~/git-repos/
[student@workstation ~]$ cd ~/git-repos/
```

- 1.2. Clone the `https://git.lab.example.com/student/review-cr2.git` repository and then change into the cloned repository.

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/review-cr2.git
Cloning into 'review-cr2'...
...output omitted...
[student@workstation git-repos]$ cd review-cr2
```

- 1.3. Create the `exercise` branch and switch to it.

```
[student@workstation review-cr2]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

2. Modify the `ansible.cfg` file so that privilege escalation is not performed by default.

Because the tasks in the `firewall`, `haproxy`, `apache`, and `webapp` roles require privilege escalation, enable privilege escalation at the play level in the `deploy_apache.yml`, `deploy_haproxy.yml`, and `deploy_webapp.yml` playbooks.

- 2.1. Edit the `ansible.cfg` file to remove the `become=true` entry from the `privilege_escalation` block, or use `become=false` in the `privilege_escalation` block. Alternatively, remove the entire `[privilege_escalation]` block from the `ansible.cfg` file. If you choose the last option, then the file contains the following content:

```
[defaults]
inventory=inventory
remote_user=devops
collections_paths=./collections:/usr/share/ansible/collections
```

- 2.2. Enable privilege escalation for the `Ensure Apache is deployed` play in the `deploy_apache.yml` playbook. Add the `become: true` line.

```

- name: Ensure Apache is deployed
 hosts: web_servers
 force_handlers: true
 gather_facts: false
 become: true

 roles:
 # Use the apache_firewall_rules.yml.j2 template to
 # generate the firewall rules.
 - role: apache
 firewall_rules: []
```

- 2.3. Enable privilege escalation for the play in the `deploy_haproxy.yml` playbook. Add the `become: true` line.

```

- name: Gather web_server facts
 hosts: web_servers
 gather_facts: true
 tasks: []

- name: Ensure HAProxy is deployed
 hosts: lb_servers
 force_handlers: true
 become: true

 roles:
 # The "haproxy" role has a dependency on the "firewall"
 # role. the "firewall" role requires a "firewall_rules"
```

```
variable be defined.
- role: haproxy
 haproxy_backend_port: "{{ apache_port }}"
 # all backend servers are active; none are disabled.
 haproxy_backend_pool: "{{ groups['web_servers'] }}"
 haproxy_active_backends: "{{ groups['web_servers'] }}"
```

- 2.4. Enable privilege escalation for the play in the `deploy_webapp.yml` playbook. Add the `become: true` line.

```

- name: Ensure Web App is deployed
 hosts: web_servers
 become: true

 roles:
 - role: webapp
```

3. Edit each of the tasks in the `roles/firewall/tasks/main.yml` tasks file to use filters to set default values for specific variables if they do not have a value set, as follows:

- For the `state` option, if `item['state']` is not set, then set it to `enabled` by default.
- For the `zone` option, if `item['zone']` is not set, then omit the `zone` option.
- In the `Ensure Firewall Port Configuration` task, in the `port` option to the `firewalld` module, if `item['protocol']` is not set, then set it to `tcp`. Use the `lower` filter to ensure that its value is in lowercase.

- 3.1. Edit the `roles/firewall/tasks/main.yml` file. For the `state` option of all three `firewalld` tasks, add the `default('enabled')` filter to the `item['state']` Jinja2 expression.

The updated file should consist of the following content:

```
- name: Ensure Firewall Sources Configuration
 ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['source'] is defined
 notify: reload firewalld

- name: Ensure Firewall Service Configuration
 ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
 loop: "{{ firewall_rules }}"
 when: item['service'] is defined
 notify: reload firewalld
```

**Chapter 10 |** Comprehensive Review

```
- name: Ensure Firewall Port Configuration
ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] }}"
 zone: "{{ item['zone'] }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['port'] is defined
notify: reload firewalld
```

- 3.2. For the `zone` option of all three `firewalld` tasks, add the `default('omit')` filter to the `item['zone']` Jinja2 expression.

The task file should now contain the following content:

```
- name: Ensure Firewall Sources Configuration
ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] | default('omit') }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['source'] is defined
notify: reload firewalld

- name: Ensure Firewall Service Configuration
ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] | default('omit') }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['service'] is defined
notify: reload firewalld

- name: Ensure Firewall Port Configuration
ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] }}"
 zone: "{{ item['zone'] | default('omit') }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['port'] is defined
notify: reload firewalld
```

- 3.3. In the third `firewalld` task, `Ensure Firewall Port Configuration`, replace `{{ item['protocol'] }}` with `{{ item['protocol'] | default('tcp') | lower }}`. Save your work.

The completed task file should contain the following content:

```
- name: Ensure Firewall Sources Configuration
ansible.posix.firewalld:
 source: "{{ item['source'] }}"
 zone: "{{ item['zone'] | default('omit') }}"
```

```

permanent: true
state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['source'] is defined
notify: reload firewalld

- name: Ensure Firewall Service Configuration
ansible.posix.firewalld:
 service: "{{ item['service'] }}"
 zone: "{{ item['zone'] | default(omit) }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['service'] is defined
notify: reload firewalld

- name: Ensure Firewall Port Configuration
ansible.posix.firewalld:
 port: "{{ item['port'] }}/{{ item['protocol'] | default('tcp') | lower }}"
 zone: "{{ item['zone'] | default(omit) }}"
 permanent: true
 state: "{{ item['state'] | default('enabled') }}"
loop: "{{ firewall_rules }}"
when: item['port'] is defined
notify: reload firewalld

```

- 3.4. Test the changes that you made to the `roles/firewall/tasks/main.yml` file by running the `test_firewall_role.yml` playbook. If you completed the preceding step successfully, then the playbook runs without errors.

```

[student@workstation review-cr2]$ ansible-navigator run \
> -m stdout test_firewall_role.yml

PLAY [Test Firewall Role] ****
...output omitted...

PLAY RECAP ****
serverf.lab.example.com : ok=6 changed=5 unreachable=0 failed=0 ...

```

4. Examine the `deploy_apache.yml` playbook. It calls the `apache` role to ensure that Apache HTTP Server is deployed, which itself calls the `firewall` role. It also defines the `firewall_rules` variable that the `firewall` role uses to configure the `firewalld` service. The `8008/tcp` port for `firewalld` and the IPv4 address of the load balancer (`172.25.250.10`) must be enabled in the `internal` zone for `firewalld`.  
Edit the playbook. Change the value of the `firewall_rules` variable to a Jinja2 expression that uses the `template` lookup plug-in to dynamically generate the variable's setting from the `apache_firewall_rules.yml.j2` Jinja2 template. Use the `from_yaml` filter in the Jinja2 expression to convert the resulting value from a text string into a YAML data structure that Ansible can interpret.  
One correct solution results in the following contents in the `deploy_apache.yml` Ansible Playbook:

```

- name: Ensure Apache is deployed
 hosts: web_servers
 force_handlers: true
 gather_facts: false
 become: true

 roles:
 # Use the apache_firewall_rules.yml.j2 template to
 # generate the firewall rules.
 - role: apache
 firewall_rules: "{{ lookup('ansible.builtin.template',
 'apache_firewall_rules.yml.j2') | from_yaml }}"

```

**Note**

In the preceding example, make sure you enter the expression for the `firewall_rules` variable on a single line.

If you use linting rules that indicate that lines should not exceed 80 characters, then you might define the Jinja2 expression over multiple lines by defining the `firewall_rules` variable using a greater than sign. For example:

```

firewall_rules: >
 {{ lookup('ansible.builtin.template', 'apache_firewall_rules.yml.j2')
 | from_yaml }}

```

5. Examine the `templates/apache_firewall_rules.yml.j2` template that your playbook now uses to set the `firewall_rules` variable.

The template contains a Jinja2 `for` loop that creates a rule for each host in the `lb_servers` group, setting the source IP address from the value of the `load_balancer_ip_addr` variable.

This solution is not ideal. It requires you to set `load_balancer_ip_addr` as a host variable for each load balancer in the `lb_servers` group. To remove this manual maintenance, use gathered facts from these hosts to set that value instead.

Edit the `templates/apache_firewall_rules.yml.j2` file to replace the `load_balancer_ip_addr` host variable in the Jinja2 `for` loop with the `hostvars[server]['ansible_facts']['default_ipv4']['address']` fact.

The completed `templates/apache_firewall_rules.yml.j2` template should contain the following content:

```

- port: {{ apache_port }}
 protocol: TCP
 zone: internal
{% for server in groups['lb_servers'] %}
- zone: internal
 source: "{{ hostvars[server]['ansible_facts']['default_ipv4']['address'] }}"
{% endfor %}

```

Save the template.

**Chapter 10 |** Comprehensive Review

6. Enable the `timer` and `profile_tasks` callback plug-ins for the project. The two plug-ins are part of the `ansible.posix` collection. If desired, you can specify the FQCNs for the callback plug-ins.

Edit the `ansible.cfg` configuration file and add the plug-ins to the `callbacks_enabled` directive. The modified file contains the following content:

```
[defaults]
inventory=inventory
remote_user=devops
collections_paths=./collections:/usr/share/ansible/collections
callbacks_enabled=ansible.posix.timer,ansible.posix.profile_tasks
```

7. Using the `ee-supported-rhel8:latest` automation execution environment, run the `site.yml` playbook and analyze the output to find the most time-expensive task.
- 7.1. Run the `site.yml` playbook using the `stdout` mode. Your output should look similar to the following.

```
[student@workstation review-cr2]$ ansible-navigator run site.yml -m stdout
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=7 changed=6 unreachable=0 failed=0 ...
serverb.lab.example.com : ok=14 changed=9 unreachable=0 failed=0 ...
serverc.lab.example.com : ok=14 changed=9 unreachable=0 failed=0 ...
serverd.lab.example.com : ok=14 changed=9 unreachable=0 failed=0 ...
servere.lab.example.com : ok=14 changed=9 unreachable=0 failed=0 ...
serverf.lab.example.com : ok=14 changed=9 unreachable=0 failed=0 ...
Playbook run took 0 days, 0 hours, 1 minutes, 32 seconds
Friday 06 January 2023 20:10:46 +0000 (0:00:00.967) 0:01:32.206 ****
=====
apache : Ensure httpd packages are installed ----- 36.38s
apache : Ensure SELinux allows httpd connections to a remote database -- 25.72s
haproxy : Ensure haproxy packages are present ----- 10.91s
apache : restart httpd ----- 1.86s
Gathering Facts ----- 1.63s
...output omitted...
```

- 7.2. The previous output sorted the tasks based on how long it took for each task to complete. The following task took the most time:
- ```
apache : Ensure httpd packages are installed ----- 36.38s
```
8. Refactor the most expensive task to make it more efficient. Add the `apache_installer` tag to the task.
- 8.1. Identify the file that contains the `Ensure httpd packages are installed` task.

```
[student@workstation review-cr2]$ grep -Rl 'Ensure httpd packages are installed'
roles/apache/tasks/main.yml
...output omitted...
```

- 8.2. Edit the `roles/apache/tasks/main.yml` task file to remove the loop from the `yum` task. The modified file contains the following content:

```
---  
# tasks file for apache  
  
- name: Ensure httpd packages are installed  
  ansible.builtin.yum:  
    name:  
      - httpd  
      - php  
      - git  
      - php-mysqlnd  
    state: present  
...output omitted...
```

- 8.3. Add the `apache_installer` tag to the `yum` task. The modified file contains the following content:

```
---  
# tasks file for apache  
  
- name: Ensure httpd packages are installed  
  ansible.builtin.yum:  
    name:  
      - httpd  
      - php  
      - git  
      - php-mysqlnd  
    state: present  
  tags: apache_installer  
  
- name: Ensure SELinux allows httpd connections to a remote database  
  ansible.posix.seboolean:  
    name: httpd_can_network_connect_db  
    state: true  
    persistent: true  
  
- name: Ensure httpd service is started and enabled  
  ansible.builtin.service:  
    name: httpd  
    state: started  
    enabled: true  
  
- name: Ensure configuration is deployed  
  ansible.builtin.template:  
    src: httpd.conf.j2  
    dest: /etc/httpd/conf/httpd.conf  
    owner: root  
    group: root  
    mode: 0644  
    setype: httpd_config_t  
  notify: restart httpd
```

- 8.4. Run the `site.yml` playbook with the `ee-supported-rhel8:latest` automation execution environment and the `apache_installer` tag. Verify that the `Ensure httpd` packages are installed task takes less time to complete.

```
[student@workstation review-cr2]$ ansible-navigator run site.yml \
> -m stdout --tags apache_installer
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=1    changed=0    unreachable=0    failed=0 ...
serverb.lab.example.com : ok=3    changed=0    unreachable=0    failed=0 ...
serverc.lab.example.com : ok=3    changed=0    unreachable=0    failed=0 ...
serverd.lab.example.com : ok=3    changed=0    unreachable=0    failed=0 ...
servere.lab.example.com : ok=3    changed=0    unreachable=0    failed=0 ...
serverf.lab.example.com : ok=3    changed=0    unreachable=0    failed=0 ...
Playbook run took 0 days, 0 hours, 0 minutes, 7 seconds
Tuesday 10 January 2023 20:42:09 +0000 (0:00:01.036)      0:00:07.963 ****
=====
apache : Ensure httpd packages are installed ----- 4.02s
Gathering Facts ----- 1.52s
Gathering Facts ----- 1.37s
Gathering Facts ----- 1.04s
```

9. Add a `pre_tasks` section and task to the play in the `update_webapp.yml` playbook. This task should disable the web servers on the HAProxy load balancer. Without this task, external clients might experience problems due to unforeseen deployment issues with the web application.

Configure the new task as follows:

- Use the `community.general.haproxy` module.
- Disable the host by using the `inventory_hostname` variable in the app back end.
- Delegate each action to the load balancer. The `{{ groups['lb_servers'][0] }}` Jinja2 expression provides the name of this load balancer.

Edit the `update_webapp.yml` file, adding the `pre_tasks` task to disable the web server in the load balancer.

```
...output omitted...
pre_tasks:
  - name: Remove web server from service during the update
    community.general.haproxy:
      state: disabled
      backend: app
      host: "{{ inventory_hostname }}"
      delegate_to: "{{ groups['lb_servers'][0] }}"
...output omitted...
```

10. In the play in the `update_webapp.yml` playbook, add a task to the `post_tasks` section after the smoke test to re-enable each web server in the HAProxy load balancer. This task is similar in structure to the `community.general.haproxy` task in the `pre_tasks` section.

Add the task after the smoke test with the `state: enabled` directive.

```
...output omitted...
post_tasks:
  - name: Smoke Test - Ensure HTTP 200 OK
    ansible.builtin.uri:
      url: "http://localhost:{{ apache_port }}"
      status_code: 200
    become: true

  - name: Enable healthy server in load balancers
    community.general.haproxy:
      state: enabled
      backend: app
      host: "{{ inventory_hostname }}"
    delegate_to: "{{ groups['lb_servers'][0] }}"
```

11. Configure the play in the `update_webapp.yml` playbook to run in batches. This change mitigates the effects of unforeseen deployment errors. Ensure that the playbook uses no more than three batches to complete the upgrade of all web server hosts. Set the first batch to consist of 5% of the hosts in the play, the second batch 35% of the hosts, and the final batch to consist of all remaining hosts.

Add an appropriate setting for `max_fail_percentage` to the play to ensure that playbook execution stops if any host fails a task during the upgrade.

Add the `max_fail_percentage` directive to the play and set its value to 0, to stop execution on any failure.

Add the `serial` directive and set the value to a list of three elements, 5%, 35%, and 100%, to ensure that all servers are updated in the last batch. The entire playbook should contain the following content:

```
---
- name: Upgrade Web Application
  hosts: web_servers
  become: true
  max_fail_percentage: 0
  serial:
    - "5%"
    - "35%"
    - "100%"

  pre_tasks:
    - name: Remove web server from service during the update
      community.general.haproxy:
        state: disabled
        backend: app
        host: "{{ inventory_hostname }}"
      delegate_to: "{{ groups['lb_servers'][0] }}"

  roles:
    - role: webapp

  post_tasks:
    - name: Smoke Test - Ensure HTTP 200 OK
      ansible.builtin.uri:
```

```

    url: "http://localhost:{{ apache_port }}"
    status_code: 200
    become: true

    - name: Enable healthy server in load balancers
      community.general.haproxy:
        state: enabled
        backend: app
        host: "{{ inventory_hostname }}"
      delegate_to: "{{ groups['lb_servers'][0] }}"

```

12. Use the `ansible-navigator` command to run the `update_webapp.yml` playbook. The playbook performs a rolling update.

```

[student@workstation review-cr2]$ ansible-navigator run \
> -m stdout update_webapp.yml
...output omitted...
TASK [Remove web server from service during the update] *****
Wednesday 04 January 2023 19:27:58 +0000 (0:00:01.568)      0:00:01.591 *****
changed: [serverb.lab.example.com -> servera.lab.example.com]
...output omitted...
TASK [Enable healthy server in load balancers] *****
Wednesday 04 January 2023 19:28:01 +0000 (0:00:00.705)      0:00:04.888 *****
changed: [serverb.lab.example.com -> servera.lab.example.com]
...output omitted...
TASK [Remove web server from service during the update] *****
Wednesday 04 January 2023 19:28:03 +0000 (0:00:01.132)      0:00:06.517 *****
changed: [serverc.lab.example.com -> servera.lab.example.com]
...output omitted...
TASK [Enable healthy server in load balancers] *****
Wednesday 04 January 2023 19:28:05 +0000 (0:00:00.503)      0:00:08.753 *****
changed: [serverc.lab.example.com -> servera.lab.example.com]
...output omitted...
TASK [Remove web server from service during the update] *****
Wednesday 04 January 2023 19:28:07 +0000 (0:00:01.316)      0:00:10.566 *****
changed: [serverd.lab.example.com -> servera.lab.example.com]
changed: [servere.lab.example.com -> servera.lab.example.com]
changed: [serverf.lab.example.com -> servera.lab.example.com]
...output omitted...
TASK [Enable healthy server in load balancers] *****
Wednesday 04 January 2023 19:28:10 +0000 (0:00:00.627)      0:00:13.911 *****
changed: [serverd.lab.example.com -> servera.lab.example.com]
changed: [servere.lab.example.com -> servera.lab.example.com]
changed: [serverf.lab.example.com -> servera.lab.example.com]
...output omitted...

```

Notice that Ansible first updates the `serverb.lab.example.com` machine, which is the only machine in the first batch (5% of 5 machines). Then Ansible updates the `serverc.lab.example.com` machine, which is the only machine in the second batch (35% of 5 machines). Finally, Ansible updates the remaining machines (100% of the machines in the last batch).

Chapter 10 | Comprehensive Review

13. Verify that web browser requests from workstation to the load balancer on servera succeed.

```
[student@workstation review-cr2]$ curl servera  
This is serverb. (version v1.0)  
[student@workstation review-cr2]$ curl servera  
This is serverc. (version v1.0)
```

14. Add the changed files, commit the changes, and push them to the Git repository. If prompted, use Student@123 as the password.

```
[student@workstation review-cr2]$ git add .  
[student@workstation review-cr2]$ git commit -m "Project updates"  
[exercise ff11b64] Project updates  
 6 files changed, 29 insertions(+), 14 deletions(-)  
[student@workstation update-review]$ git push -u origin exercise  
Password for 'https://student@git.lab.example.com': Student@123  
...output omitted...
```

Evaluation

As the student user on the workstation machine, use the lab command to grade your work. Correct any reported failures and rerun the command until successful.

**Note**

Make sure you commit and push your changes to the Git repository before grading your work.

```
[student@workstation ~]$ lab grade review-cr2
```

Finish

On the workstation machine, change to the student user home directory and use the lab command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr2
```

▶ Lab

Creating and Using Ansible Content Collections and Automation Execution Environments

Create an Ansible Content Collection and an automation execution environment, and develop a playbook to test the automation execution environment from automation controller.

Outcomes

- Create an Ansible Content Collection that includes roles.
- Create a custom automation execution environment.
- Upload an Ansible Content Collection and an automation execution environment to a private automation hub.
- Configure automation controller to use a custom automation execution environment from a private automation hub.
- Create automation controller resources such as credentials, projects, and inventories.
- Create and launch a new job template using a specific automation execution environment.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/review-cr3/` directory and creates a Git repository that includes a playbook to test your work. Use `Student@123` as the Git password when you push changes to this remote repository.

```
[student@workstation ~]$ lab start review-cr3
```

The `/home/student/review-cr3/lab-resources.txt` file lists all the resources that you configure during the exercise. You can use that file to copy and then paste those resources.

To log in to the web UI of private automation hub, use `student` as the username and `redhat123` as the password. To log in to the web UI of automation controller, use `admin` as the username and `redhat` as the password. Use the following URLs to access those services:

- Private automation hub: <https://hub.lab.example.com>
- Automation controller: <https://controller.lab.example.com>

You can log in to each of the computer systems that you manage with the `student` user, with `student` as the password. The password for the `root` user account on those machines is `redhat`.

Specifications

- Create an Ansible Content Collection as follows:
 - Name the collection `review.system`.
 - Initialize a directory for the collection on `workstation` in the `/home/student/review-cr3` directory.
 - The collection must require Ansible version `>=2.9.10`.
 - Add the `iscsi_target_create` role to the collection. Copy the role from the `/home/student/review-cr3/roles/` directory to the correct location in the collection's directory structure.
 - The `iscsi_target_create` role requires `ansible.posix` collection version 1.0.0 or later. Add that dependency to the collection.
 - When you build the `review.system` collection, copy the resulting compressed tar archive to `/home/student/review-cr3`.
 - On your private automation hub, create a namespace for the collection, upload it, and approve it. The namespace must specify **Content Developers** as the namespace owners.
- Create a custom automation execution environment as follows:
 - Create the `/home/student/review-cr3/ee-build` directory and use this directory for the configuration files needed to build a custom automation execution environment. Example files are in the `/home/student/review-cr3/examples/` directory on the `workstation` machine.
 - The build environment must be configured to retrieve Ansible Content Collections from your private automation hub. Use the private automation hub web UI to generate an authentication token.
 - The automation execution environment must be tagged `hub.lab.example.com/system/ee-review-rhel8:v1.0`.
 - The automation execution environment must include the `review.system`, `ansible.posix`, and `community.general` Ansible Content Collections.
 - The automation execution environment must include your private automation hub's TLS CA certificate. Copy the `/home/student/review-cr3/Containerfile` and `/etc/pki/tls/certs/classroom-ca.pem` files to the `context/` directory of your build environment.
 - Use `hub.lab.example.com/ee-minimal-rhel8:latest` as the base container image and `hub.lab.example.com/ansible-builder-rhel8:latest` as the builder image.
 - Publish the execution environment's container image on your private automation hub.
- Prepare an Ansible Playbook that tests your `review.system` Ansible Content Collection and custom automation execution environment, as follows:
 - The playbook must be named `demo.yml`. A partially completed playbook is available in the Git repository at `git@git.lab.example.com:student/iscsi.git`. Clone that repository to the `/home/student/git-repos` directory on `workstation`.
 - Create the `review3` branch to store your modifications.

- In your branch, edit the `demo.yml` playbook. The play in that playbook must call the `review.system.iscsi_target_create` role and set the `iscsi_target_create_disk` variable to the value `vdb`. These changes must be committed to your local repository and your new branch pushed to the remote repository. Use `Student@123` as the Git password when you push changes to the remote repository.
- Configure your automation controller with the resources and a job template that you can use to run your playbook to test your Ansible Content Collection and custom automation execution environment, as follows:
 - In the automation controller web UI, create a machine credential resource that enables access to the managed nodes. Use the following settings:

Field	Value
Name	DevOps
Organization	Default
Credential Type	Machine
Username	devops
Password	redhat
Privilege Escalation Method	sudo
Privilege Escalation Username	root

- Create a credential resource that enables access to the Git repository. Use the following settings:

Field	Value
Name	GitLab
Organization	Default
Credential Type	Source Control
Username	student
SCM Private Key	Content of the <code>/home/student/.ssh/gitlab_rsa</code> file.

- Create an inventory resource named `Review 3` that belongs to the `Default` organization.

Field	Value
Name	Review 3
Organization	Default

- Add the `serverc.lab.example.com` managed node to the `Review 3` inventory.

Field	Value
Name	serverc.lab.example.com

- Create a project resource. Use the following settings:

Field	Value
Name	iSCSI
Organization	Default
Source Control Type	Git
Source Control URL	git@git.lab.example.com:student/iscsi.git
Source Control Credential	GitLab
Options	Allow Branch Override <i>(selected)</i>

- Create an automation execution environment resource for your custom automation execution environment. Use the following settings:

Field	Value
Name	Review 3
Image	hub.lab.example.com/system/ee-review-rhel8:v1.0
Pull	Always pull the container before running.
Organization	Default
Registry credential	Automation Hub Container Registry

- Create a job template resource that you can use to run your playbook on the serverc.lab.example.com managed node. Use the following settings:

Field	Value
Name	Configure iSCSI
Inventory	Review 3
Project	iSCSI
Execution environment	Review 3
Source Control Branch	review3
Playbook	demo.yml
Credentials	DevOps

- When everything is configured, start a job from the `Configure iSCSI` job template resource. To confirm that the job is successful, you can verify that the `/etc/target/saveconfig.json` file is created on the `serverc.lab.example.com` machine.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade review-cr3
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr3
```

► Solution

Creating and Using Ansible Content Collections and Automation Execution Environments

Create an Ansible Content Collection and an automation execution environment, and develop a playbook to test the automation execution environment from automation controller.

Outcomes

- Create an Ansible Content Collection that includes roles.
- Create a custom automation execution environment.
- Upload an Ansible Content Collection and an automation execution environment to a private automation hub.
- Configure automation controller to use a custom automation execution environment from a private automation hub.
- Create automation controller resources such as credentials, projects, and inventories.
- Create and launch a new job template using a specific automation execution environment.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates the `/home/student/review-cr3/` directory and creates a Git repository that includes a playbook to test your work. Use `Student@123` as the Git password when you push changes to this remote repository.

```
[student@workstation ~]$ lab start review-cr3
```

The `/home/student/review-cr3/lab-resources.txt` file lists all the resources that you configure during the exercise. You can use that file to copy and then paste those resources.

To log in to the web UI of private automation hub, use `student` as the username and `redhat123` as the password. To log in to the web UI of automation controller, use `admin` as the username and `redhat` as the password. Use the following URLs to access those services:

- Private automation hub: <https://hub.lab.example.com>
- Automation controller: <https://controller.lab.example.com>

You can log in to each of the computer systems that you manage with the `student` user, with `student` as the password. The password for the `root` user account on those machines is `redhat`.

- On the workstation machine, in the /home/student/review-cr3/ directory, create an Ansible Content Collection named `review.system`. Create the `meta/runtime.yml` file and set the `requires_ansible` parameter to `>=2.9.10`.

- Change to the /home/student/review-cr3/ directory and then run the `ansible-galaxy collection init` command to create the collection:

```
[student@workstation ~]$ cd ~/review-cr3/  
[student@workstation review-cr3]$ ansible-galaxy collection init review.system  
- Collection review.system was created successfully
```

- Create the `~/review-cr3/review/system/meta/` directory.

```
[student@workstation review-cr3]$ mkdir review/system/meta
```

- Create the `~/review-cr3/review/system/meta/runtime.yml` file with the following content:

```
---  
requires_ansible: '>=2.9.10'
```

- Copy the `iscsi_target_create` role to the collection's `roles/` directory. Declare the `ansible.posix` collection as a dependency. Build the collection and then copy the resulting `.tar.gz` file to the `/home/student/review-cr3/` directory.

- Copy the role that is provided to you in the `~/review-cr3/roles/` directory to the `~/review-cr3/review/system/roles/` directory:

```
[student@workstation review-cr3]$ ls roles/  
iscsi_target_create  
[student@workstation review-cr3]$ cp -r roles/iscsi_target_create \  
> review/system/roles/
```

- Edit the `review/system/galaxy.yml` file to declare the `ansible.posix` collection as a required dependency of your new collection:

```
...output omitted...  
# Collections that this collection requires to be installed for it to be usable.  
# The key of the dict is the  
# collection label 'namespace.name'. The value is a version range  
# L(specifiers,https://python-semanticversion.readthedocs.io/en/latest/  
#requirement-specification). Multiple version  
# range specifiers can be set and are separated by ','  
dependencies:  
  ansible.posix: '>=1.0.0'  
...output omitted...
```

- In the `~/review-cr3/review/system/` directory, build the collection.

```
[student@workstation review-cr3]$ cd review/system/
[student@workstation system]$ ansible-galaxy collection build
Created collection for review.system at /home/student/review-cr3/review/system/
review-system-1.0.0.tar.gz
```

- 2.4. Copy the resulting `review-system-1.0.0.tar.gz` file from the `~/review-cr3/review/system/` directory to the `~/review-cr3/` directory.

```
[student@workstation system]$ cp review-system-1.0.0.tar.gz ~/review-cr3/
```

3. Publish the `review.system` collection. Use the private automation hub web UI to create the `review` namespace. Upload the `review.system` collection to that namespace, and approve the collection after you upload it.
- 3.1. Open a web browser and navigate to `https://hub.lab.example.com`. Log in with `student` as the username and `redhat123` as the password.
 - 3.2. To create the namespace, navigate to **Collections > Namespaces** and then click **Create**.
- Create the namespace by using the following information and then click **Create**.

Field	Value
Name	review
Namespace owners	Content Developers



Important

The Content Developers group must be a namespace owner for group members, such as the `student` user, to upload to the namespace.

- 3.3. Upload the collection from the `/home/student/review-cr3/review-system-1.0.0.tar.gz` file to the `review` namespace.
To do so, click **Upload collection**, select the `/home/student/review-cr3/review-system-1.0.0.tar.gz` file, and then click **Upload**. Wait for the upload to complete.
- 3.4. To approve the collection, navigate to **Collections > Approval** and then click **Approve**.
- 3.5. To confirm that the collection is published, navigate to **Collections > Collections**, and then select **Published** in **Filter by repository**. The `system` collection is displayed.
4. Prepare the configuration files to build a custom automation execution environment.
 - Create the `~/review-cr3/ee-build/` directory to store the configuration files.
 - Create an `ee-build/execution-environment.yml` file to control the build process. An example file is available in the `~/review-cr3/examples/` directory.
 - Set the `hub.lab.example.com/ee-minimal-rhel8:latest` container image as the base image.
 - Set the `hub.lab.example.com/ansible-builder-rhel8:latest` container image as the builder image.

- 4.1. Change to the ~/review-cr3 directory and create the ee-build/ directory.

```
[student@workstation system]$ cd ~/review-cr3/  
[student@workstation review-cr3]$ mkdir ee-build
```

- 4.2. Copy the ~/review-cr3/examples/execution-environment.yml example file to the ee-build/ directory.

```
[student@workstation review-cr3]$ cp examples/execution-environment.yml ee-build/
```

- 4.3. Edit the ee-build/execution-environment.yml file.

Set the EE_BASE_IMAGE parameter to hub.lab.example.com/ee-minimal-rhel8:latest and the EE_BUILDER_IMAGE parameter to hub.lab.example.com/ansible-builder-rhel8:latest.

Remove the python and system parameters from the dependencies section.

The completed ee-build/execution-environment.yml file should consist of the following content:

```
---  
version: 1  
  
build_arg_defaults:  
  EE_BASE_IMAGE: hub.lab.example.com/ee-minimal-rhel8:latest  
  EE_BUILDER_IMAGE: hub.lab.example.com/ansible-builder-rhel8:latest  
  
ansible_config: ansible.cfg  
  
dependencies:  
  galaxy: requirements.yml
```

5. Create the ee-build/requirements.yml file and configure it to install the review.system and community.general collections into the automation execution environment. You do not need to declare the ansible.posix collection because the review.system collection already specifies it as a dependency.

- 5.1. In the ~/review-cr3/ee-build/ directory, create the requirements.yml file to list the collections that you want in the new automation execution environment. The completed ee-build/requirements.yml file should consist of the following content:

```
---  
collections:  
  - name: review.system  
  - name: community.general
```

6. Create an ee-build/ansible.cfg file to configure access to the private automation hub at <https://hub.lab.example.com> so that the build process can retrieve the collections. An example file is available in the ~/review-cr3/examples/ directory. Get a token from the private automation hub web UI and then edit the ansible.cfg file to update the token parameters.

Chapter 10 | Comprehensive Review

- 6.1. Copy the `~/review-cr3/examples/ansible.cfg` file to the `ee-build/` directory.

```
[student@workstation review-cr3]$ cd ee-build  
[student@workstation ee-build]$ cp ~/review-cr3/examples/ansible.cfg .
```

You update the `token` parameters in that file in the next step.

- 6.2. Retrieve the API token from the private automation hub web UI at `https://hub.lab.example.com`.

Navigate to **Collections > API token management** and then click **Load token**. Click the **Copy to clipboard** icon.

Edit the `~/review-cr3/ee-build/ansible.cfg` file and then paste the token from the clipboard as the value for the `token` parameters. Your token is probably different from the one shown in the following example.

```
[galaxy]  
server_list = published_repo, rh-certified_repo, community_repo  
  
[galaxy_server.published_repo]  
url=https://hub.lab.example.com/api/galaxy/content/published/  
token=c6aec560d9d0a8006dc6d8f258092e09a53fd7bd  
  
[galaxy_server.rh-certified_repo]  
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/  
token=c6aec560d9d0a8006dc6d8f258092e09a53fd7bd  
  
[galaxy_server.community_repo]  
url=https://hub.lab.example.com/api/galaxy/content/community/  
token=c6aec560d9d0a8006dc6d8f258092e09a53fd7bd
```

7. Install the `ansible-builder` package on `workstation`. In the `~/review-cr3/ee-build/` directory, use the `ansible-builder` command to create the `ee-build/context/` directory by performing the first stage of the build process. Configure the container image to include your private automation hub's TLS CA certificate. This build process can then retrieve collections from the lab environment's private automation hub. Use the two provided files as follows:

- Copy `~/review-cr3/Containerfile` to `~/review-cr3/ee-build/context/Containerfile`.
- Copy `/etc/pki/tls/certs/classroom-ca.pem` to `~/review-cr3/ee-build/context/classroom-ca.pem`.

- 7.1. Use the `yum` command to install the `ansible-builder` package.

```
[student@workstation ee-build]$ sudo yum install ansible-builder  
[sudo] password for student: student  
...output omitted...
```

- 7.2. Run the `ansible-builder create` command from the `ee-build/` directory:

```
[student@workstation ee-build]$ ansible-builder create
Complete! The build context can be found at: /home/student/review-cr3/ee-build/
context
```

- 7.3. Copy the /home/student/review-cr3/Containerfile and /etc/pki/tls/certs/classroom-ca.pem files into the context/ directory.

```
[student@workstation ee-build]$ cp ~/review-cr3/Containerfile context/
[student@workstation ee-build]$ cp /etc/pki/tls/certs/classroom-ca.pem context/
```

8. Build the automation execution environment container image. Set hub.lab.example.com/system/ee-review-rhel8:v1.0 as a tag for the container image. After you build the container image for your automation execution environment, push the container image to the private automation hub.

- 8.1. Use the podman login command to log in to the private automation hub at hub.lab.example.com:

```
[student@workstation ee-build]$ podman login hub.lab.example.com
Username: student
Password: redhat123
Login Succeeded!
```

- 8.2. Run the podman build command from the ee-build/ directory. Add the -t option to specify the hub.lab.example.com/system/ee-review-rhel8:v1.0 tag.

```
[student@workstation ee-build]$ podman build -f context/Containerfile \
> -t hub.lab.example.com/system/ee-review-rhel8:v1.0 context
```

- 8.3. Confirm that the new container image is available locally:

```
[student@workstation ee-build]$ podman images
REPOSITORY                                TAG      IMAGE ID      CREATED        SIZE
hub.lab.example.com/system/ee-review-rhel8  v1.0    65d5f183c35  2 minutes ago  435 MB
...output omitted...
```

- 8.4. Push the container image to the private automation hub:

```
[student@workstation ee-build]$ podman push \
> hub.lab.example.com/system/ee-review-rhel8:v1.0
```

9. Clone the https://git.lab.example.com/student/iscsi.git Git repository into the /home/student/git-repos directory and then create the review3 branch.

- 9.1. From a terminal, create the /home/student/git-repos directory if it does not exist, and then change into it.

```
[student@workstation ee-build]$ mkdir -p ~/git-repos/
[student@workstation ee-build]$ cd ~/git-repos/
```

Chapter 10 | Comprehensive Review

- 9.2. Clone the `https://git.lab.example.com/student/iscsi.git` repository and then change into the cloned repository:

```
[student@workstation git-repos]$ git clone \
> https://git.lab.example.com/student/iscsi.git
Cloning into 'iscsi'...
...output omitted...
[student@workstation git-repos]$ cd iscsi/
```

- 9.3. Create the `review3` branch and switch to it.

```
[student@workstation iscsi]$ git checkout -b review3
Switched to a new branch 'review3'
```

- 10.** In the branch, update the `demo.yml` playbook. The playbook must call the `review.system.iscsi_target_create` role. Set the `iscsi_target_create_disk` role variable to `vdb`. When done, commit and push your changes.

- 10.1. Edit the `demo.yml` playbook. The completed file should consist of the following content:

```
---
- name: Testing the iscsi_target_create role in the review.system collection
  hosts: all
  become: true

  tasks:
    - name: Ensure the iSCSI target is configured
      ansible.builtin.include_role:
        name: review.system.iscsi_target_create
    vars:
      iscsi_target_create_disk: vdb
```

- 10.2. Commit and then push your changes. If prompted, use `Student@123` as the Git password.

```
[student@workstation iscsi]$ git add demo.yml
[student@workstation iscsi]$ git commit -m "Updating the demo playbook"
[review3 6657707] Updating the demo playbook
 1 file changed, 2 insertions(+), 2 deletions(-)
[student@workstation iscsi]$ git push -u origin review3
Password for 'https://student@git.lab.example.com': Student@123
...output omitted...
To git.lab.example.com:student/iscsi.git
 * [new branch]      review3 -> review3
Branch 'review3' set up to track remote branch 'review3' from 'origin'.
```

- 11.** Access the automation controller web UI at `https://controller.lab.example.com` to create the resources. Use `admin` as the username and `redhat` as the password.

- 11.1. To create the machine credential resource, navigate to **Resources > Credentials**, click **Add**, complete the form with the following information, and then click **Save**.

Field	Value
Name	DevOps
Organization	Default
Credential Type	Machine
Username	devops
Password	redhat
Privilege Escalation Method	sudo
Privilege Escalation Username	root

- 11.2. To create the source control credential resource, navigate to **Resources > Credentials**, click **Add**, complete the form with the following information, and then click **Save**.

Field	Value
Name	GitLab
Organization	Default
Credential Type	Source Control
Username	student
SCM Private Key	Content of the /home/student/.ssh/gitlab_rsa file.

**Note**

If you choose to browse for the file, then right-click anywhere in the directory navigation and select **Show Hidden Files**. With this option enabled you can see the .ssh directory in the /home/student directory.

- 11.3. To create the **Review 3** inventory resource, navigate to **Resources > Inventories**, click **Add > Add Inventory**, specify the name of the inventory, and then click **Save**.

Field	Value
Name	Review 3
Organization	Default

- 11.4. Within the **Review 3** inventory, click the **Hosts** tab and click **Add**. Create the host with the **serverc.lab.example.com** name and then click **Save**.

Field	Value
Name	serverc.lab.example.com

Chapter 10 | Comprehensive Review

- 11.5. To create the iSCSI project resource, navigate to **Resources > Projects**, click **Add**, complete the form with the following information, and then click **Save**.

Field	Value
Name	iSCSI
Organization	Default
Source Control Type	Git
Source Control URL	git@git.lab.example.com:student/iscsi.git
Source Control Credential	GitLab
Options	Allow Branch Override (selected)

- 11.6. To create the **Review 3** automation execution environment resource, navigate to **Administration > Execution Environments**, click **Add**, complete the form with the following information, and then click **Save**.

Field	Value
Name	Review 3
Image	hub.lab.example.com/system/ee-review-rhel8:v1.0
Pull	Always pull the container before running.
Organization	Default
Registry credential	Automation Hub Container Registry

- 11.7. To create the **Configure iSCSI** job template resource, navigate to **Resources > Templates**, click **Add > Add job template**, complete the form with the following information, and then click **Save**.

Field	Value
Name	Configure iSCSI
Inventory	Review 3
Project	iSCSI
Execution environment	Review 3
Source Control Branch	review3
Playbook	demo.yml
Credentials	DevOps

12. In the automation controller web UI, start a job from the **Configure iSCSI** job template. Verify that the `demo.yml` playbook correctly configures the `serverc.lab.example.com` managed node.
- 12.1. Navigate to **Resources > Templates** and then click the **Launch Template** icon for the **Configure iSCSI** job template. Wait for the job to complete.
 - 12.2. Confirm the successful execution of the job by verifying that the role created the `/etc/target/saveconfig.json` file on the managed node.

```
[student@workstation iscsi]$ ssh serverc ls /etc/target/saveconfig.json  
/etc/target/saveconfig.json
```

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade review-cr3
```

Finish

On the `workstation` machine, change to the `student` user home directory and use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish review-cr3
```

