For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Red Hat
Training and Certification

Student Workbook (ROLE)

Red Hat Ansible Automation Platform 2.3 DO457

Network Automation with Red Hat Ansible Automation Platform

Edition 1

DO457-RHAAP2.3-en-1-20231115
Copyright ©2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Red Hat
Learning Community

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 comunity 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

Red Hat

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.

Copyright ©2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Network Automation with Red Hat Ansible Automation Platform

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Red Hat Ansible Automation Platform 2.3 DO457 Network Automation with Red Hat Ansible Automation Platform
Edition 1 20231115
Publication date 20231115

Authors:	Ed Parenti, Alejandra Ramírez Palacios, Michael Phillips, Michael Kelly
Course Architect:	Steve Bonneville
DevOps Engineer:	Trey Feagle
Editor:	David O'Brien

© 2023 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are © 2023 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: **Artur Glogowski, Wasim Raja, Chetan Tiwary**

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Document Conventions **ix**
Admonitions ... ix
Inclusive Language ... x

Introduction **xi**
Network Automation with Red Hat Ansible Automation Platform ... xi
Orientation to the Classroom Environment ... xii
Performing Lab Exercises ... xvi

1. Introducing Red Hat Ansible Automation Platform **1**
Network Administration and Red Hat Ansible Automation Platform ... 2
Quiz: Network Administration and Red Hat Ansible Automation Platform ... 10
Preparing for Ansible Development ... 14
Guided Exercise: Preparing for Ansible Development ... 23
Summary ... 28

2. Implementing an Ansible Playbook **29**
Building an Ansible Inventory ... 30
Guided Exercise: Building an Ansible Inventory ... 38
Configuring Automation Content Navigator ... 43
Guided Exercise: Configuring Automation Content Navigator ... 47
Writing and Running Playbooks ... 50
Guided Exercise: Configuring Device Login Banners ... 62
Implementing Multiple Plays ... 65
Guided Exercise: Creating a Cross-vendor Configuration Backup ... 72
Lab: Implementing an Ansible Playbook ... 77
Summary ... 88

3. Managing and Running Playbooks **89**
Managing Ansible Project Materials Using Git ... 90
Guided Exercise: Managing Ansible Project Materials Using Git ... 101
Running Playbooks in Automation Controller ... 110
Guided Exercise: Running a Playbook in Automation Controller to Configure SNMP ... 118
Troubleshooting Playbooks ... 123
Guided Exercise: Troubleshooting Playbooks ... 139
Troubleshooting Ansible Network Communication ... 150
Guided Exercise: Troubleshooting Ansible Network Communication ... 154
Lab: Managing and Running Playbooks ... 166
Summary ... 176

4. Managing Variables and Facts **177**
Managing Variables ... 178
Guided Exercise: Making Flexible Playbooks by Using Variables ... 187
Managing Facts ... 197
Guided Exercise: Gathering Facts from Managed Network Devices ... 205
Using Surveys to Set Variables ... 212
Guided Exercise: Configuring System Settings Using Surveys ... 218
Transforming Data with Filters ... 224
Guided Exercise: Using Filters to Parse Command Output ... 230
Lab: Managing Variables and Facts ... 239
Summary ... 257

5. Implementing Task Control **259**
Writing Loops and Conditional Tasks ... 260
Guided Exercise: Configuring and Verifying Specific Device Settings ... 268
Handling Task Failure ... 274
Guided Exercise: Recovering From a Failed Configuration ... 279
Building a Workflow Job Template ... 295

Guided Exercise: Back up, Configure, Recover from Errors, and Restore from Backups 303
Lab: Implementing Task Control 308
Summary 326

6. Simplifying Playbooks with Roles and Ansible Content Collections 327
Including and Importing Files 328
Guided Exercise: Configuring a Multi-vendor Network 335
Ansible Role Structure 342
Quiz: Ansible Role Structure 349
Creating Roles 353
Guided Exercise: Creating a Network Automation Role 360
Deploying Roles from External Content Sources 365
Guided Exercise: Running a Playbook that Requires a Role 370
Getting Roles and Modules from Ansible Content Collections 377
Guided Exercise: Installing and Using an Ansible Content Collection 384
Lab: Simplifying Playbooks with Roles and Ansible Content Collections 392
Summary 407

7. Automating Network Administration Tasks 409
Gaining Infrastructure Awareness 410
Guided Exercise: Gaining Infrastructure Awareness 417
Generating Configuration Settings from Jinja2 Templates 427
Guided Exercise: Generating Configuration Settings from Jinja2 Templates 432
Simplifying Tasks with Platform-independent Modules 444
Guided Exercise: Simplifying Tasks with Platform-independent Modules 454
Performing Rolling Configuration Updates 466
Guided Exercise: Performing Rolling Configuration Updates 473
Summary 485

8. Comprehensive Review 487
Comprehensive Review 488
Lab: Installation and Configuration 491
Lab: Creating Playbooks 502
Lab: Scaling Playbooks 519
Lab: Running Playbooks with Automation Controller 534

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Introduction

Network Automation with Red Hat Ansible Automation Platform

Network Automation with Red Hat Ansible Automation Platform (DO457) is designed for network administrators or infrastructure automation engineers who want to use network automation to centrally manage the switches, routers, and other devices in the organization's network infrastructure.

Course Objectives

- Automate the configuration and management of your network infrastructure using Red Hat Ansible Automation Platform.
- Remotely automate the configuration of network devices, test and validate the current network state, and perform compliance checks to detect and correct configuration drift.

Audience

- Network administrators, network automation engineers, and infrastructure automation engineers who are responsible for deploying, managing, and automating the network infrastructure of their organization or enterprise.

Prerequisites

- Experience with network administration, including a solid understanding of TCP/IP, routers, and managed switches. Students should be familiar with managing network devices from the command line, preferably with one or more of Cisco IOS/IOS XE, IOS XR, or NX-OS; Juniper Junos; or Arista EOS.
- The students will work with text files and run commands in a Red Hat Enterprise Linux environment. A working knowledge of Linux, including how to edit text files and run commands from the shell, and how to use SSH to log in to remote systems. Knowledge equivalent to "Red Hat System Administration I" (RH124) or better is recommended.
- Prior Ansible knowledge is not required.

Orientation to the Classroom Environment

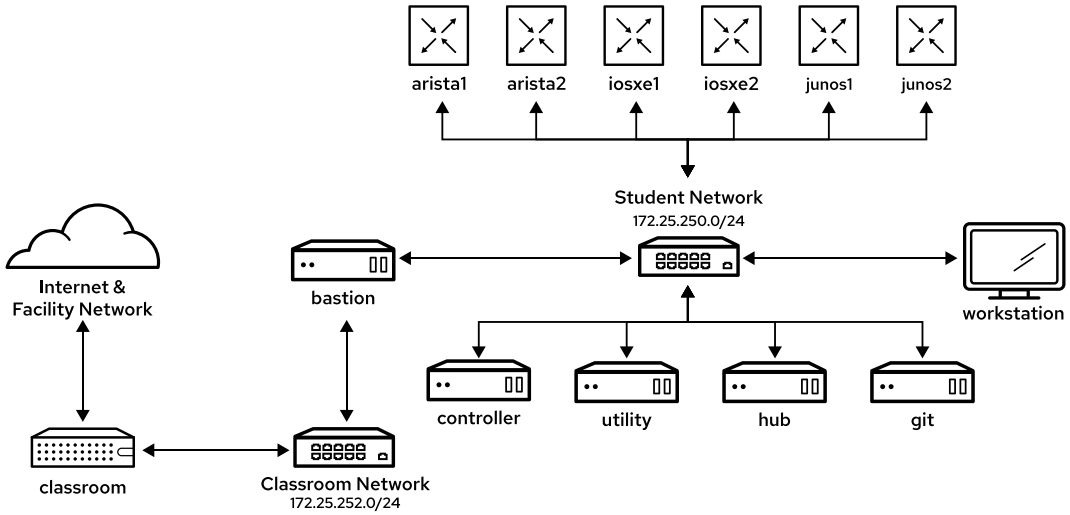


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is `workstation`.

The `bastion` machine must always be running.

All machines are in the `lab.example.com` DNS domain.

Classroom Machines

Machine name	IP addresses	Role
`bastion.lab.example.com`	172.25.250.254	Router that links VMs to central servers
`workstation.lab.example.com`	172.25.250.9	Graphical workstation used for system administration
`utility.lab.example.com`	172.25.250.220	Server hosting course-specific support services
`git.lab.example.com`	172.25.250.5	Gitlab server
`hub.lab.example.com`	172.25.250.6	Private automation hub
`controller.lab.example.com`	172.25.250.7	Automation controller
`iosxe1.lab.example.com`	172.25.250.20	Cisco IOS XE node 1
`iosxe2.lab.example.com`	172.25.250.21	Cisco IOX XE node 2

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Machine name	IP addresses	Role
`junos1.lab.example.com`	172.25.250.22	Juniper Networks Junos node 1
`junos2.lab.example.com`	172.25.250.23	Juniper Networks Junos node 2
`arista1.lab.example.com`	172.25.250.24	Arista Network EOS node 1
`arista2.lab.example.com`	172.25.250.25	Arista Networks EOS node 2

Controlling Your Systems

You are assigned remote computers in a Red Hat Online Learning (ROLE) classroom. Self-paced courses are accessed through a web application that is hosted at `rol.redhat.com [http://rol.redhat.com]`. Log in to this site with your Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

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

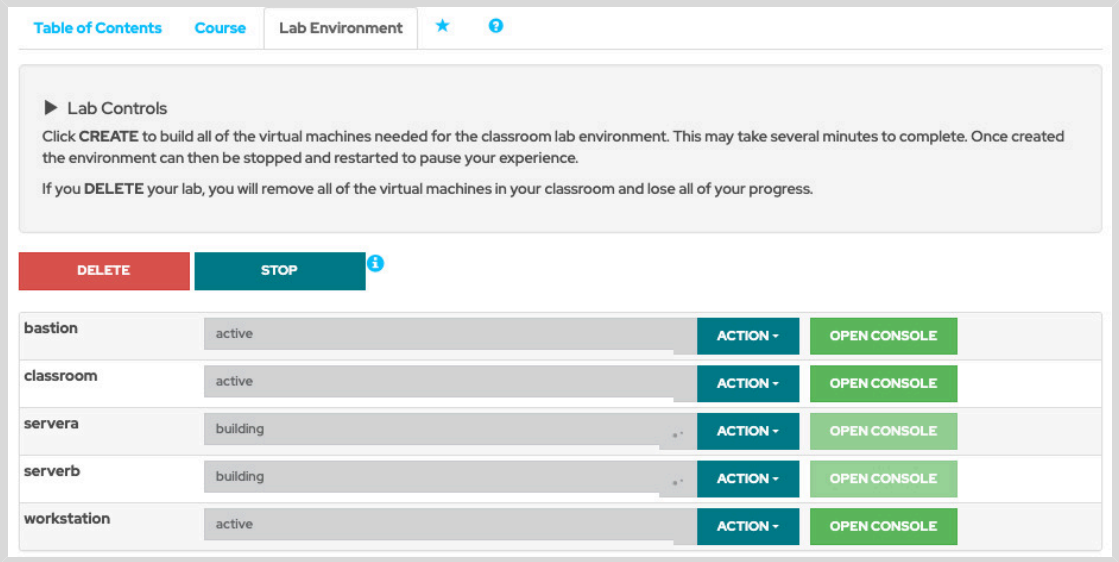


Figure 0.2: An example course lab environment management page

Machine States

Virtual machine state	Description
building	The virtual machine is being created.
active	The virtual machine is running and available. If it just started, it still might be starting services.
stopped	The virtual machine is shut down. On starting, the virtual machine boots into the same state it was in before shutdown. The disk state is preserved.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Classroom Actions

Button or action	Description
CREATE	Create the ROLE classroom. Creates and starts all the virtual machines that are needed for this classroom.
CREATING	The ROLE classroom virtual machines are being created. Creation can take several minutes to complete.
DELETE	Delete the ROLE classroom. Destroys all virtual machines in the classroom. **All saved work on those systems' disks is lost.**
START	Start all virtual machines in the classroom.
STARTING	All virtual machines in the classroom are starting.
STOP	Stop all virtual machines in the classroom.

Machine Actions

Button or action	Description
OPEN CONSOLE	Connect to the system console of the virtual machine in a new browser tab. You can log in directly to the virtual machine and run commands, when required. Normally, log in to the `workstation` virtual machine only, and from there, use `ssh` to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.
ACTION > Shutdown	Gracefully shut down the virtual machine, preserving disk contents.
ACTION > Power Off	Forcefully shut down the virtual machine, preserving disk contents. This action is equivalent to removing the power from a physical machine.
ACTION > Reset	Forcefully shut down the virtual machine and reset associated storage to its initial state. **All saved work on that system's disks is lost.**

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

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION** > **Reset** on every virtual machine in the list.

If you want to return the classroom environment to its original state at the start of the course, then click **DELETE** to remove the entire classroom environment. After the lab is deleted, then click **CREATE** to provision a new set of classroom systems.

Warning
The **DELETE** operation cannot be undone. All completed work in the classroom environment is lost.

The Auto-stop and Auto-destroy Timers

The Red Hat Online Learning enrollment entitles you to a set allotment of computer time. To help to conserve your allotted time, the ROLE classroom uses timers, which shut down or delete the classroom environment when the appropriate timer expires.

To adjust the timers, locate the two + buttons at the bottom of the course management page. Click the auto-stop + button to add another hour to the auto-stop timer. Click the auto-destroy + button to add another day to the auto-destroy timer. Be careful to keep the timers set while you are working, so that your environment is not unexpectedly shut down. Be careful not to set the timers unnecessarily high, which could waste your subscription time allotment.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Performing Lab Exercises

You might see the following lab activity types 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 to test your learning. You work through a set of high-level steps, based on the guided exercises in that chapter, but the steps do not walk you through every command. A solution is provided with 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, and might cover content from the entire course. You work through a specification of what to accomplish in the activity, without receiving the specific steps to do so. Again, a solution is provided with 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 a specified activity name from 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 as follows:

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

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

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

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

finish
: The `finish` action cleans up resources that were configured during the exercise. You can perform 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 then press the `Tab` key twice.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 1

Introducing Red Hat Ansible Automation Platform

Goal

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

Objectives

- Describe the motivation for automating network administration tasks with Ansible, fundamental Ansible concepts, and the basic architecture of Ansible.
- Install automation content navigator and other development tools on a workstation for Ansible automation development and testing.

Sections

- Network Administration and Red Hat Ansible Automation Platform (and Quiz)
- Preparing for Ansible Development (and Guided Exercise)

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Network Administration and Red Hat Ansible Automation Platform

Objectives

- Describe the motivation for automating network administration tasks with Ansible, fundamental Ansible concepts, and the basic architecture of Ansible.

Automation and Network Administration

For many years, network engineering and administration has relied mostly on manual tasks performed through graphical or command-line user interfaces. Network engineers often work from checklists, configuration templates, or a memorized routine to perform standard tasks.

This approach is error-prone and difficult to maintain. It is easy to skip steps and make mistakes. Verification that the steps were performed properly or that they result in the expected outcome is often limited.

By managing devices manually and independently, it is easy for many devices that are supposed to have the same configuration to be different in minor (or major) ways. This can make maintenance more difficult and introduce errors or instability into your infrastructure.

Automation can help avoid the problems caused by manual configuration.

Automation can help you perform the following tasks:

- Ensure that your network infrastructure is quickly and correctly deployed and configured.
- Perform routine or repetitive tasks with minimal effort.
- Free up your time and enable you to focus on more critical tasks.
- Reduce risks associated with network infrastructure upgrades or changes.
- Respond to service outages and quickly restore service.
- Roll out security configuration updates quickly and effectively.

Using Ansible for NetDevOps

NetDevOps means applying the methods, strategies, and best practices of DevOps to networking. Using Ansible for network automation provides you with implementation flexibility and the ability to mitigate risks. Network automation is a foundational component of a NetDevOps culture, but you do not have to take an all or nothing approach.

With Ansible, you can implement automation in phases. You can start with read-only automation by performing the following tasks:

- Gather an inventory of your network.
- Create backups of device configurations.
- Gather the configuration states of your network devices.
- View specific statistics debugging information to help you resolve issues.

When you are ready, you can continue your automation adoption journey by performing the following tasks:

- Automate the configuration of certain devices or services.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Automate specific standard configuration settings, such as logging and reporting, or security settings.

You can then continue to roll out automation to your entire infrastructure.

Infrastructure as Code

A good automation system allows you to implement *infrastructure as code* (IaC) practices. IaC means that you can use a machine readable automation language to define and describe the required state of your infrastructure.

Ideally, this automation language should also be easy for humans to read, because then you can readily understand the current state. This code is then applied to your infrastructure to ensure that it is in the expected or desired state.

If the automation language is represented as simple text files, then it is easy to manage in a version control system. The advantage of this is that every change can be uploaded to the version control system, ensuring that you have an ongoing history of changes. If you want to revert to an earlier known-good configuration, then you can pull that version and apply it to your infrastructure.

IaC techniques help you to accurately and consistently manage your network infrastructure, and provide a foundation for NetDevOps workflows. You can define the desired configuration in the automation language, and then a peer can review the accuracy of those changes. You can then use that automation to ensure that the network is in the state that you expect.

Mitigating Human Error

Using task automation and IaC practices can help you ensure that your network is consistently configured. This means that you need to become accustomed to making changes by updating your automation code, rather than manually applying the changes to devices. Otherwise, you run the risk of losing manually applied changes the next time you apply changes using automation.

You can use code review, peer review by multiple subject matter experts, and document the procedure within the automation content to reduce your operational risks.

To mitigate human error, changes must be made through automation.

What Is Ansible?

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

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

The following goals were used during the design of Ansible.

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

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Think Declaratively

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

Ansible has a number of important strengths:

- *Cross-platform support*: Ansible provides agentless support for Linux, Windows, UNIX, and network devices, in physical, virtual, cloud, and container environments.
- *Precise application descriptions*: Every change can be made by Ansible Playbooks, and every aspect of your application environment can be described and documented.
- *Easy to manage in version control*: Ansible Playbooks and projects are plain text. They can be treated like source code and placed in your existing version control system.
- *Support for dynamic inventories*: The list of machines that Ansible manages can be dynamically updated from external sources to capture the correct, current list of all managed nodes all the time, regardless of infrastructure or location.
- *Orchestration that integrates with other systems*: Puppet, Jenkins, Red Hat Satellite, and other systems that exist in your environment can be leveraged and integrated into your Ansible workflow.
- *Human-readable automation*: Ansible Playbooks, written as YAML text files, are easy to read and help ensure that everyone understands what the playbooks do.

Figure 1.1: Ansible across the application lifecycle

Communication is the key to DevOps and NetDevOps. Ansible is the first automation language that can be read and written across IT.

Ansible Concepts and Architecture

The Ansible architecture consists of two types of machines: *control nodes* and *managed nodes*. Ansible is installed and run from a control node, and this machine also has copies of your Ansible project files.

The Ansible architecture is agentless. Typically, when you run an Ansible Playbook, the control node connects to the managed host by using SSH by default. For managed nodes that are network devices, you might access the device command-line interface (CLI) over SSH, send XML messages over SSH, or connect to an API using the HTTP or HTTPS protocol.

Because the Ansible architecture is agentless, you do not need to have an Ansible-specific agent installed on managed nodes, and do not need to configure any additional communication between the control node and managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Inventory Management

Managed nodes are listed in an *inventory*, which also organizes those systems into groups for easier collective management. You can define the inventory statically in a text file, or dynamically using scripts that obtain group and host information from external sources.

Playbooks and Plays

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

Tasks and Modules

Each task runs a *module*, a small piece of code (written in Python, PowerShell, or some other language), with specific arguments. Each module is essentially a tool in your toolkit. Ansible ships with a core set of useful modules, and you can install additional modules using Ansible content collections. Modules can perform a wide variety of automation tasks, such as modifying system files, installing software, or making API calls.

Idempotent State Management

Tasks, plays, and playbooks are designed to be *idempotent*. This means that you can safely run a playbook against the same nodes multiple times. When your systems are in the correct state, the playbook makes no changes when you run it.

Using Plug-ins

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

Red Hat Ansible Automation Platform 2 Components

Red Hat Ansible Automation Platform 2 includes a number of distinct components, which 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 needed for automation code, such as loops, conditionals, and other Ansible imperatives. It also provides the framework and basic command-line tools to drive automation.

Red Hat Ansible Automation Platform 2.3 provides Ansible Core 2.14 in the `ansible-core` RPM package.

Ansible Content Collections

Historically, Ansible provided many 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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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, such as those belonging to 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, called *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.

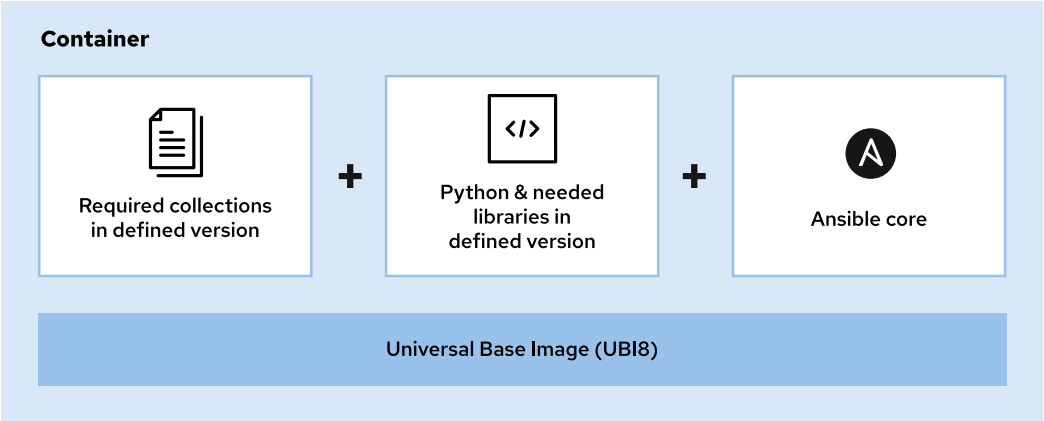


Figure 1.2: Parts of an automation execution environment

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.

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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The automation controller design separates the control node (providing the web UI and API) from the automation execution environments (which run in containers). You can deploy containerized automation execution environments from a container registry. Replacing execution environments for a particular Ansible Playbook is a matter of a simple configuration change in the web UI.

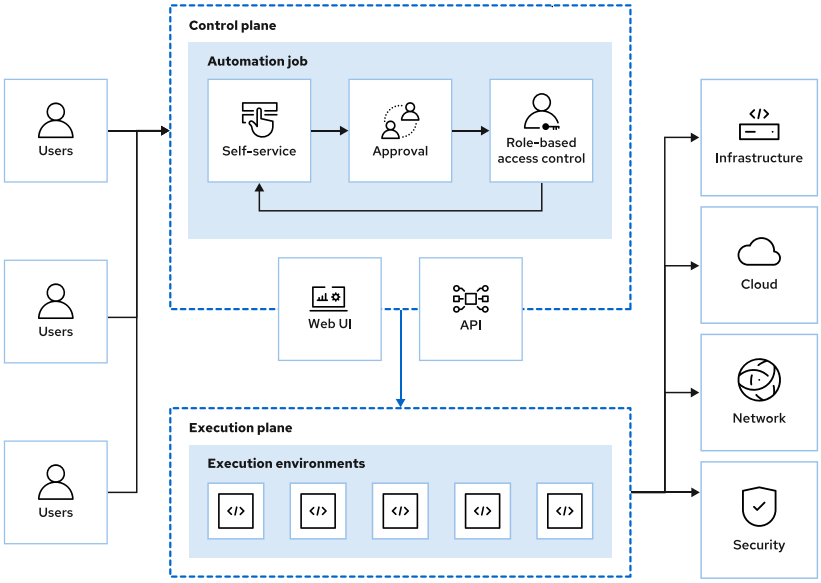


Figure 1.3: Components of automation controller

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

> **Note**
> Automation controller and automation content navigator both support automation execution environments. This simplifies the transition from developing playbooks to running them in production, because you do not 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 of your playbooks from development to production should be almost seamless.

Automation Hub

Automation hub provides a way to manage and distribute automation content. A public service at `console.redhat.com` provides access to Red Hat Ansible Certified Content Collections, which 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 obtain automation content.

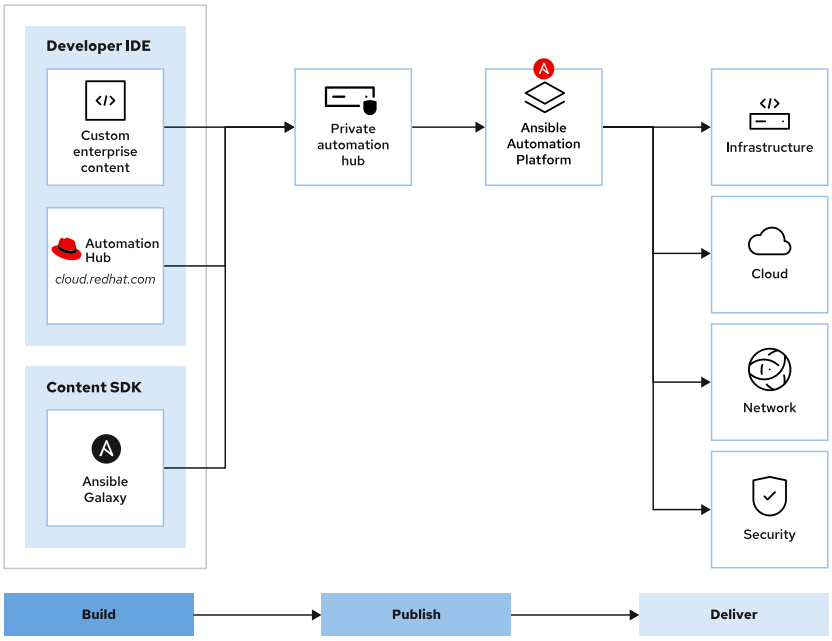


Figure 1.4: Creator experience: Working with Ansible Automation Platform

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 helps 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 often use.

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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

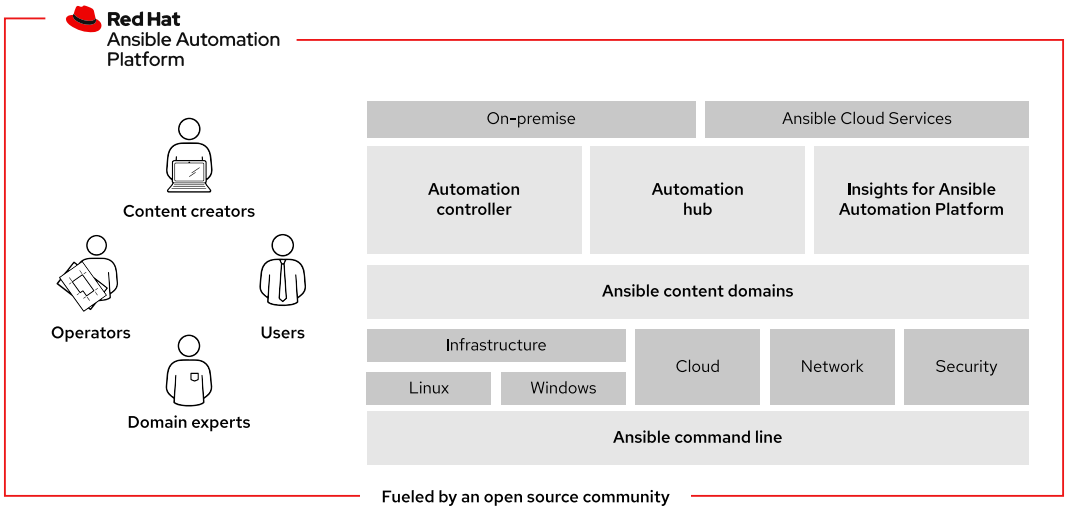


Figure 1.5: Architecture of Red Hat Ansible Automation Platform 2

References

Ansible
https://www.ansible.com

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

Red Hat Ansible Automation Platform
https://www.redhat.com/en/technologies/management/ansible

Ansible for Network Automation - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/index.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Quiz

Network Administration and Red Hat Ansible Automation Platform

Choose the correct answers to the following questions:

1. Which two terms best describe the Ansible architecture? (Choose two.)

a. Agentless
b. Client/Server
c. Stateless
d. Idempotent

2. Which file defines the actions that Ansible performs on managed nodes?

a. Inventory
b. Manifest
c. Playbook
d. Script

3. Which syntax is used to define Ansible Playbooks?

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

4. Which three of the following are components of an Ansible Playbook? (Choose three.)

a. Plays
b. Tasks
c. Procedures
d. Modules

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

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. Which one of the following is NOT a component of Red Hat Ansible Automation Platform?

a. Ansible Core

b. Ansible Content Collections

c. Automation content navigator

d. Red Hat Ansible for Red Hat OpenShift

e. Automation execution environments

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Network Administration and Red Hat Ansible Automation Platform

Choose the correct answers to the following questions:

1. Which two terms best describe the Ansible architecture? (Choose two.)

a. Agentless
b. Client/Server
c. Stateless
d. Idempotent

2. Which file defines the actions that Ansible performs on managed nodes?

a. Inventory
b. Manifest
c. Playbook
d. Script

3. Which syntax is used to define Ansible Playbooks?

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

4. Which three of the following are components of an Ansible Playbook? (Choose three.)

a. Plays
b. Tasks
c. Procedures
d. Modules

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

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. Which one of the following is NOT a component of Red Hat Ansible Automation Platform?

a. Ansible Core

b. Ansible Content Collections

c. Automation content navigator

d. Red Hat Ansible for Red Hat OpenShift

e. Automation execution environments

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Preparing for Ansible Development

Objectives

- Install automation content navigator and other development tools on a workstation for Ansible automation development and testing.

Installing Automation Content Navigator

To use automation content navigator, you must install the `ansible-navigator` RPM package on your control node. If you use the Visual Studio Code (VS Code) integrated development environment (IDE) for playbook development, then consider installing the `ansible-core` and `ansible-lint` RPM packages. Although automation content navigator does not require these packages, VS Code attempts to perform syntax and style checking by running the `ansible-lint` command on playbooks that you open.

You do not need to install these Ansible packages 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, then trial entitlements are available from `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.
  - If your control node runs Red Hat Enterprise Linux 8, then run:

```
[root@host ~]# subscription-manager repos \
--enable ansible-automation-platform-2.3-for-rhel-8-x86_64-rpms
```

  - If your control node runs Red Hat Enterprise Linux 9, then run:

```
[root@host ~]# subscription-manager repos \
--enable ansible-automation-platform-2.3-for-rhel-9-x86_64-rpms
```

Note
You do not need to run these `subscription-manager` commands in the lab environment because your machines are already configured to install packages from a local RPM repository.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

– Install the `ansible-core`, `ansible-navigator`, and `ansible-lint` RPM packages.

```
[root@host ~]# dnf install ansible-core ansible-navigator ansible-lint
```

Running Automation Content Navigator

Use the `ansible-navigator` command to run automation content navigator. 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 disable interactive mode. When using the default `interactive` mode (rather than `stdout` mode), you can interactively examine playbook output.

The following table briefly describes the subcommands available to automation content navigator. You can run most subcommands from either the command line or from within an interactive automation content navigator session, but some commands do not support the `-m stdout` option. When running a subcommand in an interactive session, type `:` to start the command, such as `:config`.

Subcommand	Description
`builder`	Build an execution environment (container image).
`collections`	Explore available collections.
`config`	Explore the current Ansible configuration.
`doc`	Review the documentation for a module or plug-in.
`exec`	Run a command within an execution environment.
`images`	Explore execution environment images.
`inventory`	Explore an inventory.
`lint`	Lint a file or directory for common errors and issues.
`replay`	Explore a previous run using a playbook artifact.
`run`	Run a playbook.
`settings`	Review the current `ansible-navigator` settings.
`welcome`	Start at the welcome page.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

By default, automation content navigator uses a special container image known as an automation execution environment.

You can use the `--execution-environment-image` (`--eei`) option to select a specific container image to use as the automation execution environment. For example, you could use the `--eei registry.redhat.io/ansible-automation-platform-23/ee-supported-rhel8:latest` option to specify a container image. The `latest` tag indicates that you want the latest version of that image.

Before you can download an automation execution environment, you might need to log in to a container registry, such as `registry.redhat.io` or a private automation hub.

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

Running Playbooks

You can run playbooks using automation content navigator from the command line with the `ansible-navigator run` command or interactively with the `:run` subcommand. You might enter the following command to run a playbook with automation content navigator (add `-m stdout` to disable interactive mode):

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

PLAY [Sample Playbook] *********************************************************

TASK [Display message] *********************************************************
ok: [localhost] => {
    "msg": "Hello, world!"
}

PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=0   ...  failed=0    skipped=0   ...
```

When you run a playbook in interactive mode, automation content navigator updates the output as the playbook runs.

```
[user@host ~]$ ansible-navigator run hello.yml
```

While the playbook runs, the bottom right of the terminal window displays a status of `Running`. After the playbook finishes, the bottom right of the terminal window displays a status of either `Successful` or `Failed`.

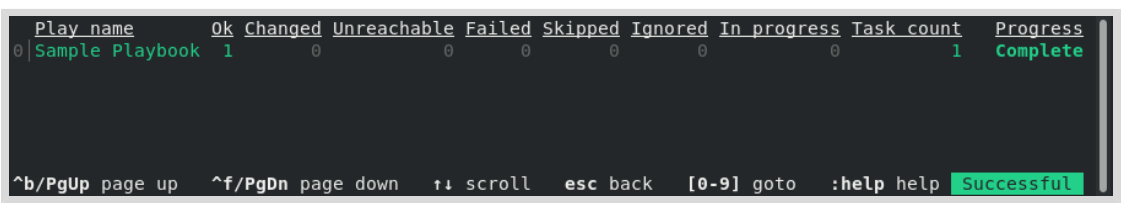


Figure 1.6: Playbook output when using interactive mode

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The display summarizes the status of the tasks of each play in the playbook. You can examine the details for a play by entering the number to the left of the play name.

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

Reviewing Previous Playbook Runs

Automation content navigator provides a 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 uses names such as `site-artifact-2023-06-01T20:09:57.939910+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, using the time zone offset from UTC (`+00:00` indicates that the time was recorded in UTC).

You can review a previous playbook run by using either the `ansible-navigator replay` command from the command line or the `:replay` subcommand in interactive mode. Instead of running a playbook, replaying a playbook displays the output from a previous playbook run.

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. For example, you can disable playbook artifacts by adding the `--pae false` option to the `ansible-navigator` command.

Reading Documentation

Automation content navigator provides access to documentation for plug-ins, such as modules, lookups, and callbacks.

You can access documentation by running the `ansible-navigator doc` command (or the `:doc` subcommand in interactive mode) and passing the name of a module. The following example displays documentation for the `cisco.ios.ios_hostname` module and disables interactive mode. Module documentation describes options and includes examples.

```
[user@host ~]$ ansible-navigator doc cisco.ios.ios_hostname -m stdout
> CISCO.IOS.IOS_HOSTNAME    (/usr/share/ansible/collections/ansible_collectio...

        This module provides declarative management of hostname on
        Cisco IOS devices.

ADDED IN: version 2.7.0 of cisco.ios

  * note: This module has a corresponding action plugin.
```

```
OPTIONS (= is mandatory):

...output omitted...
```

If you do not know the name of a module, then you can run the `ansible-navigator collections` command to interactively explore documentation for Ansible content collections that are available within an automation execution environment. For example, after running the `ansible-navigator collections` command, you might select the `cisco.ios` collection and then display documentation for the `ios_banner` module.

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.

You can also 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:

```
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'
```

Using VS Code to Edit Playbooks

Ansible uses text files and you can create these files using any text editor. This course provides Visual Studio Code (VS Code), available from `https://code.visualstudio.com`, as one possible option.

Overview of the VS Code Interface

The VS Code interface contains five primary components:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

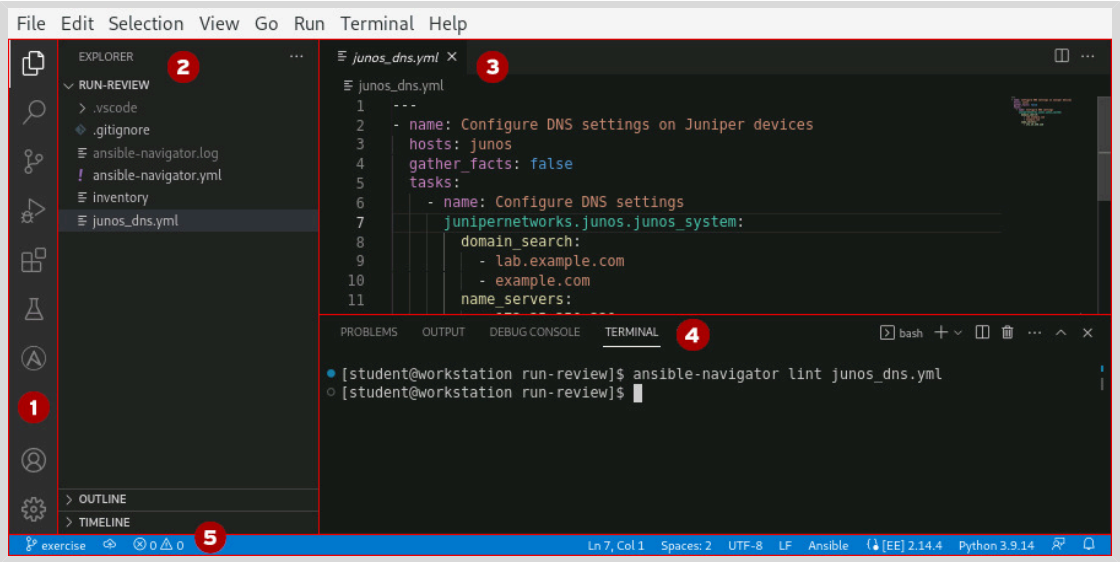


Figure 1.7: The VS Code user interface

1. **The Activity Bar**

   At the far left, the Activity Bar contains shortcuts to change the view of the Side Bar. By default, the Activity Bar contains shortcuts for the Explorer, Search, Source Control, Run and Debug, and Extensions views.

   You can also access these activity views from the **View** menu. You can expand or collapse the Side Bar by clicking a shortcut icon more than once.

2. **Side Bar**

   Located immediately to the right of the Activity Bar, the Side Bar area displays the selected Activity view, such as the Explorer view.

3. **Editor Groups**

   The top-right region of VS Code contains one or more groups of editors. By default, there is only one editor group. Any active editor takes up the entire region of the editor group.

   Click the **Split Editor Right** icon in the upper right to create a second editor group. With two editor groups, you can edit two different files side by side.

4. **Panels**

   Located below the editors, individual panels provide different output or debugging information for activities in VS Code. By default, four panels are provided:
   - PROBLEMS
   - OUTPUT
   - DEBUG CONSOLE
   - TERMINAL

5. **Status Bar**

   At the bottom, the Status Bar area provides information about the open project and files as you edit.

Using the VS Code Integrated Terminal

VS Code integrates the terminal from your operating system, enabling you to execute arbitrary terminal commands in VS Code. To view the integrated terminal, click **View** > **Terminal**.

If you need more space to view the terminal, then you can modify its size by dragging the border between the Editor Groups and Panels, or by clicking the **Maximize Panel Size** icon (^).

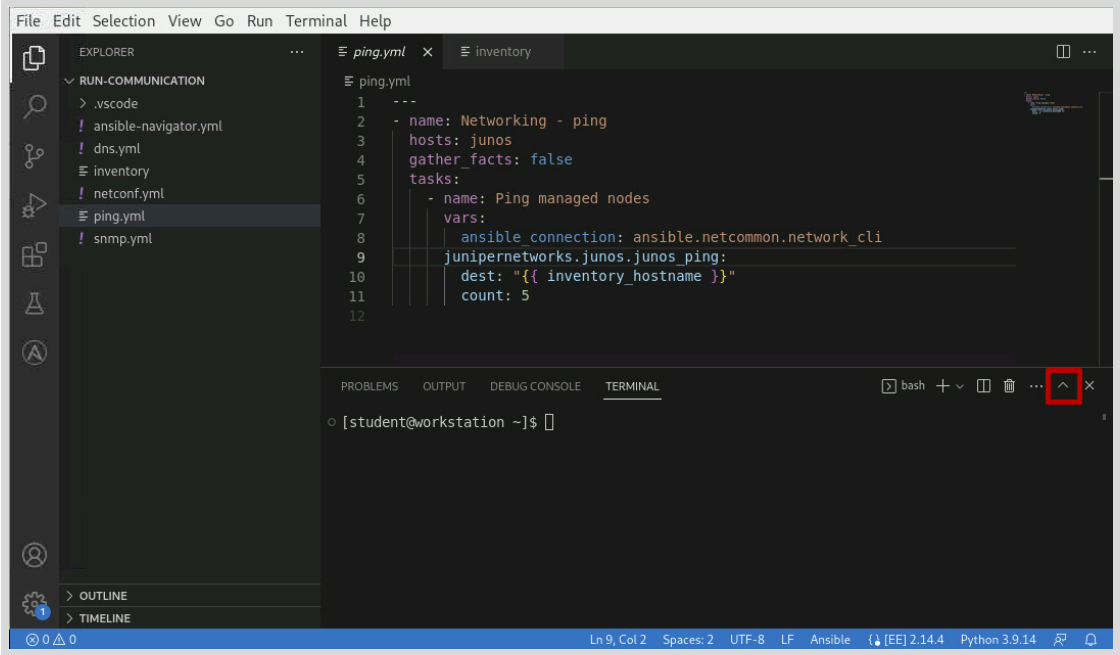


Figure 1.8: Maximizing a VS Code terminal

If you prefer to open a terminal in the **Editor Groups** location by default, then click **File** > **Preferences** > **Settings**, search for `Terminal > Integrated: Default Location`, and then select **editor** from the menu.

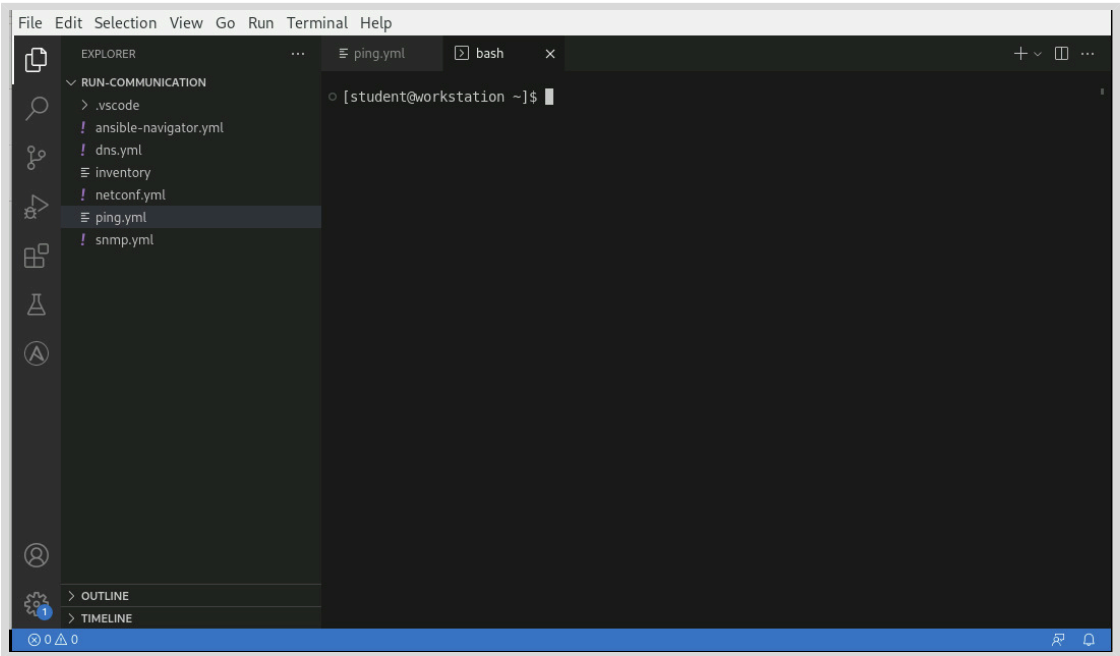


Figure 1.9: The VS Code terminal in the editor location

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

You can open multiple terminals in the VS Code terminal panel. The terminal panel contains a list of open terminals. To display a terminal, select it from the list. To close a terminal, click **Kill Terminal**.

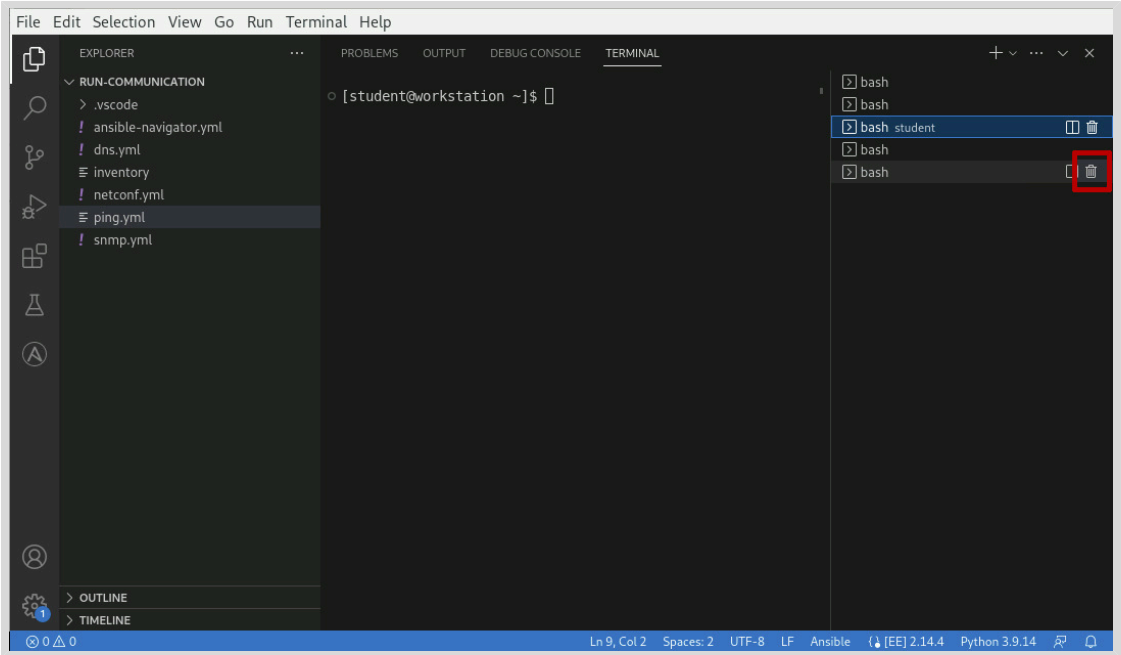


Figure 1.10: Killing a VS Code terminal

Configuring VS Code

You can customize VS Code by modifying its various settings and by using extensions.

Configuring Extensions

VS Code provides an extension framework to encourage the integration of software development capabilities. Any user or organization can contribute extensions. After an extension is developed, it is advertised and published on the VS Code marketplace.

The Ansible (`redhat.ansible`) and YAML (`redhat.vscode-yaml`) VS Code extensions have been installed in the classroom environment. You can search for VS Code extensions at `https://marketplace.visualstudio.com`, or manage extensions by navigating to **View** > **Extensions** within VS Code.

> **Important**
> Although Red Hat provides some VS Code extensions, VS Code is a third-party tool and is not supported by Red Hat.

You can configure the Ansible extension from within VS Code by selecting **View** > **Extensions** and then clicking the **Ansible** extension. Click the **Manage** icon and then click **Extension Settings**.

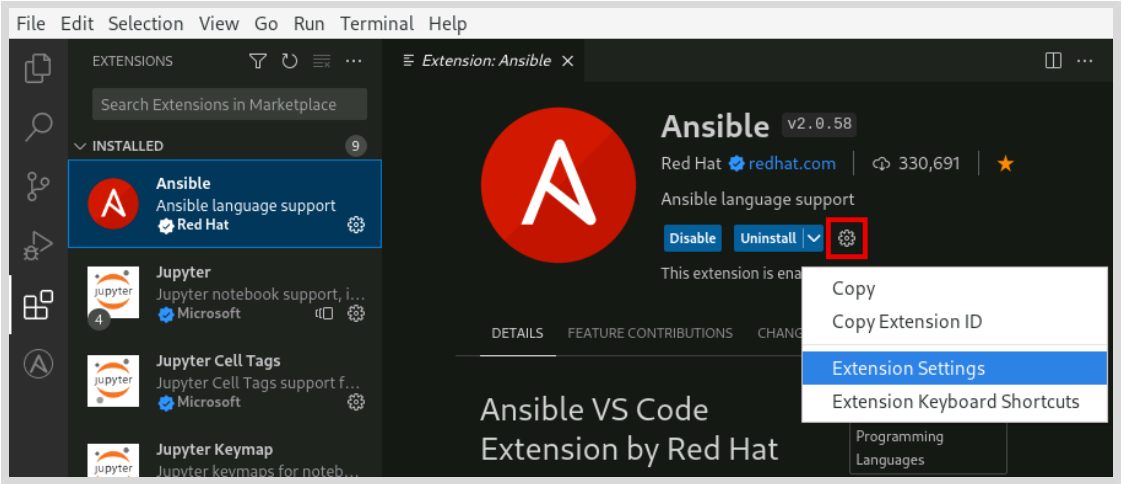


Figure 1.11: Configure the Ansible extension in VS Code

For example, the lab environment has the `ansible-lint` Python module installed in a Python virtual environment, and VS Code can activate this Python virtual environment. If you install the `ansible-lint` RPM package, then remove the entry from the **Ansible** > **Python: Activation Script** field. If linting is not necessary, then disable Ansible linting.

Configuring Settings

In addition to extension settings, you can update general settings. Navigate to **File** > **Auto Save** to automatically save file changes. Navigate to **File** > **Preferences** > **Settings** to update settings, such as font size, tab size, and how the editor renders white space.

References

Automation Content Navigator Creator Guide
https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.3/html-single/automation_content_navigator_creator_guide/

Visual Studio Code
https://code.visualstudio.com

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Preparing for Ansible Development

Install automation content navigator on a control node that runs Red Hat Enterprise Linux.

Outcomes

- Prepare the `workstation` machine to run playbooks by using an automation execution environment with the `ansible-navigator` command.
- Examine Visual Studio Code settings.
- 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 environment for this exercise, and to ensure that all required resources are available.

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

Instructions

Install development tools for Red Hat Ansible Automation Platform on `workstation`, which runs Red Hat Enterprise Linux.

A code editor, Visual Studio Code (VS Code), has already been installed. You use it in this course to perform various actions that you could also complete from the command line. These actions include viewing files, editing files, cloning Git repositories, committing and pushing changes to Git repositories, and running automation content navigator.

Test the development tools by using VS Code to run a test Ansible Playbook.

1. Install software that you use throughout the course.

   1.1. Check your system for the `ansible-core`, `ansible-navigator`, `ansible-lint`, and `podman` RPM packages.

```
[student@workstation ~]$ sudo dnf list ansible-core ansible-navigator \
ansible-lint podman
[sudo] password for student: student
Red Hat Enterprise Linux 9 for x86_64 - BaseOS (RPMs)  100 kB/s | 4.1 kB     ...
Red Hat Enterprise Linux 9 for x86_64 - AppStream (RPM 134 kB/s | 4.5 kB     ...
Installed Packages
ansible-core.x86_64      2.14.4-1.el9ap   @ansible-automation-platform-2.3-fo...
podman.x86_64            2:4.2.0-11.el9_1 @rhel-9-for-x86_64-appstream-rpms
Available Packages
ansible-lint.noarch      6.8.2-1.el9ap    ansible-automation-platform-2.3-for...
ansible-navigator.noarch 2.2.0-1.el9ap    ansible-automation-platform-2.3-for...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1.2. The `ansible-core` and `podman` packages are already installed. Install the `ansible-navigator` and `ansible-lint` packages.

```
[student@workstation ~]$ sudo dnf install ansible-navigator ansible-lint
...output omitted...
Is this ok [y/N]: y
...output omitted...
Complete!
```

2. Download the `hub.lab.example.com/ee-supported-rhel8` automation execution environment from the lab environment's private automation hub. Automation content navigator uses it to run the Ansible Playbook.

2.1. 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.

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

2.2. Use the `podman pull` command to download the automation execution environment.

```
[student@workstation ~]$ podman pull hub.lab.example.com/ee-supported-rhel8
Trying to pull hub.lab.example.com/ee-supported-rhel8:latest...
Getting image source signatures
Copying blob 9d6b728989c4 done
Copying blob a94bd44d2f55 done
Copying blob c7e856e03741 done
Copying blob 8b41bcf810f7 done
Copying config 024856bccc done
Writing manifest to image destination
Storing signatures
024856bccc5fbafddc07cbf758adf1e42d041f478a7c14427b2ca4a9b84294f3
```

3. Examine VS Code settings. Confirm that the **Ansible** extension is installed.

3.1. Open VS Code. Click **Activities** on the top bar, type `code` in the **Type to search** field, and press `Enter` or click the **Visual Studio Code** icon.

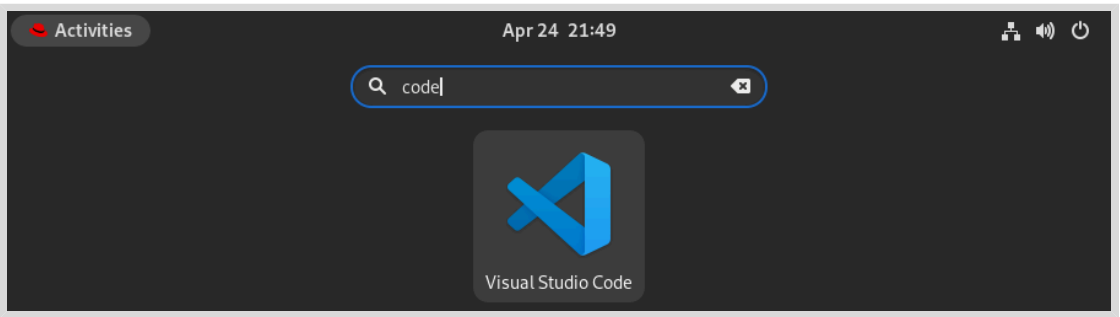


Figure 1.12: Searching for VS Code

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.2. Navigate to **View** > **Extensions** to display a list of installed and recommended extensions. Notice that the **Ansible** extension has been installed in the lab environment.

3.3. Click any extension, such as the **Ansible** extension, to display details about the extension. You can enable or disable an installed extension. You can uninstall or install another version of an extension.

> **Important**
> Do not uninstall the Ansible extension. If your lab environment does not provide outside internet access, then you cannot reinstall the extension.

3.4. You can adjust the settings for installed extensions. To adjust the settings for the **Ansible** extension, click the **Manage** icon and then click **Extension Settings**.

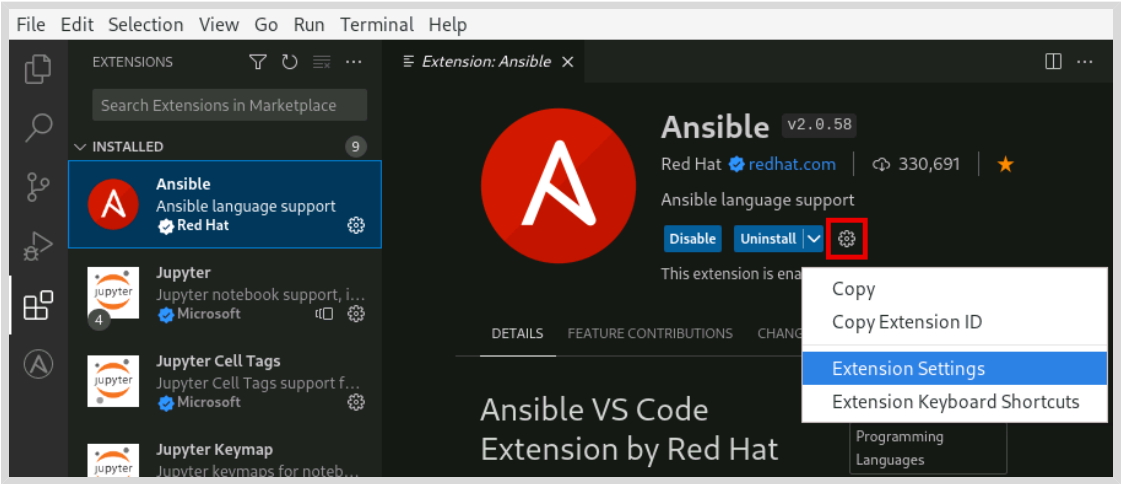


Figure 1.13: VS Code extension settings for Ansible

Extension settings are a subset of all VS Code settings. You can access all VS Code settings by clearing the filter, such as the `@ext:redhat.ansible` filter, or by closing the **Settings** tab and then navigating to **File** > **Preferences** > **Settings**.

You can enable or disable the **Auto Save** setting directly from the **File** menu.

3.5. *(Optional)* Adjust VS Code settings for your preferences, such as enabling the auto save feature and changing the font size.

3.6. Close the **Settings** tab and any other tabs that you opened.

4. From within VS Code, open the `/home/student/intro-prepare` directory and view the `sample.yml` playbook.

4.1. Navigate to **View** > **Explorer** and then navigate to **File** > **Open Folder**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

If navigating to **File** > **Open Folder** does not appear to do anything, then VS Code probably opened the **Open Folder** window behind the current window. Click the date and time in the top bar and then click the **Visual Studio Code** notification.

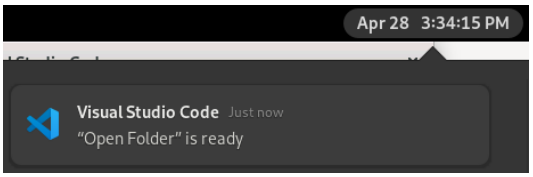


Figure 1.14: VS Code open folder notification

4.2. Navigate to **Home** > **intro-prepare** and then click **Open**.

The first time you open a folder, VS Code displays a window asking if you trust the authors of the files in the folder. Select **Trust the authors of all files in the parent folder 'student'** and then click **Yes, I trust the authors**.

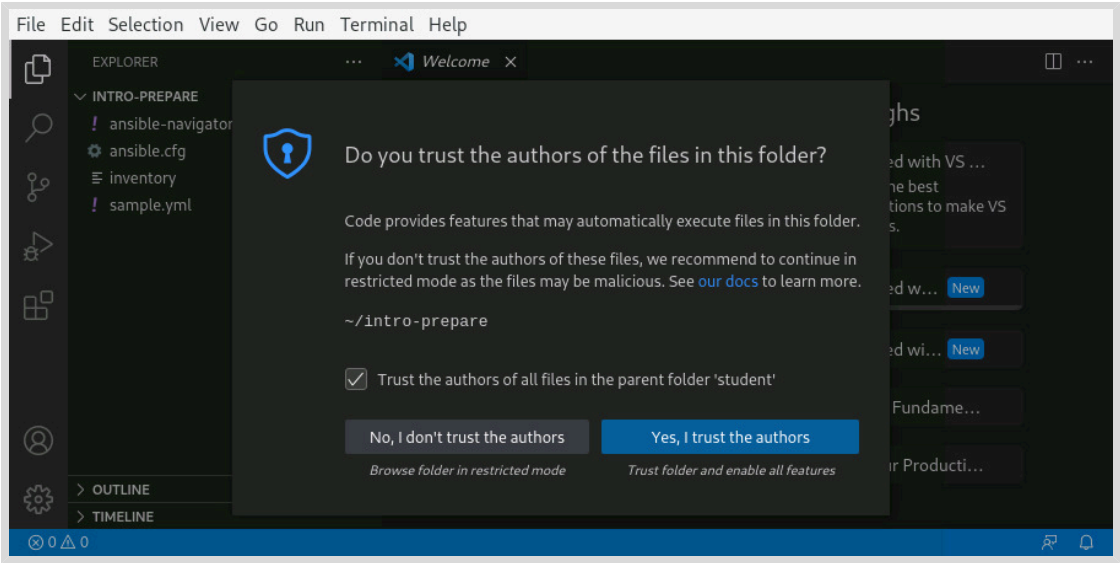


Figure 1.15: Trusting the authors of files in VS Code

4.3. Click the `sample.yml` playbook. As soon as you open the file, VS Code uses the `ansible-lint` command to check the playbook for errors. The lower left corner of the application displays the result of these checks. The `sample.yml` playbook does not produce any errors or warnings.

Figure 1.16: Bottom bar of VS Code

5. Run the `sample.yml` playbook. You can either run the playbook from a GNOME terminal or from a VS Code terminal.

5.1. To use a VS Code terminal, navigate to **Terminal** > **New Terminal**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

Red Hat Enterprise Linux 9 updated the default SELinux policy to disallow commands with text relocation libraries. Because of this change, opening a VS Code terminal generates an SELinux security alert. Note that this policy change does not affect the GNOME terminal.

If you are quick enough, then you can click **Show** as soon as you see the SELinux alert notification. Otherwise, you can access the notification by clicking the date and time in the top bar and clicking **New SELinux security alert**.

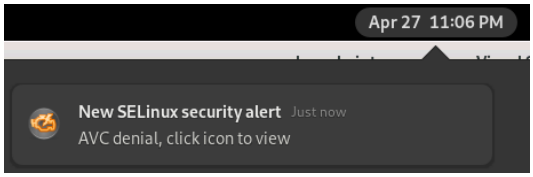


Figure 1.17: SELinux security alert

You can safely ignore this alert. From the **SELinux Alert Browser** window, click **Ignore** and then close the alert browser window.

More information about this security enhancement can be found by searching for `BZ#2055822` at: `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/9.0_release_notes/index#enhancement_security`

5.2. Use the `ansible-navigator run` command to run the `sample.yml` playbook. The playbook displays your configured VS Code settings. Your settings might be different from the following output.

```
[student@workstation intro-prepare]$ ansible-navigator run sample.yml -m stdout
...output omitted...
TASK [Display /home/student/.config/Code/User/settings.json] ******************
ok: [workstation] => {
    "slurp_settings['content'] | b64decode": {
        "editor.fontSize": 11,
        "files.autoSave": "afterDelay",
        "redhat.telemetry.enabled": false
    }
}

PLAY RECAP *********************************************************************
workstation                : ok=3    changed=0    unreachable=0    failed=0   ...
```

5.3. Close the **sample.yml** tab.

Finish

On the `workstation` machine, 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 intro-prepare
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- Automation is a key technique to mitigate human error and quickly ensure that your network is in a consistent, correct state.
- Ansible is a desired-state engine. It approaches the problem of how to automate technology deployments by expressing them in terms of the required end state of your systems.
- The Ansible architecture consists of two types of machines: control nodes and managed nodes.
- Ansible is agentless. Managed nodes do not need any special agent software.
- Red Hat Ansible Automation Platform is a supported distribution of Ansible that includes Ansible Core, selected Ansible Content Collections, automation content navigator, automation execution environments, automation controller, automation hub, private automation hub, and access to hosted services.
- A playbook is a text file that contains a list of one or more plays to run in order.
- Automation content navigator (`ansible-navigator`) is used to run Ansible Playbooks in an automation execution environment provided by a Linux container image.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 2

Implementing an Ansible Playbook

Goal

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

Objectives

- Describe Ansible inventory concepts and manage a static inventory file.
- Describe where automation content navigator configuration files might be located and edit them to apply changes to default settings.
- Write a basic Ansible Playbook and run it using the automation content navigator.
- Write a playbook that uses multiple plays and per-play privilege escalation, and effectively use automation content navigator to find new modules in available Ansible Content Collections and use them to implement tasks for a play.

Sections

- Building an Ansible Inventory (and Guided Exercise)
- Configuring Automation Content Navigator (and Guided Exercise)
- Writing and Running Playbooks (and Guided Exercise)
- Implementing Multiple Plays (and Guided Exercise)

Lab

- Implementing an Ansible Playbook

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Building an Ansible Inventory

Objectives

- Describe Ansible inventory concepts and manage a static inventory file.

The Purpose of an Ansible Inventory

An *inventory* file defines nodes in your network topology that Ansible manages. You can organize the managed nodes in groups. Groups can contain child groups, and the managed nodes can be members of many groups. You can also set variables in the inventory that apply to one managed node, or a group of them.

You use the inventory to define the set of nodes against which you run one or more tasks in a playbook.

The inventory can be static or dynamic. A *static inventory* is a single text file with a list of the managed nodes. The managed nodes and groups in a static inventory are fixed as long as you do not manually change them. A *dynamic inventory* generates a list of nodes and groups in real time. To generate a dynamic inventory, you can use a script that queries an external source, such as a configuration management database (CMDB). You can create the script, take it from the open source community, and so on.

The use of dynamic inventories is beyond the scope of this course. You can find more information about dynamic inventories at `https://docs.ansible.com/ansible/latest/inventory_guide/intro_dynamic_inventory.html`

Building a Static Ansible Inventory

You can build a static Ansible inventory using different formats. In this course, you focus on learning the INI-style format. You can use Visual Studio Code or any other editor to create the file.

The following example is a simple static inventory file written in the INI-style format. This Ansible inventory contains a simple list of managed nodes. You can use hostnames or IP addresses for the managed nodes.

```
iosxe1.lab.example.com
iosxe2.lab.example.com
junos.lab.example.com
arista.lab.example.com
```

You can improve the Ansible inventory by defining groups of managed nodes and including variables that apply to them.

Defining Groups of Managed Nodes

Use brackets to define groups of managed nodes, and add the list of nodes to the group. You can use blank lines between groups for better readability.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The following example contains three groups of managed nodes in the Ansible inventory. The `cisco` group contains the `iosxe1.lab.example.com` and `iosxe2.lab.example.com` managed nodes. The `juniper` and `arista` groups contain only one managed node each.

```
[cisco]
iosxe1.lab.example.com
iosxe2.lab.example.com

[juniper]
junos.lab.example.com

[arista]
arista.lab.example.com
```

A managed node can belong to more than one group in the Ansible inventory. In the previous example, suppose that you have the managed nodes distributed in the `dc1` and `dc2` data centers. If you plan to run some tasks on the nodes of one of the data centers, then consider defining groups with that information.

In the following inventory, all the managed nodes belong to more than one group. For example, the `iosxe1.lab.example.com` node belongs to the `cisco` group and to the `dc1` group. You can use as many groups as you need to organize your inventory.

```
[cisco]
iosxe1.lab.example.com
iosxe2.lab.example.com

[juniper]
junos.lab.example.com

[arista]
arista.lab.example.com

[dc1]
iosxe1.lab.example.com
junos.lab.example.com

[dc2]
iosxe2.lab.example.com
arista.lab.example.com
```

Special Groups of Managed Nodes

All Ansible inventories contain two special groups: the `all` and the `ungrouped` groups.

`all`
: This group contains all the managed nodes in the Ansible inventory.

`ungrouped`
: This group contains all managed nodes that do not belong to any group. Ungrouped nodes appear at the top of an inventory file before the first group is defined.

You are not required to define either the `all` or the `ungrouped` groups in the Ansible inventory because they always exist.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Creating Nested Groups

The inventory can include groups of groups. You can create such groups by adding the `:children` suffix to the parent group. In the following example, the `routers` group includes the `cisco`, `juniper`, and `arista` groups.

```
[cisco]
iosxe1.lab.example.com
iosxe2.lab.example.com

[juniper]
junos.lab.example.com

[arista]
arista.lab.example.com

[routers:children]
cisco
juniper
arista
```

A group can have both individually managed nodes and groups of managed nodes as members. The individually managed nodes should be listed in the group, and the members that are groups should be placed as children.

In the following example, the parent `cisco` group consists of three managed nodes: the two members of the `ios` group, and the `nxos.lab.example.com` managed node.

```
[ios]
iosxe1.lab.example.com
iosxe2.lab.example.com

[cisco]
nxos.lab.example.com

[cisco:children]
ios
```

Using Ranges to Simplify the Ansible Inventory

Sometimes the managed nodes in your Ansible inventory might have a pattern in their names. You can use a special syntax to add a range of managed hosts to the inventory. Ranges can be numeric or alphabetical.

In the following example, instead of writing twenty lines under the group name, you can define a range of managed nodes in the `cisco` group using a simplified syntax:

```
[cisco]
iosxe[01:20].lab.example.com
```

The meaning of the `[START:END]` syntax is as follows:

- *START* defines the start of the range.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- *END* defines the end of the range.

The ranges include all values from *START* to *END*. Consider the following examples:

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

If leading zeros are included in numeric ranges, they are used in the pattern. The second example above does not match `core-rtr-7.example.com` but does match `core-rtr-07.example.com`.

You can also specify a range where there are increments between the sequence in the name of the managed nodes.

Suppose that you have distributed your managed nodes from the previous example in the `dc1` and `dc2` data centers. The managed nodes with an odd number in their names (`iosxe01.lab.example.com`, `iosxe03.lab.example.com`, and so on) are in the `dc1` data center. The managed nodes with an even number in their names (`iosxe02.lab.example.com`, `iosxe04.lab.example.com`, and so on) are in the `dc2` data center. The simplified Ansible inventory looks like the following:

```
[cisco]
iosxe[01:20].lab.example.com

[dc1]
iosxe[01:20:2].lab.example.com

[dc2]
iosxe[02:20:2].lab.example.com
```

The meaning of the `[START:END:STRIDE]` syntax is as follows:

- *START* defines the start of the range.
- *END* defines the end of the range.
- *STRIDE* defines the increment between the sequence of elements in the range.

Including Variables in the Ansible Inventory

You can define variables for managed nodes in the Ansible inventory file. These variables might be used by your Ansible Playbook, or they might be used to provide additional information about how Ansible connects to those managed nodes. For example, you might use the `ansible_host` variable to point to an IP address if the managed node does not have a DNS entry.

In the following example, a play that uses the `nxos.lab.example.com` managed node connects to it by using the `10.80.1.251` IP address.

```
nxos.lab.example.com ansible_host=10.80.1.251
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

You can define variables for single managed nodes (*host variables*), and variables that affect an entire group in your Ansible inventory (*group variables*). You can use the host variables and the group variables later in your playbooks.

In the following example, you define the `private_ip` host variable with the `172.25.250.23` value in the Ansible inventory. You can then use the `private_ip` value as the IP address of the `nxos.lab.example.com` managed node in a private network that is reachable by the control node.

```
nxos.lab.example.com private_ip=172.25.250.23
```

To define group variables, add the `:vars` suffix to the group in the Ansible inventory. For example, Ansible treats the managed nodes in the Ansible inventory as network devices if you define the `ansible_connection` variable with the `netcommon.network_cli` value. You might define this variable for specific groups of network devices, or you might create a group that contains all your network devices, such as the `network_devices` group.

```
[network_devices:vars]
ansible_connection=ansible.netcommon.network_cli
```

The `ansible_network_os` variable is commonly used by network devices to indicate the network platform to which the managed node corresponds. In the following example, the `ansible_network_os` variable is set to `cisco.ios.ios` for the managed nodes in the `cisco` group, and is set to `arista.eos.eos` for the managed nodes in the `arista` group.

```
[cisco]
iosxe1.lab.example.com
iosxe2.lab.example.com

[arista]
arista.lab.example.com

[cisco:vars]
ansible_network_os=cisco.ios.ios

[arista:vars]
ansible_network_os=arista.eos.eos
```

Using Automation Content Navigator to Browse the Ansible Inventory

To display the contents of your Ansible inventory, you can use the `ansible-navigator` command with the `inventory` subcommand.

```
[user@host ~]$ ansible-navigator inventory
```

To specify an Ansible inventory path that is not the default, use the `--inventory` (or the `-i`) option.

```
[user@host ~]$ ansible-navigator inventory --inventory ~/my_inventory
```

The following output shows the two possible options to explore the Ansible inventory.

```
student@workstation:~/intro-inventory — /usr/bin/python3 /bin/ansible-navigator inventory
  Title            Description
0 Browse groups    Explore each inventory group and group members members
1 Browse hosts     Explore the inventory with a list of all hosts

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

Figure 2.1: Browsing the Ansible inventory with automation content navigator

Browsing Groups

Type `0` in the main Ansible inventory page to explore the groups and group members of the Ansible inventory.

```
student@workstation:~/intro-inventory — /usr/bin/python3 /bin/ansible-navigator inventory
  Name                          Taxonomy                    Type
0 ungrouped                     all                         group
1 dc1                           all                         group
2 dc2                           all                         group
3 routers                       all                         group

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

Figure 2.2: Listing the Ansible inventory groups

Enter the number of the group that you want to explore. If the number of the group is greater than nine, type a colon before the group number, and then press `Enter`.

For example, to list the managed nodes in the `dc2` group in the previous output, type 2 or `:2` followed by `Enter`.

```
student@workstation:~/intro-inventory — /usr/bin/python3 /bin/ansible-navigator inventory
  Name                                          Taxonomy          Type
0 iosxe2.lab.example.com                        all▸dc2           host
1 arista.lab.example.com                        all▸dc2           host

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

Figure 2.3: Managed nodes in the dc2 group.

You can continue pressing the number of the item you want to explore, or you can press `Esc` to go back one level. Press `Esc` as many times as you need to go back some levels, to return to the main page, or to exit the `ansible-navigator` command. You can also type `:q` to exit.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Browsing Managed Nodes

Type `1` in the main Ansible inventory page to explore the list of managed nodes in your Ansible inventory.

```
student@workstation:~/intro-inventory — /usr/bin/python3 /bin/ansible-navigator inventory
  Inventory hostname
0 arista.lab.example.com
1 iosxe1.lab.example.com
2 iosxe2.lab.example.com
3 junos.lab.example.com

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

Figure 2.4: Listing the Ansible inventory managed nodes

To obtain more information about a managed node, type the number of the node, or type a colon, the node number, and `Enter`.

The following output shows the information for the `iosxe1.lab.example.com` managed node from the previous example.

```
student@workstation:~/intro-inventory — /usr/bin/python3 /bin/ansible-navigator inventory
[iosxe1.lab.example.com] cisco.iosxr.iosxr
0 ---
1 ansible_connection: ansible.netcommon.network_cli
2 ansible_network_os: cisco.iosxr.iosxr
3 inventory_hostname: iosxe1.lab.example.com

^b/PgUp page up^f/PgDn page down↑↓ scrollesc back- previous+ next[0-9] goto:help help
```

Figure 2.5: Specific information about a managed node

Exploring the Ansible Inventory in `stdout` Mode

By default, the `ansible-navigator` command uses the text-based user interface. To disable this interactive mode, run `ansible-navigator inventory` using the `--mode stdout` (or `-m stdout`) option.

When browsing an Ansible inventory using the `-m stdout` option, you have to specify one of the following actions:

- To create an inventory graph, use the `--graph` action.
- To specify a managed node, use the `--host` *`HOST`* action.
- To output all host information in JSON format, use the `--list` action.

The following example shows the output using the `--graph` action:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[user@host ~]$ ansible-navigator inventory -m stdout --graph
@all:
  |--@ungrouped:
  |--@dc1:
  |  |--iosxe1.lab.example.com
  |  |--junos.lab.example.com
  |--@dc2:
  |  |--iosxe2.lab.example.com
  |  |--arista.lab.example.com
  |--@routers:
  |  |--@cisco:
  |  |  |--iosxe1.lab.example.com
  |  |  |--iosxe2.lab.example.com
  |  |--@juniper:
  |  |  |--junos.lab.example.com
  |  |--@arista:
  |  |  |--arista.lab.example.com
```

The `--graph` action can also be used to create the inventory graph of only one group of the inventory:

```
[user@host ~]$ ansible-navigator inventory -m stdout --graph routers
@routers:
  |--@cisco:
  |  |--iosxe1.lab.example.com
  |  |--iosxe2.lab.example.com
  |--@juniper:
  |  |--junos.lab.example.com
  |--@arista:
  |  |--arista.lab.example.com
```

References

How to Build Your Inventory - Ansible Documentation
https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

Groups and Variables in an Inventory File - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/network_best_practices_2.5.html#groups-and-variables-in-an-inventory-file

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Building an Ansible Inventory

Create a new static inventory containing managed nodes and groups.

Outcomes

- Create a custom static inventory using Visual Studio Code.
- Use automation content navigator to explore the Ansible Inventory.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your environment for this exercise, and to ensure that all required resources are available.

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

Instructions

1. From Visual Studio Code (VS Code) on the `workstation` machine, open the `/home/student/implement-inventory` directory.

   1.1. Open VS Code. Click **View** > **Explorer** to open the **Primary Side Bar**.

   1.2. Click **File** > **Open Folder**.

   1.3. Click **Home** > **implement-inventory** and then click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

2. Create a custom static inventory named `inventory` in the `/home/student/implement-inventory` directory. Use the following table to organize the Ansible inventory. Assign each managed node to multiple groups for best management, based on its operating system and the data center location.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Managed node	Operating system	Data center
`iosxe1.lab.example.com`	Cisco IOS XE	Raleigh
`iosxe2.lab.example.com`	Cisco IOS XE	Johannesburg
`nxos.lab.example.com`	Cisco NX-OS	Raleigh
`junos.lab.example.com`	Juniper Junos	London
`arista-leaf1.lab.example.com`	Arista EOS	Johannesburg
`arista-leaf2.lab.example.com`	Arista EOS	Raleigh
`arista-leaf3.lab.example.com`	Arista EOS	London

2.1. Click **File** > **New File**.

2.2. Enter `inventory` as the file name, and then click **Create New File**.

2.3. Click **Create File** in the **Files** window behind VS Code. The **Files** window closes after the `inventory` file is created.

2.4. Edit the Ansible inventory file so that it contains the following content, and then click **File** > **Save**:

```
[ios]
iosxe[1:2].lab.example.com

[nxos]
nxos.lab.example.com

[junos]
junos.lab.example.com

[eos]
arista-leaf[1:3].lab.example.com

[raleigh]
iosxe1.lab.example.com
nxos.lab.example.com
arista-leaf2.lab.example.com

[johannesburg]
iosxe2.lab.example.com
arista-leaf1.lab.example.com

[london]
junos.lab.example.com
arista-leaf3.lab.example.com
```

3. Explore the Ansible inventory by using the `ansible-navigator` command in `stdout` mode. Create an inventory graph of all the Ansible inventory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.1. In VS Code, click **Terminal** > **New Terminal**.

3.2. Run the `ansible-navigator inventory -m stdout --graph` command to obtain the graph of all the Ansible inventory.

```
[student@workstation implement-inventory]$ ansible-navigator inventory \
 -m stdout --graph
@all:
  |--@ungrouped:
  |--@ios:
  |  |--iosxe1.lab.example.com
  |  |--iosxe2.lab.example.com
  |--@nxos:
  |  |--nxos.lab.example.com
  |--@junos:
  |  |--junos.lab.example.com
  |--@eos:
  |  |--arista-leaf1.lab.example.com
  |  |--arista-leaf2.lab.example.com
  |  |--arista-leaf3.lab.example.com
  |--@raleigh:
  |  |--iosxe1.lab.example.com
  |  |--nxos.lab.example.com
  |  |--arista-leaf2.lab.example.com
  |--@johannesburg:
  |  |--iosxe2.lab.example.com
  |  |--arista-leaf1.lab.example.com
  |--@london:
  |  |--junos.lab.example.com
  |  |--arista-leaf3.lab.example.com
```

4. Use the following table to define group variables in the Ansible inventory. Create the `network_devices` group that includes the `ios`, `nxos`, `junos`, and `eos` groups. Use VS Code to modify the Ansible inventory.

Managed node	Network platform	Network device
`iosxe1.lab.example.com`	`cisco.ios.ios`	Yes
`iosxe2.lab.example.com`	`cisco.ios.ios`	Yes
`nxos.lab.example.com`	`cisco.nxos.nxos`	Yes
`junos.lab.example.com`	`junipernetworks.junos.junos`	Yes
`arista-leaf1.lab.example.com`	`arista.eos.eos`	Yes
`arista-leaf2.lab.example.com`	`arista.eos.eos`	Yes
`aarista-leaf3.lab.example.com`	`arista.eos.eos`	Yes

4.1. Edit the Ansible inventory file to append the following content to the `inventory` file, and then click **File** > **Save**:

```
[network_devices:children]
ios
nxos
junos
eos

[network_devices:vars]
ansible_connection=ansible.netcommon.network_cli

[ios:vars]
ansible_network_os=cisco.ios.ios

[nxos:vars]
ansible_network_os=cisco.nxos.nxos

[junos:vars]
ansible_network_os=junipernetworks.junos.junos

[eos:vars]
ansible_network_os=arista.eos.eos
```

4.2. Select **File** > **Save**.

5. Run automation content navigator in interactive mode in the VS Code terminal. Explore the managed nodes entries in the inventory file and obtain the detail for the `arista-leaf3.lab.example.com` node.

5.1. Run the `ansible-navigator inventory` command in the VS Code terminal.

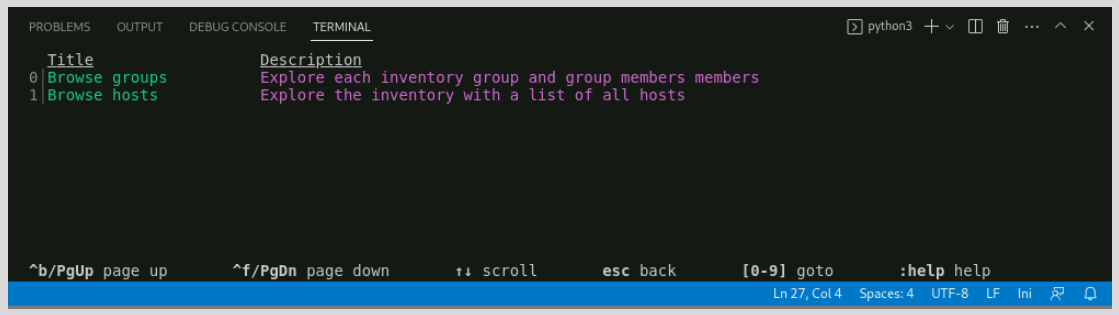


Figure 2.6: Browsing the Ansible inventory

5.2. Type `1` in the main Ansible inventory page to list the managed nodes.

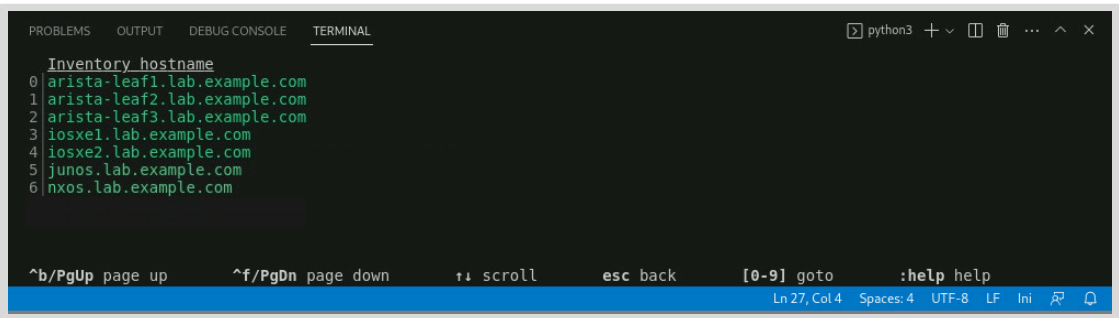


Figure 2.7: List of managed nodes

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

5.3. Type 2 to obtain inventory information about the `arista-leaf3.lab.example.com` node.

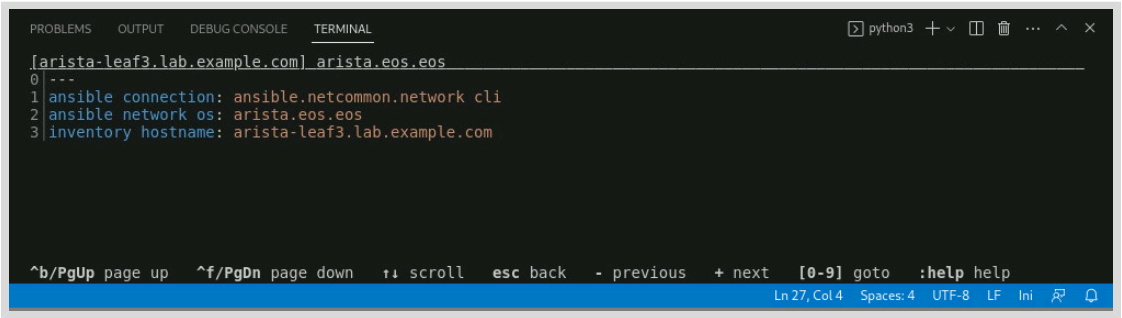


Figure 2.8: Inventory information for the `arista-leaf3.lab.example.com` managed node

5.4. Type `:q` and then press `Enter` to exit the `ansible-navigator` command.

6. From VS Code, click **File** > **Close Folder** to close the `/home/student/implement-inventory` directory.

Finish

On the `workstation` machine, 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 implement-inventory
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Configuring Automation Content Navigator

Objectives

- Describe where automation content navigator configuration files might be located and edit them to apply changes to default settings.

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 that you use the `ansible-navigator` command.

The following details apply to settings files:

- 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 the 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.

Generating a Settings File

You can use the `ansible-navigator settings` command to generate settings files. You can use the `--sample` option to output a sample `ansible-navigator` configuration file in YAML format. You can use the `--effective` option instead, to output a configuration file that matches the current effective configuration for `ansible-navigator`. In either case, you can redirect the output of the command into a file to save it, edit it, and use it.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 command options, 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.

Defining Inventory Sources

Like automation content navigator settings, Ansible configuration settings can be defined in multiple places. You can define Ansible configuration settings in an `ansible.cfg` file, in a `~/.ansible.cfg` file, or in the `/etc/ansible/ansible.cfg` file.

Although `ansible.cfg` files can contain several settings, the files almost always define a path to an inventory source. For example:

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

If you only intend to define an inventory source in an `ansible.cfg` file, then rather than creating that file, you can add inventory entries in the automation content navigator configuration file. The following example shows how to define inventory sources (or entries) in the `ansible` key:

```
---
ansible-navigator:
  ansible:
    inventory:
      entries:
        - ./inventory
```

If you need to place additional settings in an `ansible.cfg` file, such as settings for roles or Ansible Content Collections, then you might define the inventory in the `ansible.cfg` file and not in the `ansible-navigator.yml` file. Tools that are included with the `ansible-core` package use the `ansible.cfg` file and not the `ansible-navigator.yml` file.

> **Important**
> If an `ansible.cfg` file and an `ansible-navigator.yml` file define *different* inventories, then the `ansible-navigator` command uses the inventories defined in the `ansible-navigator.yml` file.

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 equivalent to using the `--pull-policy` (or `--pp`) option. You can also pass additional arguments, such as disabling TLS verification.

For example, consider the following `ansible-navigator` command:

```
[user@host ~]$ ansible-navigator run site.yml --pp missing \
--eei registry.redhat.io/ansible-automation-platform-23/ee-29-rhel8
```

You can eliminate the `--pp` and `--eei` 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: registry.redhat.io/ansible-automation-platform-23/ee-29-rhel8:latest
    pull:
      policy: missing
```

Setting the Default Mode to Standard Output

The `mode` key takes the same options as the `--mode` (or `-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:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
ansible-navigator:
  mode: stdout
```

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 often 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.

```
---
ansible-navigator:
  playbook-artifact:
    enable: false
```

You can also temporarily disable playbook artifacts by running the `ansible-navigator run` command with the `--pae false` option.

References

Automation Content Navigator Configuration Settings
https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/2.3/html-single/automation_content_navigator_creator_guide/index#assembly-settings-navigator_ansible-navigator

Ansible Navigator Documentation
https://ansible-navigator.readthedocs.io/

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring Automation Content Navigator

Edit an Ansible configuration file to customize your Ansible environment.

Outcomes

- Generate an automation content navigator configuration 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 installs the required software, downloads an automation execution environment, and creates a project directory with the files needed for the exercise.

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

Instructions

An automation content navigator configuration file simplifies the `ansible-navigator` commands that you use to run playbooks. Instead of repeatedly adding the same options on the command line, you can add those settings to the configuration file. Although you could memorize the syntax of the automation content navigator configuration file, this exercise demonstrates how to generate the file.

1. Open the `/home/student/implement-navigator` directory in Visual Studio Code or change to that directory in a GNOME terminal. Complete the following substeps if you choose VS Code.

   1.1. In VS Code, click **View** > **Explorer** and then click **File** > **Open Folder**.

> **Important**
> If clicking **File** > **Open Folder** does not appear to do anything, then VS Code probably opened the **Open Folder** window behind the current window. Click the date and time in the top bar and then click the **Visual Studio Code** notification.

   1.2. Navigate to **Home** > **implement-navigator** and then click **Open**.

> **Note**
> If prompted, then select **Trust the authors of all files in the parent folder 'student'** and then click **Yes, I trust the authors**.

2. Generate an automation content navigator configuration file that uses the existing settings.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.1. Use the `ansible-navigator settings --effective` command in a terminal to display the effective settings. Add the `--eei` and `--pp` options to specify an automation execution environment image and a pull policy. Redirect the command output into a file named `sample.yml`.

```
[student@workstation implement-navigator] ansible-navigator settings \
--effective --eei hub.lab.example.com/ee-supported-rhel8 \
--pp missing > sample.yml
```

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.

2.2. Rename the `sample.yml` file to `ansible-navigator.yml`.

```
[student@workstation implement-navigator] mv sample.yml ansible-navigator.yml
```

3. Update the `ansible-navigator.yml` file with the desired settings.

3.1. Open the `ansible-navigator.yml` file in an editor.

3.2. Remove lines from the file, add the highlighted inventory lines, and then update the highlighted playbook artifact line so that the file contains the following content. When finished, save the file.

```
---
ansible-navigator:
  ansible:
    inventory:
      entries:
        - ./inventory (1)
  execution-environment:
    image: hub.lab.example.com/ee-supported-rhel8:latest (2)
    pull:
      policy: missing (3)
  mode: stdout (4)
  playbook-artifact:
    enable: false (5)
```

(1) If the Ansible configuration (such as an `ansible.cfg` file) defines an inventory, then the generated file adds an inventory entry with an absolute path. Because this exercise does not create or use an `ansible.cfg` file, add an entry for the project `inventory` file.

(2) The generated file uses the value of the `--eei` option. If a tag is not specified, then the `latest` tag is added to the container image name.

(3) The generated file uses the value of the `--pp` option.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4. The generated file **always** uses `stdout` mode. When prompting for a password, you must use `stdout` mode. Otherwise you can change the value to `interactive` mode if desired.

5. Playbook artifacts are enabled by default. Use a value of `false` if you need to prompt for a password, such as a connection password.

> **Note**
> This exercise removes lines so that you can focus on a small group of settings, but the lines do not need to be removed.

4. Test the configuration file by using the `ansible-navigator` command to run the `gather_facts.yml` playbook. The value of your `net_serialnum` variable might be different from the following output:

```
[student@workstation implement-navigator] ansible-navigator run gather_facts.yml
...output omitted...
        "net_hostname": "iosxe2.lab.example.com",
        "net_image": "bootflash:packages.conf",
        "net_iostype": "IOS-XE",
        "net_model": "C8000V",
        "net_operatingmode": "autonomous",
        "net_python_version": "3.9.13",
        "net_serialnum": "98U5E5QQ9AD",
        "net_system": "ios",
        "net_version": "17.06.02",
        "network_resources": {}
    }
}

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0   ...
```

5. Either close the `/home/student/implement-navigator` directory in VS Code by selecting **File** > **Close Folder**, or return to the `student` home directory in a GNOME terminal by running the `cd` command.

```
[student@workstation implement-navigator]$ cd
```

Finish

On the `workstation` machine, 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 implement-navigator
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Writing and Running Playbooks

Objectives

- Write a basic Ansible Playbook and run it using the automation content navigator.

Formatting an Ansible Playbook

A playbook is a text file written in YAML format, using Standard American English, and is normally saved with a `.yml` extension.

Playbooks use space characters for indentation to indicate the structure of the data.

YAML does not place strict requirements on how many spaces are used for the indentation, but two basic rules apply:

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

> **Important**
> You can only use space characters for indentation; do not use tab characters.

YAML files contain lists and dictionaries.

Lists contains items that are all indented at the same level. An item in a YAML list starts with a single dash followed by a space:

```
- show version
- show interfaces
- show ip route
```

Dictionaries contain a set of `key: value` pairs indented at the same level. The colon must be followed by a space:

```
port: 22
delay: 60
timeout: 600
```

A playbook contains a list of plays. Each play is a dictionary of directives that perform operations such as naming the play, listing the hosts (managed nodes) operated on by the play, and so on. Plays also contain a `tasks` directive that consists of a list of tasks that the play runs.

The following playbook example contains one play with a single task.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Network automation example
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure a Cisco IOS device
      cisco.ios.ios_config:
        lines:
            - service timestamps debug datetime msec
            - service timestamps log datetime msec localtime
            - service password-encryption
        save_when: changed
```

You can add blank lines in your playbooks for readability.

YAML comments begin with the number sign (#).

Playbook Syntax

A playbook usually begins with a line consisting of three dashes (`---`) to indicate the start of the document. It might end with three dots (`...`) to indicate the end of the document, although in practice this is often omitted.

Between those markers, the playbook is defined as a list of plays. Playbooks contain one or more plays. Plays must define, at a minimum, the managed nodes to target and at least one task to execute.

The first line of the previous example playbook starts with a dash and a space, to indicate the start of the list of plays, then the name of the first play is defined. The `name` line associates a string with the play as a label that identifies the play. Defining the name of a play is optional, but is recommended because it helps to document your playbook and makes playbook output easier to understand. This is especially useful when a playbook contains multiple plays.

```
---
- name: Network automation example
```

The next line of the play specifies the hosts on which to run the tasks that are defined in the play. The `hosts` line takes a host pattern as a value, such as the names of managed nodes or groups in the inventory.

```
  hosts: ios
```

After `hosts`, this play sets `gather_facts: false` to disable standard fact gathering. This is an example of a directive that affects how the play runs.

The next line contains the `tasks` keyword, which specifies the list of tasks that this play runs.

Tasks call *modules*, which perform your desired action. Modules have specific parameters that you can use to specify the action of the module.

This example play has a single task, which runs the `cisco.ios.ios_config` module with the `lines` and `save_when` parameters.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The `lines` parameter specifies the configuration lines you want to appear in the running configuration. The `save_when` parameter determines when to copy the running configuration to the startup configuration.

For more information on the available parameters for the `cisco.ios.ios_config` module, visit https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html.

```
tasks:
  - name: Configure a Cisco IOS device
    cisco.ios.ios_config:
      lines:
          - service timestamps debug datetime msec
          - service timestamps log datetime msec localtime
          - service password-encryption
      save_when: changed
```

The `tasks` line is the part of the play that lists, in order, the tasks to run against the managed nodes.

Important

The order in which the plays and tasks are listed in a playbook is important, because Ansible runs plays in order from top to bottom.

If a parameter is set to its default value, you do not have to specify it in the play. Although, you can still specify a parameter for documentation purposes and to clarify the desired outcome of the play for anyone else who reads the playbook.

For example, the following play specifies the `match` and `replace` parameters, even though their value is the default value.

```
tasks:
  - name: Configure a Cisco IOS device
    cisco.ios.ios_config:
      lines:
          - service timestamps debug datetime msec
          - service timestamps log datetime msec localtime
          - service password-encryption
      match: line
      replace: line
      save_when: changed
```

Network Automation Modules

Several types of network automation modules are available.

Facts modules

Ansible *facts* are variables that are discovered by Ansible on a managed node by using facts modules. Ansible uses the `setup` module as the default facts module. You enable it by specifying `gather_facts: true` (or `gather_facts: yes`).

Facts modules use an *_facts* suffix. For example, the following are facts modules:

- `cisco.ios.ios_facts`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- `cisco.nxos.nxos_facts`
- `junipernetworks.junos.junos_facts`
- `arista.eos.eos_facts`

> **Important**
> When creating playbooks for network automation, always specify `gather_facts: false` (or `gather_facts: no`) and use the vendor-specific `_facts` modules if the managed node OS supports a specific `_facts` module.

Command modules

Command modules allow you to send commands to network devices. You might use these modules to create playbooks that automate the gathering of statistics, resetting interfaces, or troubleshooting a commonly occurring issue.

Command modules use an *_command* suffix. For example, the following are command modules:

- `cisco.ios.ios_command`
- `cisco.nxos.nxos_command`
- `junipernetworks.junos.junos_command`
- `arista.eos.eos_command`

Configuration modules

Configuration modules send configuration commands to network devices.

Configuration modules use the `_config` suffix, such as the `cisco.ios.ios_config` module.

> **Important**
> Configuration modules change the running configuration, but do not necessarily affect the startup configuration. If you want your changes to be persistent, then you must check documentation for the module that you are using.
>
> Most Cisco configuration modules include the `save_when` option with `never` as the default value. With this value, the module never copies the running configuration to the startup configuration. Set a different value for the `save_when` option if you want the module to copy the running configuration to the startup configuration.
>
> Like Cisco configuration modules, the Arista configuration module includes the `save_when` option with `never` as the default value.
>
> The `junipernetworks.junos.junos_config` module uses the `confirm` option with `0` as the default value. With this value, the module *immediately* commits the changes so that the changes are persistent.

Resource modules

Network resource modules simplify and standardize the management of network devices across vendors.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Network resource modules enable reading and configuring specific configuration sections, such as VLANs, interfaces, Simple Network Management Protocol (SNMP), Open Shortest Path First (OSPF), ACLs, and others. Network resource modules provide a consistent experience across different network devices.

For more information on network resource modules, visit https://docs.ansible.com/ansible/latest/network/user_guide/network_resource_modules.html

For example, you can use the following network resource modules to configure Layer 3 interfaces:

- `cisco.ios.ios_l3_interfaces`
- `arista.eos.eos_l3_interfaces`
- `junipernetworks.junos.junos_l3_interfaces`

And the following are network resource modules you can use to configure ACLs:

- `cisco.ios.ios_acls`
- `arista.eos.eos_acls`
- `junipernetworks.junos.junos_acls`

Important
As with their associated configuration modules, Cisco and Arista resource modules change the running configuration, but do not change the startup configuration. You might use handlers to make persistent changes.

Using Handlers to Make Persistent Changes to Managed Network Nodes

Handlers are tasks that respond to a notification triggered by other tasks. Tasks only notify their handlers when the status of a task evaluates as changed. Each handler is triggered by its name.

If no task notifies the handler by name, then the handler does not run. If one or more tasks notify the handler, then the handler only runs a single time after all other tasks in the play have completed. Because handlers are tasks, you can use the same modules in handlers that you use for any other task.

For Linux hosts, you normally use handlers to reboot hosts and restart services. For managed network nodes, you can use handlers to save the running configuration to the startup configuration.

Important
Use unique names for your handlers. When multiple handlers are defined with the same name, only the last handler defined with the shared name runs.

Handlers can be considered as *inactive* tasks that only get triggered when explicitly invoked using a `notify` statement. The following playbook demonstrates saving the running configuration to the startup configuration for Cisco managed nodes. The `ios_save_changes` handler is only triggered if the `cisco.ios.ios_banner` makes a change.

```
---
- name: Configure login banners
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure banner on IOS devices ❶
      cisco.ios.ios_banner:
        banner: login
        text: Managed by Ansible
        state: present
      notify: ios_save_changes ❷

  handlers: ❸
    - name: ios_save_changes ❹
      cisco.ios.ios_config: ❺
        save_when: always ❻
```

❶ The task that notifies the handler.

❷ The `notify` statement indicates that the task needs to trigger a handler.

❸ The `handlers` keyword indicates the start of the list of handler tasks.

❹ The name of the handler. This name must match the name invoked by tasks.

❺ The module to use for the handler.

❻ By using the `always` value for the `save_when` option, if this handler is triggered, then the module always copies the running configuration to the startup configuration.

Important

Because handlers run after the completion of all other tasks in the play, any playbook error can stop a handler from running. Even after you fix a playbook error, you might find that a handler does not run. A task that made a change in a previous playbook run might not change, and therefor not trigger a handler, on additional playbook runs.

One way to force a task to trigger a handler is to temporarily add the `changed_when: true` line to a task, such as in the following example:

```
- name: Configure banner on IOS devices
  cisco.ios.ios_banner:
    banner: login
    text: Managed by Ansible
    state: present
  changed_when: true
  notify: ios_save_changes
```

After making the temporary change, run the playbook and verify that the handler runs. If successful, then remove the `changed_when: true` line from the previously modified task.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Finding Modules for Tasks

Modules are the tools that plays use to perform tasks. Hundreds of modules have been written that perform different tasks.. You can usually find a tested, special-purpose module that does what you need, often as part of the default automation execution environment.

Ansible Core 2.11 and later package the modules that you use for tasks in sets called *Ansible Content Collections*. Each Ansible Content Collection contains a selection of related Ansible content, including modules and documentation.

The `ansible-core` package provides a single Ansible Content Collection named `ansible.builtin`. These modules are always available. Visit https://docs.ansible.com/ansible/latest/collections/ansible/builtin/ for a list of modules contained in the `ansible.builtin` collection.

In addition, the default automation execution environment used by `ansible-navigator` in Red Hat Ansible Automation Platform 2.3, `ee-rhel8-supported`, includes a number of other Ansible Content Collections. You can browse these collections by running the `ansible-navigator collections` command.

In the `ansible-navigator` interactive text-based user interface (TUI), you can type a colon (:) followed by the word `collections` to list the available collections.

```
Welcome
──────────────────────────────────────────────────────────────────────────────

Some things you can try from here:
- :collections                                  Explore available collections
- :config                                       Explore the current ansible configuration
- :doc <plugin>                                 Review documentation for a module or plugin
- :help                                         Show the main help page
- :images                                       Explore execution environment images
- :inventory -i <inventory>                     Explore an inventory
- :log                                          Review the application log
- :lint <file or directory>                     Lint Ansible/YAML files (experimental)
- :open                                         Open current page in the editor
- :replay                                       Explore a previous run using a playbook artifact
- :run <playbook> -i <inventory>                Run a playbook in interactive mode
- :settings                                     Review the current ansible-navigator settings
- :quit                                         Quit the application

happy automating,

-winston
```

Figure 2.9: The `ansible-navigator` TUI menu

Type a colon (:) followed by the line number of a collection to get more information about it, including the list of modules and other Ansible content that it provides. You can do the same thing with the line number of a module to get documentation about that module. Press `Esc` to go back to the preceding list.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  Name                          Version   Shadowed   Type        Path
0 amazon.aws                    5.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/amazon/aws
1 ansible.builtin               2.14.4    False      contained   /usr/lib/python3.9/site-packages/ansible
2 ansible.controller            4.3.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/controller
3 ansible.netcommon             4.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/netcommon
4 ansible.network               1.2.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/network
5 ansible.posix                 1.3.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/posix
6 ansible.security              1.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/security
7 ansible.snmp                  1.0.1     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/snmp
8 ansible.utils                 2.7.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/utils
9 ansible.windows               1.12.0    False      contained   /usr/share/ansible/collections/ansible_collections/ansible/windows
0 ansible.yang                  1.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/ansible/yang
1 arista.eos                    6.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/arista/eos
2 cisco.asa                     4.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/cisco/asa
3 cisco.ios                     4.4.0     False      contained   /usr/share/ansible/collections/ansible_collections/cisco/ios
4 cisco.iosxr                   5.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/cisco/iosxr
5 cisco.nxos                    4.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/cisco/nxos
6 cloud.common                  2.1.2     False      contained   /usr/share/ansible/collections/ansible_collections/cloud/common
7 frr.frr                       2.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/frr/frr
8 ibm.qradar                    2.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/ibm/qradar
9 junipernetworks.junos         4.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/junipernetworks/junos
0 kubernetes.core               2.3.2     False      contained   /usr/share/ansible/collections/ansible_collections/kubernetes/core
1 openvswitch.openvswitch       2.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/openvswitch/openvswitch
2 redhat.insights               1.0.7     False      contained   /usr/share/ansible/collections/ansible_collections/redhat/insights
3 redhat.openshift              2.2.0     False      contained   /usr/share/ansible/collections/ansible_collections/redhat/openshift
4 redhat.rhv                    2.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/redhat/rhv
5 redhat.satellite              3.3.0     False      contained   /usr/share/ansible/collections/ansible_collections/redhat/satellite
6 servicenow.itsm               1.3.3     False      contained   /usr/share/ansible/collections/ansible_collections/servicenow/itsm
7 splunk.es                     2.1.0     False      contained   /usr/share/ansible/collections/ansible_collections/splunk/es
8 trendmicro.deepsec            2.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/trendmicro/deepsec
9 vmware.vmware_rest            2.2.0     False      contained   /usr/share/ansible/collections/ansible_collections/vmware/vmware_rest
0 vyos.vyos                     4.0.0     False      contained   /usr/share/ansible/collections/ansible_collections/vyos/vyos
```

Figure 2.10: The `ansible-navigator` TUI collections menu

Note

The following list of resources provides additional Ansible Content Collections:

- The automation hub offered through the Red Hat Hybrid Cloud Console at `https://content.redhat.com/ansible/automation-hub`
- The community's Ansible Galaxy website at https://galaxy.ansible.com

Your organization might also install and configure a private automation hub to provide more collections.

These collections can be installed in the `collections` directory of your Ansible project.

Red Hat does not provide formal support for community Ansible Content Collections; it only supports Red Hat Certified Ansible Content Collections.

Modules are named using a fully qualified collection name (FQCN). This allows the same name to be used for different modules in two Ansible Content Collections without causing conflicts.

For example, the `copy` module provided by the `ansible.builtin` Ansible Content Collection has `ansible.builtin.copy` as its FQCN.

Important

To avoid errors, it is a good practice to always use FQCNs in your playbooks.

Most modules are idempotent, which means that they can be run safely multiple times, and if the system is already in the correct state, they do nothing.

For example, if you run the play from the preceding example a second time, it should report no changes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Running Playbooks

When you run a playbook, output is generated to show the play and tasks being executed. The output also reports the results of each executed task.

The following example shows the content of the `netautomate.yml` playbook, the command to run that playbook, and the result of running it.

```
[user@host networkautomation]$ cat netautomate.yml
---
- name: Network automation example
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure a Cisco IOS device
      cisco.ios.ios_config:
        lines:
            - service timestamps debug datetime msec
            - service timestamps log datetime msec localtime
            - service password-encryption
        save_when: changed
[user@host networkautomation]$ ansible-navigator run \
netautomate.yml -m stdout

PLAY [Network automation example] *********************************************

TASK [Configure a Cisco IOS device] *******************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines
should be similar to how they appear if present in the running configuration on
device
changed: [switch1.lab.example.com]
changed: [switch2.lab.example.com]

PLAY RECAP ********************************************************************
switch1.lab.example.com     : ok=0    changed=1    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
switch2.lab.example.com     : ok=0    changed=1    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
```

The name for each play and task is displayed when the playbook is run.

Notice also that the status of the `Configure a Cisco IOS device` task is `changed` for each managed node. This means that the task changed something on that host to ensure that its specification was met.

Increasing Output Verbosity

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Configuring the Output Verbosity of Playbook Execution

Option	Description
`-v`	Displays task results.
`-vv`	Displays task results and task configuration.
`-vvv`	Displays extra information about connections to managed nodes.
`-vvvv`	Adds extra verbosity options to the connection plug-ins, including users who executed the scripts on the managed nodes, and what scripts have been executed.

Syntax Verification

Before executing a playbook, it is good practice to validate its syntax. You can use the `ansible-navigator run --syntax-check` command to validate the syntax of a playbook.

The following example shows the successful syntax validation of a playbook.

```
[user@host networkautomation]$ ansible-navigator run \
netautomate.yml -m stdout --syntax-check
playbook: /home/user/networkautomation/netautomate.yml
```

When syntax validation fails, a syntax error is reported. The output also includes the approximate location of the syntax issue in the playbook. The following example shows the failed syntax validation of a playbook where the space separator is missing after the `name` attribute for the play.

```
[user@host networkautomation]$ ansible-navigator run \
netautomate.yml -m stdout --syntax-check
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
 from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to be in '/home/user/networkautomation/netautomate.yml': line 6,
 column 27, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

    - name:Configure a Cisco IOS device
      cisco.ios.ios_config:
                           ^ here
Please review the log for errors.
```

Executing a Dry Run

You can use the `--check` option to run a playbook in *check mode*, which performs a "dry run" of the playbook. This causes Ansible to report what changes would have occurred if the playbook were executed, but does not make any actual changes to managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The following example shows the dry run of the example playbook. In this case, the dry run reports that the task would make a change on the managed host.

```
[user@host networkautomation]$ ansible-navigator run \
netautomate.yml -m stdout --check

PLAY [Network automation example] *********************************************

TASK [Configure a Cisco IOS device] *******************************************
[WARNING]: Skipping command copy running-config startup-config due to
check_mode.  Configuration not copied to non-volatile storage
[WARNING]: To ensure idempotency and correct diff the input configuration lines
should be similar to how they appear if present in the running configuration on
device
changed: [switch1.lab.example.com]
changed: [switch2.lab.example.com]

PLAY RECAP ********************************************************************
switch1.lab.example.com     : ok=0    changed=1    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
switch2.lab.example.com     : ok=0    changed=1    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
```

Idempotency

In general, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple times. If the targeted managed nodes are already in the correct state, no changes should be made.

For example, assume that the playbook from the previous example is run again:

```
PLAY [Network automation example] *********************************************

TASK [Configure a Cisco IOS device] *******************************************
ok: [switch1.lab.example.com]
ok: [switch2.lab.example.com]

PLAY RECAP ********************************************************************
switch1.lab.example.com     : ok=1    changed=0    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
switch2.lab.example.com     : ok=1    changed=0    unreachable=0    failed=0
 skipped=0    rescued=0    ignored=0
```

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

Basic rules - Ansible Documentation
https://docs.ansible.com/ansible/latest/dev_guide/style_guide/basic_rules.html

Intro to playbooks - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

Working with playbooks - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

Validating tasks: check mode and diff mode - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_checkmode.html

cisco.ios.ios_config module - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

Network Resource Modules - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/network_resource_modules.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring Device Login Banners

Write and run an Ansible Playbook.

Outcomes

- Create and run a playbook to configure the login banners on Cisco devices running the IOS XE operating system.

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:

- Directories are created for this exercise.
- The `podman`, `ansible-core`, and `ansible-navigator` packages are installed.
- An `ansible.cfg` file and an `ansible-navigator.yml` file are created for this exercise.
- An `inventory` file is created for this exercise.

```
[student@workstation ~]$ lab start implement-playbooks
```

Instructions

1. Open Visual Studio Code (VS Code) on the `workstation` machine and open the `~/implement-playbooks` directory.

   1.1. Click **Activities** on the `workstation` top bar, enter `code` in the **Type to search** field, and then press `Enter`.

   1.2. In VS Code, click **File** > **Open Folder**. Click **Home** > **implement-playbooks** and then click **Open**.

> **Note**
> If prompted, then select **Trust the authors of all files in the parent folder 'student'** and then click **Yes, I trust the authors**.

2. Use VS Code on the `workstation` machine to create a playbook named `banner.yml`. Add a task to the playbook that configures the login banners for managed nodes in the `ios` inventory group.

   2.1. In VS Code, click **File** > **New Text File**.

   2.2. Click **Select a language** and then click **Ansible (ansible)**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.3. Create a playbook with the following contents, and save the file as `banner.yml`.

```
---
- name: Configure login banners
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure banner on IOS devices
      cisco.ios.ios_banner:
        banner: login
        text: Managed by Ansible
        state: present
      notify: ios_save_changes

  handlers:
    - name: ios_save_changes
      cisco.ios.ios_config:
        save_when: always
```

3. Open a terminal, change to the `~/implement-playbooks` directory, and use the `ansible-navigator` command to run the `banner.yml` playbook.

```
[student@workstation ~] cd ~/implement-playbooks
[student@workstation implement-playbooks]$ ansible-navigator run banner.yml \
-m stdout

PLAY [Configure login banners] ************************************************

TASK [Configure banner on IOS devices] ****************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

RUNNING HANDLER [ios_save_changes] ********************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP ********************************************************************
iosxe1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0   ...
```

4. Verify that the login banner was correctly configured on the `iosxe1.lab.example.com` and `iosxe2.lab.example.com` managed nodes, and then change back to the `/home/student` directory.

4.1. Connect to the `iosxe1.lab.example.com` managed node by using SSH and verify the login banner. Then log out from the `iosxe1.lab.example.com` managed node.

```
[student@workstation implement-playbooks]$ ssh iosxe1.lab.example.com

Managed by Ansible
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
iosxe1.lab.example.com#exit
Connection to iosxe1.lab.example.com closed by remote host.
Connection to iosxe1.lab.example.com closed.
```

4.2. Connect to the `iosxe2.lab.example.com` managed node by using SSH and verify the login banner. Then log out from the `iosxe2.lab.example.com` managed node.

```
[student@workstation implement-playbooks]$ ssh iosxe2.lab.example.com

Managed by Ansible



iosxe2.lab.example.com#exit
Connection to iosxe2.lab.example.com closed by remote host.
Connection to iosxe2.lab.example.com closed.
```

5. Close the `/home/student/implement-playbooks` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

5.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/implement-playbooks` directory.

5.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation implement-playbooks]$ cd
[student@workstation ~]$
```

Finish

On the `workstation` machine, 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 implement-playbooks
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Implementing Multiple Plays

Objectives

- Write a playbook that uses multiple plays and per-play privilege escalation, and effectively use automation content navigator to find new modules in available Ansible Content Collections and use them to implement tasks for a play.

Writing Multiple Plays

A playbook is a YAML file containing a list of one or more plays. Each play in the playbook is written as a top-level list item in the playbook.

A single play is an ordered list of tasks to execute against hosts selected from the inventory. If a playbook contains multiple plays, each play can apply its tasks to a separate set of hosts.

You can create a playbook with multiple plays to orchestrate a complex deployment involving different tasks on different hosts.

The following example shows a playbook with two plays. The first play runs against the `ios` inventory group, and the second play runs against the `junos` inventory group.

Although both plays create configuration backups, the plays use different inventory groups and modules from collections that are specific to each network platform.

The `configbackups` directory must exist.

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

- name: First play - Back up Cisco IOS configurations
  hosts: ios
  gather_facts: false
  tasks:
    # This task uses the ios_config module from the cisco.ios collection.
    - name: Back up IOS configurations
      cisco.ios.ios_config:
        backup: true
        backup_options:
          filename: {{ inventory_hostname }}.cfg
          dir_path: /configbackups

- name: Second play - Back up Juniper Junos configurations
  hosts: junos
  gather_facts: false
  tasks:
    # This task uses the junos_config module from the junipernetworks.junos
 collection.
    - name: Back up Junos configurations
      junipernetworks.junos.junos_config:
        backup: true
```

```
        backup_options:
          filename: {{ inventory_hostname }}.cfg
          dir_path: /configbackups
```

Privilege Escalation

Many network devices require credentials to enter a privileged mode where you can make configuration changes. You can configure privilege escalation in a play to automatically send privileged credentials to managed nodes.

Use the `become` Boolean keyword to enable or disable privilege escalation for an individual play or task.

Set `become` to `true` to enable privilege escalation, or set it to `false` to disable it.

```
become: true
```

If `become` is set to `true`, use the `become_method` keyword in the play to specify the privilege escalation method to use for that play. For network devices, set the `become_method` to `enable`.

```
become_method: enable
```

Important

For privilege escalation to work successfully on managed network nodes, you must set the connection type to either `connection: ansible.netcommon.network_cli` or `connection: ansible.netcommon.httpapi`.

The `connection: ansible.common.netconf` connection type does not support privilege escalation. To use this connection type, you must create a user on the managed nodes with the correct privilege level.

You can also enable privilege escalation by configuring the `ansible_become` and `ansible_become_method` connection variables. One way to configure these variables is in the inventory file, as shown in the following example:

```
ansible_network_os: cisco.ios.ios
ansible_connection: ansible.netcommon.network_cli
ansible_become: yes
ansible_become_method: enable
ansible_user: student
```

Selecting Modules

The large number of modules packaged with Ansible provides you with many tools for common tasks.

The `ansible.builtin` collection provides modules to copy files, read files from a remote inventory system, use templates to dynamically generate configuration files, and debug playbooks.

The following table lists some useful `ansible.builtin` modules:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Ansible Built-in Modules

Module	Description
`ansible.builtin.copy`	Copy files from the local machine to remote hosts.
`ansible.builtin.fetch`	Copy files from remote hosts to the local machine.
`ansible.builtin.file`	Create and manage files and directories.
`ansible.builtin.lineinfile`	Ensure a particular line is or is not in a file.
`ansible.builtin.slurp`	Read a file from a remote host.
`ansible.builtin.template`	Dynamically generate text files and transfer them to managed nodes.
`ansible.builtin.service`	Manage Linux services.
`ansible.builtin.debug`	Print statements during playbook execution.
`ansible.builtin.get_url`	Download files over HTTP, HTTPS, or FTP.
`ansible.builtin.uri`	Interact with web services.
`ansible.builtin.git`	Manage git repositories.

The automation execution environment that is used in this course, `ee-supported-rhel8`, includes several collections for managing network devices. This course uses modules from the following collections:

- `cisco.ios`
- `cisco.nxos`
- `junipernetworks.junos`
- `arista.eos`
- `paloalto.panos`

Module Documentation

To display a list of the collections and modules available in your current automation execution environment, run the `ansible-navigator doc -l` command.

```
[user@host ~]$ ansible-navigator doc -l
...output omitted...
ansible.builtin.copy                      Copy files to remote locations
...output omitted...
cisco.ios.ios_config                      Module to manage configuration sectio...
...output omitted...
cisco.nxos.nxos_command                   Run arbitrary command on Cisco NXOS d...
...output omitted...
junipernetworks.junos.junos_interfaces    Junos Interfaces resource module
...output omitted...
```

Search the output of this command by typing a slash (/) at the colon prompt followed by the keyword, and then pressing `Enter`.

> **Note**
> For more details on network collections and connection types, visit the Ansible network platform options at https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html.

As an alternative to running the `ansible-navigator collections` command in interactive mode, you can use the `ansible-navigator doc` *`module_name`* command to display detailed documentation for a module.

If you specify the `-m stdout` option, formatted documentation is displayed to your terminal. If you do not specify that option, leaving `ansible-navigator` in interactive mode, then you can scroll through the documentation in YAML format.

Module documentation includes a description of what the module is for, a list of the attributes that you can use to control the module in a task, examples of how to use the module, and other metadata.

You can also view a summary of all the options you can use with a module by running the `ansible-navigator -s doc` *`module_name`* command:

```
[user@host ~]$ ansible-navigator -s doc cisco.ios.ios_config
- name: Module to manage configuration sections.
  ios_config:
      after:                # The ordered set of commands to append to the ...
      backup:               # This argument will cause the module to create...
      backup_options:       # This is a dict object containing configurable...
      before:               # The ordered set of commands to push on to the...
      defaults:             # This argument specifies whether or not to col...
...output omitted...
```

YAML Syntax

This section investigates some YAML syntax elements that you might see in playbooks.

YAML Comments

You can also use comments to aid readability. In YAML, everything to the right of the number sign (#) is a comment. If there is content to the left of the comment, precede the number sign with a space.

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

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

YAML Strings

Strings in YAML do not normally need to be put in quotation marks even if the string contains spaces.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
this is a string
```

You must enclose strings in either double or single quotation marks if the string includes a colon character followed by a space.

Escape characters are not parsed in strings enclosed in single quotation marks. You must enclose strings in double quotation marks to parse escape characters such as the newline character (\n).

```
'Play 1: Configure SNMP settings'
```

```
"This is a private system\nUnauthorized use of this system is prohibited"
```

You can write multiline strings in either of two ways; using the vertical bar (|) character or the greater-than (>) character.

Use the vertical bar (|) character to denote that newline characters within the string are to be preserved.

```
banner_variable: |
  This system is restricted to authorized users.
  Unauthorized access is prohibited.
  If you are unauthorized, terminate access now.
```

A newline character is also added to the end of the string. Use (|-) if you do not want a newline character added to the end of the string.

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

```
config_line: >
  description "This interface connects
  to interface1 on switch2 in the top
  of rack 10 in datacenter 4"
```

A newline character is still added to the end of the string. Use a greater-than symbol followed by a dash (>-) if you do not want a newline character added to the end of the string.

YAML Dictionaries

Dictionary key-value pairs are typically written as an indented block, as follows:

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

Dictionary key-value pairs can also be written in an inline block format inside braces, as follows:

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Avoid the inline block format because it is harder to read. However, this format is sometimes used with Ansible *roles*. When a playbook includes a list of roles, it is more common to use this syntax to make it easier to distinguish roles included in a play from the variables being passed to a role.

YAML Lists

Lists are typically written with the normal single-dash syntax:

```
hosts:
  - router1
  - router2
  - router3
```

Lists can also be written in an inline format in square braces, as follows:

```
hosts: [router1, router2, router3]
```

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

Obsolete Playbook Shorthand

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

```
tasks:
  - name: Shorthand form
    ansible.builtin.service: name=httpd enabled=true state=started
```

Normally you would write the same task as follows:

```
tasks:
  - name: Normal form
    ansible.builtin.service:
      name: httpd
      enabled: true
      state: started
```

You should generally avoid the shorthand form and use the normal form. Syntax highlighting tools in text editors such as VS Code can help you more effectively if you use the normal format than if you use the shorthand format.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

Become and Network Automation - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_privilege_escalation.html#become-and-network-automation

Ansible.Builtin - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/index.html

Network Platform Options - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html

YAML Syntax - Ansible Documentation
https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Creating a Cross-vendor Configuration Backup

Write and use a playbook containing multiple plays.

Outcomes

- Create a playbook with multiple plays to back up the configurations of Cisco IOS XE and Juniper Junos devices.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your environment for this exercise, and to ensure that all required resources are available.

```
[student@workstation ~]$ lab start implement-plays
```

Instructions

1. Open the `~/implement-plays` directory in VS Code and create the `inventory` file.

2. Add the Cisco IOS XE and Juniper Junos managed nodes, and the `all`, `ios`, and `junos` groups and variables to the `inventory` file.

   2.1. Add the `iosxe1.lab.example.com`, `iosxe2.lab.example.com`, `junos1.lab.example.com` and `junos2.lab.example.com` devices to the `inventory` file.

```
iosxe1.lab.example.com
iosxe2.lab.example.com
junos1.lab.example.com
junos2.lab.example.com
```

   2.2. Add the `all`, `ios`, and `junos` groups to the `inventory` file.

```
[all]
iosxe1.lab.example.com
iosxe2.lab.example.com
junos1.lab.example.com
junos2.lab.example.com

[ios]
iosxe1.lab.example.com
iosxe2.lab.example.com
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[junos]
junos1.lab.example.com
junos2.lab.example.com
```

2.3. Add the inventory variables to the `inventory` file. The final inventory file must contain the following content:

```
[all]
iosxe1.lab.example.com
iosxe2.lab.example.com
junos1.lab.example.com
junos2.lab.example.com

[ios]
iosxe1.lab.example.com
iosxe2.lab.example.com

[junos]
junos1.lab.example.com
junos2.lab.example.com

[all:vars]
ansible_user=student
ansible_ssh_private_key_file=~/.ssh/lab_rsa

[ios:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios

[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

3. In VS Code, create a playbook called `backupconfigs.yml`. Add a play to back up the configuration on the Cisco IOS XE managed nodes in the `ios` inventory group.

3.1. In VS Code, create a playbook called `backupconfigs.yml`. Add a play named "Backup IOS XE configurations". Set the `hosts:` keyword to `ios` and the `gather_facts` keyword to `false`.

```
---
- name: Backup IOS XE configurations
  hosts: ios
  gather_facts: false
  tasks:
```

3.2. Add a task named "Backup IOS XE configurations" to back up the configuration on the Cisco IOS XE managed nodes in the `ios` inventory group. Use the `{{ inventory_hostname }}` variable to name the backup file in the *`inventory_hostname`*`.txt` format.

```
    - name: Backup IOS XE configurations
      cisco.ios.ios_config:
        backup: true
        backup_options:
          dir_path: /home/student/implement-plays
          filename: "{{ inventory_hostname }}.txt"
```

4. Add a play to back up the configuration on the Juniper Junos managed nodes in the `junos` inventory group.

4.1. Add a play named "Backup Juniper Junos configurations". Set the `hosts:` keyword to `junos` and the `gather_facts` keyword to `false`.

```
- name: Backup Juniper Junos configurations
  hosts: junos
  gather_facts: false
  tasks:
```

4.2. Add a task named "Enable the netconf service on port 830" to enable the `netconf` service by using the `junipernetworks.junos.junos_netconf` module.

Set the `ansible_connection` variable to `ansible.netcommon.network_cli` to override the value set in the `inventory` file.

The `junipernetworks.junos.junos_config` module requires that the `netconf` service be enabled on the managed Junos nodes.

```
    - name: Enable the netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present
```

4.3. Add another task named "Backup Junos configurations" to back up the configuration by using the `junipernetworks.junos.junos_config` module.

Use the `{{ inventory_hostname }}` variable to name the backup file in the *`inventory_hostname`*`.txt` format.

This task uses the value of the `ansible_connection` variable set in the inventory file, `ansible.netcommon.netconf`.

```
    - name: Backup Junos configurations
      junipernetworks.junos.junos_config:
        backup: true
        backup_options:
          dir_path: /home/student/implement-plays
          filename: "{{ inventory_hostname }}.txt"
```

The completed playbook must contain the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Backup IOS XE configurations
  hosts: ios
  gather_facts: false
  tasks:
    - name: Backup IOS XE configurations
      cisco.ios.ios_config:
        backup: true
        backup_options:
          dir_path: /home/student/implement-plays
          filename: "{{ inventory_hostname }}.txt"

- name: Backup Juniper Junos configurations
  hosts: junos
  gather_facts: false
  tasks:
    - name: Enable the netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present

    - name: Backup Junos configurations
      junipernetworks.junos.junos_config:
        backup: true
        backup_options:
          dir_path: /home/student/implement-plays
          filename: "{{ inventory_hostname }}.txt"
```

Save the file.

5. Open a terminal and use the `ansible-navigator run` command to run the playbook:

```
[student@workstation implement-plays]$ ansible-navigator run \
backupconfigs.yml -m stdout

PLAY [Backup IOS XE configurations] ******************************************

TASK [Backup IOS XE configurations] ******************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY [Backup Juniper Junos configurations] ***********************************

TASK [Enable the netconf service on port 830] ********************************
changed: [junos1.lab.example.com]

TASK [Backup Junos configurations] *******************************************
changed: [junos1.lab.example.com]
```

```
PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
junos1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0 ...
```

6. Verify that the configuration backups were successful:

```
[student@workstation implement-plays]$ ls *.txt
iosxe1.lab.example.com.txt  iosxe2.lab.example.com.txt  junos1.lab.example.com.txt
```

7. Close the `/home/student/implement-plays` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

   7.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/implement-plays` directory.

   7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation implement-plays]$ cd
```

Finish

On the `workstation` machine, 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 implement-plays
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Implementing an Ansible Playbook

Configure and perform administrative tasks on managed nodes using a playbook.

Outcomes

- Create a custom Ansible inventory.
- Generate an automation content navigator configuration file.
- Create and run an Ansible playbook to configure the Message of the Day (MOTD) banners on network devices.

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 the required software, downloads an automation execution environment, and creates the project directory for the exercise.

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

Instructions

1. Create an Ansible inventory file named `inventory` in the `/home/student/implement-review` directory. Use the information in the following table to organize the groups in the Ansible inventory. Create as many inventory groups of managed nodes as needed.

Managed node	Operating system
`iosxe1.lab.example.com`	IOS XE
`iosxe2.lab.example.com`	IOS XE
`junos1.lab.example.com`	Junos
`junos2.lab.example.com`	Junos

Use the information in the following tables to define group variables for all the managed nodes in the Ansible inventory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

IOS Devices

Variable	Value
`ansible_user`	`student`
`ansible_ssh_private_key_file`	`~/.ssh/lab_rsa`
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`cisco.ios.ios`

Junos Devices

Variable	Value
`ansible_user`	`student`
`ansible_ssh_private_key_file`	`~/.ssh/lab_rsa`
`ansible_connection`	`ansible.netcommon.netconf`
`ansible_network_os`	`junipernetworks.junos.junos`

The variables from the tables control the way Ansible connects to the managed nodes:

- `ansible_user` specifies which username to use when connecting to the managed node.
- `ansible_ssh_private_key_file` specifies the private key file to use as authentication when connecting to the managed node.
- `ansible_connection` determines the connection type to the managed node.
- `ansible_network_os` informs Ansible of the network platform of the managed node.

2. Generate the automation content navigator configuration file in the `/home/student/implement-review` directory. Configure this file as follows:
   - Add an entry for the Ansible inventory file.
   - Use the `hub.lab.example.com/ee-supported-rhel8:latest` image as the default automation execution environment.
   - Set the pull policy to `missing`.
   - Set the mode to `stdout`.
   - Disable the creation of playbook artifacts.
3. Create a playbook called `motd_banner.yml` in the `/home/student/implement-review` directory. Add two plays to the playbook with the necessary entries to set up a MOTD banner on the IOS and Junos managed nodes. The MOTD banner is a message displayed

immediately after logging in to the managed node. The banner must read "This is an Ansible-managed node".

> **Note**
> Remember to always give meaningful names to plays and tasks, to make clear what they are doing.
>
> You can use the `cisco.ios.ios_banner` module to configure the banner in the IOS devices and the `junipernetworks.junos.junos_banner` module to configure the banner in the Junos devices.
>
> For the IOS managed nodes, use a handler that runs the `cisco.ios.ios_config` module to ensure that the banner changes are persistent.
>
> The `junipernetworks.junos.junos_banner` module requires that the `netconf` service be enabled on the Junos managed nodes. You can use the `junipernetworks.junos.junos_netconf` module to enable the Junos `netconf` service on port 830.

4. Run the `motd_banner.yml` playbook using automation content navigator. Optionally, verify the MOTD banner on managed nodes for the IOS and Junos devices.
5. Click **File** > **Close Folder** in VS Code to close the `/home/student/implement-review` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

```
[student@workstation implement-review]$ cd
```

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 implement-review
```

Finish

As the `student` user on the `workstation` machine, 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 implement-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Implementing an Ansible Playbook

Configure and perform administrative tasks on managed nodes using a playbook.

Outcomes

- Create a custom Ansible inventory.
- Generate an automation content navigator configuration file.
- Create and run an Ansible playbook to configure the Message of the Day (MOTD) banners on network devices.

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 the required software, downloads an automation execution environment, and creates the project directory for the exercise.

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

Instructions

1. Create an Ansible inventory file named `inventory` in the `/home/student/implement-review` directory. Use the information in the following table to organize the groups in the Ansible inventory. Create as many inventory groups of managed nodes as needed.

Managed node	Operating system
`iosxe1.lab.example.com`	IOS XE
`iosxe2.lab.example.com`	IOS XE
`junos1.lab.example.com`	Junos
`junos2.lab.example.com`	Junos

Use the information in the following tables to define group variables for all the managed nodes in the Ansible inventory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

IOS Devices

Variable	Value
`ansible_user`	`student`
`ansible_ssh_private_key_file`	`~/.ssh/lab_rsa`
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`cisco.ios.ios`

Junos Devices

Variable	Value
`ansible_user`	`student`
`ansible_ssh_private_key_file`	`~/.ssh/lab_rsa`
`ansible_connection`	`ansible.netcommon.netconf`
`ansible_network_os`	`junipernetworks.junos.junos`

The variables from the tables control the way Ansible connects to the managed nodes:

- `ansible_user` specifies which username to use when connecting to the managed node.
- `ansible_ssh_private_key_file` specifies the private key file to use as authentication when connecting to the managed node.
- `ansible_connection` determines the connection type to the managed node.
- `ansible_network_os` informs Ansible of the network platform of the managed node.

1.1. Open the `/home/student/implement-review` directory in Visual Studio Code.

- In VS Code, click **View** > **Explorer** and then click **File** > **Open Folder**.
- Navigate to **Home** > **implement-review** and then click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'** in the pop-up window, and then click **Yes, I trust the authors**.

1.2. Create a custom static inventory named `inventory` in the `/home/student/implement-review` directory. Complete the following substeps if you do it in VS Code:

- Click **File** > **New File**, enter `inventory` as the file, and then click **Create New File**.
- Click **Create File** in the **Files** window behind VS Code. The **Files** window closes after the inventory file is created.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1.3. Edit the Ansible inventory file so that it contains the following content:

```
[ios]
iosxe[1:2].lab.example.com

[junos]
junos[1:2].lab.example.com

[network_devices:children]
ios
junos

[network_devices:vars]
ansible_user=student
ansible_ssh_private_key_file=~/.ssh/lab_rsa

[ios:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios

[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

2. Generate the automation content navigator configuration file in the `/home/student/implement-review` directory. Configure this file as follows:

- Add an entry for the Ansible inventory file.
- Use the `hub.lab.example.com/ee-supported-rhel8:latest` image as the default automation execution environment.
- Set the pull policy to `missing`.
- Set the mode to `stdout`.
- Disable the creation of playbook artifacts.

2.1. Use the `ansible-navigator settings --effective` command in a terminal to display the effective settings. Add the `--eei` and `--pp` options to the automation content navigator command to specify the automation execution environment image and the `missing` pull policy. Redirect the command output into a file named `sample.yml`.

If you are using VS Code, click **Terminal** > **New Terminal** to open a terminal.

```
[student@workstation implement-review] ansible-navigator settings \
--effective --eei hub.lab.example.com/ee-supported-rhel8 \
--pp missing > sample.yml
```

2.2. Rename the `sample.yml` file to `ansible-navigator.yml`.

```
[student@workstation implement-review] mv sample.yml ansible-navigator.yml
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.3. Edit the `ansible-navigator.yml` file removing the lines not related to what was requested, and adding an entry for the inventory file. After updating the `ansible-navigator.yml` file, it must have the following content.

```
---
ansible-navigator:
  ansible:
    inventory:
      entries:
        - ./inventory
  execution-environment:
    image: hub.lab.example.com/ee-supported-rhel8:latest
    pull:
      policy: missing
  mode: stdout
  playbook-artifact:
    enable: false
```

Note

This lab solution removes lines so that you can focus on a small group of settings, but the lines do not need to be removed.

3. Create a playbook called `motd_banner.yml` in the `/home/student/implement-review` directory. Add two plays to the playbook with the necessary entries to set up a MOTD banner on the IOS and Junos managed nodes. The MOTD banner is a message displayed immediately after logging in to the managed node. The banner must read "This is an Ansible-managed node".

Note

Remember to always give meaningful names to plays and tasks, to make clear what they are doing.

You can use the `cisco.ios.ios_banner` module to configure the banner in the IOS devices and the `junipernetworks.junos.junos_banner` module to configure the banner in the Junos devices.

For the IOS managed nodes, use a handler that runs the `cisco.ios.ios_config` module to ensure that the banner changes are persistent.

The `junipernetworks.junos.junos_banner` module requires that the `netconf` service be enabled on the Junos managed nodes. You can use the `junipernetworks.junos.junos_netconf` module to enable the Junos `netconf` service on port 830.

3.1. Either in the GNOME terminal or in VS Code, create the `motd_banner.yml` file in the `/home/student/implement-review` directory. Add the following entry to the beginning of the file to indicate YAML format.

```
---
```

3.2. Add the following entry in the playbook file to define the start of the first play.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
- name: Configure a banner on IOS devices
```

3.3. Add the following entries to indicate that the play applies to the IOS network devices, and to disable fact gathering for these managed nodes. Make sure that the beginning of the entries are indented two spaces.

```
  hosts: ios
  gather_facts: false
```

3.4. Add the following entries to define the beginning of the task list and the first task. Indent the beginning of the task list two spaces, and indent the line for the first task four spaces. Pay attention to adding two more spaces to the indentation of each lower level.

```
  tasks:
    - name: Configure a MOTD banner
      cisco.ios.ios_banner:
        banner: motd
        text: This is an Ansible-managed node
        state: present
```

3.5. Changes made by the `cisco.ios.ios_banner` module are not persistent. Add the highlighted lines so that if the `Configure a MOTD banner` task results in a change, the task triggers a handler that saves the running configuration to the startup configuration.

Indent the `notify` line at the same level as the module name, and indent the `handlers` line at the same level as the `tasks` line.

```
  tasks:
    - name: Configure a MOTD banner
      cisco.ios.ios_banner:
        banner: motd
        text: This is an Ansible-managed node
        state: present
      notify: ios_save_changes

  handlers:
    - name: ios_save_changes
      cisco.ios.ios_config:
        save_when: always
```

3.6. Add the following entries to the `motd_banner.yml` playbook to denote the start of the second play. Indicate that the play applies to the Junos network devices, and disable fact gathering for these managed nodes.

Make sure that the beginning of the entry is not indented, and indent the beginning of the next entries two spaces.

```
- name: Configure a banner on Junos devices
  hosts: junos
  gather_facts: false
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.7. Add the following entries to define the beginning of the task list and the first task for the second play. The task must enable the `netconf` service on port 830. Set the `ansible_connection` variable to `ansible.netcommon.network_cli` to override the value set in the inventory file.

Pay attention to the indentation for each level.

- Indent the beginning of the task list two spaces, and indent the first task four spaces.
- The lines for the variables definition and the `junipernetworks.junos.junos_netconf` module have six spaces of indentation.
- The defined `ansible_connection` variable and the parameters for the `junipernetworks.junos.junos_netconf` module are indented eight spaces.

```
  tasks:
    - name: Enable the netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present
```

3.8. Add the following entries to define the second task for the second play. Use the `junipernetworks.junos.junos_banner` module to configure the MOTD message. Indent the line for the task name four spaces. Indent the beginning of the module entry six spaces, and indent the beginning of the module parameters eight spaces.

```
    - name: Configure a MOTD banner
      junipernetworks.junos.junos_banner:
        banner: motd
        text: This is an Ansible-managed node
        state: present
```

3.9. The complete `motd_banner.yml` playbook reads as follows:

```
---
- name: Configure a banner on IOS devices
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure a MOTD banner
      cisco.ios.ios_banner:
        banner: motd
        text: This is an Ansible-managed node
        state: present
      notify: ios_save_changes

  handlers:
    - name: ios_save_changes
      cisco.ios.ios_config:
        save_when: always
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
- name: Configure a banner on Junos devices
  hosts: junos
  gather_facts: false
  tasks:
    - name: Enable the netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present

    - name: Configure a MOTD banner
      junipernetworks.junos.junos_banner:
        banner: motd
        text: This is an Ansible-managed node
        state: present
```

Note
You can use different names for the plays and tasks than this solution uses.

4. Run the `motd_banner.yml` playbook using automation content navigator. Optionally, verify the MOTD banner on managed nodes for the IOS and Junos devices.

4.1. Use the `ansible-navigator run` command to run the `motd_banner.yml` playbook. Read through the generated output to ensure that all tasks completed successfully.

```
[student@workstation implement-review]$ ansible-navigator run motd_banner.yml

PLAY [Configure a banner on IOS devices] ***************************************

TASK [Configure a MOTD banner] *************************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

RUNNING HANDLER [ios_save_changes] *********************************************
changed: [iosxe2.lab.example.com]
changed: [iosxe1.lab.example.com]

PLAY [Configure a banner on Junos devices] *************************************

TASK [Enable the netconf service on port 830] **********************************
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

TASK [Configure a MOTD banner] *************************************************
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0   ...
```

```
iosxe2.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0  ...
junos1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0  ...
junos2.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0  ...
```

5. Click **File** > **Close Folder** in VS Code to close the `/home/student/implement-review` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

```
[student@workstation implement-review]$ cd
```

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 implement-review
```

Finish

As the `student` user on the `workstation` machine, 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 implement-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- An *inventory* defines nodes in your network topology that Ansible manages. A *static inventory* can be defined in a text file.
- A *play* is an ordered list of tasks that run against nodes selected from the inventory.
- A *playbook* is a text file, written in YAML format, that contains a list of one or more plays to run in order.
- A task is implemented using standardized code packaged as Ansible modules.
- Ansible modules are packaged into *Ansible Content Collections*, which are a way to distribute Ansible content, which can consist of modules, roles, plug-ins, and playbooks.
- In general, tasks in an Ansible Playbook are idempotent, which means that they do not change the managed node if the node's current state matches its desired state.
- The `ansible-navigator settings` command can be used to review and manage that command's configuration settings. The `--sample` option generates a sample configuration file, and the `--effective` option generates a configuration file that reflects the current configuration.
- The `ansible-navigator doc` and `ansible-navigator collections` commands can list modules in your automation execution environments, and provide documentation and example code snippets of how to use them in playbooks.
- The `ansible-navigator run` command is used to run playbooks and validate playbook syntax.
- Several types of network automation modules are available, such as facts modules, command modules, config modules, and resource modules.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 3

Managing and Running Playbooks

Goal

Manage automation code in version control and run Ansible Playbooks from a centrally managed automation controller.

Objectives

- Create and manage Ansible Playbooks in a Git repository, following recommended practices.
- Navigate and describe the automation controller web UI, and successfully launch a job using a job template, project, credential, and inventory.
- Identify issues in Ansible Playbooks and repair them.
- Verify that Ansible can communicate with managed network devices, and troubleshoot any problems with those connections.

Sections

- Managing Ansible Project Materials Using Git (and Guided Exercise)
- Running Playbooks in Automation Controller (and Guided Exercise)
- Troubleshooting Playbooks (and Guided Exercise)
- Troubleshooting Ansible Network Communication (and Guided Exercise)

Lab

- Managing and Running Playbooks

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Managing Ansible Project Materials Using Git

Objectives

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

Introducing Git

Git is a *distributed version control system (DVCS)* that enables you to collaboratively manage changes to files.

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 to files 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 changes 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 changes, you *commit* the changes and then *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 files 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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Initial Git Configuration

Because Git users often modify projects with multiple contributors, Git records the user's name and email address on each of their commits. You can define these values 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
```

Starting the Git Workflow

When you first start working on an existing project, you *clone* a remote repository to create a local repository in a subdirectory of 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.

To clone a repository from the command line, use the `git clone` command, followed by the URL that specifies the location of the remote repository that you want to clone.

For example, the following command clones the `netconfig.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:netconfig.git
```

To clone a repository in VS Code, click the **Source Control** icon in the left navigation menu, or click **View** > **Source Control** to display the Source Control panel. Click **Clone Repository** in the Source Control panel.

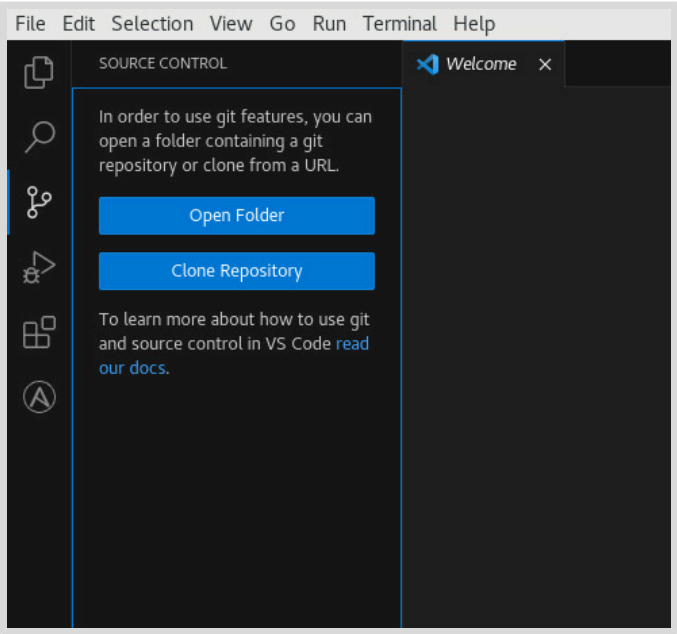


Figure 3.1: Cloning a repository in VS Code

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 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.

To add all changed files in the current directory and subdirectories, use the `git add .` command. To add all changed files in every directory in the entire working tree, use the `git add -A` command.

> **Note**
> The `git add .` and `git add -A` commands can simultaneously add multiple files to the staging area.
>
> If you are working on multiple changes at the same time, then you might want to add each file separately so that you can organize the files into different commits for better tracking of changes.

To view staged files in VS Code, click the **Source Control** icon in the left navigation menu.

To include a file in the next commit, click the plus (+) symbol next to the file to add the file to the Staged Changes section. Click the minus (-) symbol to remove changed files from the Staged Changes section.

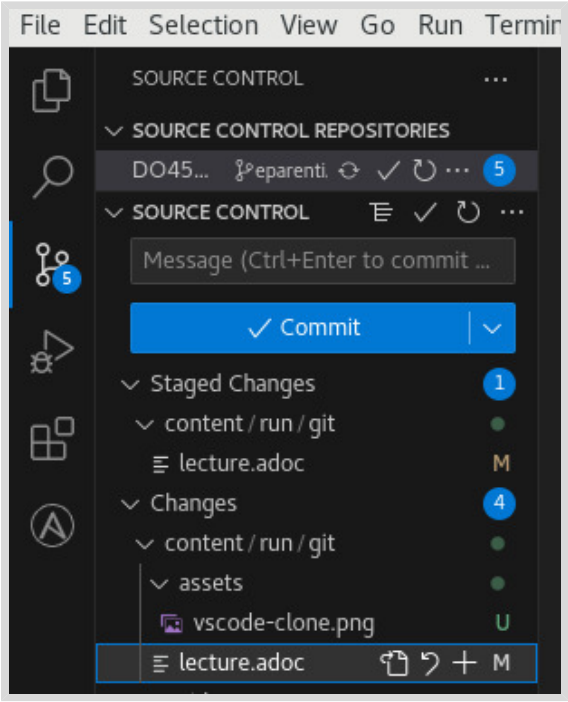


Figure 3.2: Staging changes in VS Code

Note that a capital **M** character appears next to modified files.

To commit your staged changes, type a commit message in the upper text box and click **Commit**. This saves your changes to the local Git repository.

Commit messages do not have to be long, but they should be meaningful so that they are useful.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

Meaningful and concise commit messages are the key to maintaining a clear history for an Ansible project. Use descriptive commit messages and add references to issues, tickets, and related people where applicable.

Synchronizing Changes

From the command line, the `git push` command uploads commits you made in your 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.

Likewise, the `git pull` command downloads changes to your local repository from the remote repository. 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 often to stay current with the changes that others are making to the project in the remote repository.

To synchronize changes by using `git pull` and `git push` in VS Code, click the **More Actions** menu (...) and then click **Pull** or **Push**, or choose an option in the **Pull, Push** submenu.

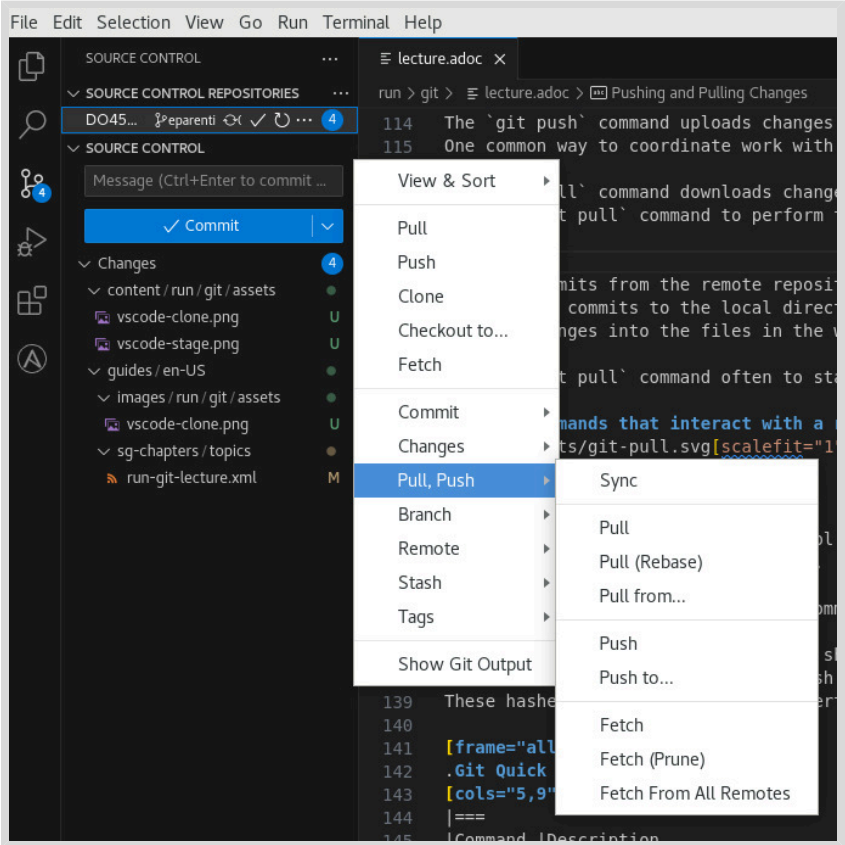


Figure 3.3: Synchronizing local changes with a remote repository

Working with Branches and References

You can use *branches* to update your project and work on new development without altering your main branch. When you are satisfied with your changes, you can *merge* the changes in your branch to the main development branch, integrating them.

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

Overview of Git Branching

Git version control features a branching model to track code changes. All Git repositories have a base branch. Many projects prefer to use `main` as the name of the base branch. When you create a commit in your repository, the base branch is updated with the new commit. Git uses the term *HEAD* to describe the pointer to where you are in the currently active branch.

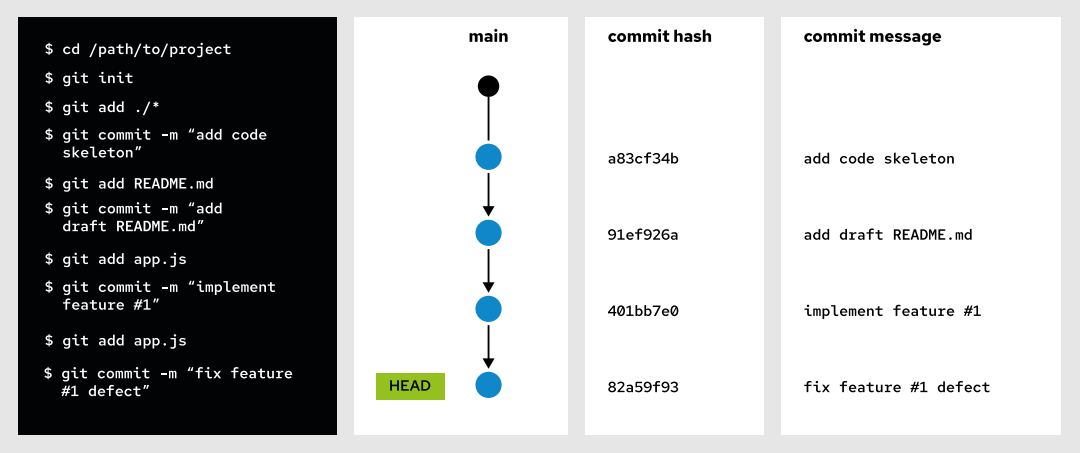


Figure 3.4: Commits to the main branch of a Git repository.

By convention, the `main` or `master` branch in a Git repository contains the latest, stable version of the source code. To implement a new feature or functionality, create a branch from the `main` branch. This new branch, called a *feature branch*, contains commits corresponding to code changes for the new feature. The `main` branch is not affected by commits to the feature branch.

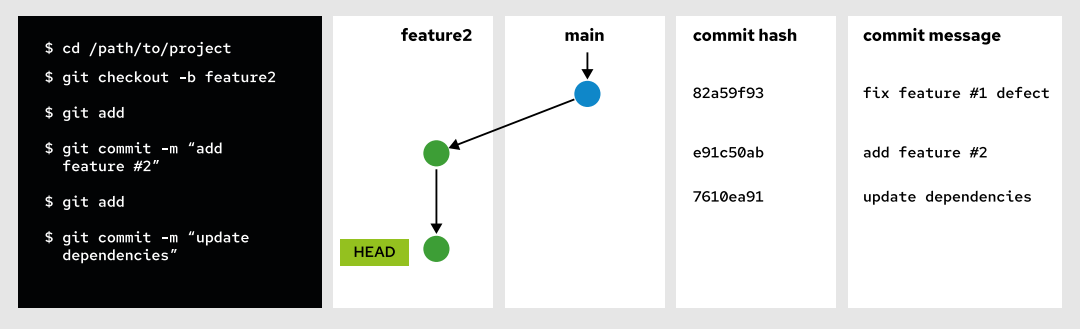


Figure 3.5: Commits to a feature branch of a Git repository.

When you use a branch for feature development, you can commit and share your code often without impacting the stability of code in the `main` branch. After ensuring that the code in the

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

feature branch is complete, tested, and reviewed, you are ready to merge the branch into another branch, such as the `main` branch. *Merging* is the process of combining the commit histories from two separate branches into a single branch.

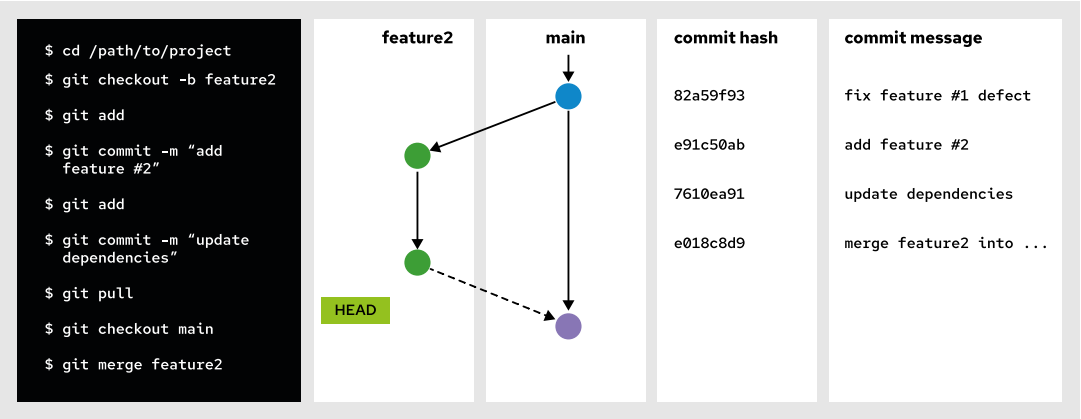


Figure 3.6: Merging a feature branch into the main branch.

Merge Conflicts

Git has sophisticated mechanisms to merge code changes from one branch into another branch. However, if there are changes to the same file in both branches, then a *merge conflict* can occur.

A merge conflict indicates that Git cannot automatically determine how to integrate changes from both branches. When this happens, Git inserts markers in each affected file to indicate the sections that contain content conflicts from both branches.

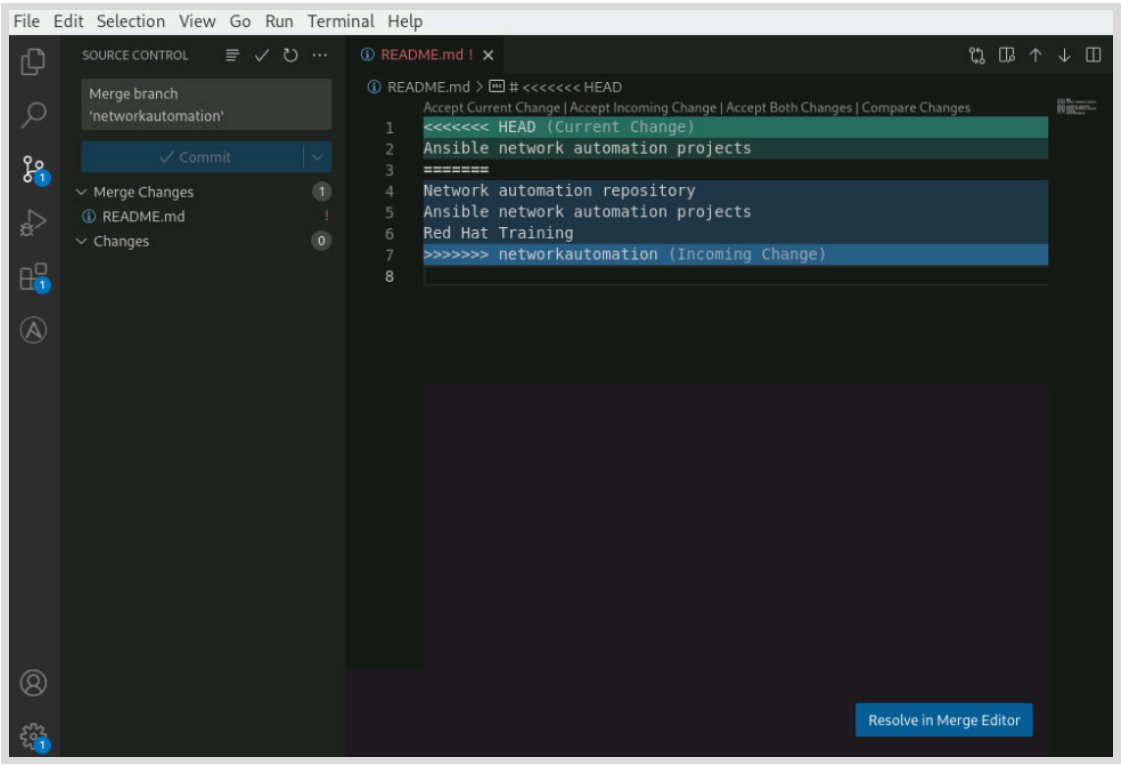


Figure 3.7: Viewing a merge conflict in VS Code.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For each conflict in a file, replace the content between the merge conflict markers (<<<<<<<< and >>>>>>>> inclusive) with the correct content from one branch or the other, or an appropriate combination of content from both branches.

You can also click **Resolve in Merge Editor** and resolve the merge conflict there.

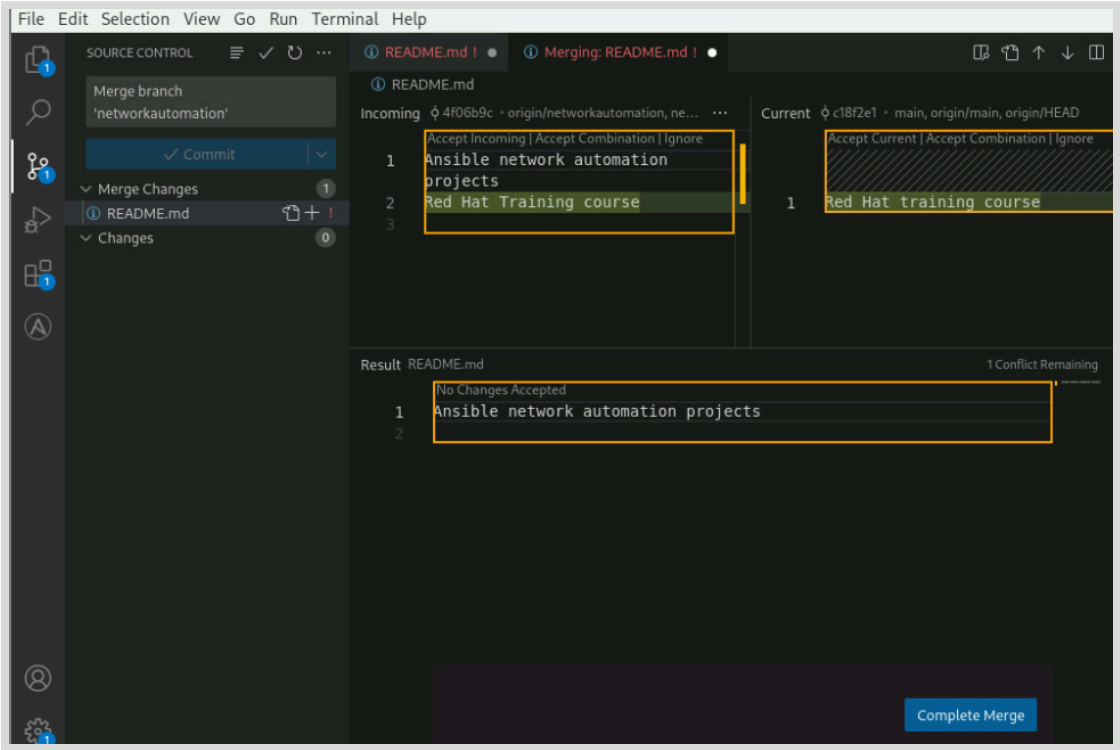


Figure 3.8: Resolving a merge conflict in VS Code.

After you reconcile the content for each conflict in a file, you need to save, stage, and commit the file changes. You can then synchronize with the remote repository.

> **Note**
> For more information on merge conflicts, see **Basic Merge Conflicts** at `https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging`

Creating Branches

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

From the command line, you can use the `git branch` command to create a 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.

To create a branch in VS Code, click the **Source Control** icon in the left navigation menu, or click **View** > **Source Control** to display the Source Control panel. Click the **Views and More Actions** icon (...) and then click **Branch** > **Create Branch**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

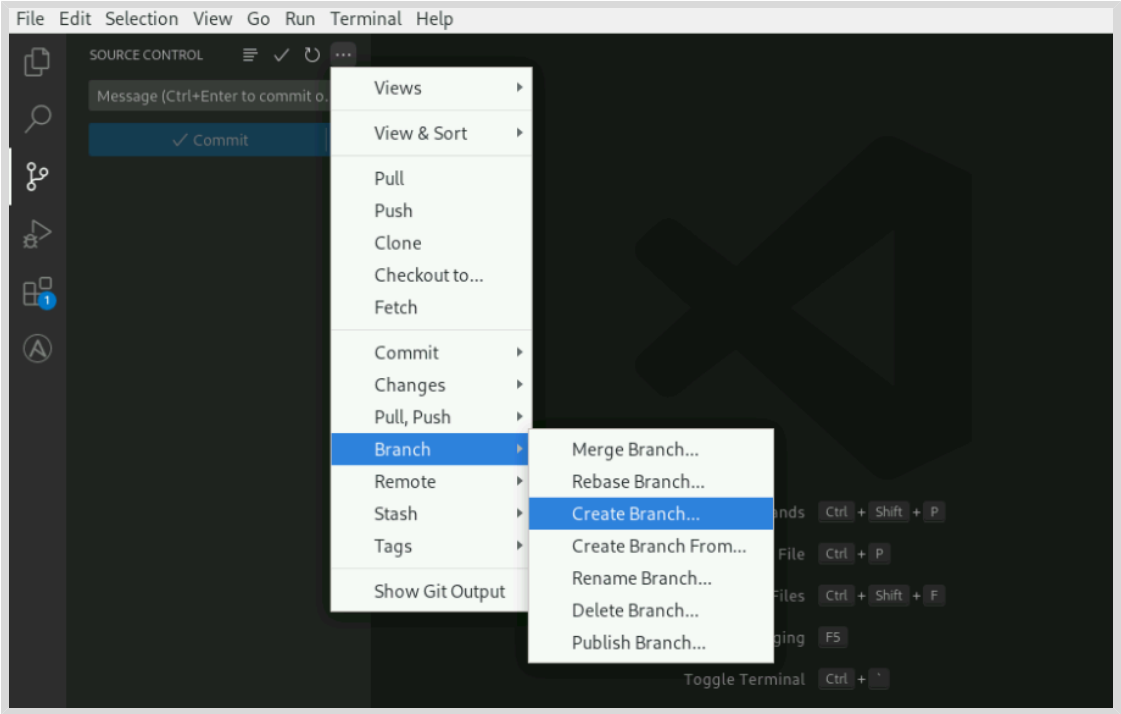


Figure 3.9: Creating a new branch in VS Code.

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.

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.

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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Git Quick Reference

Command	Description
`git status`	Display the status of files in the working tree.
`git log`	Display the commit log messages.
`git show` *`commit-hash`*	Show what was in the change set for a particular commit hash.
`git revert` *`commit-hash`*	Create a 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.

Important

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

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

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
ios.yml                   # playbook for IOS managed nodes
junos.yml                 # playbook for Junos managed nodes

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

roles/
    requirements.yml      # specifies roles needed by this project
  ...additional roles...
```

Ansible Roles and Ansible Content Collections

Ansible Roles and Ansible Content Collections are ways that you can reuse code from other Ansible projects.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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.

A role or collection can have its own Git repository. Then instead of keeping a copy of the role or collection in your Ansible project's repository, automation controller can retrieve it directly from the relevant role or collection repository when it runs playbooks that belong to the project.

To set this up, create `requirements.yml` files in your `roles/` and `collections/` directories. Then 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.

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

Each line in a `.gitignore` file represents a pattern to match, and 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. You can use blank lines to improve readability, and any line that starts with a number sign (#) 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.)

For example, 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.

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

```
roles/**
!roles/requirements.yml
collections/ansible_collections
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

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

Git
https://git-scm.com

Sample Ansible Setup: Sample Directory Layout - Ansible Documentation
https://docs.ansible.com/ansible/6/user_guide/sample_setup.html#sample-directory-layout

Git Branching and Merging
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Using Git Source Control in VS Code
https://code.visualstudio.com/docs/sourcecontrol/overview

Introduction to Git in VS Code
https://code.visualstudio.com/docs/sourcecontrol/intro-to-git

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 them to the original repository.

Outcomes

- Clone a Git repository to the `workstation` machine.
- Create a playbook to configure the Simple Network Management Protocol (SNMP) on Cisco IOS XE managed nodes.
- Stage the playbook file, commit the changes, and synchronize with the remote repository.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your environment for this exercise, and to ensure that all required resources are available. The command also removes any Simple Network Management Protocol (SNMP) configuration from the Cisco IOS XE managed nodes.

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

Instructions

1. Open a terminal and verify that SNMP is not configured on the `iosxe1.lab.example.com` managed node.

   1.1. Connect to the `iosxe1.lab.example.com` managed node by using SSH:

```
[student@workstation ~]$ ssh student@iosxe1.lab.example.com

iosxe1.lab.example.com#
```

   1.2. Display the SNMP configuration for the `iosxe1.lab.example.com` managed node. The command output indicates that the SNMP agent is not enabled:

```
iosxe1.lab.example.com#show snmp
%SNMP agent not enabled
iosxe1.lab.example.com#
```

   1.3. Return to the `workstation` machine:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
iosxe1.lab.example.com#exit
Connection to iosxe1.lab.example.com closed by remote host.
Connection to iosxe1.lab.example.com closed.
[student@workstation ~]$
```

2. Clone the `git@git.lab.example.com:student/snmp` Git repository to the `/home/student/git-repos` directory. Create a branch named `exercise` and check out that branch.

2.1. 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
[student@workstation git-repos]$
```

2.2. Clone the `git@git.lab.example.com:student/snmp` repository, and then change to the repository directory:

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

2.3. Create and check out the `exercise` branch:

```
[student@workstation snmp]$ git branch exercise
[student@workstation snmp]$ git checkout exercise
Switched to branch 'exercise'
```

Note

You can also use the `git checkout -b exercise` command to create and check out the `exercise` branch by using a single command:

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

3. Open VS Code and create a playbook named `cisco_snmp.yml` in the `/home/student/git-repos/snmp` directory.

3.1. Open VS Code and click **File** > **Open Folder**. Navigate to **Home** > **git-repos** > **snmp** and then click **Open**.

3.2. In VS Code, click **File** > **New Text File**. Click **Select a language** and then click **Ansible**.

3.3. Create a playbook with the following content and save it as `cisco_snmp.yml`:

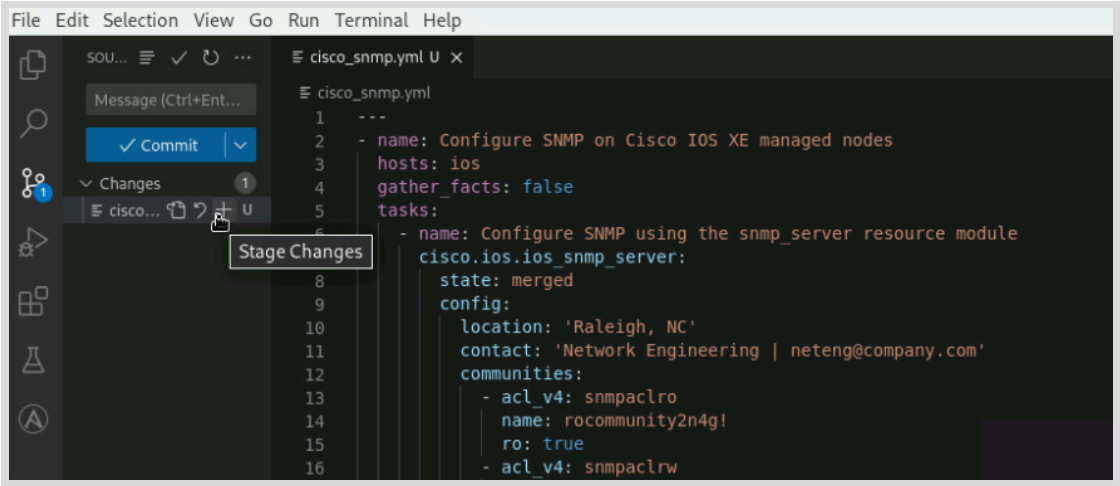
For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Configure SNMP on Cisco IOS XE managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure SNMP using the snmp_server resource module
      cisco.ios.ios_snmp_server:
        state: merged
        config:
          location: 'Raleigh, NC'
          contact: 'Network Engineering | neteng@company.com'
          communities:
            - acl_v4: snmpaclro
              name: rocommunity2n4g!
              ro: true
            - acl_v4: snmpaclrw
              name: rwcommunityd7g$v
              rw: true
          hosts:
            - community_string: 'm0n1t0r'
              host: 172.25.1.10
              informs: true
              version: 2c
          traps:
            bgp:
              enable: true
            eigrp: true
            hsrp: true
            ospf:
              cisco_specific:
                error: true
                lsa: true
                retransmit: true
            snmp:
              authentication: true
              coldstart: true
              linkdown: true
              linkup: true
              warmstart: true
```

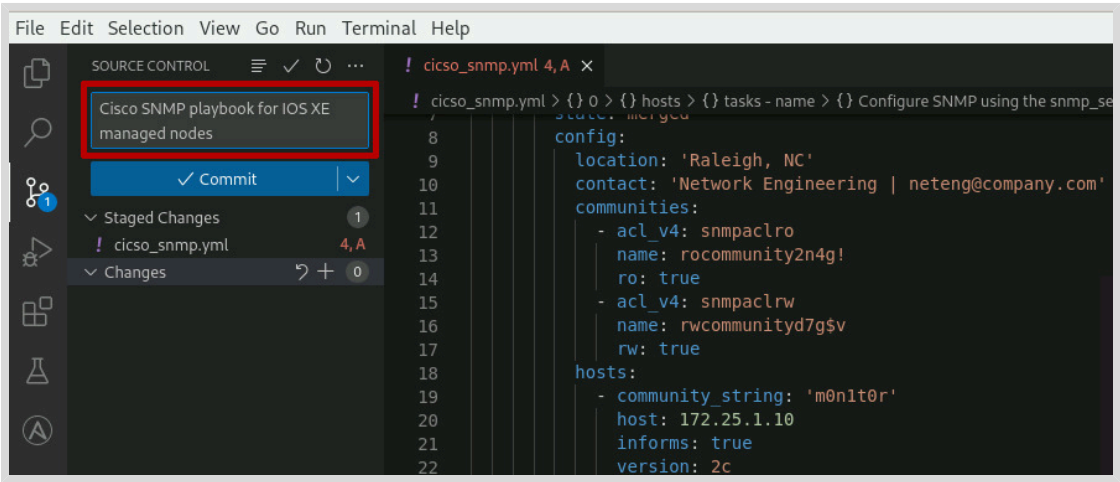
4. In VS Code, stage and commit the SNMP playbook.

4.1. Click the **Source Control** icon in the left navigation menu, or click **View** > **Source Control** to display the Source Control panel.

4.2. Hover over your mouse over the file. A plus (+) symbol next to the file appears. Click the plus symbol to add the file to the Staged Changes section.



4.3. In the Message field in the Primary Side Bar, enter "Cisco SNMP playbook for IOS XE managed nodes" as the commit message and click **Commit**.



5. Return to the terminal and run the `cisco_snmp.yml` playbook.

5.1. Copy all files from the `~/run-git` directory to the `~/snmp` directory. This copies the `ansible-navigator.yml` and `inventory` files to the repository directory so that the playbook runs. Add and commit these files to complete the project:

```
[student@workstation snmp]$ cp ~/run-git/* .
[student@workstation snmp]$ git add ansible-navigator.yml inventory
[student@workstation snmp]$ git commit -m "Add automation navigator config file
 and managed node inventory file"
[exercise 08eb493] Add automation navigator config file and managed node inventory
 file
 2 files changed, 5 insertions(+), 1 deletion(-)
[student@workstation snmp]
```

5.2. Run the `cisco_snmp.yml` playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation snmp]$ ansible-navigator run cisco_snmp.yml

PLAY [Configure SNMP on Cisco IOS XE managed nodes] ***************************

TASK [Configure SNMP using the snmp_server resource module] *******************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
```

6. In a terminal, verify that the job successfully deployed SNMP.

6.1. Connect to the `iosxe1.lab.example.com` managed node as the `student` user:

```
[student@workstation snmp]$ ssh student@iosxe1.lab.example.com

iosxe1.lab.example.com#
```

6.2. Display the SNMP configuration for the managed node:

```
iosxe1.lab.example.com#show snmp
...output omitted...
SNMP informs: enabled
    Informs in flight 0/25 (current/max)
    Logging to 172.25.1.10.162
        0 sent, 0 in-flight, 0 retries, 0 failed, 0 dropped
```

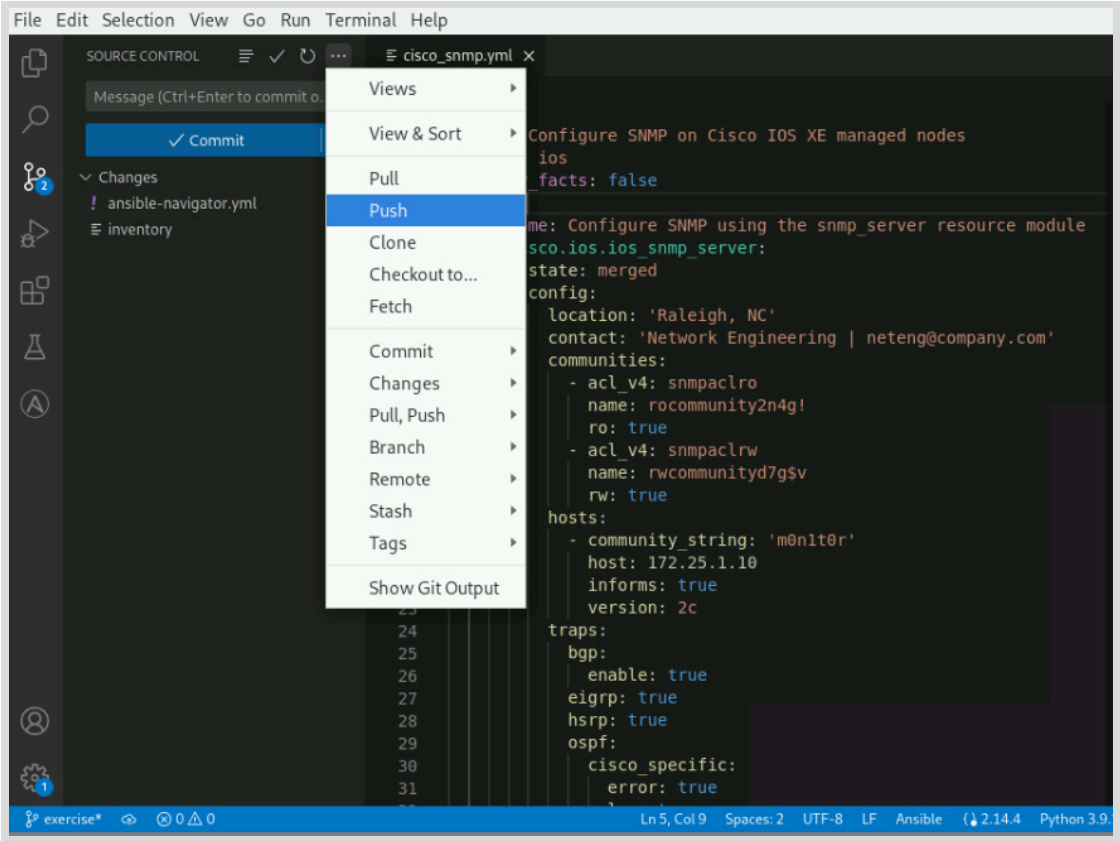
6.3. Return to the `workstation` machine:

```
iosxe1.lab.example.com#exit
Connection to iosxe1.lab.example.com closed by remote host.
Connection to iosxe1.lab.example.com closed.
[student@workstation snmp]$
```

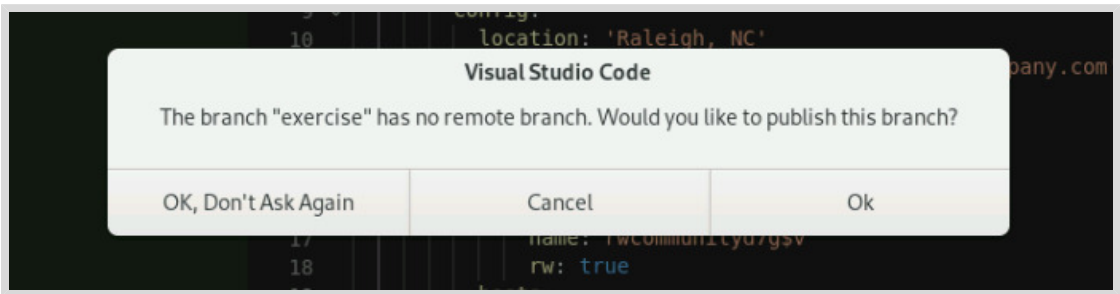
7. In the VS Code **Source Control** panel, synchronize the local commits with the remote repository.

7.1. Click the **Views and More Actions** icon (...) and then click **Push**.

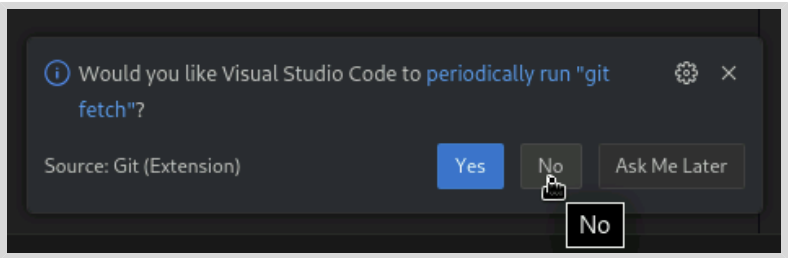
For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.



7.2. You might see a message asking you to publish the `exercise` branch. Click `Ok`.



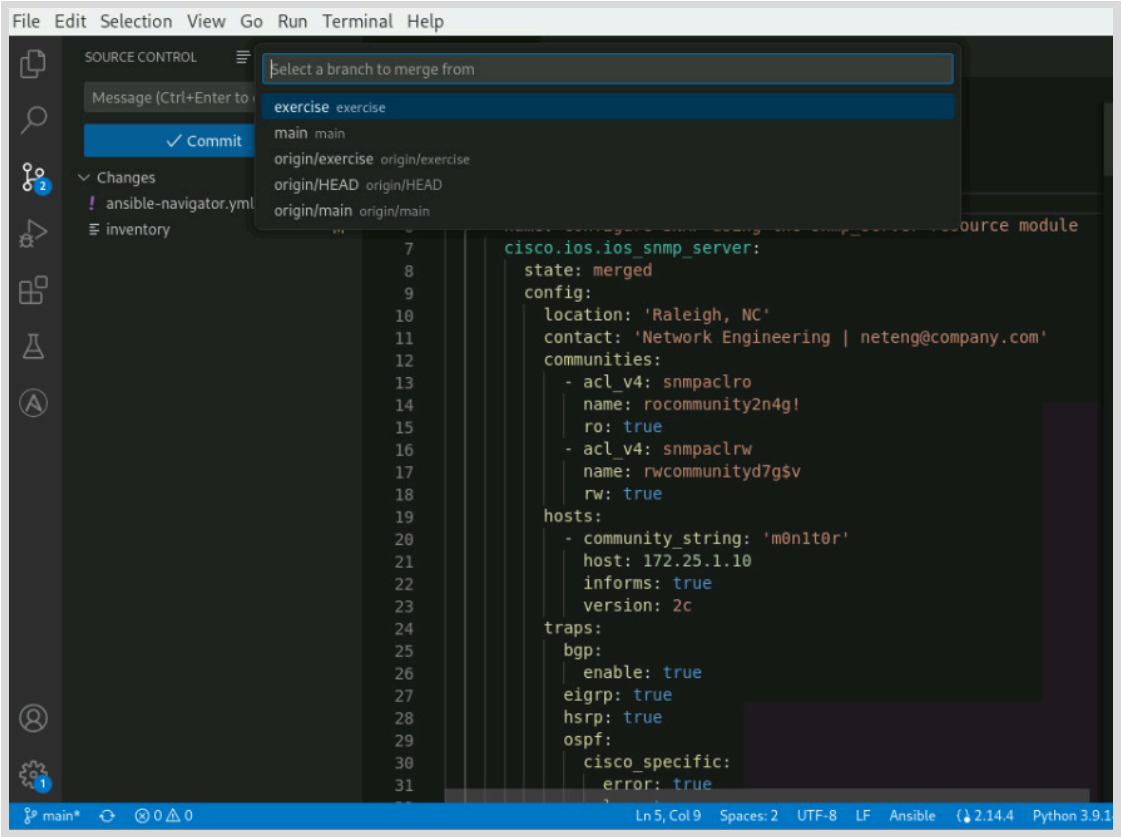
7.3. You might see another message asking you to periodically run `git fetch`. Click `No`.



8. Merge the `exercise` branch into the `main` branch.

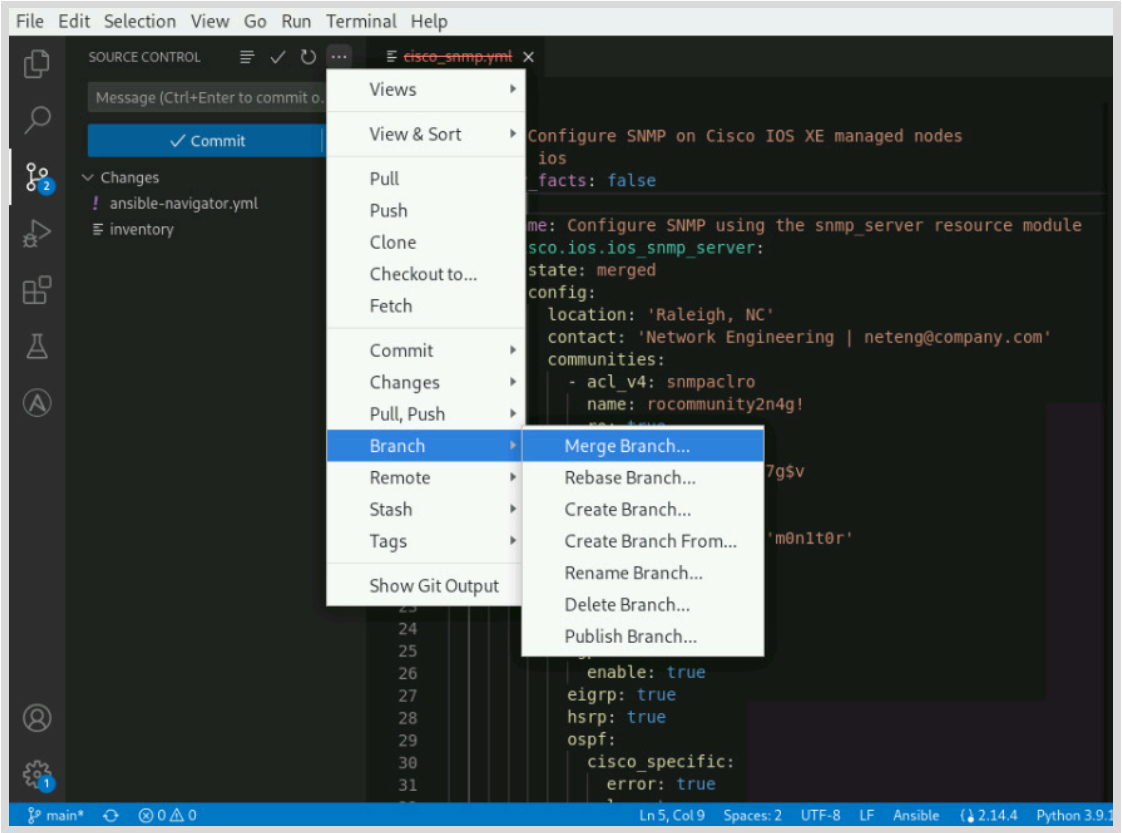
8.1. Click **exercise** on the bottom status bar to display the menu to change branches. Select the **main** branch.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.



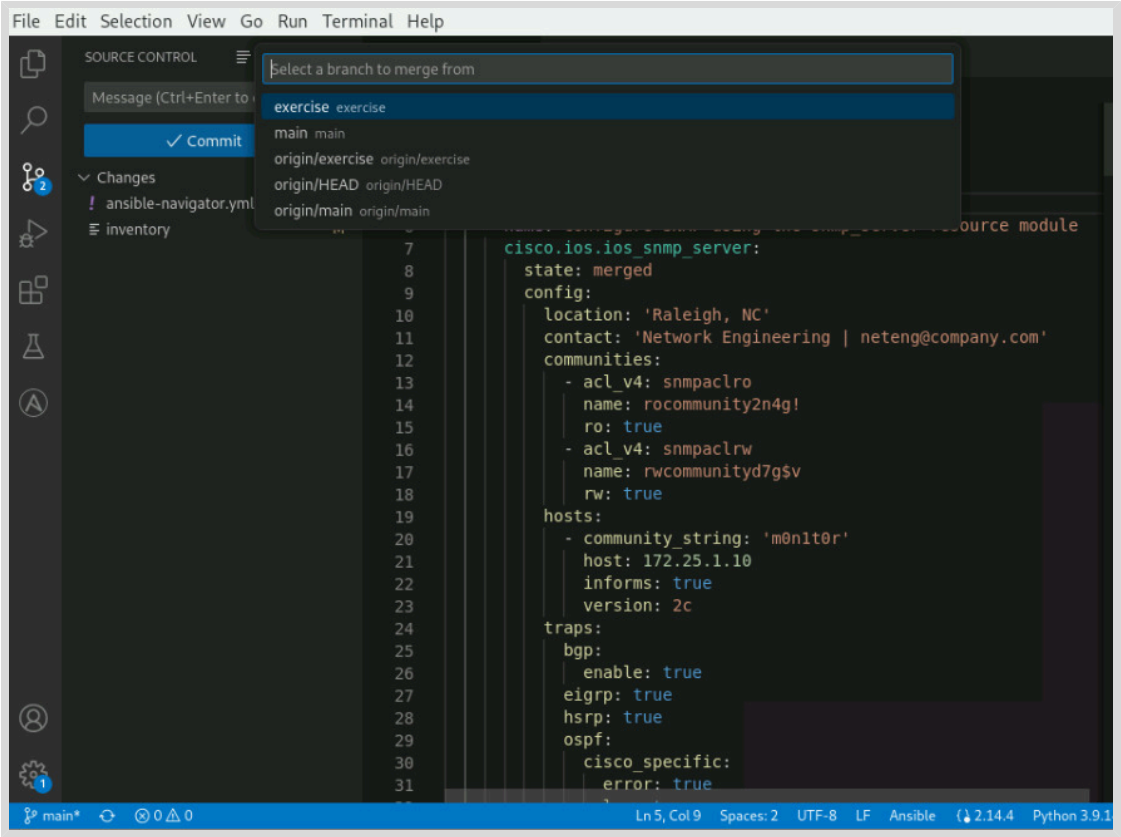
8.2. Click the **Views and More Actions** icon (...) and then click **Branch** > **Merge Branch**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.



8.3. Select the `exercise` branch.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.



9. Close the `/home/student/git-repos/snmp` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

   9.1. Click **File** > **Close Folder** to close the `/home/student/git-repos/snmp` directory.

   9.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation snmp]$ cd
```

Finish

On the `workstation` machine, 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 run-git
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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.

You run playbooks by creating and then running job templates. Several resources must exist before you can create a job template.

Typically, the following resources are required:

- 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 extensive 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/Ansible Automation Hub API Token

Create the credentials resource so that automation controller can download content collections and roles from Ansible Galaxy, Ansible 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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Container Registry

Create this type of credential if you need to authenticate before you can pull 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 change 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 the credential details.

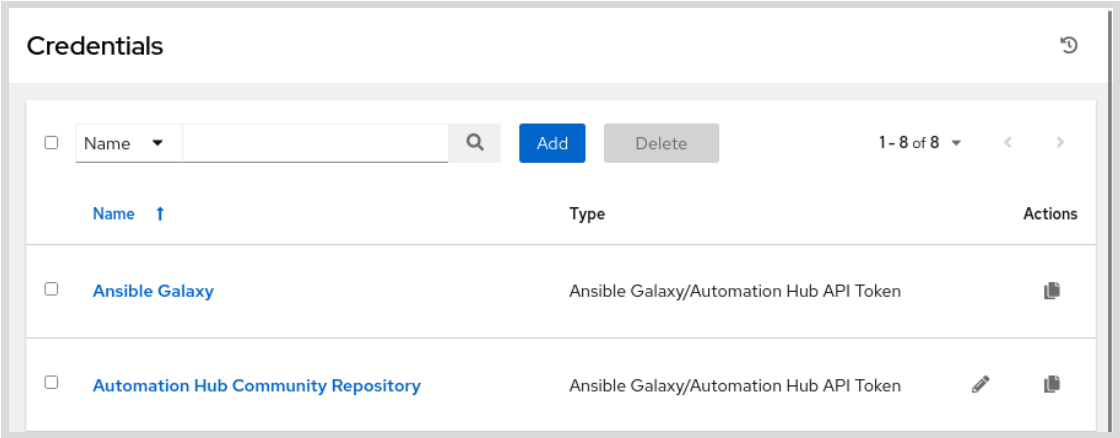


Figure 3.18: List of existing credentials

Creating a Machine Credential

To create a machine credential, navigate to **Resources** > **Credentials** and then click **Add**. Specify a name for the credential and choose the machine credential type. Specify any additional settings and then click **Save**.

Setting	Description
Username	Automation controller connects to the managed hosts as this user (similar to the `remote_user` setting in an `ansible.cfg` 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 same as the `become_method` setting in an `ansible.cfg` file or the `ansible_become_method` setting in an inventory file). Use the `enable` method for network devices.
Privilege Escalation Username	The user to become for tasks that require privilege escalation (the same as the `become_user` setting in an `ansible.cfg` file).
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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Creating a Source Control Credential

To create a 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 source code management (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 SCM private 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.

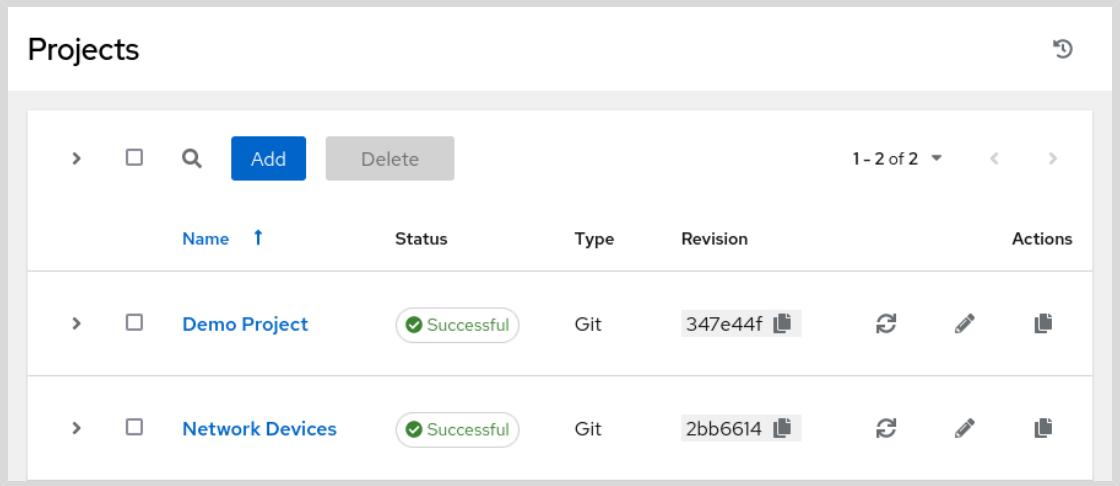


Figure 3.19: List of existing projects

To create a 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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Choose a source control credential type, such as Git, specify any 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 *existing* 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 following:

- 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

You can create inventory resources within automation controller, just as you can with a local inventory file or directory. Inventories can contain groups of hosts as well as ungrouped hosts. You can configure variables that apply to the entire inventory, to a specific inventory group, and to a specific inventory host. 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. Static inventories are inventories that you manually create and manage within automation controller. Static inventories display **Disabled** in the **Sync Status** column. You can also create inventories that query an inventory source, such as an inventory defined in an automation controller project. Inventories that contain content from an inventory source display the status of the most recent inventory synchronization attempt.

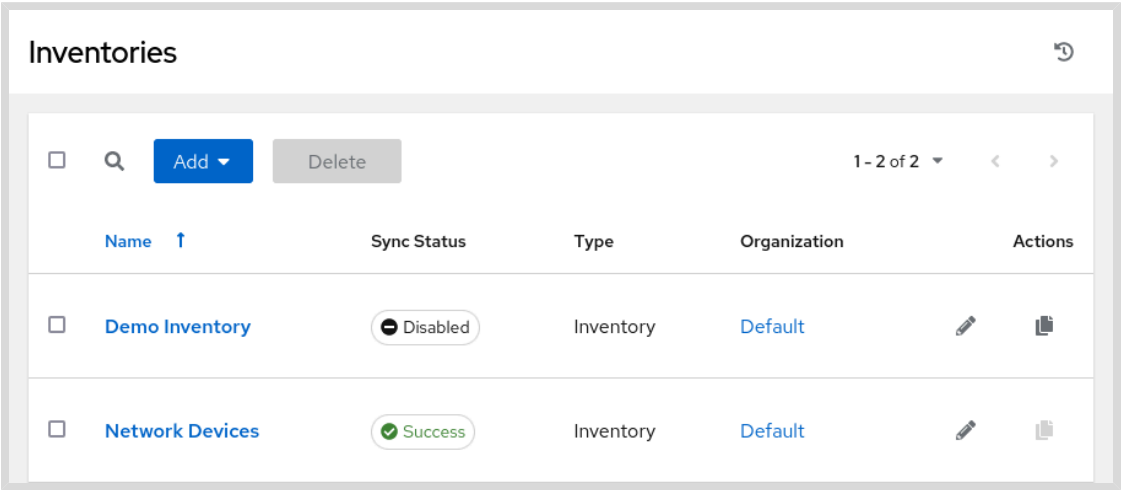


Figure 3.20: List of existing inventories

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.

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 choose **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**.

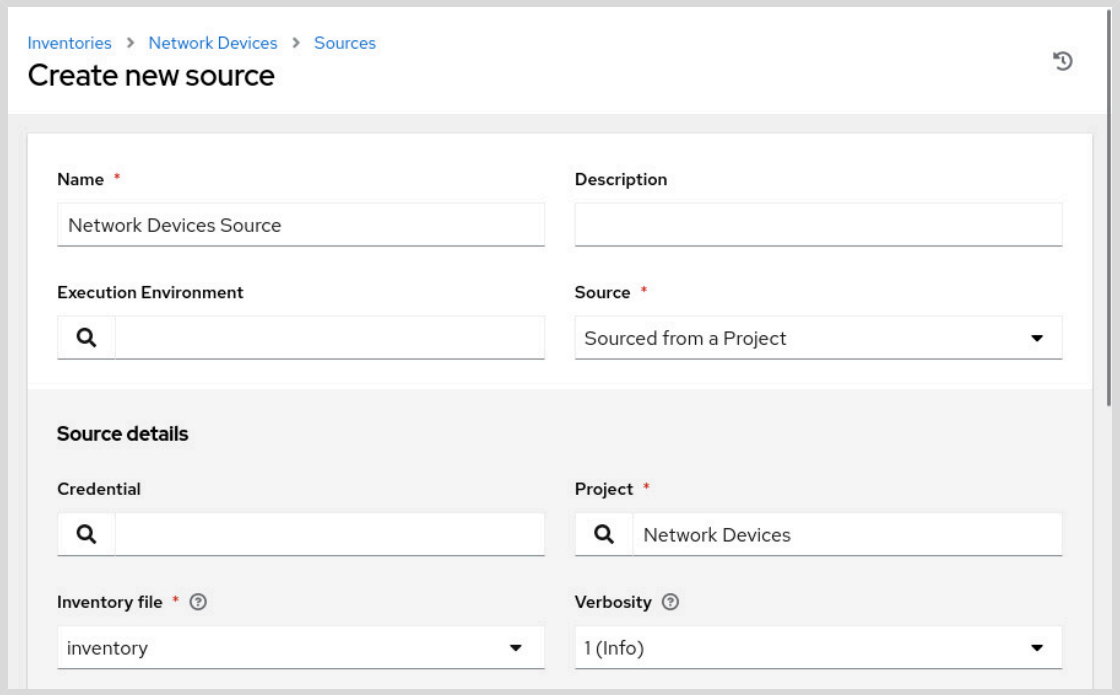


Figure 3.21: Adding a new inventory source from a project

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 specifies the information that you need to run a playbook. A job template includes an inventory and might specify variables to override, or tags to either use or skip.

The **Templates** page lists existing templates. Expand template information by clicking the > icon to the left of any template name. Initiate a job run by clicking the **Launch Template** icon.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

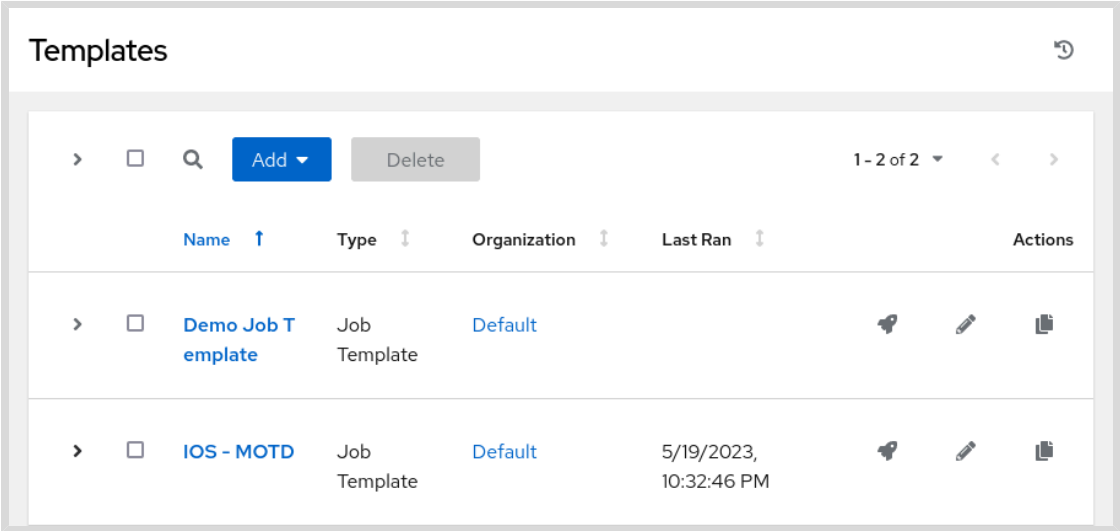


Figure 3.22: List of existing job templates

To create a job template, navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. You have to 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.

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.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

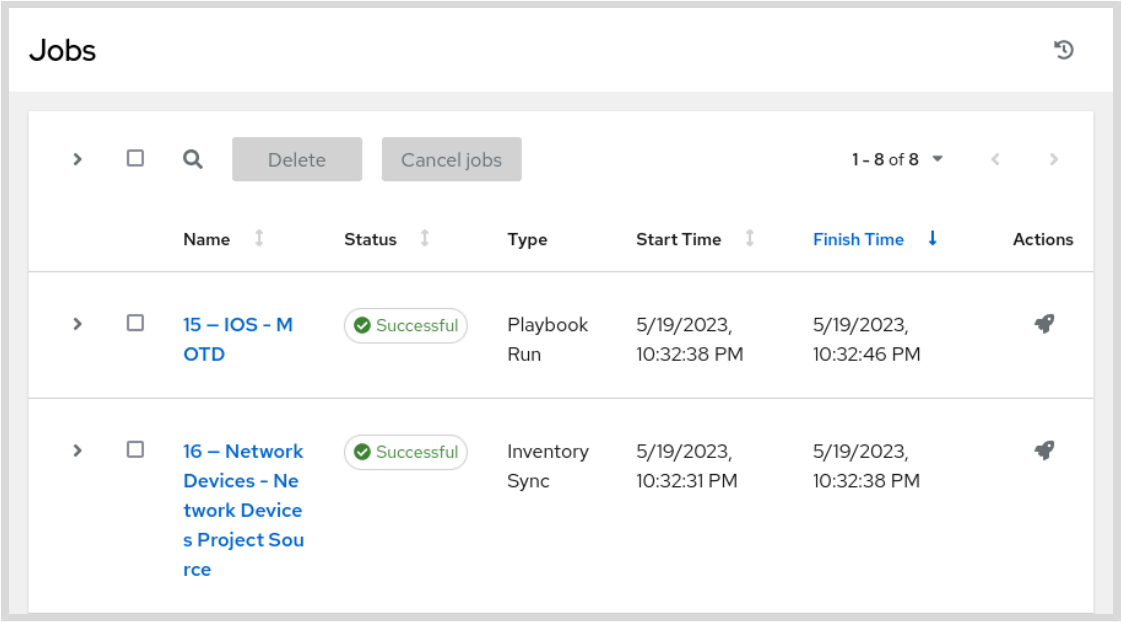


Figure 3.23: List of jobs

References

Automation Controller User Guide

https://docs.ansible.com/automation-controller/4.3/html/userguide/index.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Running a Playbook in Automation Controller to Configure SNMP

Navigate the automation controller web UI and launch an automation job.

Outcomes

- Create source control and machine credentials.
- Create a project.
- Create an inventory that uses an inventory file from the project.
- Create a 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 environment for this exercise, and to ensure that all required resources are available. This command also removes any Simple Network Management Protocol (SNMP) configuration from the Cisco IOS XE managed nodes.

```
[student@workstation ~]$ lab start run-controller
```

Instructions

This exercise provides you with an Ansible project in a Git repository that contains a playbook and an inventory file.

1. Verify that SNMP is not configured on the `iosxe2.lab.example.com` managed node.

   1.1. Connect to the `iosxe2.lab.example.com` managed node as the `student` user:

```
[student@workstation ~]$ ssh student@iosxe2.lab.example.com


iosxe2.lab.example.com#
```

   1.2. Display the SNMP configuration for the managed node. The command output indicates that the SNMP agent is not enabled:

```
iosxe2.lab.example.com#show snmp
%SNMP agent not enabled
```

   1.3. Return to the `workstation` machine:

```
iosxe2.lab.example.com#exit
Connection to iosxe2.lab.example.com closed by remote host.
Connection to iosxe2.lab.example.com closed.
```

2. Automation controller must have a *source control credential* to access content from a Git repository. Create a source control credential from the automation controller web UI.

   2.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

   2.2. Click the main menu icon at the top left of the page. Navigate to **Resources** > **Credentials** and click **Add**. Configure the credential to use authentication information for the `git.lab.example.com` server. Create the credential with the following settings and then click **Save**:

Important

If a field displays a search icon, then you can click that icon to select from the available resources for that field. If you choose to type a value, rather than selecting a value, then you might see an error message displayed until automation controller can validate your entry. If you cannot type a value in a field, then you must click the search icon.

Field	Value
Name	`Git Projects`
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.

3. To run playbooks against each managed node, automation controller must have a *machine credential* that contains authentication information. Create a machine credential for the Cisco IOS XE managed nodes that contains authentication information for the devices in the lab environment. Navigate to **Resources** > **Credentials** and click **Add**. Create the credential with the following settings and then click **Save**:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Field	Value
Name	`Cisco IOS XE Auth`
Organization	`Default`
Credential Type	`Machine`
Username	`student`
SSH Private Key	Content of the `/home/student/.ssh/lab_rsa` file.
Privilege Escalation Method	`enable`

4. Create a project named `SNMP`. This project points to a Git repository that contains a playbook and an inventory file. Navigate to **Resources** > **Projects** and then click **Add**. Create the project with the following settings and then click **Save**:

Field	Value
Name	`SNMP`
Organization	`Default`
Source Control Type	`Git`
Source Control URL	`git@git.lab.example.com:student/snmp`
Source Control Branch/Tag/Commit	`exercise`
Source Control Credential	`Git Projects`
Options	`Delete`
	`Allow Branch Override`

Note
In this exercise, you configure the project to use a specific branch, but you also provide the ability to override this selection. You can select a different branch when you create a job template that uses this project.

After you click **Save**, automation controller immediately attempts to synchronize the project. The **Last Job Status** field updates to indicate that the project synchronization succeeded.

Important
If synchronization fails, ensure that the project specifies the correct source control URL and credentials and then attempt the project synchronization again.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

5. Create an inventory for this exercise named `Cisco IOS XE` 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.

5.1. Navigate to **Resources** > **Inventories** and then click **Add** > **Add inventory**.

5.2. Create an inventory with the following settings and then click **Save**:

Field	Value
Name	`Cisco IOS XE`
Organization	`Default`

5.3. From the **Sources** tab, click **Add**. Create an inventory source with the following settings and then click **Save**:

Field	Value
Name	`Cisco IOS XE Source`
Source	`Sourced from a Project`
Project	`SNMP`
Inventory file	`inventory`

5.4. Click **Sync** to synchronize the inventory source. The **Last Job Status** field updates to indicate that the synchronization succeeded.

5.5. Click the **Back to Sources** tab and then click the **Hosts** tab. The inventory source imported the `iosxe1.lab.example.com` and `iosxe2.lab.example.com` hosts.

5.6. Click the **Groups** tab. The inventory source imported the `ios` group.

5.7. Click the **ios** group. Notice how the inventory variables for the `ios` group were also imported. Click the **Hosts** tab. The `ios` group contains the `iosxe1.lab.example.com` and `iosxe2.lab.example.com` hosts.

6. Create a job template to run the `cisco_snmp.yml` playbook. The playbook ensures that SNMP is configured on network devices in the `ios` inventory host group.

6.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. Create the job template with the following settings, and then click **Save**:

> **Important**
> You must click the search icon to select credentials for the job template.

Field	Value
Name	`Cisco SNMP`
Inventory	`Cisco IOS XE`
Project	`SNMP`
Playbook	`cisco_snmp.yml`
Credentials	`Cisco IOS XE Auth`

6.2. Click **Launch** to run the job template. The resulting job succeeds.

6.3. Navigate to **Views** > **Jobs** and look for a numbered job with `Cisco SNMP` in the name. By default, automation controller lists the most recent jobs at the top. The `Cisco SNMP` job displays a status of **Successful**.

6.4. *(Optional)* Click the name of the `Cisco SNMP` job to review the job output.

7. Verify that the job successfully deployed SNMP.

7.1. Connect to the `iosxe2.lab.example.com` managed node as the `student` user:

```
[student@workstation ~]$ ssh student@iosxe2.lab.example.com


iosxe2.lab.example.com#
```

7.2. Display the SNMP configuration for the managed node:

```
iosxe2.lab.example.com#show snmp
...output omitted...
SNMP informs: enabled
    Informs in flight 0/25 (current/max)
    Logging to 172.25.1.10.162
        0 sent, 0 in-flight, 0 retries, 0 failed, 0 dropped
```

7.3. Return to the `workstation` machine:

```
iosxe2.lab.example.com#exit
Connection to iosxe2.lab.example.com closed by remote host.
Connection to iosxe2.lab.example.com closed.
```

Finish

On the `workstation` machine, 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 run-controller
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Troubleshooting Playbooks

Objectives

- Identify issues in Ansible Playbooks and repair them.

Debugging Playbooks

The output provided by the `ansible-navigator run` command is a good starting point for troubleshooting issues with your plays.

Consider the following output from a playbook run:

```
PLAY [Configure the Junos managed nodes] ****************************************

TASK: [Enable the netconf service on port 830] ***********************************
changed: [junos1]
ok: [junos2]

TASK [Configure DNS settings] ****************************************************
changed: [junos1]
changed: [junos2]

PLAY RECAP ***********************************************************************
junos1 : ok=2  changed=2  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
junos2 : ok=2  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

The previous output shows a PLAY header with the name of the play being run, followed by one or more TASK headers for the tasks in that play. Each of the TASK headers represents the associated task in the play, and it is run against all the managed nodes specified by the `hosts` parameter in the play.

As the task runs against each managed node, the name of the managed node is displayed under the TASK header, along with the task result for that node. Task results can be `ok`, `fatal`, `changed`, or `skipping`.

Task Results

Result	Description
`ok`	There is no need to made any changes on the managed node. It is already in the desired state.
`fatal`	There was a problem implementing a change on the managed node.
`changed`	The task implemented a change on the managed node. It is now in the desired state.
`skipping`	The task was not run on the managed node. It might or might not be in the desired state.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

At the bottom of the output for each play, the `PLAY RECAP` section displays the number of tasks run for each managed node, by task result.

You can increase the verbosity of the output from `ansible-navigator run` by adding one or more `-v` options. The `ansible-navigator run -v` command provides additional debugging information, with up to four levels of verbosity.

Configuring the Output Verbosity of Playbook Execution

Option	Description
`-v`	Displays task results.
`-vv`	Displays task results and task configuration.
`-vvv`	Displays extra information about connections to managed nodes.
`-vvvv`	Adds extra verbosity options to the connection plug-ins, including users who executed the scripts on the managed nodes, and what scripts have been executed.

Debugging by Displaying Variable Values

The `ansible.builtin.debug` module can provide you insight into what is happening in the play. You can create a task that uses this module to display the value for a given variable at a specific point in the play. This can help you to debug tasks that use variables to communicate with each other (for example, using the output of a task as the input to the following one).

The following example uses the `msg` and `var` parameters of the `ansible.builtin.debug` module, after gathering the interface facts for the IOS manged nodes:

```
- name: Gather IOS interface facts
  cisco.ios.facts:
    gather_subset: interfaces

- name: Display the operating system version running on the managed node
  ansible.builtin.debug:
    msg: "Running the {{ ansible_net_version }} OS version"

- name: Display the IPv4 addresses configured on the managed node
  ansible.builtin.debug:
    var: ansible_net_all_ipv4_addresses
```

The first task using the `ansible.builtin.debug` module displays the value at run time of the `ansible_net_version` variable as part of a message printed to the output of the `ansible-navigator run` command.

The second task using the `ansible.builtin.debug` module displays the value of the `ansible_net_all_ipv4_addresses` variable.

You can use the `verbosity` parameter to specify whether the `ansible.builtin.debug` module task is executed. The value for the `verbosity` parameter correlates to the number of `-v` options that are specified when you run the playbook. The default value of the `verbosity` parameter is 0.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

In the following example, the verbosity is set to 2 for a task using the `ansible.builtin.debug` module. The task is executed only if you run the playbook with the `-vv` option in the `ansible-navigator run` command.

```
- name: Display the network device serial number
  ansible.builtin.debug:
    var: ansible_net_serialnum
    verbosity: 2
```

Working with Command Modules

You can have a playbook that includes command modules to send commands to your managed nodes. A playbook task might fail to run because a command does not yet produce the required output on the managed node, but if you wait a certain amount of time, the condition is eventually met.

For example, a network interface is being connected. You execute the playbook and it fails. It fails because the network interface is not connected yet.

Conditionals and Command Module Arguments

Use conditionals to evaluate the result from a command that is executed on the managed node and then wait for a certain result before continuing.

The following table illustrates the unique conditionals you can use to evaluate the command result in a managed node:

Networking Module Conditionals

Option	Result from executing a command on the managed node
`eq`	Result is equal to what follows the conditional.
`neq`	Result is not equal to what follows the conditional.
`gt`	Result has a value greater than the value that follows the conditional.
`ge`	Result has a value greater than or equal to the value that follows the conditional.
`lt`	Result has a value less than the value that follows the conditional.
`le`	Result has a value less than or equal to the value that follows the conditional.
`contains`	Result contains the string that follows the conditional.

The `wait_for` command argument can be used to specify one or more conditions to evaluate against the output of the command. The task does not continue to execute until all conditions are true. If the conditions are not true within the configured number of retries, then the task fails.

You can control how many retries to do, and how much time to leave between one and the other. By default, the number of retries is set to 9. Use the `retries` command argument to modify this value. By default, the interval in seconds between retries is 1. Modify this interval using the `interval` command argument.

The following example shows a task that runs the `show interfaces` command on the managed nodes, and then waits for the result of that task to contain the text `GigabitEthernet1 is up`. It retries the task 19 times, waiting two seconds between each retry.

```
- name: Wait for the second network interface
  cisco.ios.ios_command:
    commands:
      - show interfaces
    wait_for:
      - result[0] contains 'GigabitEthernet1 is up'
    interval: 2
    retries: 19
```

Including Prompts in the Playbook

When working directly with network devices, sometimes there is a prompt confirmation message that must be answered to perform some actions or changes on the devices. By using Ansible, you can handle these kinds of prompt confirmations using the `ansible.netcommon.cli_command` module with the `prompt` parameter.

> **Note**
> The `cisco.ios.ios_command` and `cisco.nxos.nxos_command` modules include the `prompt` parameter to provide answers to interactive prompts when commands require it.
>
> You can also use the `ansible.netcommon.cli_command` module to handle prompts. For more information on the `ansible.netcommon.cli_command` module, visit https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/cli_command_module.html.

The `prompt` parameter is accompanied by the `answer` parameter. The `prompt` parameter specifies the prompt message that is received on the network device, or a list of these prompt messages when there are more than one. Likewise, the `answer` parameter specifies the answers to the prompt.

The value of the `prompt` parameter must not only match the network device prompt request, but also follow the rules for Python regular expressions, as indicated in https://docs.ansible.com/ansible/latest/network/user_guide/network_working_with_command_output.html#handling-prompts-in-network-modules.

In the following example, the task uses the `cisco.ios.ios_command` to send the `reload` command to the managed nodes. Then, the prompt message from the network device and the corresponding answer are specified.

```
- name: Reboot the managed nodes
  cisco.ios.ios_command:
    commands:
      - command: 'reload'
        prompt: 'Proceed with reload\?'
        answer: 'y'
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Reviewing Playbooks for Errors

Several issues can occur during a playbook run, many related to the syntax of the playbook, or due to connectivity issues with the managed nodes (for example, an error in the hostname of the managed node in the inventory file).

A number of tools are available that you can use to review your playbook for syntax errors and other problems before you run it.

Following Good Practices

One of the best ways to simplify debugging playbooks is to follow good practices when writing them in the first place. The following list describes some recommended practices for playbook development:

- Use a concise description of the purpose of the play or task when naming plays and tasks. The play name or task name is displayed when the playbook is executed. This also helps document what each play or task is supposed to accomplish, and possibly why it is needed.
- Use comments to add inline documentation about tasks.
- Make effective use of vertical white space. In general, organize task attributes vertically to make them easier to read.
- Consistent horizontal indentation is critical. Use spaces, not tabs, to avoid indentation errors. You can set up your text editor to insert spaces when you press the `Tab` key to make this easier.
- Try to keep the playbook as simple as possible. Only use the features that you need.

> **Note**
> Some Ansible practitioners at Red Hat have been working on an *unofficial* set of recommended practices for creating Ansible automation content, based on their own experiences in the field. See https://redhat-cop.github.io/automation-good-practices.
>
> Although not officially endorsed by Red Hat at this time, it can be a useful starting point for developing good practices of your own.

Making Use of Ansible Lint

To help you follow good practices, Red Hat Ansible Automation Platform 2.3 provides a tool called Ansible Lint, which uses a set of predefined rules to look for possible issues with your playbook.

Use the `ansible-navigator lint` command to review a file for common errors and issues:

```
[user@host ~]$ ansible-navigator lint playbook.yml
```

Not all the issues that Ansible Lint reports break your playbook, but a reported issue might indicate the presence of a more serious error. The output provided for the `ansible-navigator lint` command shows the issue found, the line where the issue was found, and the rule that allowed Ansible Lint to find the issue.

For example, assume that you have the following `ios_information.yml` playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Gather information from IOS devices
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather information from IOS devices
      ios_command:
        commands:
        - show version
        - show interfaces
```

Use the `ansible-navigator lint` command to validate it:

```
[user@host ~]$ ansible-navigator lint ios_information.yml -m stdout
fqcn[action]: Use FQCN for module actions, such
 `<namespace>.<collection>.ios_command`. (warning) ❶
ios_information_lint.yml:7 Action `ios_command` is not FQCN.

yaml[indentation]: Wrong indentation: expected 10 but found 8 ❷
ios_information_lint.yml:10

yaml[trailing-spaces]: Trailing spaces ❸
ios_information_lint.yml:11

yaml[empty-lines]: Too many blank lines (1 > 0) ❹
ios_information_lint.yml:12
```

❶ Line 7 of the playbook, (`ios_command:`). The issue was detected by the `fqcn[action]` rule. Ansible Lint indicates the use of the fully qualified collection name (FQCN) for the module name on that task. It should be `cisco.ios.ios_command` instead of just `ios_command`.

❷ Line 10 of the playbook, (`- show version`). The issue was detected by the `yaml[indentation]` rule. Ansible Lint indicates incorrect indentation. It expected 10 spaces but found 8. You need to increase the indentation by two spaces.

❸ Line 11 of the playbook, (`- show interfaces`). The issue was detected by the `yaml[trailing-spaces]` rule. Ansible Lint indicates an issue related to trailing spaces. Although it is not visible, there are blank spaces at the end of the line. It is not a problem with the playbook directly, but many developers prefer not to have trailing white space in files stored in version control to avoid unnecessary differences as they edit the files.

❹ The playbook ends with one or more blank lines, detected by the `yaml[empty-lines]` rule. Again, this is not a problem with the playbook directly, but it is good practice to remove this last blank line.

Playbook Artifacts and Log Files

Red Hat Ansible Automation Platform can log the output of playbook runs that you make from the command line in a number of different ways.

- Automation content navigator can produce *playbook artifacts* that store information about runs of playbooks in JSON format.
- You can log information about playbook runs to a text file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Playbook Artifacts from Automation Content Navigator

The `ansible-navigator` command produces playbook artifact files by default each time you run a playbook. These files record information about the playbook run, and can be used to review the results of the run when it completes, to troubleshoot issues, or be kept for compliance purposes. Each playbook artifact file is named based on the name of the playbook you ran, followed by `artifact`, and then the time stamp of when the playbook was run, ending with the `.json` file extension.

For example, if you run the command `ansible-navigator run networking.yml` at 20:00 UTC on May 31, 2023, the resulting file name of the artifact file would be:

```
networking-artifact-2023-05-31T20:00:04.019343+00:00.json
```

You can review the contents of these files with the `ansible-navigator replay` command. If you include the `-m stdout` option, then the output of the playbook run is printed to your terminal as if it had just run. If you omit that option, you can examine the results of the run interactively.

Consider the following playbook where you run the `show interfaces` command on Junos managed nodes. The playbook uses the `ansible.netcommon.cli_command` module to run the command:

```
---
- name: Gather interfaces information for Junos devices
  hosts: junos
  gather_facts: false
  tasks:
    - name: Run command on remote managed nodes
      ansible.netcommon.cli_command:
        command: show interfaces
```

The `junos` managed nodes were defined in the inventory file for the project:

```
[junos]
junos[1:2].lab.example.com

[junos:vars]
ansible_user=developer
ansible_ssh_private_key_file=~/.ssh/lab_rsa
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

If you run the playbook then it fails. Using Ansible Lint, you do not find any issues in the predefined rules.

To troubleshoot further, you run `ansible-navigator replay` in interactive mode on the resulting artifact file, which opens the following output in your terminal:

```
  Play name          Ok Changed Unreachable Failed Skipped Ignored In progress Task count  Progress
0 Gather interfaces   0       0           0      2       0       0           0          2  Complete

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

Figure 3.24: Initial replay screen

You can press 0 to display details of the play. The output shows the failed results in each of the managed nodes. In this case, to simplify and show the problem, the play only has one task. If there is more than one task, they are listed in this output.

```
  Result Host                 Number Changed Task                 Task action           Duration
0 Failed junos1.lab.example.co     0 False   Run command on remote ansible.netcommon.cli_      1s
1 Failed junos2.lab.example.co     1 False   Run command on remote ansible.netcommon.cli_      1s

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

Figure 3.25: Play results by managed node and task

It looks like the `Run command on remote managed nodes` task failed on both managed nodes. By entering 0 or 1 you can see the failure for the first or the second managed node, respectively.

The following terminal output shows the failed result for the first managed node. You find a similar result for the other node.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
Play name: Gather interfaces information for Junos devices:0
Task name: Run command on remote managed nodes
Failed: junos1.lab.example.com Connection type ansible.netcommon.netconf is not valid for this modu
 0 ---
 1 duration: 0.75042
 2 end: '2023-05-31T17:21:06.916214'
 3 event_loop: null
 4 host: junos1.lab.example.com
 5 ignore_errors: null
 6 play: Gather interfaces information for Junos devices
 7 play_pattern: junos
 8 playbook: /home/developer/junos_error.yml
 9 remote_addr: junos1.lab.example.com
10 res:
11   _ansible_no_log: null
12   changed: false
13   msg: Connection type ansible.netcommon.netconf is not valid for this module
14 resolved_action: ansible.netcommon.cli_command
15 start: '2023-05-31T17:21:06.165794'
16 task: Run command on remote managed nodes
17 task_action: ansible.netcommon.cli_command
18 task_args: ''
19 task_path: /home/developer/junos_error.yml:7

^b/PgUp page up^f/PgDn page down↑↓ scrollesc back- previous+ next[0-9] goto:help help   Failed
```

Figure 3.26: Task results for one of the managed nodes

The task is attempting to use the `ansible.netcommon.cli_command` module to execute the `show interfaces` command on the managed node, but the connection type used by this playbook is not valid for that module.

You might discover that the `ansible.netcommon.netconf` connection type was configured as the default for the Junos managed nodes in the inventory file:

```
[junos:vars]
ansible_user=developer
ansible_ssh_private_key_file=~/.ssh/lab_rsa
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

Using the same play but with the `ansible.netcommon.network_cli` connection type, the result is successful on both managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
Play name: Gather interfaces information for Junos devices:1
Task name: Run command on remote managed nodes
Ok: junos2.lab.example.com
  0 ---
  1 duration: 2.301246
  2 end: '2023-05-31T18:24:42.747808'
  3 event_loop: null
  4 host: junos2.lab.example.com
  5 play: Gather interfaces information for Junos devices
  6 play_pattern: junos
  7 playbook: /home/developer/junos_error.yml
  8 remote_addr: junos2.lab.example.com
  9 res:
 10   _ansible_no_log: null
 11   changed: false
 12   invocation:
 13     module_args:
 14       answer: null
 15       check_all: false
 16       command: show interfaces
 17       newline: true
 18       prompt: null
 19       sendonly: false
 20   stdout: |-
 21     Physical interface: cbp0, Enabled, Physical link is Up
 22       Interface index: 130, SNMP ifIndex: 501
 23       Type: Ethernet, Link-level type: Ethernet, MTU: 9192
 24       Device flags   : Present Running
^b/PgUp page up^f/PgDn page down↑↓ scrollesc back- previous+ next[0-9] goto:help help  Successful
```

Figure 3.27: Successful task results in the managed nodes

Important

You might not want to save playbook artifacts for several reasons:

- You are concerned about sensitive information being saved in the log file.
- You need to provide interactive input to automation content navigator.
- You do not want the files to clutter up the project directory.

You can keep the files from being generated by disabling the playbook artifacts in the `ansible-navigator.yml` configuration file inside the project directory:

```
ansible-navigator:
  playbook-artifact:
    enable: false
```

Logging Output to a Text File

Use the `logging` key in the `ansible-navigator.yml` configuration file to set the parameters related to logs that are controlled by automation content navigator.

The following parameters apply to the `logging` key:

`append`
: The default is `true` and means that the log messages are appended to an existing log file. If you set this value to `false` then every new log message creates a new log file.

`file`
: Specifies the full path to the log file.

`level`
: Choose between `debug`, `info`, `warning`, `error`, and `critical` levels to show in the log file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

> **Important**
> If you configure automation content navigator to write log files to `/var/log`, then Red Hat recommends that you configure `logrotate` to manage them.

The following example shows an automation content navigator configuration where you specify a `debug` log level for troubleshooting:

```
ansible-navigator:
  logging:
    append: true
    file: /tmp/ansible-navigator.log
    level: debug
```

Debugging in VS Code

When using VS Code to create or edit playbooks, you can see syntax errors as you write, find information for the modules or collections in your plays, or detect if the playbook breaks a predefined Ansible Lint rule.

You first need to install the Ansible and YAML VS Code extensions in VS Code. Adjust the VS Code extension settings for Ansible to use your `ee-supported-rhel8` automation execution environment. This step is required because Ansible Lint is an operating system package included in it.

To adjust the Ansible extension settings, navigate to **View** > **Extensions** in VS Code. Select the **Ansible** extension, click the **Manage** icon for the extension, and then click **Extension Settings**.

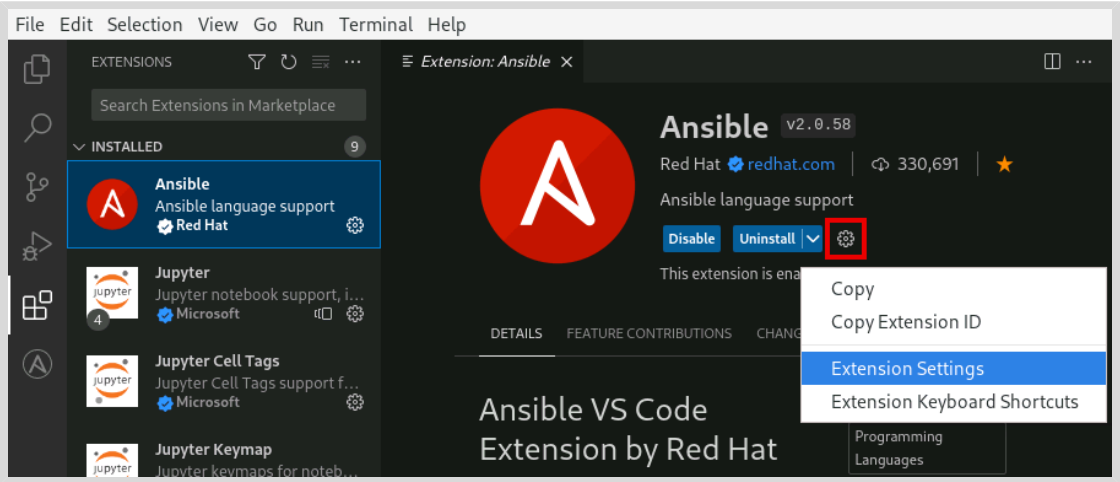


Figure 3.28: Adjusting the VS Code extension settings for Ansible

Scroll down to the **Ansible>Execution Environment:Enabled** settings option and enable the use of an automation execution environment. Specify the automation execution environment to use in the **Ansible>Execution Environment:Image** settings option. Adjust other automation execution environment related settings for your preference.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

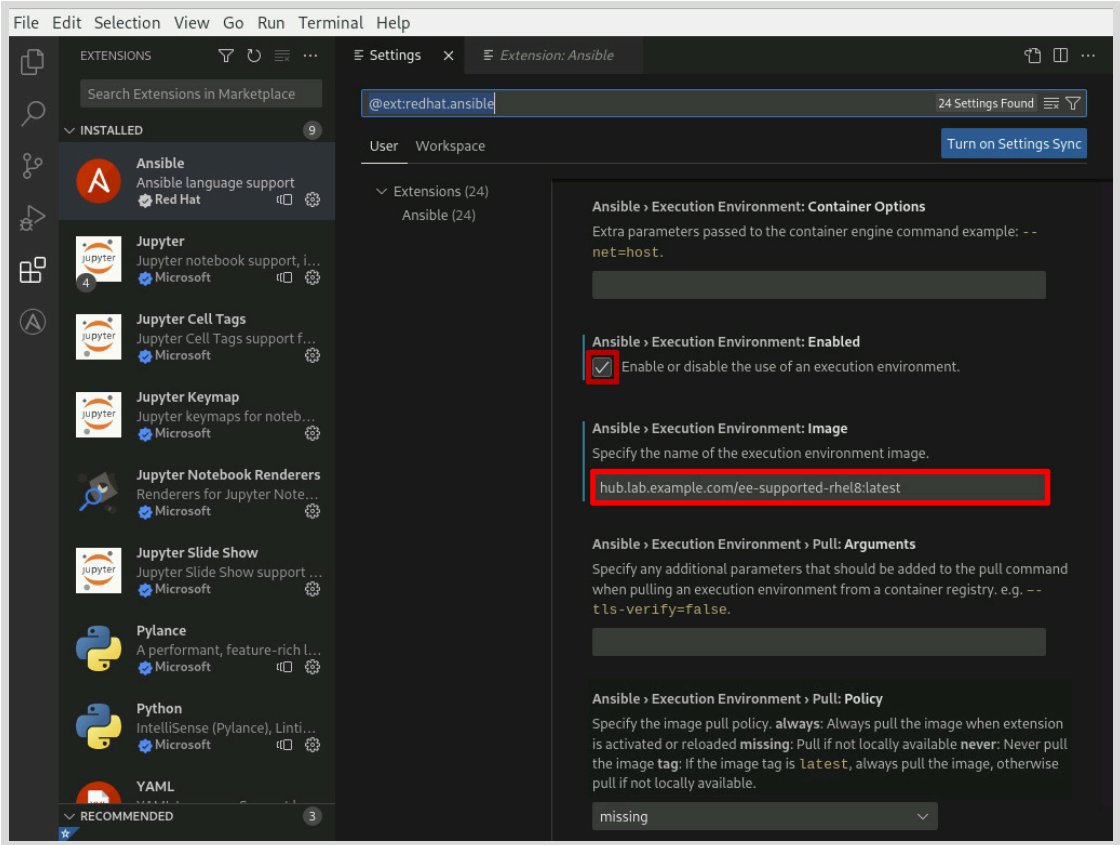


Figure 3.29: Automation execution environment settings section

Close the **Settings** tab and any other tabs that you opened.

Consider the following playbook in VS Code:

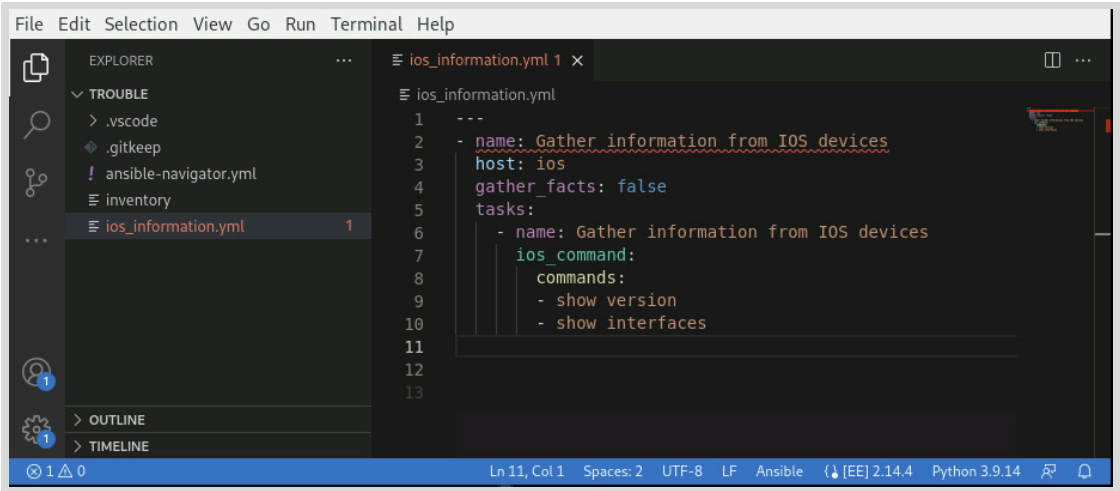


Figure 3.30: Playbook with issues opened in VS Code

The playbook is analyzed in levels: first the plays, then the attributes, and so on. The scan stops at the first problem found. Notice that for this example there are several indications regarding one issue in the playbook.

- The second line of the playbook is underlined in red, indicating an issue. If you hover over the line, a message indicates that the `host` attribute is not valid for the play.

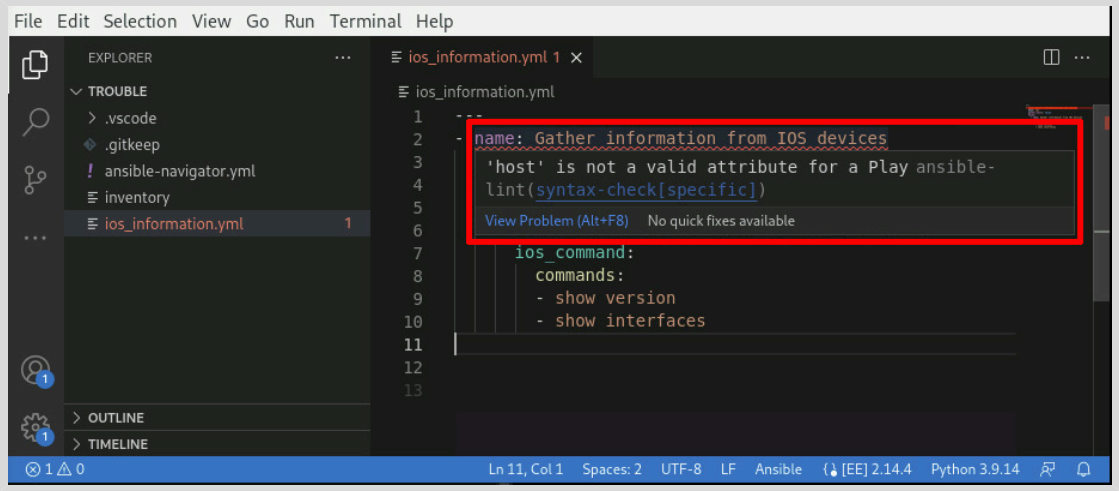


Figure 3.31: Ansible Lint checker in VS Code

- The color for the `host` attribute is different from the color for the `name`, `gather_facts`, and `tasks` attributes. In the example, you are using VS Code to verify an existing playbook, but when writing a new file, the color can be an indicator to avoid you making a mistake.

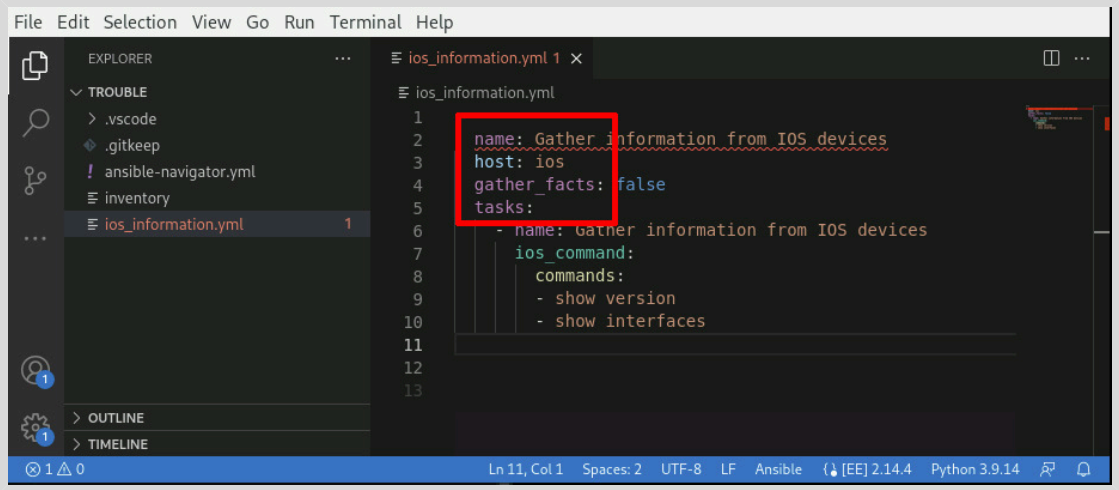


Figure 3.32: Issue related to the host attribute

- The name of the playbook is in red. If you hover over the number at the end of the playbook name line, a message indicates that there is one problem in this file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

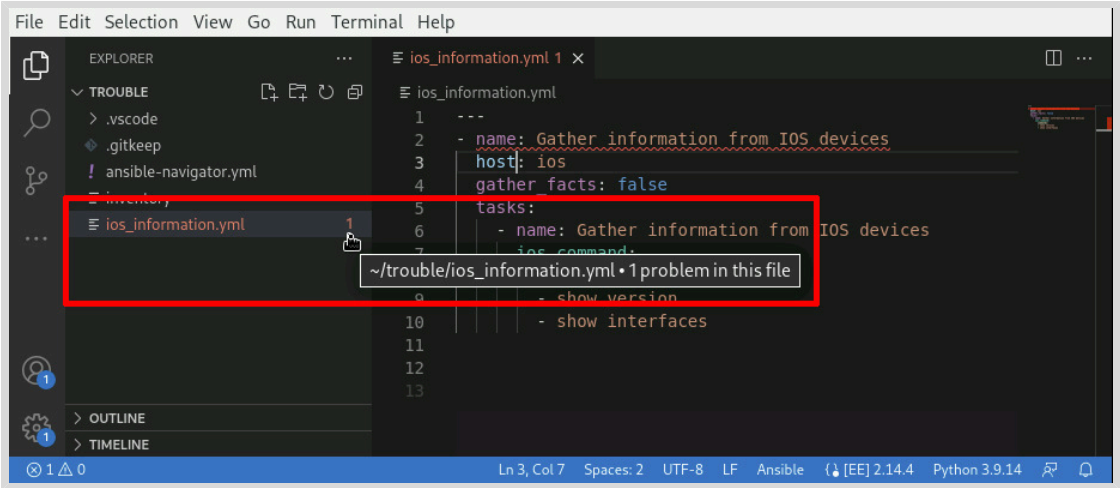


Figure 3.33: Showing the number of problems in the file

- The Status Bar in VS Code displays a summary with the result of these checks. The first symbol in this bar shows the number of problems found in the playbook.

As you can see, in a single view you can detect that there is at least one problem with the playbook. VS Code also gives you a direct view into the problems found.

From VS Code, select **View** > **Problems** to see the issues detected.

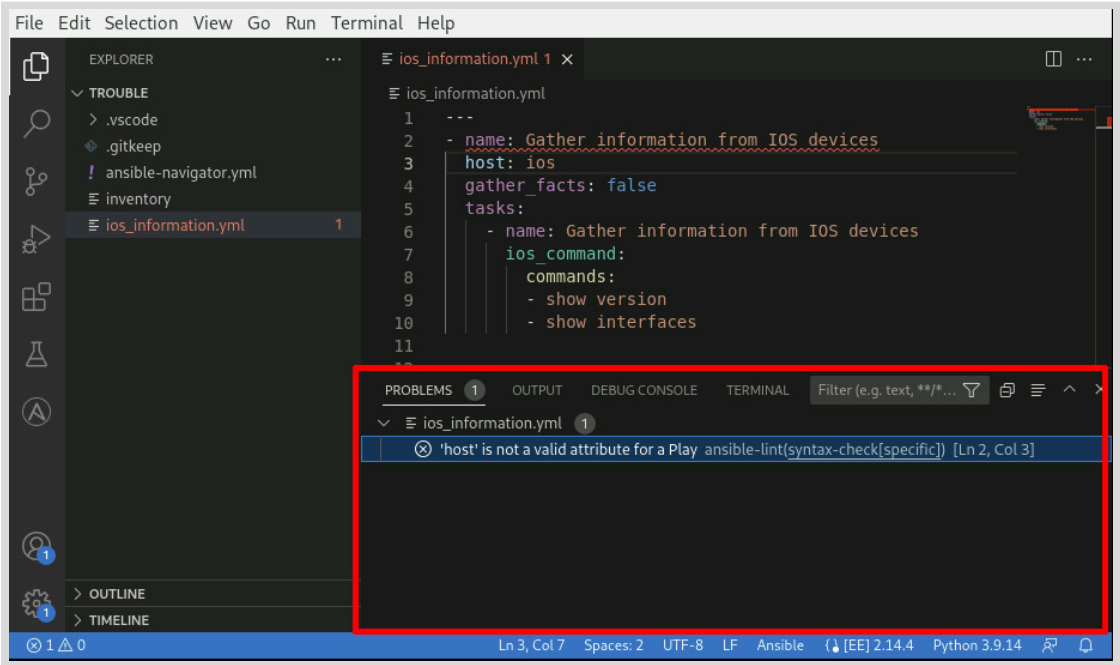


Figure 3.34: Section in VS Code to show the problems found

If you fix the `hosts` attribute in the example playbook, the first thing that happens is that the attribute acquires the same color as the rest of the play attributes. After saving the file, the Status Bar shows the starting of a new Ansible Lint verification.

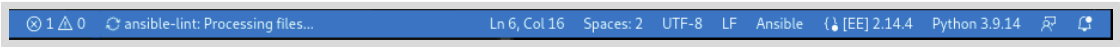


Figure 3.35: Status Bar in VS Code indicates that Ansible Lint is verifying the file

In this example, new issues are detected in the playbook after you fix the `hosts` attribute for the play.

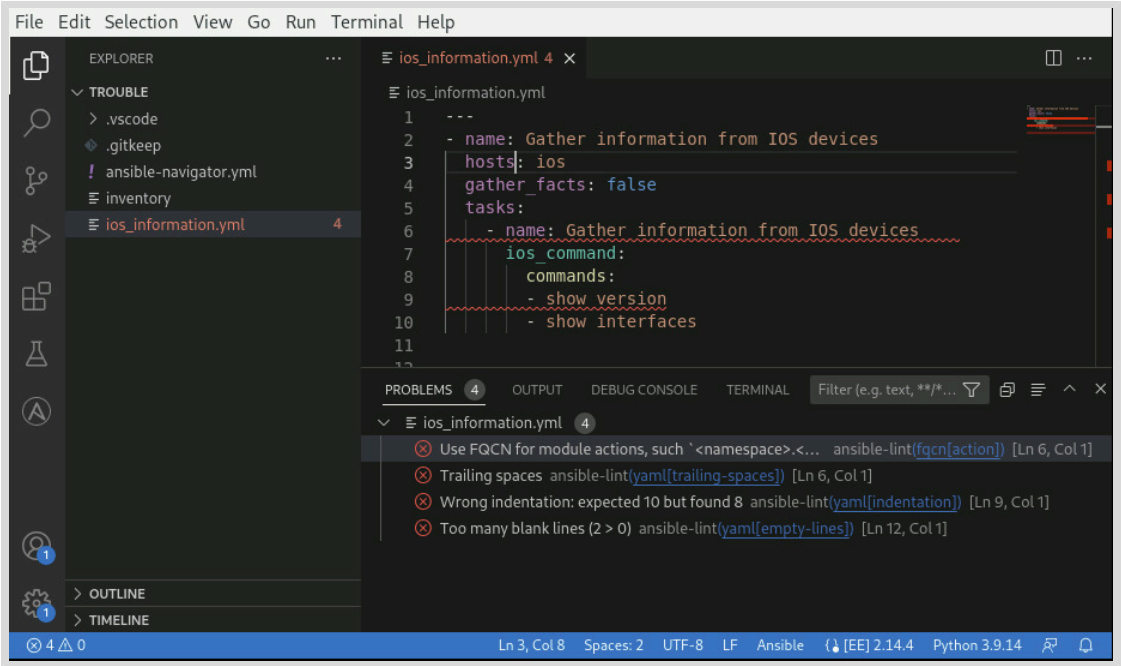


Figure 3.36: New detected issues in the playbook

Ansible Lint in VS Code works similarly as on the command line. You review each one of the detected problems, and fix them. An advantage is that you have access to more information in a simple way.

For example, the first issue now detected corresponds to not using the FQCN when referring to the `ios_command` module in the play. You can hover over the module to obtain the FQCN, its description, and additional notes.

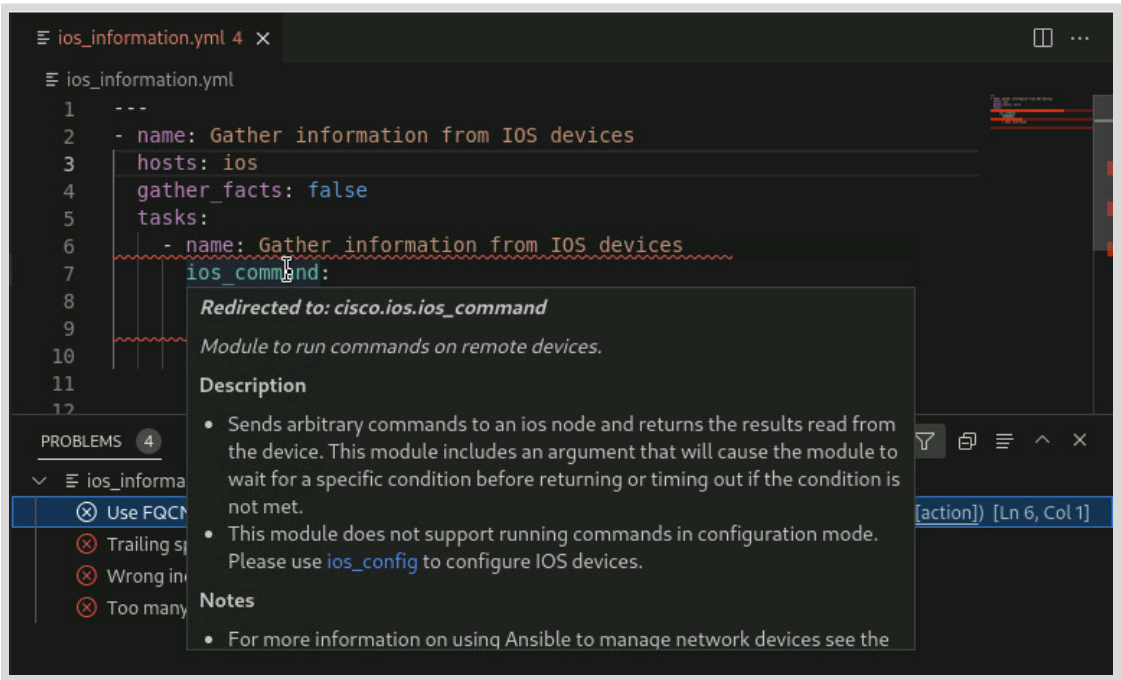


Figure 3.37: Information about the `ios_command` module.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Another advantage is the autocomplete feature, which provides you with suggestions that avoid making mistakes when typing. All these suggestions are based on the collections inside the automation execution environment image configured in VS Code.

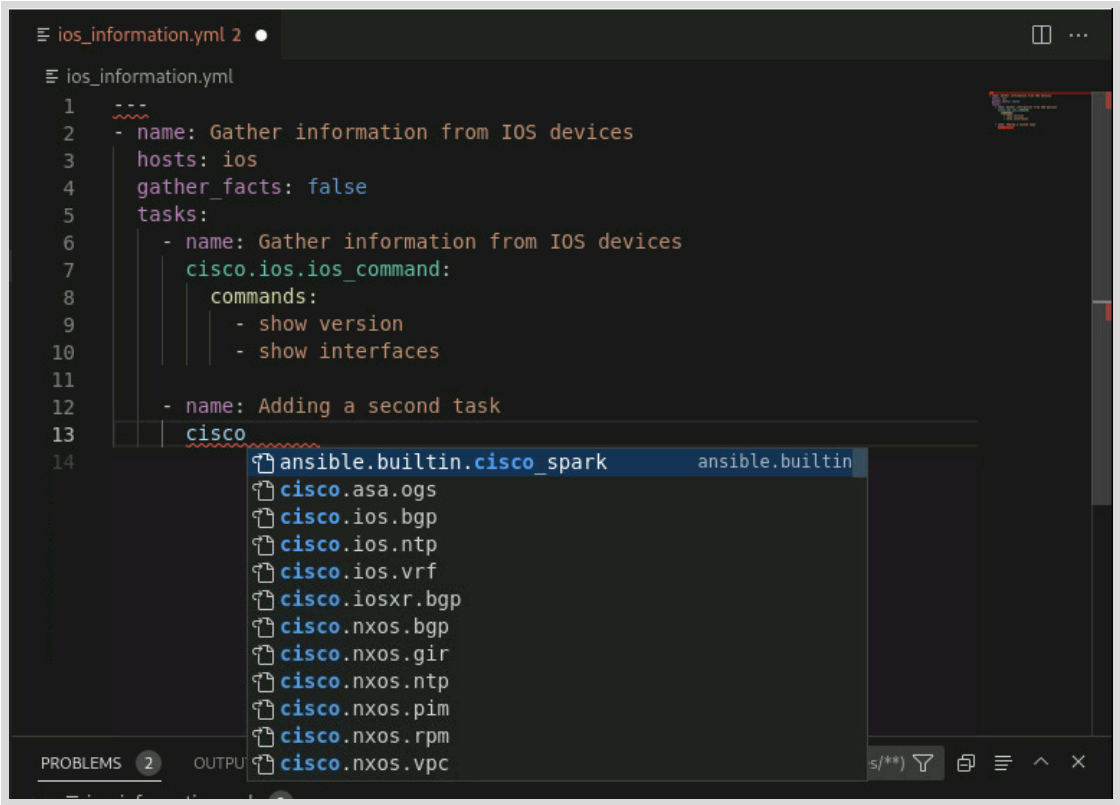


Figure 3.38: The autocomplete VS Code feature provides suggestions to use in the playbook.

References

ansible.builtin.debug module - Print Statements During Execution - Ansible Documentation
https://docs.ansible.com/ansible/latest/modules/debug_module.html

Working with Command Output and Prompts in Network Modules - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/network_working_with_command_output.html

Ansible Tips and Tricks - Ansible Documentation
https://docs.ansible.com/ansible/latest/tips_tricks/index.html

Good Practices for Ansible
https://redhat-cop.github.io/automation-good-practices/

Ansible Lint Documentation
https://ansible-lint.readthedocs.io/

Deep Dive on Ansible VScode Extension
https://www.ansible.com/blog/deep-dive-on-ansible-vscode-extension

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Troubleshooting Playbooks

Troubleshoot a playbook that has been given to you that does not work properly.

Outcomes

- Troubleshoot and resolve issues in playbooks.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start run-trouble
```

Instructions

You have been provided with a playbook called `vlan_data.yml`. The playbook collects the VLANs configured on the IOS managed nodes, and saves that information in local files. Each local file is named after the managed node, followed by the `_vlan` suffix, ending with the `yml` file extension.

The playbook is not working. Find out what the problem is and fix it.

1. To use Ansible Lint from Visual Studio Code (VS Code) on the `workstation` machine, enable the use of the `hub.lab.example.com/ee-supported-rhel8:latest` automation execution environment.

   1.1. Open VS Code. Click **View** > **Extensions** and then click in the **Ansible** extension.

   1.2. Click the **Manage** icon and then click **Extension Settings**.

   1.3. Scroll down to the **Ansible>Execution Environment:Enabled** settings option. Enable the use of an automation execution environment.

   1.4. Scroll down to the **Ansible>Execution Environment:Image** settings option. Specify `hub.lab.example.com/ee-supported-rhel8:latest` as the name of the image to use.

   1.5. Close the **Settings** and **Extension:Ansible** tabs.

2. Open the `/home/student/run-trouble` directory in VS Code and review the `vlan_data.yml` playbook. Set the Python environment by selecting the recommended interpreter path.

   2.1. Click **File** > **Open Folder**.

   2.2. Navigate to **Home** > **run-trouble** and click **Open**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

2.3. Click the `vlan_data.yml` playbook. The playbook runs against the `ios` inventory group of managed nodes, and has two tasks. The first task gathers the VLAN information. The second task saves the collected information in local files.

```
---
- name: Creating a file with VLANs data
  hosts: ios
  gather_facts: false
  task:
    - name: Gather IOS interface facts
      cisco.ios.ios_facts:
       gather_network_resources: vlans

    - name: Copying the vlan configuration to a file
      ansible.builtin.copy:
        content: "{{ vlans | to_nice_yaml }}"
        dest: "{{ inventory_hostname }}_vlan.yml"
        mode: 0600
```

2.4. Click **Select python environment** in the status bar, and then click the recommended option.

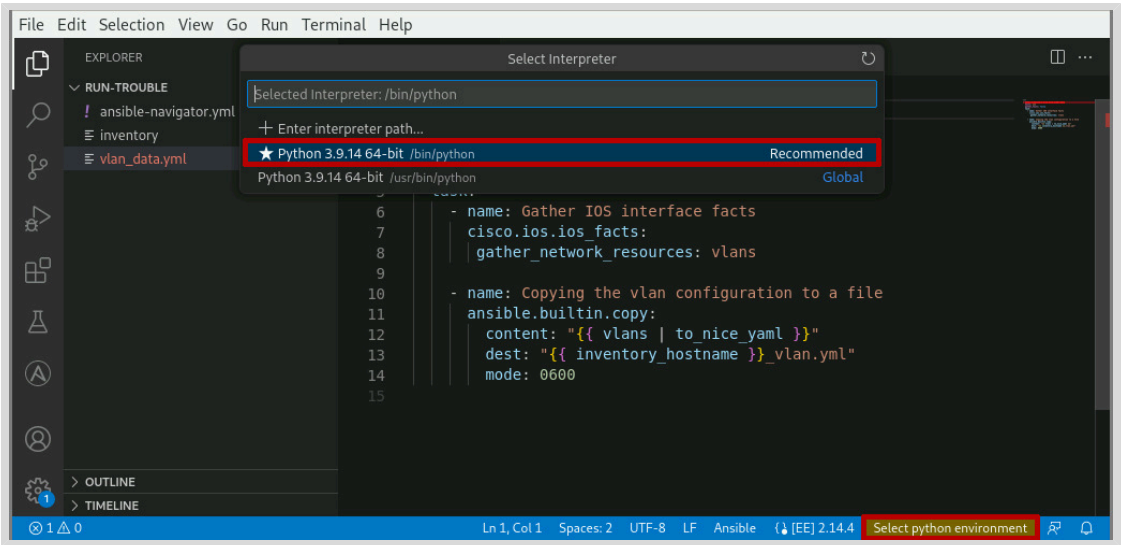


Figure 3.39: Selecting the Python environment

2.5. After setting the Python language, the status bar must display the following information:

- The playbook is recognized as an Ansible file.
- The automation execution environment is enabled and configured.
- The Python language is set.

Figure 3.40: VS Code status bar

3. Find and fix the problems in the `vlan_data.yml` playbook detected by Ansible Lint. Open the VS Code problems panel to follow the Ansible Lint issues detected.

3.1. Click **View** > **Problems** to open the problems panel in VS Code.

3.2. Ansible Lint detects an issue in the play definition. The broken rule is syntax related.

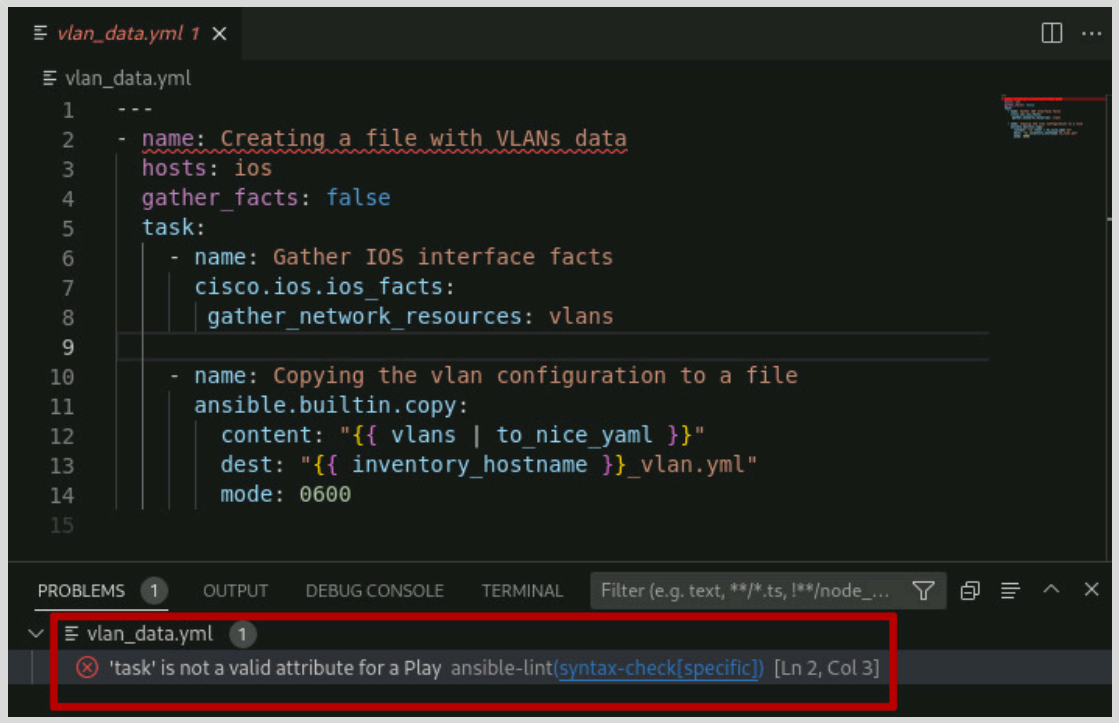


Figure 3.41: Ansible Lint detects a syntax issue

3.3. Fix the playbook by using `tasks` instead of `task` for the play attribute. Notice how the color that VS Code uses for the `tasks` attribute is now the same as that used for the other play attributes. If you hover over the `tasks` attribute, Ansible Lint displays information about what the `tasks` attribute is.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
vlan_data.yml 1
vlan_data.yml
1   ---
2   - name: Creating a file with VLANs data
3     Main list of tasks to execute in the play, they run after roles and before
4     post_tasks.
5     tasks:
6       - name: Gather IOS interface facts
7         cisco.ios.ios_facts:
8           gather_network_resources: vlans
9
10      - name: Copying the vlan configuration to a file
11        ansible.builtin.copy:
12          content: "{{ vlans | to_nice_yaml }}"
13          dest: "{{ inventory_hostname }}_vlan.yml"
14          mode: 0600
15
```

Figure 3.42: Ansible Lint provides information about the `tasks` attribute

3.4. Save the playbook. Ansible Lint detects a new issue in the eighth line of the file. The broken rule is related to the indentation in YAML format. Ansible Lint expects eight indentation spaces but it found seven.

```
vlan_data.yml 1
vlan_data.yml
1   ---
2   - name: Creating a file with VLANs data
3     hosts: ios
4     gather_facts: false
5     tasks:
6       - name: Gather IOS interface facts
7         cisco.ios.ios_facts:
8           gather_network_resources: vlans
9
10      - name: Copying the vlan configuration to a file
11        ansible.builtin.copy:
12          content: "{{ vlans | to_nice_yaml }}"
13          dest: "{{ inventory_hostname }}_vlan.yml"
14          mode: 0600
15
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   Filter (e.g. text, **/*.ts, !**/node_...
vlan_data.yml 1
Wrong indentation: expected 8 but found 7  ansible-lint(yaml[indentation])  [Ln 8, Col 1]
```

Figure 3.43: Ansible Lint detects an indentation issue

3.5. Fix the playbook by adding one more indentation space on the eighth line. Save the playbook. Ansible Lint no longer detects any issues.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

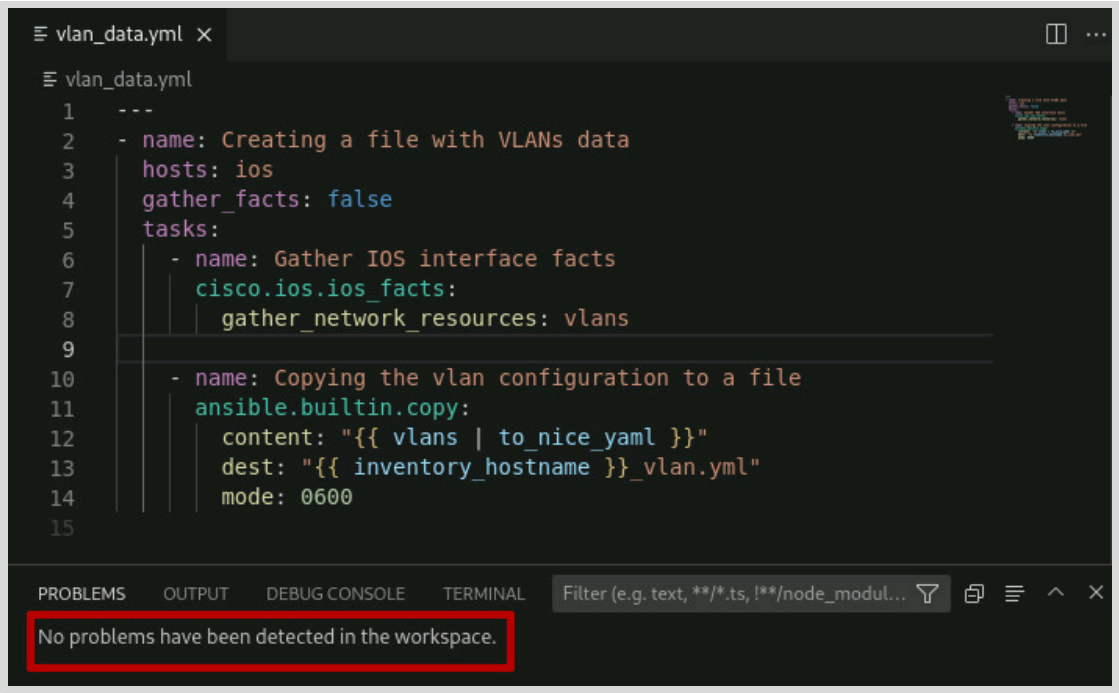


Figure 3.44: The playbook no longer breaks any Ansible Lint rules

4. Verify that the modified playbook works as expected. If it fails, use automation content navigator to generate a playbook artifact that enables you to perform a deeper analysis.

> **Note**
> In a production network, where the playbook has passed the Ansible Lint revisions but does not work as expected, you can share the playbook artifacts with others to assist with troubleshooting.

4.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/run-trouble` directory in a GNOME terminal. The **Terminal** tab in VS Code is in the same panel as the **Problems** tab.

4.2. Use the `ansible-navigator run` command to run the playbook. The output shows a fatal result for the second task in the play.

```
[student@workstation run-trouble]$ ansible-navigator run vlan_data.yml

PLAY [Creating a file with VLANs data] ************************************

TASK [Gather IOS interface facts] *****************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Copying the vlan configuration to a file] ******************************
fatal: [iosxe1.lab.example.com]: FAILED! => {"msg": "to_nice_yaml - 'vlans' is
 undefined. 'vlans' is undefined"}
fatal: [iosxe2.lab.example.com]: FAILED! => {"msg": "to_nice_yaml - 'vlans' is
 undefined. 'vlans' is undefined"}
```

```
PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=1    changed=0    unreachable=0    failed=1    ...
iosxe2.lab.example.com     : ok=1    changed=0    unreachable=0    failed=1    ...

Please review the log for errors.
```

4.3. Run the playbook again. This time use the `--pae true` option to generate one artifact file.

```
[student@workstation run-trouble]$ ansible-navigator run vlan_data.yml --pae true
```

4.4. Locate the generated playbook artifact. The name of the playbook artifact file starts with the name of the playbook, followed by `-artifact-`, the time when the playbook was run, and ending with the `json` file extension.

```
[student@workstation run-trouble]$ ls vlan_data-artifact*.json
vlan_data-artifact-2023-06-12T04:14:14.286421+00:00.json
```

5. Use the playbook artifact to explore the previous run and troubleshoot the issues found.

5.1. Replay the generated playbook artifact using the `ansible-navigator replay` command with the `-m interactive` option:

```
[student@workstation run-trouble]$ ansible-navigator replay -m interactive \
your-artifact-file-name
```

5.2. The initial replay screen from automation content navigator shows the failed status for the `Creating a file with VLANs data` play. Press 0 to display details of the play. The output shows that the first task was successful in both managed nodes. As expected, the second task failed on both managed nodes.

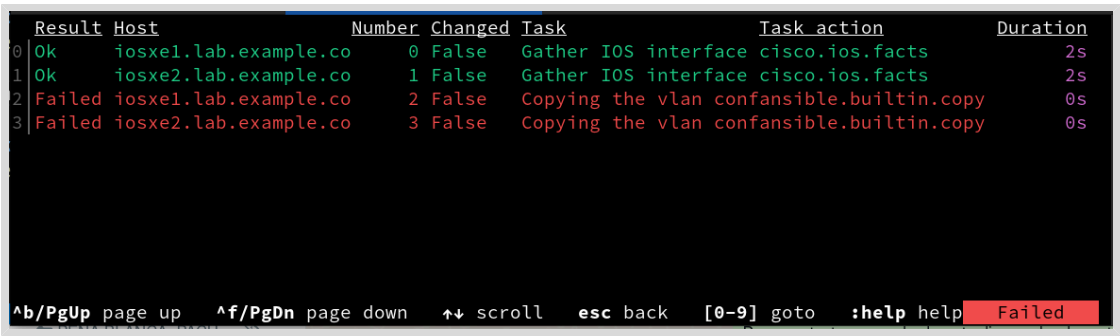


Figure 3.45: Replay screen for the playbook artifact

5.3. Press 2 to display the details of the second task in the `iosxe1.lab.example.com` managed node. The message indicates that the `vlans` variable is undefined.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
Play name: Creating a file with VLANs data:2
Task name: Copying the vlan configuration to a file
Failed: iosxe1.lab.example.com to_nice_yaml - 'vlans' is undefined. 'vlans' is undefined
 0 ---
 1 duration: 0.028802
 2 end: '2023-06-09T04:14:13.820299'
 3 event_loop: null
 4 host: iosxe1.lab.example.com
 5 ignore_errors: null
 6 play: Creating a file with VLANs data
 7 play_pattern: ios
 8 playbook: /home/student/run-trouble/vlan_data.yml
 9 remote_addr: iosxe1.lab.example.com
10 res:
11   _ansible_no_log: false
12   msg: to_nice_yaml - 'vlans' is undefined. 'vlans' is undefined
13 resolved_action: ansible.builtin.copy
14 start: '2023-06-09T04:14:13.791497'
^b/PgUp page up^f/PgDn page down↑↓ scrollesc back- previous+ next[0-9] goto:help hel   Failed
```

Figure 3.46: The variable used in the second task is undefined

5.4. Press `Esc` to go back to the previous page. Because the second task only saves the information obtained in the first task, the defined variable that provides information about the VLANs is in the first task.

5.5. Press 0 to inspect the details of the first task in the `iosxe1.lab.example.com` managed node. Notice that the `ansible_network_resources` variable specifies the VLANs for the managed node.

```
Play name: Creating a file with VLANs data:0
Task name: Gather IOS interface facts
Ok: iosxe1.lab.example.com
 0 ---
 1 duration: 2.252545
 2 end: '2023-06-09T04:14:13.736739'
 3 event_loop: null
 4 host: iosxe1.lab.example.com
 5 play: Creating a file with VLANs data
 6 play_pattern: ios
 7 playbook: /home/student/run-trouble/vlan_data.yml
 8 remote_addr: iosxe1.lab.example.com
 9 res:
10   _ansible_no_log: null
11   ansible_facts:
12     ansible_net_api: cliconf
13     ansible_net_gather_network_resources:
14     - vlans
15     ansible_net_gather_subset:
16     - default
17     ansible_net_hostname: iosxe1.lab.example.com
18     ansible_net_image: bootflash:packages.conf
19     ansible_net_iostype: IOS-XE
20     ansible_net_model: C8000V
21     ansible_net_operatingmode: autonomous
22     ansible_net_python_version: 3.9.13
23     ansible_net_serialnum: 916NT1210KQ
24     ansible_net_system: ios
25     ansible_net_version: 17.06.02
26     ansible_network_resources:
27       vlans:
28       - mtu: 1500
29         name: default
30         shutdown: disabled
31         state: active
^b/PgUp page up^f/PgDn page down↑↓ scrollesc back- previous+ next[0-9] goto:help hel   Failed
```

Figure 3.47: Variable specifying the information regarding to VLANs

5.6. Type `:q` and then press `Enter` to exit the `ansible-navigator` command.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. Verify that the `ansible_network_resources` variable provides you with the information you want to save to the local files.

6.1. Edit the playbook in VS Code. Add a task between the two existing tasks, and then save the file. Use the `ansible.builtin.debug` module with the `var` parameter to print the value for the `ansible_network_resources` variable:

```
- name: Displaying the variable to debug
  ansible.builtin.debug:
    var: ansible_network_resources
```

Notice how as you create the task in VS Code, the autocomplete feature helps you to avoid making mistakes.

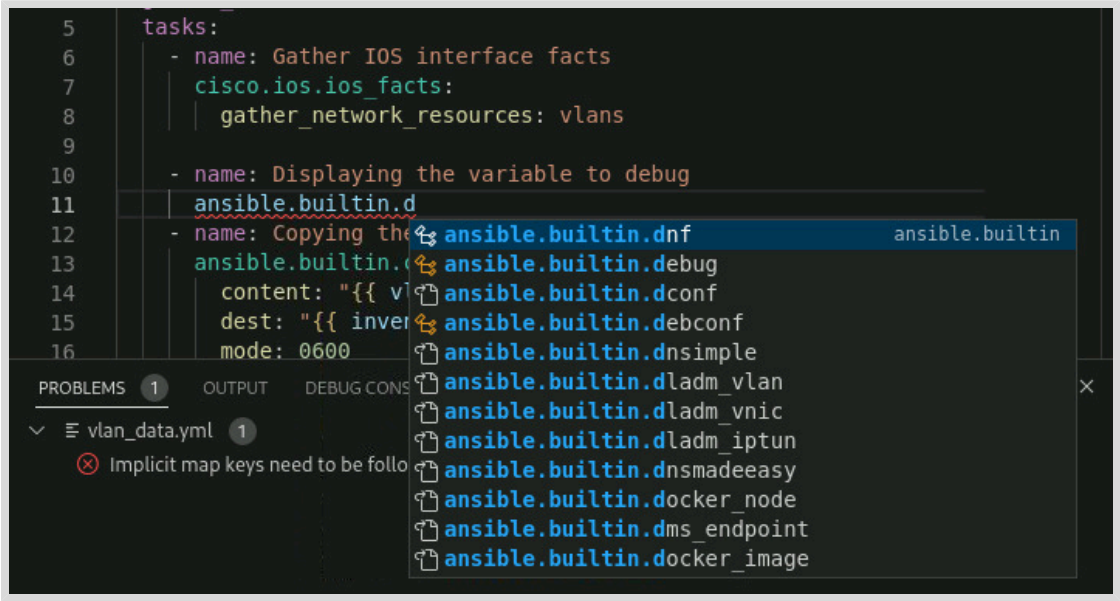


Figure 3.48: The autocomplete feature in VS Code provides suggestions as you type

6.2. Go back to the terminal and run the playbook again. The `ansible_network_resources` variable effectively provides the required information for both managed nodes.

```
[student@workstation run-trouble]$ ansible-navigator run vlan_data.yml

PLAY [Creating a file with VLANs data] ****************************************

TASK [Gather IOS interface facts] *********************************************
ok: [iosxe1.lab.example.com]
ok: [iosxe2.lab.example.com]

TASK [Displaying the variable to debug] ***************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_network_resources": {
        "vlans": [
            {
                "mtu": 1500,
                "name": "default",
                "shutdown": "disabled",
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
                "state": "active",
                "vlan_id": 1
            },
...output omitted...
ok: [iosxe2.lab.example.com] => {
    "ansible_network_resources": {
        "vlans": [
            {
                "mtu": 1500,
                "name": "default",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 1
            },
...output omitted...

TASK [Copying the vlan configuration to a file] ********************************
fatal: [iosxe1.lab.example.com]: FAILED! => {"msg": "to_nice_yaml - 'vlans' is
 undefined. 'vlans' is undefined"}
fatal: [iosxe2.lab.example.com]: FAILED! => {"msg": "to_nice_yaml - 'vlans' is
 undefined. 'vlans' is undefined"}

PLAY RECAP *********************************************************************
iosxe1.lab.example.com    : ok=2   changed=0   unreachable=0   failed=1      ...
iosxe2.lab.example.com    : ok=2   changed=0   unreachable=0   failed=1      ...
```

7. Now that you found what the problem is and the variable you are looking for, fix the playbook. Verify that it works as expected.

   7.1. Edit the playbook. Delete the task that you added in the previous step and use the `ansible_network_resources` variable in the second task, instead of using the undefined `vlans` variable. The fixed version of the `vlan_data.yml` playbook must contain the following content:

```
---
- name: Creating a file with VLANs data
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS interface facts
      cisco.ios.facts:
        gather_network_resources: vlans

    - name: Copying the vlan configuration to a file
      ansible.builtin.copy:
        content: "{{ ansible_network_resources | to_nice_yaml }}"
        dest: "{{ inventory_hostname }}_vlan.yml"
        mode: 0600
```

   7.2. Run the playbook again. This time the playbook ends successfully. The second task has a result of `changed` which means that the local files have been created.

```
[student@workstation run-trouble]$ ansible-navigator run vlan_data.yml

PLAY [Creating a file with VLANs data] *****************************************

TASK [Gather IOS interface facts] **********************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Copying the vlan configuration to a file] ********************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0      ...
iosxe2.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0      ...
```

7.3. Find the `iosxe1.lab.example.com_vlan.yml` and `iosxe2.lab.example.com_vlan.yml` files in the explorer area of VS Code. Select either one of them to see the VLAN data.

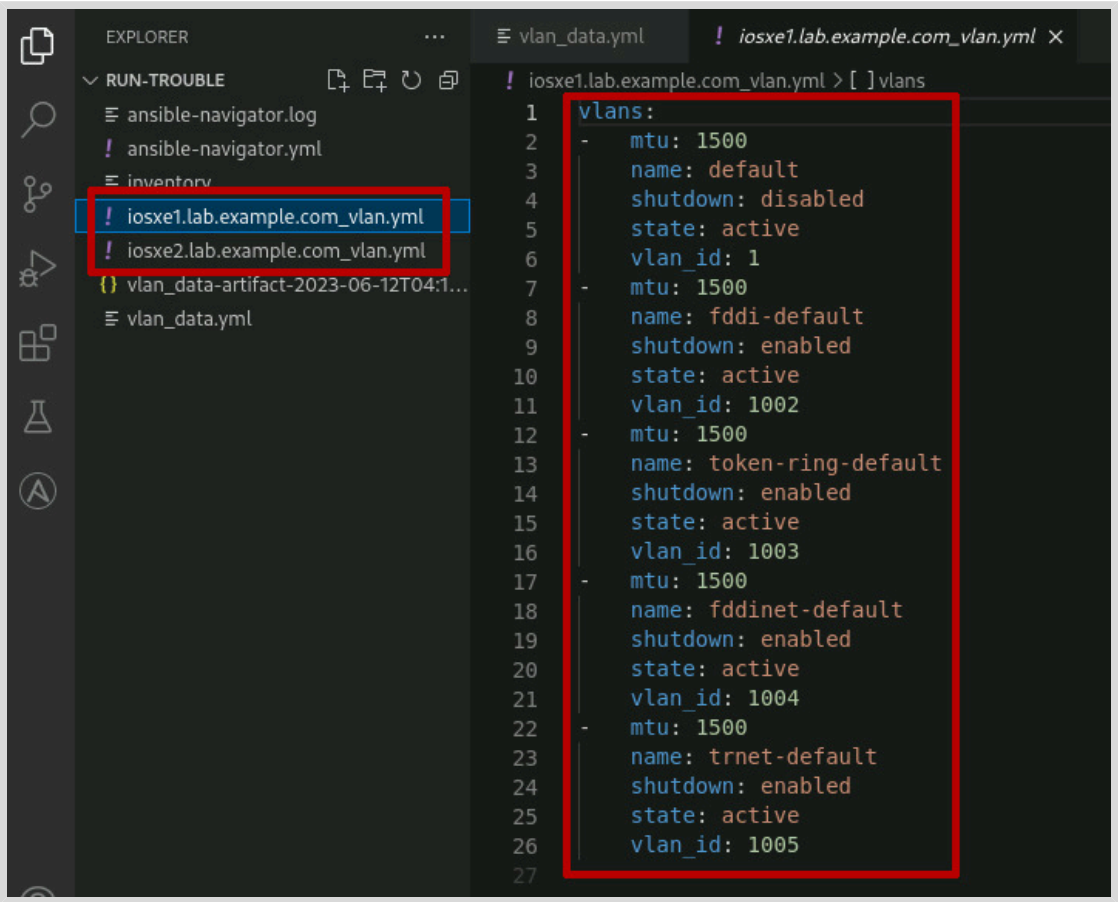


Figure 3.49: The VLAN information saved in local files

8. Close the `/home/student/run-trouble` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

8.1. Click **File** > **Close Folder** to close the `/home/student/run-trouble` directory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

8.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation run-trouble]$ cd
```

Finish

On the `workstation` machine, 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 run-trouble
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Troubleshooting Ansible Network Communication

Objectives

- Verify that Ansible can communicate with managed network devices, and troubleshoot any problems with those connections.

Troubleshooting Network Automation

Automation tasks can fail for various reasons. The following issues are the most common reasons that network automation tasks can fail.

- Connectivity issues, such as routing issues from the Ansible control node to the managed nodes.
- Authentication issues, such as an incorrect username, password, or SSH key credentials.
- Timeout issues, where commands take longer to run than the configured timeout value. A timeout issue might also be due to an authentication or connectivity issue.

Isolating and Resolving Connectivity Issues

Use the platform specific `*_ping` modules to verify connectivity to managed nodes.

For example, use the `ios_ping` module from the `cisco.ios` collection to ping Cisco managed nodes:

```
---
- name: Ping Cisco managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Ping Cisco managed nodes
      cisco.ios.ios_ping:
        dest: "{{ inventory_hostname }}"
        count: 5
```

If a ping task fails, you see an error indicating the root cause of the issue.

For example, the following error indicates that the managed node is unreachable via the IP address or hostname provided:

```
TASK [Ping Cisco managed nodes] ************************************************
fatal: [iosxe1.lab.example.com]: FAILED! => {"changed": false, "msg":
 "ssh connection failed: ssh connect failed: Timeout connecting to
 iosxe1.lab.example.com"}
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Verifying Platform Options

When writing playbooks for network automation, always verify that you are using the correct setting for each network platform option, such as `ansible_connection` and `ansible_network_os`.

Visit the network platform documentation at https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html to verify that you are using the correct settings for each network platform.

Isolating and Resolving Authentication Issues

When you run a playbook, it might fail with the error message `No authentication methods available`, or another error indicating that authentication to the managed node has failed.

```
TASK [Change configuration on managed nodes] ************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "msg": "No
 authentication methods available"}
```

This message means that the Ansible control node where you ran the playbook cannot connect via password or SSH key to the managed node.

SSH Key Issues

If you cannot connect to managed nodes using SSH keys, you can still connect by adding the `-k` option to the `ansible-navigator run` command. This option prompts for the connection password.

Because the `-k` option only prompts for **one** connection password, problems can occur if the managed nodes use different passwords.

One solution is to ensure that the hosts targeted by the playbook use the same connection password. This might mean having separate playbooks for the various types of network devices.

Another solution is to use a single playbook, but use the `--limit` option to restrict which managed nodes are targeted by the playbook:

```
[user@host ~]$ ansible-navigator run snmp.yml --limit junos1.lab.example.com
```

You can explicitly define the location of the SSH key for an inventory host or host group. Define the SSH key location by setting the `ansible_ssh_private_key_file` variable with the path to the appropriate SSH key.

Important

Although setting the `ansible_ssh_private_key_file` variable works from the CLI, it does not work from automation controller if you import the inventory (and associated variables).

Automation controller can fail with a message similar to: `No such file or directory: '/home/runner/.ssh/lab_rsa'`.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Troubleshooting Privilege Escalation

A possible authentication error is not being able to enter the privileged configuration mode of a managed node.

To resolve this issue, ensure that you set the `ansible_become` variable to `true` and the `ansible_become_method` variable to `enable`.

> **Note**
> For network automation playbooks, the only valid value for the `ansible_become_method` option is `enable`.

```
[user@host ~]$ cat inventory
...output omitted...
[ios:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios
ansible_become=true
ansible_become_method=enable
...output omitted...
```

Isolating and Resolving Timeout Issues

You might see the following error if the amount of time to wait for a command from the managed node is too low for a task to succeed:

```
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "msg": "command
 timeout triggered, timeout value is 30 secs.\nSee the timeout setting options in
 the Network Debug and Troubleshooting Guide."}
```

You also might see a timeout error due to authentication or DNS resolution errors.

To troubleshoot these types of errors, you can set the value of the `ansible_command_timeout` variable for a play or a task.

```
- name: Save the running-config
  vars:
    ansible_command_timeout: 120
  cisco.ios.ios_command:
    commands:
      - copy running-config startup-config
```

The related environment variable is the `ANSIBLE_PERSISTENT_COMMAND_TIMEOUT` variable. This environment variable is set to 30 seconds by default. You can use the `export` command to modify the value of the `ANSIBLE_PERSISTENT_COMMAND_TIMEOUT` environment variable in your current session:

```
export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=120
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Using the Long Form of Commands

Always use the long form of commands. For example, use the command `no shutdown` versus `no shut`.

When committing a configuration, the network OS converts the short form of commands into the long form of commands. For example, if you specify the command `shut`, then the configuration line is stored as `shutdown`.

The `*_config` module compares the command in your task to the line in the configuration. If you use the short form of the command, then the module returns `changed=true` every time the play runs, even though the configuration might already be correct. This means that your task is not idempotent and the configuration is updated every time the task runs.

To avoid this problem, use the long form of commands when using `*_config` modules in your tasks. You can still abbreviate the configuration statements in the play name if desired.

```
---
- name: Change interface configuration
  hosts: ios
  gather_facts: false
  tasks:
    - name: Shutdown interface Gig0/1
      cisco.ios.ios_config:
        lines:
          - shutdown
        parents: interface GigabitEthernet0/1
```

References

Network Debug and Troubleshooting Guide - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/network_debug_troubleshooting.html

Platform Options - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Troubleshooting Ansible Network Communication

Troubleshoot problems with communication between your automation execution environment and your managed network nodes.

Outcomes

- Troubleshoot and resolve network communication issues.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start run-communication
```

Instructions

You have been provided with a playbook called `snmp.yml`, which runs against the `junos` inventory group of managed nodes.

Managed Nodes in the `junos` Inventory Group

Managed node	IP address
`junos1.lab.example.com`	172.25.250.22
`junos2.lab.example.com`	172.25.250.23

DNS Name Server IP Address and Domain Name

Name server IP address	Domain name
`172.25.250.220`	`lab.example.com`

The playbook performs a connectivity verification, and then configures the Simple Network Management Protocol (SNMP) in the Juniper Junos managed nodes. You were told that the user account used to connect to the managed nodes is `student` with `Student` as the password.

The exercise also provides you with three additional playbooks, which you can use to troubleshoot connectivity issues:

- The `netconf.yml` playbook enables and configures the Network Configuration Protocol (NETCONF) system service. The playbook then checks the connectivity by waiting five seconds for port 830 to become open on the managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- The `dns.yml` playbook configures the Domain Name System (DNS) service. The playbook then verifies and displays the managed node DNS configuration.
- The `ping.yml` playbook tests that a remote destination is reachable from the Juniper Junos managed nodes.

Based on the error messages provided by Ansible when you run the `snmp.yml` playbook, troubleshoot the network communication issues. Ensure that there are no connectivity problems if more Juniper Junos managed nodes are added to the `junos` inventory group.

1. Open the `/home/student/run-communication` directory in VS Code and review the `snmp.yml` playbook.

   1.1. Open VS Code and then click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **run-communication** and then click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Click the `snmp.yml` playbook. Notice that the playbook consists of one play with two tasks to execute against the `junos` inventory group of managed nodes:

```
---
- name: Configure SNMP on Juniper managed nodes
  hosts: junos
  gather_facts: false
  tasks:
    - name: Verify connectivity between the managed nodes
       vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_ping:
        dest: "{{ inventory_hostname }}"
        count: 5

    - name: Configure SNMP on managed nodes
      junipernetworks.junos.junos_snmp_server:
        state: merged
        config:
          location: 'Raleigh, NC'
          contact: 'Network Engineering | neteng@company.com'
          description: "SNMP Server"
          communities:
            - name: rocommunity2n4g!
              authorization: read-only
            - name: rwcommunityd7g$v
              authorization: read-write
```

2. Try to run the `snmp.yml` playbook. The playbook fails at the `Configure SNMP on managed nodes` task for the `junos1.lab.example.com` managed node, and at the `Verify connectivity between the managed nodes` task for the `junos2.lab.example.com` managed node.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Isolate the error found for the `junos1.lab.example.com` managed node. By isolating each error, you can troubleshoot the issues you find one by one.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/run-communication` directory in a GNOME terminal:

```
[student@workstation ~]$ cd ~/run-communication
[student@workstation run-communication]$
```

2.2. Use the `ansible-navigator run` command to run the `snmp.yml` playbook. The output shows a fatal result for both managed nodes:

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Verify connectivity between the managed nodes] *****************************
ok: [junos1.lab.example.com]
fatal: [junos2.lab.example.com]: FAILED! => {"changed": false, "module_stderr":
 "expected string or bytes-like object", "module_stdout": "", "msg": "MODULE
 FAILURE\nSee stdout/stderr for the exact error"}
ok: [junos1.lab.example.com]

TASK [Configure SNMP on managed nodes] *******************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "msg": "Could not
 open socket to junos1.lab.example.com:830"}

PLAY RECAP ************************************************************************
junos1.lab.example.com     : ok=1    changed=0    unreachable=0    failed=1    ...
junos2.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1    ...

Please review the log for errors.
```

2.3. Run the playbook using the `--limit` option to isolate the error message for the `junos1.lab.example.com` managed node:

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml \
--limit junos1.lab.example.com

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Verify connectivity between the managed nodes] *****************************
ok: [junos1.lab.example.com]

TASK [Configure SNMP on managed nodes] *******************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "msg": "Could not
 open socket to junos1.lab.example.com:830"}

PLAY RECAP ************************************************************************
junos1.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1    ...

Please review the log for errors.
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3. Troubleshoot the `Could not open socket to junos1.lab.example.com:830` error message.

3.1. *(Optional)* Use the `ssh` command to determine if NETCONF over SSH is enabled. The error message indicates that the SSH connection to port 830 on the managed node was refused. That might indicate that the NETCONF service on the managed node is disabled.

```
[student@workstation run-communication]$ ssh junos1.lab.example.com -p 830 \
-s netconf
ssh: connect to host junos1.lab.example.com port 830: Connection refused
```

When you have many managed nodes where you need to configure or verify connectivity, running the provided `netconf.yml` playbook is a better option than using the `ssh` command.

3.2. Open the `netconf.yml` playbook in VS Code. Notice that the first task in the playbook performs the NETCONF configuration. This task is currently commented out. The second task validates that there is indeed connectivity.

```
---
- name: Networking - netconf
  hosts: junos
  gather_facts: false
  tasks:
#    - name: Enable netconf service on port 830
#      vars:
#        ansible_connection: ansible.netcommon.network_cli
#      junipernetworks.junos.junos_netconf:
#        netconf_port: 830
#        state: present

    - name: Checking connectivity
      ansible.builtin.wait_for:
        host: "{{ inventory_hostname }}"
        port: 830
        timeout: 5
```

3.3. Run the `netconf.yml` playbook against the `junos1.lab.example.com` managed node. The timeout when waiting for an answer indicates that after five seconds, port 830 was not available on the managed node.

```
[student@workstation run-communication]$ ansible-navigator run netconf.yml \
--limit junos1.lab.example.com

PLAY [Networking - netconf] ****************************************************

TASK [Checking connectivity] ***************************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "elapsed": 5,
 "msg": "Timeout when waiting for junos1.lab.example.com:830"}

PLAY RECAP *********************************************************************
```

```
junos1.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1    ...

Please review the log for errors.
```

3.4. Uncomment the task that configures the NETCONF service in the `netconf.yml` playbook and then save the playbook. Run the `netconf.yml` playbook again against the `junos1.lab.example.com` managed node. Both tasks in the playbook complete successfully.

```
[student@workstation run-communication]$ ansible-navigator run netconf.yml \
--limit junos1.lab.example.com

PLAY [Networking - netconf] ****************************************************

TASK [Enable netconf service on port 830] ***************************************
changed: [junos1.lab.example.com]

TASK [Checking connectivity] ****************************************************
ok: [junos1.lab.example.com]

PLAY RECAP **********************************************************************
junos1.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0    ...
```

3.5. Add the NETCONF configuration task to the `snmp.yml` playbook to avoid future errors related to the missing NETCONF configuration for new Juniper Junos managed nodes added to your environment.

Edit the `snmp.yml` file and add the `Enable netconf service on port 830` task before the `Verify connectivity between the managed nodes` task, and then save the playbook:

```
---
- name: Configure SNMP on Juniper managed nodes
  hosts: junos
  gather_facts: false
  tasks:
    - name: Enable netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present

    - name: Verify connectivity between the managed nodes
...output omitted...
```

4. Run the `snmp.yml` playbook again against the `junos1.lab.example.com` managed node and troubleshoot the new error message obtained.

4.1. Run the `snmp.yml` playbook against the `junos1.lab.example.com` managed node. Although the playbook still stops at the same task, the error regarding NETCONF is no longer displayed.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

Depending on the connection to the `workstation` machine, you can obtain the `SSHException('No existing session')` or the `No authentication methods available` error message. The root cause of the issue is the same.

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml \
--limit junos1.lab.example.com

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Enable netconf service on port 830] ****************************************
ok: [junos1.lab.example.com]

TASK [Verify connectivity between the managed nodes] *****************************
ok: [junos1.lab.example.com]

TASK [Configure SNMP on managed nodes] *******************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "msg":
 "SSHException: No existing session"}

PLAY RECAP ************************************************************************
junos1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=1    ...

Please review the log for errors.
```

4.2. Because you were told that the `student` user uses `Student` as the password to connect to the managed nodes, one of the following reasons is causing the failure:

- The `Student` password is incorrect.
- The `Student` password is not defined in your project.

4.3. Use the `ssh` command to log in to the `junos1.lab.example.com` managed node. Add the `-o PreferredAuthentications=password` option to authenticate using a password. When prompted, enter `Student` as the SSH connection password.

Because the connection is successful, the `Student` provided password is correct. Exit from the managed node:

```
[student@workstation run-communication]$ ssh junos1.lab.example.com \
-o PreferredAuthentications=password
student@junos1.lab.example.com's password: Student
Last login: Wed Jun 21 19:02:00 2023 from 172.25.250.9
--- JUNOS 23.1R1.8 Kernel 64-bit  JNPR-12.1-20230307.3e7c4b6_buil
student> exit

Connection to junos1.lab.example.com closed.
[student@workstation run-communication]$
```

4.4. Open the inventory file in VS Code. Inspect the variables for the `junos` inventory group used to authenticate when connecting to the managed nodes. Notice that the

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

only defined authentication variable is `ansible_user`. The playbook fails because it does not have the password to use when it tries to connect to the managed node:

```
[junos]
junos1.lab.example.com
junos2.lab.example.com

[junos:vars]
ansible_user=student
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

It is not good practice to store passwords in plain text, so leave the inventory file as it is. You can use the `-k` option for the `ansible-navigator` command to type the authentication password when you run the playbook.

4.5. Run the `snmp.yml` playbook against the `junos1.lab.example.com` managed node. Add the `-k` option to the `ansible-navigator run` command and enter `Student` as the SSH connection password when prompted.

The playbook runs successfully.

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml \
--limit junos1.lab.example.com -k
SSH password: Student

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Enable netconf service on port 830] ****************************************
ok: [junos1.lab.example.com]

TASK [Verify connectivity between the managed nodes] *****************************
ok: [junos1.lab.example.com]

TASK [Configure SNMP on managed nodes] *******************************************
changed: [junos1.lab.example.com]

PLAY RECAP ************************************************************************
junos1.lab.example.com     : ok=3    changed=1    unreachable=0    failed=0    ...
```

5. Run the `snmp.yml` playbook against the `junos2.lab.example.com` managed node to isolate the communication issue with this managed node. Investigate the connectivity error.

5.1. Run the `snmp.yml` playbook against the `junos2.lab.example.com` managed node. Add the `-k` option to the `ansible-navigator` command. When prompted, enter `Student` as the password. The playbook fails in the `Verify connectivity between the managed nodes` task:

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml \
--limit junos2.lab.example.com -k
SSH password: Student

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Enable netconf service on port 830] ****************************************
```

```
ok: [junos2.lab.example.com]

TASK [Verify connectivity between the managed nodes] ****************************
fatal: [junos2.lab.example.com]: FAILED! => {"changed": false, "module_stderr":
 "expected string or bytes-like object", "module_stdout": "", "msg": "MODULE
 FAILURE\nSee stdout/stderr for the exact error"}

PLAY RECAP *********************************************************************
junos2.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1    ...
```

5.2. In VS Code, open the `ping.yml` file. Notice that the playbook uses the `junipernetworks.junos.junos_ping` module to ping the managed nodes in the `junos` inventory group:

```
---
- name: Networking - ping
  hosts: junos
  gather_facts: false
  tasks:
    - name: Ping managed nodes
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_ping:
        dest: "{{ inventory_hostname }}"
        count: 5
```

5.3. Run the `ping.yml` playbook against the `junos2.lab.example.com` managed node. The error message when you run the `ping.yml` playbook is the same as that received when you run the `snmp.yml` playbook. This is because the `snmp.yml` uses the `junipernetworks.junos.junos_ping` module to verify the connectivity between the managed nodes:

```
[student@workstation run-communication]$ ansible-navigator run ping.yml \
--limit junos2.lab.example.com

PLAY [Networking - ping] *******************************************************

TASK [Ping managed nodes] ******************************************************
fatal: [junos2.lab.example.com]: FAILED! => {"changed": false, "module_stderr":
 "expected string or bytes-like object", "module_stdout": "", "msg": "MODULE
 FAILURE\nSee stdout/stderr for the exact error"}

PLAY RECAP *********************************************************************
junos2.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1    ...
```

5.4. Connect to the `junos2.lab.example.com` managed node and ping the managed nodes:

```
[student@workstation run-communication]$ ssh junos2.lab.example.com
...output omitted...
--- JUNOS 23.1R1.8 Kernel 64-bit  JNPR-12.1-20230307.3e7c4b6_buil
student@junos2.lab.example.com> ping junos1.lab.example.com
ping: cannot resolve junos1.lab.example.com: Host name lookup failure
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
student@junos2.lab.example.com> ping junos2.lab.example.com
ping: cannot resolve junos2.lab.example.com: Host name lookup failure

student@junos2.lab.example.com>
```

5.5. The messages suggest that the problem is related to the DNS configuration on the `junos2.lab.example.com` managed node. Exit from the `junos2.lab.example.com` managed node:

```
[student@junos2.lab.example.com> exit

Connection to junos2.lab.example.com closed.
[student@workstation run-communication]$
```

6. Solve the connectivity issue for the `junos2.lab.example.com` managed node.

6.1. Open the `dns.yml` playbook in VS Code. The first task in the playbook enables the NETCONF service in the managed node. This task is required when using the `junipernetworks.junos.junos_system` module to configure DNS on Juniper Junos managed nodes. The second task performs the DNS configuration. Both the first and second tasks are commented out in the playbook.

The last two tasks in the playbook verify and display the configuration in each managed node:

```
---
- name: Networking - dns
  hosts: junos
  gather_facts: false
  tasks:
#    - name: Enable netconf service on port 830
#      vars:
#        ansible_connection: ansible.netcommon.network_cli
#      junipernetworks.junos.junos_netconf:
#        netconf_port: 830
#        state: present

#    - name: Configure DNS settings
#      junipernetworks.junos.junos_system:
#        name_servers: 172.25.250.220

    - name: Checking DNS configuration
      junipernetworks.junos.junos_command:
        commands:
          - show configuration system name-server
      register: junos_dns

    - name: Display DNS configuration
      ansible.builtin.debug:
        var: junos_dns['stdout']
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6.2. Run the `dns.yml` playbook to verify and compare any differences in the DNS configuration for each managed node. Add the `-k` option to the `ansible-navigator` command. When prompted, enter `Student` as the password.

Note that the DNS configuration is missing in the `junos2.lab.example.com` managed node:

```
[student@workstation run-communication]$ ansible-navigator run dns.yml -k
SSH password: Student
...output omitted...
TASK [Display DNS configuration] ************************************************
ok: [junos1.lab.example.com] => {
    "junos_dns['stdout']": [
        "172.25.250.220;"
    ]
}
ok: [junos2.lab.example.com] => {
    "junos_dns['stdout']": [
        ""
    ]
}
...output omitted...
```

6.3. Uncomment the first two tasks for configuring DNS in the `dns.yml` playbook, and save the playbook. Run the `dns.yml` playbook against the `junos2.lab.example.com` managed node. Add the `-k` option to the `ansible-navigator` command and enter `Student` as the password when prompted:

```
[student@workstation run-communication]$ ansible-navigator run dns.yml \
--limit junos2.lab.example.com -k
SSH password: Student

PLAY [Networking - dns] *********************************************************

TASK [Enable netconf service on port 830] ***************************************
ok: [junos2.lab.example.com]

TASK [Configure DNS settings] ***************************************************
changed: [junos2.lab.example.com]

TASK [Checking DNS configuration] ***********************************************
ok: [junos2.lab.example.com]

TASK [Display DNS configuration] ************************************************
ok: [junos2.lab.example.com] => {
    "junos_dns['stdout']": [
        "172.25.250.220;"
    ]
}

PLAY RECAP ******************************************************************
junos2.lab.example.com     : ok=4    changed=1    unreachable=0    failed=0    ...
```

The `Configure DNS settings` task runs successfully on the `junos2.lab.example.com` managed node. This time the managed node has the correct DNS configuration.

6.4. Verify that the connectivity issue for the `junos2.lab.example.com` has been solved by running the `ping.yml` playbook again. The playbook finishes successfully.

```
[student@workstation run-communication]$ ansible-navigator run ping.yml \
--limit junos2.lab.example.com

PLAY [Networking - ping] *******************************************************

TASK [Ping managed nodes] ******************************************************
ok: [junos2.lab.example.com]

PLAY RECAP *********************************************************************
junos2.lab.example.com     : ok=1    changed=0    unreachable=0    failed=0    ...
```

6.5. Add the `Configure DNS settings` task to the `snmp.yml` playbook to avoid issues related to the missing DNS configuration on any new Juniper Junos managed node that might be added to the `junos` inventory group.

Because the DNS configuration requires enabling the `netconf` service, add the task after the `Enable netconf service on port 830` task and before the `Verify connectivity between the managed nodes` task. When done, save the playbook:

```
...output omitted...
  tasks:
    - name: Enable netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present

    - name: Configure DNS settings
      junipernetworks.junos.junos_system:
        name_servers: 172.25.250.220

    - name: Verify connectivity between the managed nodes
...output omitted...
```

7. Verify that the playbook runs successfully for all the Juniper Junos managed nodes. Close the `/home/student/run-communication` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

7.1. Run the `snmp.yml` playbook without isolating any managed node. Add the `-k` option to the `ansible-navigator` command. When prompted, enter `Student` as the password. The playbook runs successfully.

```
[student@workstation run-communication]$ ansible-navigator run snmp.yml -k
SSH password: Student
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Enable netconf service on port 830] ****************************************
ok: [junos1.lab.example.com]
ok: [junos2.lab.example.com]

TASK [Configure DNS settings] ****************************************************
ok: [junos1.lab.example.com]
ok: [junos2.lab.example.com]

TASK [Verify connectivity between the managed nodes] *****************************
ok: [junos1.lab.example.com]
ok: [junos2.lab.example.com]

TASK [Configure SNMP on managed nodes] *******************************************
ok: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

PLAY RECAP ***********************************************************************
junos1.lab.example.com     : ok=4    changed=0    unreachable=0    failed=0    ...
junos2.lab.example.com     : ok=4    changed=1    unreachable=0    failed=0    ...
```

7.2. Click **File** > **Close Folder** in VS Code to close the `/home/student/run-communication` directory.

7.3. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation run-communication]$ cd
```

Finish

On the `workstation` machine, 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 run-communication
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Managing and Running Playbooks

Commit changes to an existing Ansible project in a Git repository, use automation controller to launch a job using a playbook from that project, and review the job results.

Outcomes

- Consult module documentation to identify module requirements.
- Create a Git branch and update a remote Git repository.
- Create an automation controller machine credential, inventory, and job template.
- Launch an automation controller 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 a Git repository, creates some automation controller resources, and removes DNS settings from the Juniper Junos managed nodes.

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

Instructions

This exercise provides you with a playbook that you can use to configure DNS settings on the Juniper Junos managed nodes in your environment. The initial playbook does not work and you must fix it. After updating the playbook, create automation controller resources so that you can run the playbook as a job template.

Navigate to `https://controller.lab.example.com` to access the automation controller web UI. Log in as `admin` using `redhat` as the password.

1. Clone the `git@git.lab.example.com:student/run-review` Git repository into the `/home/student/git-repos` directory and then create the `exercise` branch.
2. The existing `/home/student/git-repos/run-review/junos_dns.yml` playbook fails with the `Could not open socket to junos1.lab.example.com:830` error message. Update the `junos_dns.yml` playbook to add a task that resolves this error.

   To solve this problem, you might consult documentation for the `junipernetworks.junos.junos_system` module.
3. Update the `/home/student/git-repos/run-review/inventory` file to remove the line that sets the `ansible_ssh_private_key_file` variable. Although this inventory file works correctly when used on the `workstation` machine, the variable does not work when running the playbook from automation controller. Automation controller cannot find the private SSH key file because the file specified by the variable is not part of the Git repository.
4. Use the `ansible-navigator lint` and `ansible-navigator run` commands to check your updated files for syntax errors and to verify that your playbook runs successfully. Add

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

the `-k` option to the `ansible-navigator run` command and, when prompted, enter `Student` as the SSH connection password. Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository. When finished, either close the opened folder in VS Code or return to the `/home/student` directory from the CLI.

5. From the automation controller web UI, update the `run-review` project resource so that the project uses the `exercise` branch.

6. Create a machine credential resource using the following information:

Field	Value
Name	`Juniper Junos Auth`
Organization	`Default`
Credential Type	`Machine`
Username	`student`
SSH Private Key	Content of the `/home/student/.ssh/lab_rsa` file.

7. Create an inventory resource that uses the `inventory` file in the project directory. Create the inventory using the following information:

Field	Value
Name	`Juniper Junos`
Organization	`Default`

Create a source for the inventory using the following information, and then synchronize the inventory source so that the inventory contains the `junos` host group and the `junos1.lab.example.com` host:

Field	Value
Name	`Juniper Junos Source`
Source	`Sourced from a Project`
Project	`run-review`
Inventory file	`inventory`
Update options	`Overwrite` *(selected)*
	`Overwrite variables` *(selected)*

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

8. Create a job template to run the `junos_dns.yml` playbook using the following information. Launch the job template and verify that the resulting job succeeds.

Field	Value
Name	`Juniper Junos DNS`
Inventory	`Juniper Junos`
Project	`run-review`
Playbook	`junos_dns.yml`
Credentials	`Juniper Junos Auth`

9. Close the `/home/student/git-repos/run-review` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

   9.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/run-review` directory.

   9.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

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 run-review
```

Finish

As the `student` user on the `workstation` machine, 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 run-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Managing and Running Playbooks

Commit changes to an existing Ansible project in a Git repository, use automation controller to launch a job using a playbook from that project, and review the job results.

Outcomes

- Consult module documentation to identify module requirements.
- Create a Git branch and update a remote Git repository.
- Create an automation controller machine credential, inventory, and job template.
- Launch an automation controller 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 a Git repository, creates some automation controller resources, and removes DNS settings from the Juniper Junos managed nodes.

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

Instructions

This exercise provides you with a playbook that you can use to configure DNS settings on the Juniper Junos managed nodes in your environment. The initial playbook does not work and you must fix it. After updating the playbook, create automation controller resources so that you can run the playbook as a job template.

Navigate to `https://controller.lab.example.com` to access the automation controller web UI. Log in as `admin` using `redhat` as the password.

1. Clone the `git@git.lab.example.com:student/run-review` Git repository into the `/home/student/git-repos` directory and then create the `exercise` branch.

   1.1. Use either VS Code or the `git clone` command to clone the `git@git.lab.example.com:student/run-review` Git repository into the `/home/student/git-repos` directory:

```
[student@workstation ~]$ cd ~/git-repos/
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/run-review
```

   1.2. Use either VS Code or the `git checkout -b exercise` command to create the `exercise` branch and check it out:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

2. The existing `/home/student/git-repos/run-review/junos_dns.yml` playbook fails with the `Could not open socket to junos1.lab.example.com:830` error message. Update the `junos_dns.yml` playbook to add a task that resolves this error.

To solve this problem, you might consult documentation for the `junipernetworks.junos.junos_system` module.

2.1. Attempt to run the `junos_dns.yml` playbook from either a VS Code terminal or a GNOME terminal. The playbook fails at the `Configure DNS settings` task with the `Could not open socket to junos1.lab.example.com:830` error message.

```
[student@workstation run-review]$ ansible-navigator run junos_dns.yml

PLAY [Configure DNS settings on Juniper devices] ******************************

TASK [Configure DNS settings] *************************************************
fatal: [junos1.lab.example.com]: FAILED! => {"changed": false, "module_stderr":
 "", "module_stdout": "\n{\"failed\": true, \"msg\": \"Could not open
 socket to junos1.lab.example.com:830\", \"invocation\": {\"module_args\":
 {\"domain_search\": [\"lab.example.com\", \"example.com\"], \"name_servers\":
 [\"172.25.250.220\"], \"state\": \"present\", \"active\": true, \"hostname
\": null, \"domain_name\": null}}}\n\n{\"failed\": true, \"msg\": \"Could not
 open socket to junos1.lab.example.com:830\", \"invocation\": {\"module_args\":
 {\"domain_search\": [\"lab.example.com\", \"example.com\"], \"name_servers\":
 [\"172.25.250.220\"], \"state\": \"present\", \"active\": true, \"hostname\":
 null, \"domain_name\": null}}}\n", "msg": "MODULE FAILURE\nSee stdout/stderr for
 the exact error"}

PLAY RECAP *********************************************************************
junos1.lab.example.com     : ok=0    changed=0    unreachable=0    failed=1 ...

Please review the log for errors.
```

2.2. The `Configure DNS settings` task uses the `junipernetworks.junos.junos_system` module to configure some DNS settings. Check the documentation for that module. The `NOTES` section indicates that the `netconf` system service is a requirement:

```
[student@workstation run-review]$ ansible-navigator doc \
junipernetworks.junos.junos_system
...output omitted...
NOTES:
      * This module requires the netconf system service be
        enabled on the remote device being managed.
...output omitted...
```

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

2.3. Interactively check the documentation for the `junipernetworks.junos.junos_netconf` module. If you do not know the

module name, then the following steps help you identify the module name. Run the `ansible-navigator collections` command. After the command enters interactive mode, type `:19` and press `Enter` to display information about the `junipernetworks.junos` collection:

```
[student@workstation run-review]$ ansible-navigator collections -m interactive
   Name                   Version  Shadowed  Type       Path
...output omitted...
19|junipernetworks.junos  4.0.0    False     contained  /usr/share/ansible/co...
...output omitted...
```

Type `:21` and press `Enter` to display information about the `junos_netconf` module:

```
   Junipernetworks.junos  Type     Added  Deprecated  Description
...output omitted...
21|junos_netconf          module   1.0.0  False       Configures the Junos Ne...
...output omitted...
```

Line 38 contains a note about using the `network_cli` connection method. Because the `inventory` file for the project defines a different `ansible_connection` method for the `junos` host group, you must override the connection method when you use the `junipernetworks.junos.junos_netconf` module.

```
36|  notes:
37|  - Tested against vSRX JUNOS version 15.1X49-D15.4, vqfx-10000 JUNOS Vers...
38|  - Recommended connection is C(network_cli). See L(the Junos OS Platform ...
```

Lines 75-78 contain an example task that uses the module. The `listens_on` option is an alias to the `netconf_port` option.

```
74|examples: |-
75|  - name: enable netconf service on port 830
76|    junipernetworks.junos.junos_netconf:
77|      listens_on: 830
78|      state: present
```

Type `:q` and then press `Enter` to exit the `ansible-navigator` command.

2.4. Use the VS Code editor to modify the `junos_dns.yml` playbook. Add the highlighted lines and then save the file. The added task overrides the `ansible_connection` variable that is currently defined in the `inventory` file for the `junos` host group. The entire playbook contains the following content:

```
---
- name: Configure DNS settings on Juniper devices
  hosts: junos
  gather_facts: false
  tasks:
    - name: Enable netconf service on port 830
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        netconf_port: 830
        state: present
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - name: Configure DNS settings
      junipernetworks.junos.junos_system:
        domain_search:
          - lab.example.com
          - example.com
        name_servers:
          - 172.25.250.220
```

Note

The `ansible-navigator doc -l -t connection` command lists available connection plug-ins. You might run this command to find the *fully qualified collection name (FQCN)* of the `network_cli` connection plug-in. You can use either `network_cli` or `ansible.netcommon.network_cli` as the value of the `ansible_connection` variable.

3. Update the `/home/student/git-repos/run-review/inventory` file to remove the line that sets the `ansible_ssh_private_key_file` variable. Although this inventory file works correctly when used on the `workstation` machine, the variable does not work when running the playbook from automation controller. Automation controller cannot find the private SSH key file because the file specified by the variable is not part of the Git repository.

3.1. Use the VS Code editor to modify the `inventory` file so that the file only contains the following content:

```
[junos]
junos1.lab.example.com

[junos:vars]
ansible_user=student
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

4. Use the `ansible-navigator lint` and `ansible-navigator run` commands to check your updated files for syntax errors and to verify that your playbook runs successfully. Add the `-k` option to the `ansible-navigator run` command and, when prompted, enter `Student` as the SSH connection password. Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository. When finished, either close the opened folder in VS Code or return to the `/home/student` directory from the CLI.

4.1. Check the modified playbook for style and syntax errors. The `ansible-navigator lint` command only returns output if the command detects potential problems:

```
[student@workstation run-review]$ ansible-navigator lint junos_dns.yml
```

4.2. Run the `junos_dns.yml` playbook and use the `-k` option. Enter `Student` as the SSH password:

```
[student@workstation run-review]$ ansible-navigator run junos_dns.yml -k
SSH password: Student
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
PLAY [Configure DNS settings on Juniper devices] ******************************

TASK [Enable netconf service on port 830] *************************************
changed: [junos1.lab.example.com]

TASK [Configure DNS settings] *************************************************
changed: [junos1.lab.example.com]

PLAY RECAP ********************************************************************
junos1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0   ...
```

4.3. Use either VS Code or the CLI to add, commit, and push the changes to the remote repository. Use a descriptive commit message, such as `Configure netconf service`. If you choose to use the CLI, then use the `git add`, `git commit`, and `git push` commands:

```
[student@workstation run-review]$ git add junos_dns.yml inventory
[student@workstation run-review]$ git commit -m "Configure netconf service"
[exercise 7cd3899] Configure netconf service
 2 files changed, 7 insertions(+), 1 deletion(-)
[student@workstation run-review]$ git push -u origin exercise
...output omitted...
To git.lab.example.com:student/run-review
 * [new branch]      exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

4.4. If you completed this step and the previous steps using VS Code, then click **File** > **Close Folder** to close the `/home/student/git-repos/run-review` directory. If you completed this step and the previous steps using the CLI, then run the `cd` command to return to the `/home/student` directory:

```
[student@workstation run-review]$ cd
```

5. From the automation controller web UI, update the `run-review` project resource so that the project uses the `exercise` branch.

5.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

5.2. Navigate to **Resources** > **Projects** and then click the **Edit Project** icon for the **run-review** project.

5.3. Enter `exercise` in the **Source Control Branch/Tag/Commit** field and then click **Save**. After you click **Save**, automation controller immediately attempts to synchronize the project. The **Last Job Status** field updates to indicate that the project synchronization succeeded.

6. Create a machine credential resource using the following information:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Field	Value
Name	`Juniper Junos Auth`
Organization	`Default`
Credential Type	`Machine`
Username	`student`
SSH Private Key	Content of the `/home/student/.ssh/lab_rsa` file.

Navigate to **Resources** > **Credentials** and click **Add**. Create the credential using the settings in the preceding table and then click **Save**.

7. Create an inventory resource that uses the `inventory` file in the project directory. Create the inventory using the following information:

Field	Value
Name	`Juniper Junos`
Organization	`Default`

Create a source for the inventory using the following information, and then synchronize the inventory source so that the inventory contains the `junos` host group and the `junos1.lab.example.com` host:

Field	Value
Name	`Juniper Junos Source`
Source	`Sourced from a Project`
Project	`run-review`
Inventory file	`inventory`
Update options	`Overwrite` *(selected)*
	`Overwrite variables` *(selected)*

7.1. Navigate to **Resources** > **Inventories** and then click **Add** > **Add inventory**. Create the inventory and then click **Save**.

7.2. From the **Sources** tab, click **Add**. Create an inventory source and then click **Save**.

7.3. Click **Sync** to synchronize the inventory source. The **Last Job Status** field updates to indicate that the synchronization succeeded.

8. Create a job template to run the `junos_dns.yml` playbook using the following information. Launch the job template and verify that the resulting job succeeds.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Field	Value
Name	`Juniper Junos DNS`
Inventory	`Juniper Junos`
Project	`run-review`
Playbook	`junos_dns.yml`
Credentials	`Juniper Junos Auth`

8.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. Create the job template with settings from the preceding table and then click **Save**.

8.2. Click **Launch** to run the job template. The resulting job succeeds.

8.3. Navigate to **Views** > **Jobs** and look for a numbered job with `Juniper Junos DNS` in the name. By default, automation controller lists the most recent jobs at the top. The `Juniper Junos DNS` job displays a status of **Successful**.

8.4. *(Optional)* Click the name of the `Juniper Junos DNS` job to review the job output.

9. Close the `/home/student/git-repos/run-review` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

9.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/run-review` directory.

9.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation run-review]$ cd
```

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 run-review
```

Finish

As the `student` user on the `workstation` machine, 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 run-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- Git is a distributed version control system (DVCS) that enables you to manage changes to files, review and restore earlier versions of files, compare two versions of the same file to identify changes, and record a log of who made what changes to files and when those changes were made.
- Git enables multiple users to collaboratively work on content for network automation projects.
- Automation controller provides a centralized location from which you can run your network automation playbooks and review the results of those 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, automation execution environment, project, and playbook to use to run your Ansible code on the managed nodes.
- To get more verbose output from the `ansible-navigator run` command to help troubleshoot problems, you can specify the `-v` option one or more times.
- Ansible Lint is a tool that uses rules to detect issues in a playbook. You can run the `ansible-lint` package inside an automation execution environment by running the `ansible-navigator lint` command or by configuring VS Code to use it.
- The `ansible-navigator` command can produce playbook artifact files that store information about playbook runs, which can be reviewed by using the `ansible-navigator replay` command.
- Three common reasons why network automation tasks fail are connectivity issues, authentication issues, and timeout issues.
- You can use the platform-specific `*_ping` modules to verify connectivity to managed nodes.
- To isolate an error, you can run a playbook with the `ansible-navigator` command, using its `--limit` option to limit the run to specific managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 4

Managing Variables and Facts

Goal

Write playbooks that use variables to simplify management of the playbook and facts to reference information about managed nodes.

Objectives

- Create and reference variables that affect particular managed nodes or groups, the play, or the global environment, and describe how variable precedence works.
- Reference data about managed nodes using Ansible facts.
- Create and use surveys in automation controller to provide information to Ansible Playbook jobs.
- Format, parse, manipulate, and define the values of variables by using filters.

Sections

- Managing Variables (and Guided Exercise)
- Managing Facts (and Guided Exercise)
- Using Surveys to Set Variables (and Guided Exercise)
- Transforming Data with Filters (and Guided Exercise)

Lab

- Managing Variables and Facts

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Managing Variables

Objectives

- Create and reference variables that affect particular managed nodes or groups, the play, or the global environment, and describe how variable precedence works.

Ansible Variables

Ansible supports the use of variables to store values that can then be reused throughout files in an Ansible project. Using Ansible variables simplifies the creation and maintenance of a project and reduces errors.

Examples of values that Ansible variables might contain include:

- Users to connect to managed nodes
- Networks to define in a project
- A network port to use
- A firewall zone

Naming Variables

Variable names must start with a letter, and they can only contain letters, numbers, and underscores.

The following table illustrates the difference between invalid and valid variable names.

Invalid variable names	Valid variable names
`vlan id`	`vlan_id`
`device.family`	`device_family`
`firewall$zone#1`	`firewall_zone_1` `firewall_zone1`
`2nd-banner-message`	`banner_message_2` `banner_message2`

Variable Precedence

You can define variables in various places in an Ansible project. If you set a variable using the same name in two places, and those settings have different values, Ansible uses the value that has the highest *precedence*.

The following simplified list shows ways to define an Ansible variable, ordered from the lowest to the highest precedence:

- Group variables defined in the inventory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Group variables defined in files in a subdirectory named `group_vars` that is in the same directory as the inventory or the playbook.
- Host variables defined in the inventory.
- Host variables defined in files in a subdirectory named `host_vars` that is in the same directory as the inventory or the playbook.
- Facts gathered from the managed nodes.
- Play variables in the playbook (the `vars` and `vars_files` sections of a play).
- Task variables (set by a `vars` section on tasks or included with the `ansible.builtin.include_tasks` module).
- Extra variables set by using the `--extra-vars` (or `-e`) option of the `ansible-navigator run` command. These variables always have the highest precedence.

For example, a variable that is set to affect the `all` inventory group of managed nodes is overridden by a variable that has the same name and is set to affect a single managed node.

One recommended practice is to choose globally unique variable names, so that you do not have to consider precedence rules. However, sometimes you might want to use precedence to cause different managed nodes or managed node groups to get different settings from your defaults.

Precedence order is covered in the *Variable Precedence: Where Should I Put a Variable?* documentation at `https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable`.

Using Variables in Playbooks

After you have declared variables, you can use them in tasks. Place the variable name in double braces ({{ }}) to reference the variables. Ansible substitutes the variable with its value when the task is executed.

```
---
- name: Configuring hostname for IOS managed node
  hosts: iosxe.example.com
  gather_facts: false
  vars:
    router_hostname: rtr01.example.com
  tasks:
    # This line reads: Set the rtr01.example.com hostname
    - name: Set the {{ router_hostname }} hostname
      cisco.ios.ios_hostname:
        config:
          # This line sets the rtr01.example.com hostname for IOS
          hostname: "{{ router_hostname }}"
        state: merged
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

When you start a value using the `{{ variable }}` expression, it is mandatory to use quotes. This prevents Ansible from interpreting the variable reference as starting a YAML dictionary. The following message appears if quotes are missing:

```
cisco.ios.ios_interfaces:
  config:
     name: {{ interface_name }}
            ^ here
We could be wrong, but this one looks like it might be an issue with
missing quotes.  Always quote template expression brackets when they
start a value. For instance:

    with_items:
      - {{ foo }}

Should be written as:

    with_items:
      - "{{ foo }}"
```

Capturing Task Output with Registered Variables

You can use the `register` keyword to define a temporary variable (with a lifetime limited to the run of your playbook) that stores the output of a task. The output is saved into a variable that can be used later in the playbook for either debugging or other purposes, such as applying a particular configuration setting based on the value of the variable.

The following tasks in a playbook show how to capture the output of the first task and use the other task to show the value of the variable, possibly for debugging purposes:

```
- name: Gather IOS vlan
  cisco.ios.ios_vlans:
    state: gathered
  register: vlans_data

- name: Show the vlan data
  ansible.builtin.debug:
    var: vlans_data
```

When you run the play, the `ansible.builtin.debug` module dumps the value of the `vlans_data` registered variable to the terminal:

```
...output omitted...

TASK [Show the vlan data] *******************************************************
ok: [iosxe.lab.example.com] => {
    "vlans_data": {
        "changed": false,
        "failed": false,
        "gathered": [
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
            {
                "mtu": 1500,
                "name": "internal",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 10
            },
            {
                "mtu": 1500,
                "name": "dmz",
                "shutdown": "enabled",
                "state": "active",
                "vlan_id": 20
            }
        ]
    }
}
...output omitted...
```

Using Dictionaries as Variables

Instead of assigning configuration data that associates the same element to multiple variables, you can use *dictionaries*. A dictionary is a data structure containing key-value pairs, where the values can also be dictionaries.

For example, consider the following snippet:

```
# Hypervisors
vlan1_id: 1000
vlan1_mtu:9000
vlan1_name: physical-servers

# Virtual machines
vlan2_id: 2000
vlan2_mtu:1500
vlan2_name: virtual-servers

# Servers in DMZ
vlan3_id: 3000
vlan3_mtu:1500
vlan3_name: dmz-servers
```

This could be rewritten as a dictionary called `vlan`:

```
vlan:
  hypervisors:
    name: physical-servers
    mtu: 9000
    id: 1000
  vms:
    name: virtual-servers
    mtu: 1500
    id: 2000
  dmz:
```

```
    name: dmz
    mtu: 1500
    id: 3000
```

You can then use the following variables to access `vlan` data:

```
- name: Returns 'dmz'
  ansible.builtin.debug:
    var: vlan['dmz']['name']

- name: Returns '9000'
  ansible.builtin.debug:
    var: vlan['hypervisors']['mtu']

- name: Returns '2000'
  ansible.builtin.debug:
    var: vlan['vms']['id']
```

Note

The `var` parameter in the `ansible.builtin.debug` module runs in Jinja2 context and implicitly uses double braces. Specify the variable without wrapping the variable in double braces.

Special Variables

A number of variables are available that you can use to change how Ansible connects to a managed node listed in the inventory. Some of these are most useful as variables specific to one managed node, but others might be relevant to all managed nodes in a group in the inventory.

`ansible_connection`

The `ansible_connection` variable defines the connection plug-in to use to access the managed node.

The following table shows some possible values for the `ansible_connection` variable.

Value	Description
`ansible.netcommon.network_cli`	Command-line connection over the Secure Shell (SSH) network protocol.
`ansible.netcommon.netconf`	Using eXtensible Markup Language (XML) over the SSH protocol.
`ansible.netcommon.httpapi`	Connection over the Hypertext Transfer Protocol Secure (HTTPS) protocol.

`ansible_network_os`

The `ansible_network_os` variable defines the managed node operating system. If you do not specify the value for the variable, Ansible tries to automatically detect the value to use. As a best practice, always define the `ansible_network_os` variable for managed network nodes.

The following table shows some possible values for the `ansible_network_os` variable.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Value	Managed Node OS
`arista.eos.eos`	Arista EOS
`cisco.ios.ios`	Cisco IOS Cisco IOS-XE
`cisco.iosxr.iosxr`	Cisco IOS-XR
`cisco.nxos.nxos`	Cisco NX-OS
`junipernetworks.junos.junos`	Juniper Junos

For more information on the `ansible_connection` and `ansible_network_os` variables, see `https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform`.

`ansible_host`
: The `ansible_host` variable defines the actual IP address or fully qualified domain name to use when connecting to the managed node, instead of using the name from the inventory file. By default, this variable has the same value as the inventory hostname.

`ansible_user`
: The `ansible_user` variable defines the user that Ansible uses to connect to the managed node. The default Ansible behavior is to connect to the managed node as the user running the Ansible Playbook on the control node.

`ansible_become`
: Set this variable to `true` to enable privilege escalation mode. The privilege escalation method is supported only for the `ansible.netcommon.network_cli` and `ansible.netcommon.httpapi` Ansible connections.

`ansible_method`
: When the privilege escalation mode is supported for a managed node, set this variable to `enable`.

Separating Variables from Your Inventory

If you are managing your inventory as a static file, then you can define variables in the inventory, either for a specific managed node or for a group of managed nodes. 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. A better approach is to move variable definitions from the inventory file into separate variable files.

Assigning Variables to Inventory Groups

You can create a subdirectory named `group_vars` in the same directory as your Ansible project. In the `group_vars` subdirectory, create a file for each group in your inventory and then add variables specific to that group of managed nodes:

```
[user@host project]$ tree -F group_vars/
group_vars/
├── arista.yml
├── ios.yml
└── junos.yml
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The above organizational structure makes it easier to locate configuration variables for any of the `arista`, `ios`, or `junos` inventory groups. That structure is sufficient when each inventory group does not contain too many variable definitions. 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 inventory group of managed nodes under the `group_vars` subdirectory, such as the `groups_vars/ios/` subdirectory, in your project directory. Ansible parses any YAML or JSON files in these subdirectories and associates the variables with an inventory group based on the parent directory.

For example, you might initially define the `ansible_network_os` and `ansible_connection` variables for the `ios` inventory group in the inventory file:

```
[ios:vars]
ansible_network_os = cisco.ios.ios
ansible_connection = ansible.netcommon.network_cli
```

Instead of defining those variables in your inventory file, create a YAML file in the `group_vars/ios` path in your project directory and the `ansible_network_os` and `ansible_connection` variables in the new file:

```
ansible_network_os: cisco.ios.ios
ansible_connection: ansible.netcommon.network_cli
```

If you use this organizational structure for variables, then place group variables with a common theme into the same file, and use a file name that indicates that common theme. A project organized according to this convention might appear as follows:

```
[user@host project]$ tree -F group_vars/
group_vars/
├── all/
│   └── common.yml
├── arista/
│   ├── acl.yml
│   ├── connection.yml
│   └── vlan.yml
├── ios/
│   ├── acl.yml
│   ├── connection.yml
│   ├── security.yml
│   └── vlan.yml
└── junos/
    ├── acl.yml
    ├── connection.yml
    ├── snmp.yml
    └── vlan.yml
```

With this organizational structure for a project, you can quickly see the variables that are defined for each group of managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 project easier to understand and maintain.

Assigning Variables to Inventory Hosts

Likewise with inventory groups, you can create a subdirectory named `host_vars` in your Ansible project directory. Then, create a file named after an inventory managed node with the variable definitions for that managed node.

Variables defined for a managed node have a higher precedence than variables defined for the managed node's inventory group.

The `group_vars` and `host_vars` subdirectories can exist not only in your project directory, which is where your playbook is located, but in the directory where your inventory file is located. These two locations might not be the same directory.

For the `group_vars` and `host_vars` subdirectories relative to the inventory, the following is true:

- If you have `group_vars` and `host_vars` subdirectories in the same directory as your playbook, then Ansible automatically includes those variables for managed nodes and groups of managed nodes.
- If 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.
- If you use an inventory directory in your project directory, and that inventory file contains multiple inventory files, then Ansible includes the `group_vars` and `host_vars` subdirectories of your inventory directory.

> **Important**
> The variables that Ansible includes from the playbook directory override the variables from the inventory directory if there is a conflict.

Overriding Variables from the Command Line

Inventory variables are overridden by variables set in a playbook, but both kinds of variables can be overridden by arguments passed to the `ansible-navigator run` command on the command line. Variables set on the command line are called *extra variables*.

Extra variables can be useful when you need to override the defined value for a variable for a one-off run of a playbook. For example:

```
[user@host ~]$ ansible-navigator run snmp_configuration.yml -e state=merged
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

Using Variables - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Variable Precedence: Where Should I Put a Variable? - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

Settings by Platform - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform

Ansible Documentation: Special Variables - Ansible Documentation
https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Making Flexible Playbooks by Using Variables

Rewrite an Ansible Playbook to customize and configure settings for specific managed nodes by using variables to set parameters that are passed to modules used by tasks in its plays.

Outcomes

- Create variable files for each inventory group of managed nodes under the `groups_var` subdirectory in the project directory.
- Separate variables from your inventory to the new variable files.
- Create new variables to be used in the tasks of the current playbooks.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

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

Instructions

You want to improve the `manage-variables` project to make it easier to find group-related variables in the project. To do this, you need to relocate inventory variables and create new variables to use in the current playbooks.

1. Open the `/home/student/manage-variables` directory in VS Code. Locate the playbooks and the inventory groups in the inventory file for the `manage-variables` project.

   1.1. Open VS Code and navigate to **File** > **Open Folder**.

   1.2. Navigate to **Home** > **manage-variables** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Notice that the project has two playbooks to configure the Domain Name System (DNS) service and two playbooks to configure the Simple Network Management Protocol (SNMP):

   - The `ios_dns.yml` playbook configures DNS on the IOS managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- The `junos_dns.yml` playbook configures DNS on the Juniper Junos managed nodes.
- The `ios_snmp.yml` playbook configures SNMP on the IOS managed nodes.
- The `junos_snmp.yml` playbooks configures SNMP on the Juniper Junos managed nodes.

1.4. Click the inventory file. The inventory file has the `ios` and `junos` inventory groups and group variables.

```
[ios]
iosxe[1:2].lab.example.com

[junos]
junos[1:2].lab.example.com

[ios:vars]
ansible_user=student
ansible_ssh_private_key_file=~/.ssh/lab_rsa
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios

[junos:vars]
ansible_user=student
ansible_ssh_private_key_file=~/.ssh/lab_rsa
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

2. Create the directory structure to hold the variables for the inventory groups.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/manage-variables` directory in a GNOME terminal:

```
[student@workstation ~]$ cd manage-variables
[student@workstation manage-variables]$
```

2.2. Create the `group_vars/all`, `group_vars/ios`, and `group_vars/junos` directories to hold the variable files for the managed nodes:

```
[student@workstation manage-variables]$ mkdir -p group_vars/all
[student@workstation manage-variables]$ mkdir -p group_vars/ios
[student@workstation manage-variables]$ mkdir -p group_vars/junos
```

2.3. *(Optional)* Verify the directories in the `group_vars` directory by using the `tree` command:

```
[student@workstation manage-variables]$ tree group_vars/
group_vars/
├── all
├── ios
└── junos

3 directories, 0 files
```

3. Create the following files under the `groups_vars` directory to store variables from the inventory file. Update the inventory file by removing the variables already defined.

   - The `all/common.yml` file to store the common variables for the managed nodes.
   - The `ios/connection.yml` file to store the `ios` inventory group variables.
   - The `junos/connection.yml` file to store the `junos` inventory group variables.

   3.1. Notice that the inventory file holds the following common variables for the `ios` and `junos` inventory groups:

Variable	Value
`ansible_user`	`student`
`ansible_ssh_private_key_file`	`~/.ssh/lab_rsa`

   3.2. Create the `group_vars/all/common.yml` file with the following content:

```
---
ansible_user: student
ansible_ssh_private_key_file: ~/.ssh/lab_rsa
```

   3.3. Notice that the inventory file holds the following variables used for connecting to the managed nodes:

Variable	Value for the `ios` inventory group
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`cisco.ios.ios`

Variable	Value for the `junos` inventory group
`ansible_connection`	`ansible.netcommon.netconf`
`ansible_network_os`	`junipernetworks.junos.junos`

   3.4. Create the `group_vars/ios/connection.yml` file with the following content:

```
---
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
```

   3.5. Create the `group_vars/junos/connection.yml` file with the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
ansible_connection: ansible.netcommon.netconf
ansible_network_os: junipernetworks.junos.junos
```

3.6. *(Optional)* Verify the directories and files in the `group_vars` directory by using the `tree` command:

```
[student@workstation manage-variables]$ tree group_vars/
group_vars
├── all
│   └── common.yml
├── ios
│   └── connection.yml
└── junos
    └── connection.yml

3 directories, 3 files
```

3.7. Update the inventory file by removing the IOS and Juniper Junos variables. The resulting inventory file should consist of the following content:

```
[ios]
iosxe[1:2].lab.example.com

[junos]
junos[1:2].lab.example.com
```

4. Make the `ios_dns.yml` and `junos_dns.yml` playbooks more flexible by using variables in their `Configure DNS settings` task.

4.1. In VS Code, click the `ios_dns.yml` playbook. Pay attention to the parameters used by the `cisco.ios.ios_system` module in the `Configure DNS settings` task:

```
...output omitted...
    - name: Configure DNS settings
      cisco.ios.ios_system:
        domain_search:
          - lab.example.com
          - example.com
        name_servers:
          - 172.25.250.220
```

4.2. Click the `junos_dns.yml` playbook. Pay attention to the parameters used by the `junipernetworks.junos.junos_system` module in the `Configure DNS settings` task:

```
...output omitted...
    - name: Configure DNS settings
      junipernetworks.junos.junos_system:
        domain_search:
          - lab.example.com
          - example.com
        name_servers:
          - 172.25.250.220
```

4.3. Create the `group_vars/ios/dns.yml` and `group_vars/junos/dns.yml` variables files with the following content:

```
---
dns_domains:
 - lab.example.com
 - example.com
dns_name_servers:
 - 172.25.250.220
```

Note

Because the new `dns_domains` and `dns_name_servers` variables are the same for both `ios` and `junos` inventory groups, another option is to define the variables in a file in the `group_vars/all/` directory.

Both options are valid. The choice depends on the organization that you want to implement in your company.

4.4. Edit the `ios_dns.yml` and `junos_dns.yml` playbooks to introduce the defined variables. Place the variable names in double braces and use quotes for both variable expressions.

- The `Configure DNS settings` task for the `ios_dns.yml` playbook must have the following content:

```
- name: Configure DNS settings
  cisco.ios.ios_system:
    domain_search: "{{ dns_domains }}"
    name_servers: "{{ dns_name_servers }}"
```

- The `Configure DNS settings` task for the `junos_dns.yml` playbook must have the following content:

```
- name: Configure DNS settings
  junipernetworks.junos.junos_system:
    domain_search: "{{ dns_domains }}"
    name_servers: "{{ dns_name_servers }}"
```

4.5. *(Optional)* Verify the directories and files in the `group_vars` directory by using the `tree` command:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation manage-variables]$ tree group_vars/
group_vars
├── all
│   └── common.yml
├── ios
│   ├── connection.yml
│   └── dns.yml
└── junos
    ├── connection.yml
    └── dns.yml

3 directories, 5 files
```

5. Make the `ios_snmp.yml` and `junos_snmp.yml` playbooks more flexible by using variables in their `Configure SNMP settings` task.

5.1. In VS Code, click the `ios_snmp.yml` playbook. Pay attention to the parameters used by the `cisco.ios.ios_snmp_server` module in the `Configure SNMP settings` task:

```
...output omitted...
    - name: Configure SNMP settings
      cisco.ios.ios_snmp_server:
        state: merged
        config:
          location: 'Raleigh, NC'
          contact: 'Network Engineering | neteng@company.com'
          communities:
            - name: rocommunity2n4g!
              ro: true
            - name: rwcommunityd7g$v
              rw: true
```

5.2. Click the `junos_dns.yml` playbook. Pay attention to the parameters used by the `junipernetworks.junos.junos_system` module in the `Configure DNS settings` task:

```
...output omitted...
    - name: Configure SNMP settings
      junipernetworks.junos.junos_snmp_server:
        state: merged
        config:
          location: 'Raleigh, NC'
          contact: 'Network Engineering | neteng@company.com'
          communities:
            - name: rocommunity2n4g!
              authorization: read-only
            - name: rwcommunityd7g$v
              authorization: read-write
```

5.3. Create the `group_vars/ios/snmp.yml` variables file with the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
snmp_state: merged
snmp_location: 'Raleigh, NC'
snmp_contact: 'Network Engineering | neteng@company.com'
snmp_communities:
  - name: rocommunity2n4g!
    ro: true
  - name: rwcommunityd7g$v
    rw: true
```

5.4. Create the `group_vars/junos/snmp.yml` variables file with the following content:

```
---
snmp_state: merged
snmp_location: 'Raleigh, NC'
snmp_contact: 'Network Engineering | neteng@company.com'
snmp_communities:
  - name: rocommunity2n4g!
    authorization: read-only
  - name: rwcommunityd7g$v
    authorization: read-write
```

5.5. Edit the `ios_snmp.yml` and `junos_snmp.yml` playbooks to introduce the defined variables. Place the variable names in double braces and use quotes for both variable expressions.

- The `Configure SNMP settings` task for the `ios_snmp.yml` must have the following content:

```
  - name: Configure SNMP settings
    cisco.ios.ios_snmp_server:
      state: "{{ snmp_state }}"
      config:
        location: "{{ snmp_location }}"
        contact: "{{ snmp_contact }}"
        communities: "{{ snmp_communities }}"
```

- The `Configure SNMP settings` task for the `junos_snmp.yml` must have the following content:

```
  - name: Configure SNMP settings
    junipernetworks.junos.junos_snmp_server:
      state: "{{ snmp_state }}"
      config:
        location: "{{ snmp_location }}"
        contact: "{{ snmp_contact }}"
        communities: "{{ snmp_communities }}"
```

5.6. *(Optional)* Verify the directories and files in the `group_vars` directory by using the `tree` command:

```
[student@workstation manage-variables]$ tree group_vars/
group_vars/
├── all
│   └── common.yml
├── ios
│   ├── connection.yml
│   ├── dns.yml
│   └── snmp.yml
└── junos
    ├── connection.yml
    ├── dns.yml
    └── snmp.yml

3 directories, 7 files
```

6. The rewritten playbooks perform the DNS and SNMP configuration tasks on the managed nodes by using variables defined in files under the `groups_var` directory. The group variable files have meaningful names, making it easier to find particular variables.

Verify that the `Configure DNS settings` and `Configure SNMP settings` tasks run successfully when you run the `ios_dns.yml`, `junos_dns.yml`, `ios_snmp.yml`, and `junos_snmp.yml` playbooks.

6.1. Run the `ios_dns.yml` playbook. The playbook runs successfully.

```
[student@workstation manage-variables]$ ansible-navigator run ios_dns.yml

PLAY [Configure DNS settings on IOS managed nodes] ******************************

TASK [Configure DNS settings] ****************************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP ***********************************************************************
iosxe1.lab.example.com    : ok=1   changed=1   unreachable=0   failed=0      ...
iosxe2.lab.example.com    : ok=1   changed=1   unreachable=0   failed=0      ...
```

6.2. Run the `junos_dns.yml` playbook. The playbook runs successfully.

```
[student@workstation manage-variables]$ ansible-navigator run junos_dns.yml

PLAY [Configure DNS settings on Juniper managed nodes] **************************

TASK [Enable netconf service on port 830] ****************************************
changed: [junos2.lab.example.com]
changed: [junos1.lab.example.com]

TASK [Configure DNS settings] ****************************************************
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
junos1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0       ...
junos2.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0       ...
```

6.3. Run the `ios_snmp.yml` playbook. The playbook runs successfully.

```
[student@workstation manage-variables]$ ansible-navigator run ios_snmp.yml

PLAY [Configure SNMP on Cisco IOS XE managed nodes] ******************************

TASK [Configure SNMP settings] ***************************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0       ...
iosxe2.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0       ...
```

6.4. Run the `junos_snmp.yml` playbook. The playbook runs successfully.

```
[student@workstation manage-variables]$ ansible-navigator run junos_snmp.yml

PLAY [Configure SNMP on Juniper managed nodes] ***********************************

TASK [Enable netconf service on port 830] ****************************************
ok: [junos2.lab.example.com]
ok: [junos1.lab.example.com]

TASK [Configure SNMP settings] ***************************************************
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

PLAY RECAP *********************************************************************
junos1.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0       ...
junos2.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0       ...
```

6.5. *(Optional)* The playbooks in the `aux` directory in the current project enable you to verify the DNS and SNMP settings on the IOS and Juniper Junos managed nodes.

- Run the `aux/ios_dns_verify.yml` auxiliary playbook to verify the DNS configuration on the IOS managed nodes.
- Run the `aux/junos_dns_verify.yml` auxiliary playbook to verify the DNS configuration on the Juniper Junos managed nodes.
- Run the `aux/ios_snmp_verify.yml` auxiliary playbook to verify the SNMP configuration on the IOS managed nodes.
- Run the `aux/junos_snmp_verify.yml` auxiliary playbook to verify the SNMP configuration on Juniper Junos managed nodes.

7. Close the `/home/student/manage-variables` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

7.1. Click **File** > **Close Folder** to close the `/home/student/manage-variables` directory.

7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation manage-variables]$ cd
```

Finish

On the `workstation` machine, 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-variables
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Managing Facts

Objectives

- Reference data about managed nodes using Ansible facts.

Ansible Facts

Ansible *facts* are variables that are automatically discovered by Ansible on a managed node.

Facts contain host specific information that can be used just like regular variables in plays, or any other statement that depends on a value collected from a managed node.

Some facts gathered for a managed node might include:

- Hostname
- Serial number
- Network operating system version
- Network interface names
- Network interface IP addresses
- Number of CPUs
- Available or free memory
- Size and free space of storage devices

Facts are a convenient way to retrieve the state of a managed node and to determine what action to take based on that state.

For example, your play might perform one of the following actions:

- Restart a managed node based on the value of a fact that was gathered, such as the state of a particular interface.
- Detect and resolve a configuration issue based on the value of a fact.
- Customize a configuration based on the network OS version that is reported by a fact.

Default Fact Gathering

In previous versions, Ansible was unable to gather facts for managed network nodes using the default fact gathering method of setting the `gather_facts` keyword to `true`.

In Ansible version 2.9 and later, if you set the `gather_facts` keyword to `true`, then Ansible tries to detect the network platform and use the appropriate `*_facts` module to gather facts.

The default setting is to gather a minimal subset of facts. This is equivalent to setting the `gather_subset` parameter to `min`.

To view a minimal subset of facts for a specific network platform, create a short playbook with a play that gathers facts by setting the `gather_facts` keyword to `true`. Then use the `ansible.builtin.debug` module to display the value of the `ansible_facts` variable.

```
---
- name: Gather and display default facts for IOS managed nodes
  hosts: ios
  gather_facts: true
  tasks:
    - name: Display facts
      ansible.builtin.debug:
        var: ansible_facts
```

The following play gathers and displays the equivalent set of facts as the previous play:

```
---
- name: Gather and display default facts for IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - min

    - name: Display facts
      ansible.builtin.debug:
        var: ansible_facts
```

Configurable Fact Gathering

Use `*_facts` resource modules to gather facts for your managed nodes. For example, for Cisco IOS or IOS XE managed nodes, use the `cisco.ios.ios_facts` resource module. For Juniper Junos managed nodes, use the `junipernetworks.junos.junos_facts` resource module.

View the collections documentation for each network platform at https://docs.ansible.com/ansible/latest/collections/ to determine which modules to use.

To view all available facts for a specific network platform, create a short playbook that gathers all facts by setting the `gather_subset` parameter to `all`. Then use the `ansible.builtin.debug` module to display the value of the `ansible_facts` variable.

```
---
- name: Gather and display all facts for IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
         gather_subset:
            - all

    - name: Display facts
      ansible.builtin.debug:
        var: ansible_facts
```

When you run the playbook, the facts are displayed in the output:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

PLAY [Gather and display facts for IOS managed nodes] *************************

TASK [Gather IOS facts] ********************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Display facts] ***********************************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_facts": {
        "net_all_ipv4_addresses": [
            "172.25.250.20"
        ],
        "net_all_ipv6_addresses": [],
        "net_api": "cliconf",
        "net_config": "Building configuration...\n\nCurrent configuration : 6062
 bytes\n!\n! Last configuration change at 15:55:19 UTC Thu Jul 6 2023\n!\nversion
 17.6\nservice timestamps debug datetime msec\nse
...output omitted...
```

The playbook displays facts in JSON format as a dictionary of variables. You can browse the output to see what facts are gathered, and to find facts that you might want to use in your plays.

The following table shows some facts that might be gathered from a managed node and which might be useful in a playbook:

Examples of Ansible Facts

Fact	Variable
Hostname	`ansible_facts['net_hostname']`
Serial Number	`ansible_facts['net_serialnum']`
Network OS	`ansible_facts['net_system']`
Network OS Version	`ansible_facts['net_version']`
IPv4 addresses	`ansible_facts['net_all_ipv4_addresses']`
Current Active Configuration	`ansible_facts['net_config']`

Note

One of two syntaxes can be used to retrieve the value of a fact, as in the following examples:

- `ansible_facts['net_hostname']` can also be written as `ansible_facts.net_hostname`
- `ansible_facts['net_config']` can also be written as `ansible_facts.net_config`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Registering Facts to a Variable

You can also register the output of the `*_facts` resource modules to a variable. Because facts are stored in the registered variable, you need to reference the facts using the registered variable's namespace, as in the following example:

```
---
- name: Gather and display facts for IOS managed nodes
  hosts: ios
  gather_facts: true
  tasks:
   - name: Gather IOS facts
     cisco.ios.ios_facts:
       gather_subset:
         - all
     register: iosfacts

   - name: Display a fact
     ansible.builtin.debug:
       var: iosfacts['ansible_facts']['ansible_net_hostname']
```

Note

Facts stored in a registered variable are referenced using that variable's namespace, and the individual facts are prefaced with the string `ansible_`.

For example, the fact `ansible_facts['net_hostname']` registered under the variable `iosfacts` is named `iosfacts['ansible_facts']['ansible_net_hostname']`.

Because gathered facts are automatically added to the `ansible_facts` variable, it is simpler and more scalable to reference them using the `ansible_facts` namespace.

When a fact is used in a playbook, Ansible dynamically substitutes the variable name for the fact with the corresponding value:

```
---
- name: Display specific facts for managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all

    - name: Display hostname and IPv4 addresses
      ansible.builtin.debug:
        msg: >-
          The IPv4 addresses of interfaces on
          {{ ansible_facts['net_hostname'] }} are
          {{ ansible_facts['net_all_ipv4_addresses'] }}
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The following output shows how Ansible was able to query the managed node and dynamically use the information to update the variable:

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

PLAY [Display specific facts for managed nodes] ******************************

TASK [Gather IOS facts] *******************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Display hostname and IPv4 addresses] ************************************
ok: [iosxe2.lab.example.com] => {
    "msg": "The IPv4 addresses of interfaces on  iosxe2.lab.example.com are
 ['172.25.250.21']"
}
ok: [iosxe1.lab.example.com] => {
    "msg": "The IPv4 addresses of interfaces on  iosxe1.lab.example.com are
 ['172.25.250.20']"
}

PLAY RECAP ********************************************************************
iosxe1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
```

Gather facts from different network platforms by using the appropriate facts module for each platform.

For example, the following playbook uses a different module in each task to gather facts based on the value of the `network_os` fact:

```
---
- name: Gather facts from different network platforms
  hosts: all
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all
      when: ansible_network_os == 'cisco.ios.ios'

    - name: Gather Junos facts
      junipernetworks.junos.junos_facts:
        gather_subset:
          - all
      when: ansible_network_os == 'junipernetworks.junos.junos'
```

Turning off Fact Gathering

You might not want to gather facts for your play for the following reasons:

- You might not be using any facts and want to speed up the play, or reduce load caused by the play on the managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- The managed nodes cannot run the `ansible.builtin.setup` module for some reason, or cannot gather all facts using that module. This is often the case for managed network nodes.

To disable fact gathering for a play, set the `gather_facts` keyword to `false`. You should do this for managed network nodes, unless they support the `ansible.builtin.setup` module for all fact gathering.

Gathering a Subset of Facts

Configure fact modules to gather all facts for a managed node by setting the `gather_subset` parameter to `all`.

Configure fact modules to only gather a subset of facts by setting the `gather_facts` parameter to the subset you want to gather. For example, to only gather facts about the configuration, set the `gather_subset` parameter to `config`:

```
- name: Collect only configuration facts
  cisco.ios.ios_facts:
    gather_subset:
      - config
```

If you want to gather all facts except a certain subset, add an exclamation point (!) in front of the subset name. Add quotes around the string because in YAML the exclamation point cannot be used at the start of an unquoted string.

```
- name: Collect all facts except for the configuration subset
  cisco.ios.ios_facts:
    gather_subset:
      - "!config"
```

Refer to the documentation for each fact module to view possible values for the `gather_subset` parameter. For example, visit `https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_facts_module.html#ansible-collections-cisco-ios-ios-facts-module` to view possible values for the `gather_subset` parameter for the `cisco.ios.ios_facts` module.

Gathering Facts as Structured Resource Data

Use the `gather_network_resources` parameter to gather facts as structured network resource data.

You can gather all resource facts or a subset of resource facts, similar to the `gather_subset` parameter. For example, when using the `cisco.ios.ios_facts` module, you can set the `gather_network_resources` parameter to values such as the following:

- `l3_interfaces`
- `acls`
- `interfaces`
- `"!lldp_interfaces,!ospf_interfaces"`
- `static_routes`
- `"!vlans"`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note
Refer to the module documentation for the full list of valid values for the `gather_network_resources` parameter.

Display network resource facts by referencing the `ansible_network_resources` variable, as in the following example:

```
---
- name: Gather facts as resource data
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_network_resources:
          - all

    - name: Display facts
      ansible.builtin.debug:
        var: ansible_network_resources
```

Using Magic Variables

Ansible sets some special variables automatically. These *magic variables* can also be useful to get information specific to a particular managed node. Magic variable names are reserved, so you should not define variables with these names.

The following list describes four of the most useful magic variables:

`hostvars`
: The `hostvars` variable contains the variables for managed nodes, and can be used to get the values for another managed node's variables. It does not include the managed node's facts if they have not yet been gathered for that host.

`group_names`
: The `group_names` variable lists all groups that the current managed node is in.

`groups`
: The `groups` variable lists all groups and managed nodes in the inventory.

`inventory_hostname`
: The `inventory_hostname` variable contains the hostname for the current managed node as configured in the inventory. This might be different from the hostname reported by facts for various reasons.

The following task causes every host that runs the play to display the serial number of the `junos1.lab.example.com` managed node. It uses the `hostvars` magic variable to access the `ansible_facts['net_serialnum']` fact for that host.

This task works as long as facts were gathered for `junos1.lab.example.com` earlier in the play or by a preceding play in the playbook:

```
    - name: Print the serial number of junos1
      ansible.builtin.debug:
        var: hostvars['junos1.lab.example.com']['ansible_facts']['net_serialnum']
```

Several other magic variables are also available. For more information, see `https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html`.

References

Ansible Facts - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html#ansible-facts

FACTS_MODULES - Ansible Documentation
https://docs.ansible.com/ansible/latest/reference_appendices/config.html#facts-modules

Collection Index - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/

Special Variables - Ansible Documentation
https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Gathering Facts from Managed Network Devices

Gather Ansible facts from managed network nodes to create a report of the operational state of the network.

Outcomes

- Create and run playbooks to gather, display, and store facts from managed network nodes.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

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

Instructions

1. Open the `/home/student/manage-facts` directory in VS Code and create a playbook file.

   1.1. Open VS Code and navigate to **File** > **Open Folder**.

   1.2. Navigate to **Home** > **manage-facts** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Click **File** > **New Text File**.

   1.4. Click **Select a language** and then click **Ansible (ansible)**.

2. In VS Code, create a playbook in the `~/manage-facts` directory to gather the default set of facts from managed nodes. Save the file as `minfacts.yml`, and then run the playbook.

   2.1. In VS Code, create a playbook to gather all facts from managed nodes.
   Save the file as `minfacts.yml`.
   The completed playbook must consist of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Gather and display facts
  hosts: ios,junos
  gather_facts: true
  tasks:
    - name: Display facts
      ansible.builtin.debug:
        var: ansible_facts
```

2.2. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/manage-facts` directory in a GNOME terminal.

Use the `ansible-navigator run` command to run the `minfacts.yml` playbook:

```
[student@workstation ~] cd ~/manage-facts
[student@workstation manage-facts]$ ansible-navigator run minfacts.yml

PLAY [Gather and display facts] ************************************************

TASK [Gathering Facts] *********************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]
ok: [junos2.lab.example.com]
ok: [junos1.lab.example.com]

TASK [Display facts] ***********************************************************
ok: [junos1.lab.example.com] => {
    "ansible_facts": {
        "net_api": "netconf",
        "net_gather_network_resources": [],
        "net_gather_subset": [
            "default"
        ],
        "net_hostname": "junos1.lab.example.com",
        "net_model": "vmx",
        "net_python_version": "3.9.13",
        "net_serialnum": "VM646555B74D",
        "net_system": "junos",
        "net_version": "23.1R1.8",
        "network_resources": {}
    }
}
ok: [junos2.lab.example.com] => {
    "ansible_facts": {
        "net_api": "netconf",
...output omitted...

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
junos1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
junos2.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0 ...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3. Notice that in the previous step, the playbook only gathered a minimal set of facts for the managed nodes. In VS Code, create a playbook to gather all facts from managed nodes. Save the file as `allfacts.yml`, run the playbook, then analyze the results.

   3.1. In VS Code, create a playbook to gather all facts from managed nodes. Save the file as `allfacts.yml`.

> **Note**
> This playbook uses the `run_once` keyword in the first task.
>
> When you set the `run_once` keyword to `true`, the task executes on the first available managed node and, if successful, does not execute on subsequent managed nodes.
>
> Because this task creates a directory, the task does not need to run again if the directory is already created.

The completed playbook must consist of the following content:

```
---
- name: Gather and display facts for managed nodes
  hosts: ios,junos
  gather_facts: false
  tasks:
    - name: Create the facts directory if it does not exist
      ansible.builtin.file:
        path: facts
        state: directory
      run_once: true

    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all
      when: ansible_network_os == 'cisco.ios.ios'

    - name: Gather Junos facts
      junipernetworks.junos.junos_facts:
        gather_subset:
          - all
      when: ansible_network_os == 'junipernetworks.junos.junos'

    - name: Save facts to file
      ansible.builtin.copy:
        content: "{{ ansible_facts | to_nice_yaml }}"
        dest: "facts/{{ inventory_hostname }}.txt"
```

   3.2. Use the `ansible-navigator run` command to run the `allfacts.yml` playbook:

```
[student@workstation manage-facts]$ ansible-navigator run allfacts.yml

PLAY [Gather and display facts for managed nodes] ****************************
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
TASK [Create the facts directory if it does not exist] ************************
changed: [iosxe1.lab.example.com]

TASK [Gather IOS facts] ********************************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
ok: [iosxe1.lab.example.com]
ok: [iosxe2.lab.example.com]

TASK [Gather Junos facts] ******************************************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
ok: [junos2.lab.example.com]
ok: [junos1.lab.example.com]

TASK [Save facts to file] ******************************************************
changed: [junos2.lab.example.com]
changed: [iosxe2.lab.example.com]
changed: [iosxe1.lab.example.com]
changed: [junos1.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0 ...
junos1.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0 ...
junos2.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0 ...
```

3.3. Analyze the created facts files to verify that all facts were gathered for each managed node. Display the first five lines of each file:

```
[student@workstation manage-facts]$ head -n5 facts/iosxe1.lab.example.com.txt
net_all_ipv4_addresses:
- 172.25.250.20
net_all_ipv6_addresses: []
net_api: cliconf
net_config: "Building configuration...\n\nCurrent configuration : 6063 bytes\n!\n!
[student@workstation manage-facts]$ head -n5 facts/iosxe2.lab.example.com.txt
net_all_ipv4_addresses:
- 172.25.250.21
net_all_ipv6_addresses: []
net_api: cliconf
net_config: "Building configuration...\n\nCurrent configuration : 6063 bytes\n!\n!
[student@workstation manage-facts]$ head -n5 facts/junos1.lab.example.com.txt
net_api: netconf
net_config: "## Last changed: 2023-07-12 21:04:00 UTC\nversion 23.1R1.8;\nsystem
 {\n
    \   host-name junos1.lab.example.com;\n    root-authentication {\n
 encrypted-password
    \"$6$5TiPJaqE$929rhpem1O3HigTCPHZTuyg4wV.IeFK2ZFWBOQG9syw.AS50/
FEHTiXmzJ1sB5ZPppRKXilY8.gXQW4e8z.yP.\";\n
    \   }\n    login {\n        user student {\n            uid 2000;\n
 class
[student@workstation manage-facts]$ head -n5 facts/junos2.lab.example.com.txt
```

```
net_api: netconf
net_config: "## Last changed: 2023-07-12 16:21:42 UTC\nversion 23.1R1.8;\nsystem
 {\n
    \   host-name junos2.lab.example.com;\n    root-authentication {\n
 encrypted-password
    \"$6$5TiPJaqE$929rhpem1O3HigTCPHZTuyg4wV.IeFK2ZFWBOQG9syw.AS50/
FEHTiXmzJ1sB5ZPppRKXilY8.gXQW4e8z.yP.\";\n
    \   }\n    login {\n        user student {\n            uid 2000;\n
 class
```

4. In VS Code, create a playbook in the `~/manage-facts` directory to gather facts in resource module format using the `gather_network_resources` parameter.

   Save the file as `resourcefacts.yml`, and then run the playbook.

   4.1. In VS Code, create a playbook to gather facts in resource module format using the `gather_network_resources` parameter.

   Save the file as `resourcefacts.yml`.

   The completed playbook must consist of the following content:

```
---
- name: Gather and display facts for IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_network_resources:
          - all

    - name: Display facts
      ansible.builtin.debug:
        var: ansible_network_resources
```

   4.2. Use the `ansible-navigator run` command to run the `resourcefacts.yml` playbook:

```
[student@workstation manage-facts]$ ansible-navigator run resourcefacts.yml

PLAY [Gather and display facts for IOS managed nodes] *************************

TASK [Gather IOS facts] ********************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Display facts] ***********************************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_network_resources": {
        "acl_interfaces": [],
        "acls": [
            {
                "acls": [
                    {
                        "acl_type": "extended",
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
                        "name": "meraki-fqdn-dns"
                    }
                ],
                "afi": "ipv4"
            }
        ],
        "bgp_address_family": {},
        "bgp_global": {},
        "hostname": {
            "hostname": "iosxe1.lab.example.com"
        },
        "interfaces": [
...output omitted...
```

5. In VS Code, create a playbook in the `~/manage-facts` directory that creates a report of models, IOS versions, hostnames, VLANs, and Layer 3 interfaces on the IOS XE managed nodes. Save the file as `factsreport.yml`, and then run the playbook.

5.1. Create the `factsreport.yml` playbook with the following content:

```
---
- name: Gather and display facts report for IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - min
        gather_network_resources:
          - vlans
          - l3_interfaces

    - name: Save facts to file
      ansible.builtin.copy:
        content: |
          Hostname: {{ ansible_facts['net_hostname'] }}
          Model: {{ ansible_facts['net_model'] }}
          Version: {{ ansible_facts['net_version'] }}
          {{ ansible_network_resources | to_nice_yaml(indent=2) }}
        dest: "facts/{{ inventory_hostname }}.report"
```

5.2. Use the `ansible-navigator run` command to run the `factsreport.yml` playbook, and then view the generated report for the `iosxe1.lab.example.com` managed node.

```
[student@workstation manage-facts]$ ansible-navigator run factsreport.yml
...output omitted...
[student@workstation manage-facts]$ cat facts/iosxe1.lab.example.com.report
Hostname: iosxe1.lab.example.com
Model: C8000V
Version: 17.06.02
l3_interfaces:
- ipv4:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - address: 172.25.250.20/24
    name: GigabitEthernet1
  - name: GigabitEthernet2
  vlans:
  ...output omitted...
```

6. Click **File** > **Close Folder** in VS Code to close the `/home/student/manage-facts` directory.

7. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation manage-facts]$ cd
```

Finish

On the `workstation` machine, 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-facts
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Using Surveys to Set Variables

Objectives

- Create and use surveys in automation controller to provide information to Ansible Playbook jobs.

Managing Variables

You should write playbooks that you can reuse in similar situations. For example, you might start by writing a playbook that configures a single managed network node. Later, you might want to use that same playbook to configure additional managed network nodes with slightly different configurations. One way to apply different configuration settings is by using variables to specify those settings.

You can set variables with specific values in many different ways, and you can override them depending on how you set the variables. For example, a role or collection can provide a default value for a variable that you might override with the value set in the inventory that you are using, or in the playbook itself.

In automation controller, you can interactively set variables in two ways:

- Automation controller can prompt you to enter extra variables when you launch a job from a job template.
- You can create a survey that presents you with a form to help you set variables when you launch a job from a job template or a workflow template.

> **Important**
> Automation controller does not support playbooks that use `vars_prompt` to interactively set variables.

Defining Extra Variables

In automation controller, you can use a job template to directly set extra variables, using either of the following two methods:

- Define the variables in YAML or JSON format in the **Variables** field of the job template. This sets the specific values for extra variables used by the job template.
- Select **Prompt on launch** for the **Variables** field of the job template. This causes automation controller to prompt you to interactively specify and modify extra variables and their values when you use the job template to launch a job.

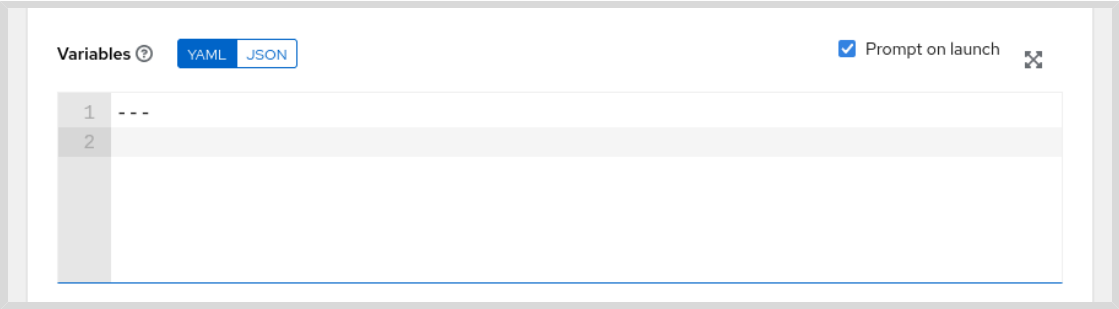


Figure 4.1: Prompt for variables when launching a job

Both methods are the same as using the `-e` or `--extra-vars` options for the `ansible-navigator` command. These variables have a high precedence. Except for variables set by a survey, these variable values override any other variables that might have the same name.

If you select **Prompt on launch** for the **Variables** field of a job template, then when you launch the job template, automation controller opens a dialog box where you can edit extra variables for the job.

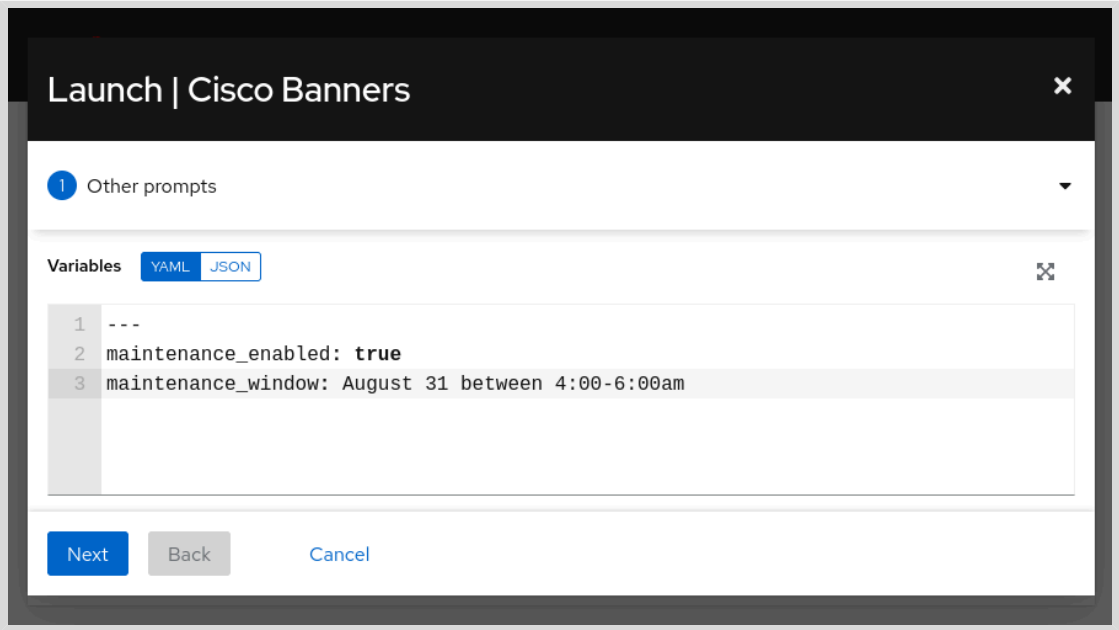


Figure 4.2: Adjusting extra variables on job launch

Important

If you relaunch a job that ran earlier (for example, by clicking the **Relaunch Job** icon for a job from the **Jobs** page), then you use the same values for its extra variables that the preceding job used. The extra variables for a job cannot be changed when you relaunch it.

Launch the job from its job template, not by relaunching a previous job, if you want to use different values for its extra variables.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Job Template Surveys

Modifying playbooks with extra variables can be challenging because you need to understand what variables are available and how the job template uses the variables. For example, misspelling the name of an extra variable does not generate an error, but it does not override the intended variable. Even if you wrote a playbook recently, you might not remember the names of all the variables used by the playbook by the time you launch the job template.

Rather than relying on remembering variable names, you can configure a job template or workflow template with a survey. When you launch a job that contains a survey, the web UI displays the survey to prompt you for information that automation controller uses to set values for extra variables. Surveys provide clearer guidance and control for setting extra variables.

Job template surveys offer several advantages over other ways to set extra variables. Users who launch the job do not need to have detailed knowledge about how extra variables work and do not need to know the names of the extra variables used by the playbook.

Important

A survey sets extra variables. The values set by surveys for variables take precedence over all other ways that value might be set.

This includes values set in the job template's **Variables** field or by using its **Prompt on launch** setting.

Managing Answers to Survey Questions

Survey questions and their descriptions can provide meaningful context to the user who launches the job template. Surveys allow you to configure how users respond to questions. If a question can only have a defined set of answers, then you might choose one of the multiple choice answer types. For example, a survey could prompt for a logging severity level. The multiple choice answers could include items such as `critical`, `warning`, and `info`.

Other survey questions might expect arbitrary text or numbers, and you can restrict the entered values by defining the minimum and the maximum number of characters that can be used. If desired, you can also provide default answers and decide which questions require an answer.

The responses to a survey question must use one of the following answer types:

Type	Description
`Text`	Single-line text.
`Textarea`	Multiline text.
`Password`	Data is treated as sensitive information.
`Multiple Choice (single select)`	A list of options from which only one option can be chosen as a response.
`Multiple Choice (multiple select)`	A list of options from which one or more options can be chosen as a response.
`Integer`	Integer number.
`Float`	Floating point decimal number.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Creating a Job Template Survey

When you create a job template, you cannot add a survey at the same time. You can only add a survey after you create the job template.

Use the following procedure to add a survey to an existing job template:

1. Navigate to **Resources** > **Templates** and click the desired job template.
2. Click the **Survey** tab and then click **Add** to open the **Add Question** page.
3. To add a question to the survey:
   a. In the **Question** field, enter the question to display. You can add an optional description in the **Description** field to provide more information about the question.
   b. In the **Answer variable name** field, enter the name of the extra variable that stores the response. You might need to consult the playbook to identify the exact variable name.
   c. From the **Answer type** list, select the answer type that you want to use for the response.
   d. If you use a multiple choice answer type, then define the list by entering the first choice in the **Multiple Choice Options** field, and then press `Enter` to display a new blank row. Continue until you have entered all desired choices.
   e. If you are not using a multiple choice answer type, then you can specify the maximum and minimum response lengths by using the **Maximum length** and **Minumum length** fields, respectively.
   f. You can also define a default value for the extra variable in the **Default answer** field. The survey starts with this value, but the user responding to the survey can choose a different value.
   g. Select **Required** to specify that a response is mandatory. Clear this field if a response is optional.
   h. Click **Save** to add the question to the survey. The survey now lists the new question.
4. Repeat the previous steps to add more questions to the survey. After adding all your questions, scroll to the top of the screen.
5. New surveys are disabled by default. To enable the survey, click **Survey Disabled** so that its label changes to **Survey Enabled**.
6. If you need to change the order of the survey questions, click **Edit Order** and drag the questions as required.

When you launch a job template that includes a survey, automation controller presents the survey before running the job. A question mark icon appears next to a question if you added a description for the survey question. Clicking the question mark icon displays the description.

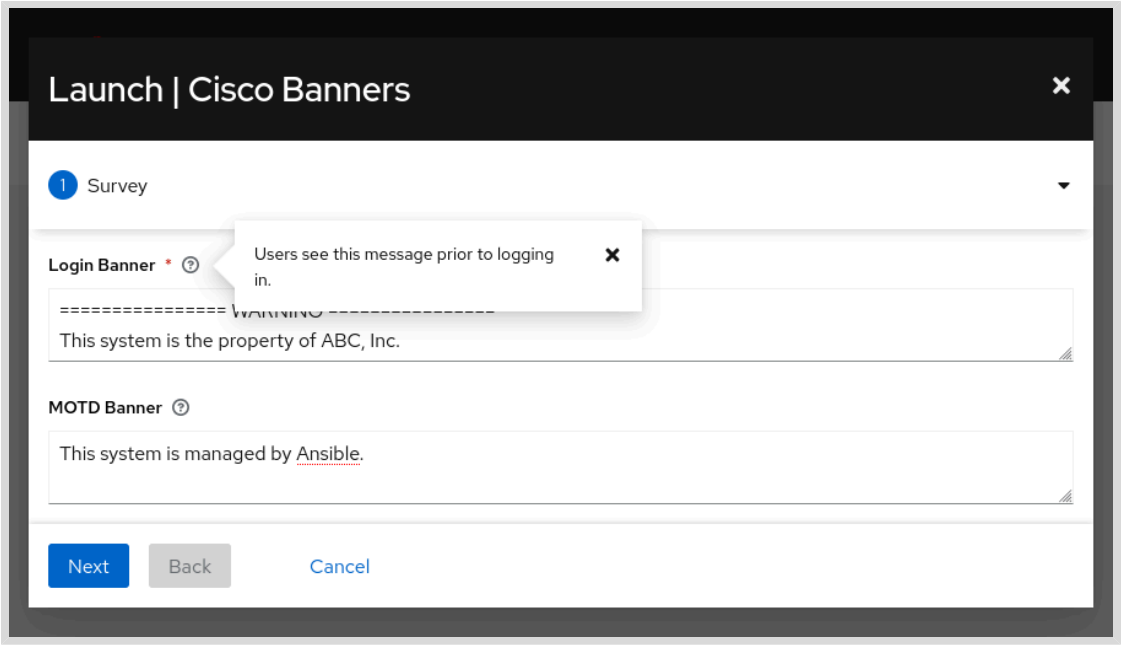


Figure 4.3: Survey prompt

After completing the survey, click **Next**. The **Preview** page displays variables and the values that the survey configures. If you are satisfied with your responses, then click **Launch** to start the job. Otherwise, click **Back** to change a response to a survey question.

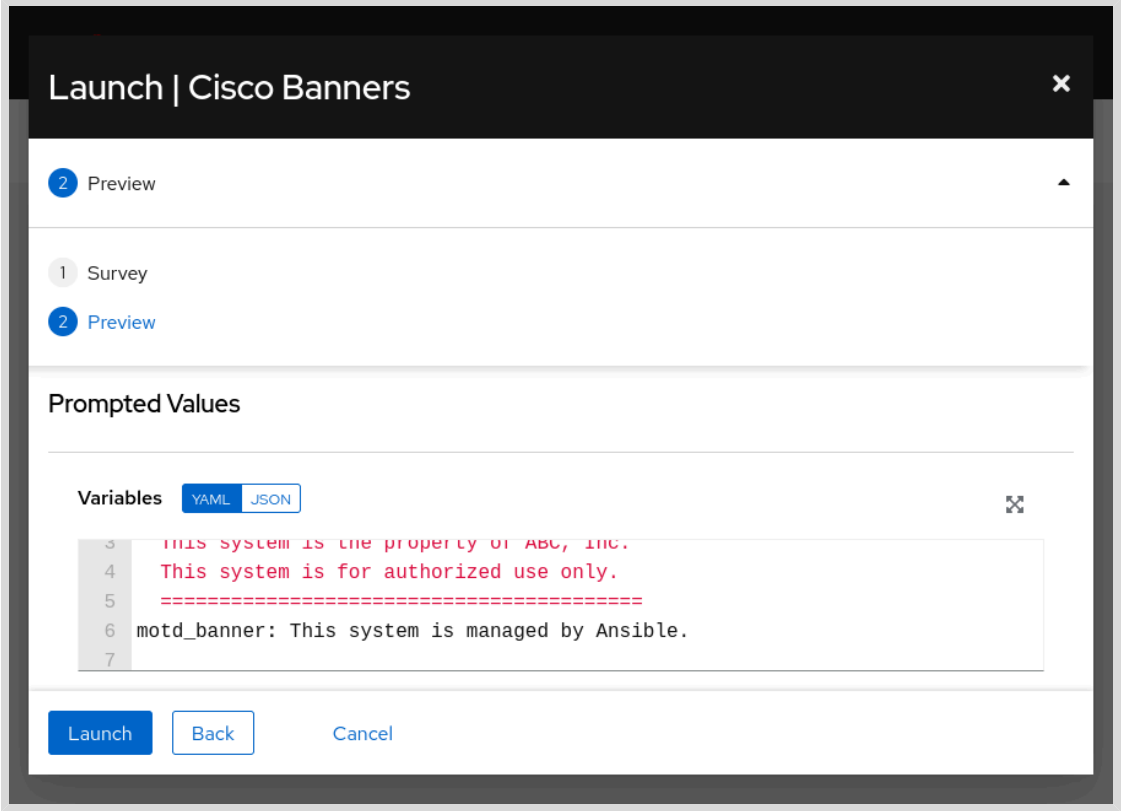


Figure 4.4: Preview page

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

Automation Controller User Guide - Surveys

https://docs.ansible.com/automation-controller/latest/html/userguide/job_templates.html#surveys

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring System Settings Using Surveys

Create a survey to provide values for variables that are used by the Ansible Playbook run by the job template, such as the device hostname and DNS settings.

Outcomes

- Create an automation controller job template.
- Add a survey to an existing job template.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available.

This command also creates two Git repositories and the following automation controller resources:

- The `Cisco IOS XE Auth` machine credential
- The `Inventories` project
- The `Common` inventory
- The `Surveys` project
- The `Check Cisco DNS` job template

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

Instructions

1. Check the existing hostname and DNS settings for the Cisco IOS managed nodes.

   1.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

   1.2. Click the main menu icon at the top left of the page. Navigate to **Resources** > **Templates** and click the **Launch Template** icon for the **Check Cisco DNS** job template.

   The job displays the following output:

```
...output omitted...
TASK [Display hostname] ***********************************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_hostname']": "iosxe1.lab.example.com"
}
ok: [iosxe2.lab.example.com] => {
```

```
    "ansible_facts['net_hostname']": "iosxe2.lab.example.com"
}

TASK [Display DNS lookup status] ************************************************
ok: [iosxe1.lab.example.com] => {
    "lookup_enabled_msg | trim": "DNS lookup enabled"
}
ok: [iosxe2.lab.example.com] => {
    "lookup_enabled_msg | trim": "DNS lookup enabled"
}

TASK [Display DNS name servers] *************************************************
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip name-server .*$',
 multiline=true) | first": "ip name-server 172.25.250.220"
}
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip name-server .*$',
 multiline=true) | first": "ip name-server 172.25.250.220"
}

TASK [Display DNS domain search] ************************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip domain list .*$',
 multiline=true)": [
        "ip domain list lab.example.com"
    ]
}
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip domain list .*$',
 multiline=true)": [
        "ip domain list lab.example.com"
    ]
}
...output omitted...
```

2. Create the `Configure Cisco DNS` job template.

   2.1. Navigate to **Resources** > **Templates** and click **Add** > **Add job template**. Create the job template with the following settings, and then click **Save**:

Field	Value
Name	`Configure Cisco DNS`
Inventory	`Common`
Project	`Surveys`
Playbook	`cisco_configuration.yml`
Credentials	`Cisco IOS XE Auth`

3. Add four survey questions to the `Configure Cisco DNS` job template.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.1. Navigate to **Resources** > **Templates** and then click the **Configure Cisco DNS** link. Click the **Survey** tab.

3.2. Use the following tables for the survey questions. For each survey question, click **Add** to begin creating the question and then click **Save** when you finish.

Survey Question 1

Field	Value
Question	`Default domain name`
Description	`This domain is used to set the hostname of the node.`
Answer variable name	`global_domain_name`
Answer type	`Text`
Required	*(selected)*
Minimum length	`1`
Maximum length	`251`
Default Answer	`lab.example.com`

Important

This exercise **does not** change DNS records. When you launch the `Configure Cisco DNS` job template, entering a value other than `lab.example.com` for the `global_domain_name` variable does set a new hostname, but you cannot reach the managed node by the new hostname.

Survey Question 2

Field	Value
Question	`Enable DNS lookups?`
Answer variable name	`ios_dns_lookup_enabled`
Answer Type	`Multiple Choice (single select)`
Required	*(selected)*
Multiple Choice Options	`true` *(selected as default)*
	`false` *(not selected as default)*

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Survey Question 3

Field	Value
Question	`Select DNS servers`
Answer variable name	`ios_name_servers`
Answer Type	`Multiple Choice (multiple select)`
Required	*(selected)*
Multiple Choice Options	`172.25.250.220` *(selected as default)*
	`1.1.1.1` *(not selected as default)*
	`8.8.8.8` *(not selected as default)*

Survey Question 4

Field	Value
Question	`Select domain search paths`
Answer variable name	`ios_domain_search`
Answer Type	`Multiple Choice (multiple select)`
Required	*(selected)*
Multiple Choice Options	`lab.example.com` *(selected as default)*
	`example.com` *(not selected as default)*
	`redhat.com` *(not selected as default)*
	`ansible.com` *(not selected as default)*

4. Enable the survey for the `Configure Cisco DNS` job template.

4.1. Navigate to **Resources** > **Templates** and click the **Configure Cisco DNS** link. Click the **Survey** tab.

4.2. Click **Survey Disabled** and verify that the label changes to **Survey Enabled**.

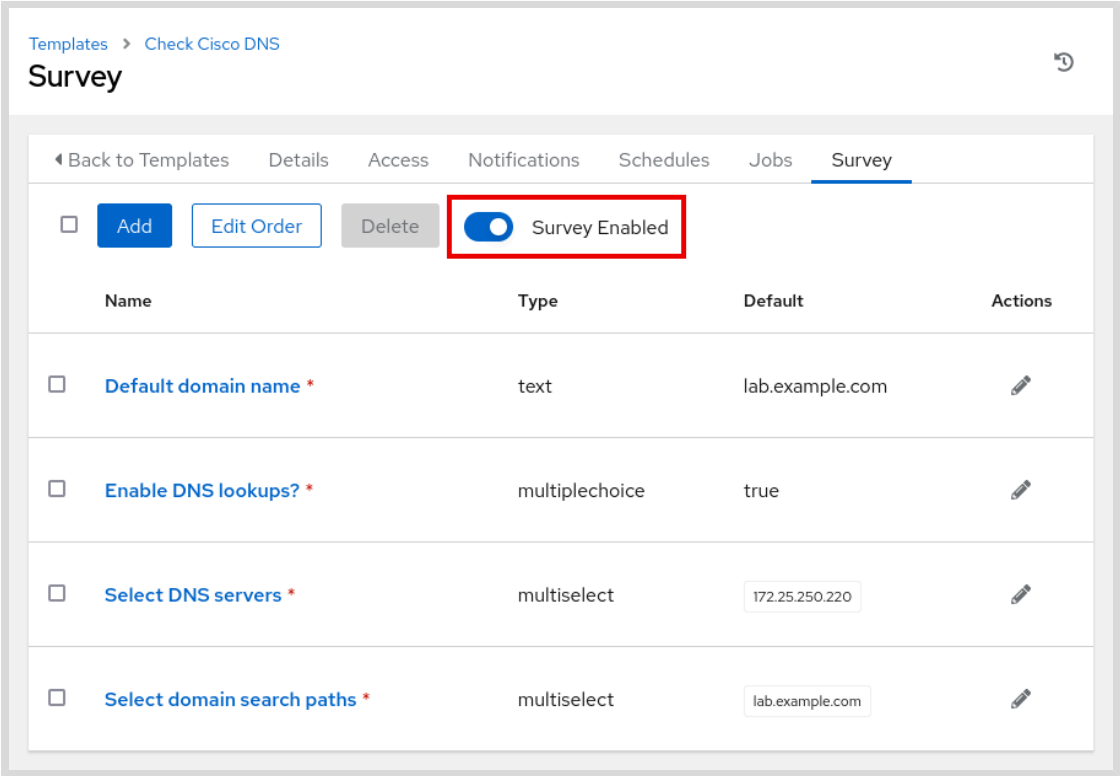


Figure 4.5: Enabled survey with questions

5. Launch the `Configure Cisco DNS` job template.

5.1. Navigate to **Resources** > **Templates** and click the **Launch Template** icon for the **Configure Cisco DNS** job template. The automation controller web UI opens a dialog box with the job template survey.

5.2. Select `1.1.1.1` as an additional entry from the **Select DNS servers** list.

5.3. Select `example.com` as an additional entry from the **Select domain search paths** list.

5.4. Click **Next** to preview the settings for the job template.

5.5. Click **Launch** to launch the job template and use the variables defined by the survey. The `Configure Cisco DNS` job displays a status of **Successful**.

6. Verify the changes made by the `Configure Cisco DNS` job template.

6.1. Navigate to **Resources** > **Templates** and click the **Launch Template** icon for the **Check Cisco DNS** job template.

The job output displays the updated DNS name servers and domain search entries:

```
...output omitted...
TASK [Display DNS name servers] ************************************************
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip name-server .*$',
 multiline=true) | first": "ip name-server 172.25.250.220 1.1.1.1"
}
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip name-server .*$',
 multiline=true) | first": "ip name-server 172.25.250.220 1.1.1.1"
}

TASK [Display DNS domain search] ************************************************
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip domain list .*$',
 multiline=true)": [
        "ip domain list lab.example.com",
        "ip domain list example.com"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^ip domain list .*$',
 multiline=true)": [
        "ip domain list lab.example.com",
        "ip domain list example.com"
    ]
}
...output omitted...
```

Finish

On the `workstation` machine, 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-surveys
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Transforming Data with Filters

Objectives

- Format, parse, manipulate, and define the values of variables by 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 inside the 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. 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 }}
```

Gathering Facts about Network Devices

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 managed nodes in your environment.

Network Information Filters

You can validate and manipulate networking 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 *Variable Length Subnet Mask (VLSM)* to *Classless Inter-domain Routing (CIDR)* subnet prefix notation, perform subnet math, or find the next usable address in a network range.

This section explores some uses of the `ansible.utils.ipaddr` filter.

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 expression returns the IP address. If the value is not an IP address, then the filter expression returns `false`.

If the value is a list, then the filter expression returns the valid IP addresses but not the invalid ones. If all the items are invalid, then the filter expression 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

If you pass a list of items to the `ansible.utils.ipaddr` filter, then the filter attempts to convert the items that appear to be valid addresses and filter out items that are not. Use the following options to the `ansible.utils.ipaddr` filter to restrict results:

Option	Description
`host`	The `host` option in the `ansible.utils.ipaddr('host')` filter expression returns valid individual IP addresses (in the correct CIDR prefix format). This filter does not return networks.
`net`	The `net` option in the `ansible.utils.ipaddr('net')` filter expression returns valid network specifications and converts the netmask into a CIDR prefix if necessary. This filter does not return individual IP addresses.
`private`	The `private` option in the `ansible.utils.ipaddr('private')` filter expression returns IP addresses or network ranges that are in ranges reserved by *Internet Assigned Numbers Authority (IANA)* to be private. These ranges include RFC 1918 address spaces and IPv6 site-local addresses at a minimum. This filter does not return public addresses and networks.
`public`	The `public` option in the `ansible.utils.ipaddr('public')` filter expression returns addresses and networks in public (globally routable) address spaces, according to IANA. This filter does not return private addresses and networks.

Manipulating IP Addresses

You can use a number of the `ansible.utils.ipaddr` options to format networking data from an IPv4 or IPv6 address.

Option	Description
`address`	The `address` option in the `ansible.utils.ipaddr('address')` filter expression returns the address without a subnet mask or prefix, such as `192.0.2.1` for the `192.0.2.1/24` address.
`netmask`	The `netmask` option in the `ansible.utils.ipaddr('netmask')` filter expression returns the variable length subnet mask, such as `255.255.255.0` for the `192.0.2.1/24` address.
`prefix`	The `prefix` option in the `ansible.utils.ipaddr('prefix')` filter expression returns the CIDR prefix, such as `24` for the `192.0.2.1/24` address.
`revdns`	The `revdns` option in the `ansible.utils.ipaddr('revdns')` filter expression returns the IPv4 or IPv6 address in DNS PTR record format, such as `1.2.0.192.in-addr.arpa.` for the `192.0.2.1/24` address.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Option	Description
`network`	The `network` option in the `ansible.utils.ipaddr('network')` filter expression returns the network's network address, such as `192.0.2.0` for the `192.0.2.1/24` address.
`broadcast`	The `broadcast` option in the `ansible.utils.ipaddr('broadcast')` filter expression returns the network's broadcast address, such as `192.0.2.255` for the `192.0.2.1/24` address.

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.

Calculating, Validating, and Reformatting Network Information

You can specify options to the `ansible.utils.ipaddr` filter to generate an address and identify valid addresses in a network.

Option	Description
integer	You can display an IP address on a given network by passing an integer to the `ansible.utils.ipaddr` filter. For example, the `{{ '192.0.2.0/24' | ansible.utils.ipaddr(5) }}` expression returns the fifth address (`192.0.2.5/24`) on the `192.0.2.0/24` network.
`range_usable`	The `range_usable` option in the `ansible.utils.ipaddr('range_usable')` filter expression returns the range of usable host addresses for a network, such as `192.0.2.1-192.0.2.254` for the `192.0.2.0/24` network.
`next_usable`	If you provide an existing address on a network, then the `next_usable` option in the `ansible.utils.ipaddr('next_usable')` filter expression returns the next address or an empty value if the end of the network has been reached. For example, the `{{ '192.0.2.5/24' | ansible.utils.ipaddr('next_usable') }}` expression returns the `192.0.2.6` address.

The following filters also belong to the `ansible.utils` collection:

`ansible.utils.network_in_usable`
: If you pass a network specification to the `ansible.utils.network_in_usable` filter, then you can specify a particular address as an option to determine if that address is usable for hosts on that network. You might use this filter to verify that an address is not reserved for broadcasts or other uses. The following expression returns the `true` Boolean value:

```
{{ '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` (an address on a different network), then the expression returns the `false` Boolean value.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

`ansible.utils.cidr_merge`

You can use the `ansible.utils.cidr_merge` filter to merge a list of subnets and individual hosts into the minimal representation. For example, consider the following variables:

```
vars:
  my_networks:
    - 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
  merged_networks: "{{ my_networks | ansible.utils.cidr_merge }}"
```

The `merged_networks` variable reduces the networks from the `my_networks` variable into the following networks:

```
    - 10.0.1.0/24
    - 10.0.2.0/23
    - 10.0.100.0/24
    - 2001:db8::/63
```

`ansible.utils.ipwrap`

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. Consider the following variables:

```
vars:
  provided:
    - 192.168.2.1
    - 10.0.0.128/25
    - 172.24.10.0/255.255.255.0
    - ff02::1
    - ::1
    - 2001::1/64
    - www.redhat.com
  brackets: "{{ provided | ansible.utils.ipwrap }}"
```

The `brackets` variable contains all the entries in the `provided` variable and formats the IPv6 addresses and networks so that they contain brackets:

```
    - 192.168.2.1
    - 10.0.0.128/25
    - 172.24.10.0/255.255.255.0
    - [ff02::1]
    - [::1]
    - [2001::1]/64
    - www.redhat.com
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

If desired, then you can use the `ansible.utils.ipwrap` filter to exclude the `www.redhat.com` list item. The following expression for the previous example does not add the `www.redhat.com` list item to the `brackets` variable.

```
brackets: "{{ provided | ansible.utils.ipaddr | ansible.utils.ipwrap }}"
```

General Purpose Filters

Several general purpose filters are available for you to use in your playbooks. The following table briefly describes some of these filters:

Filter	Description
`bool`	Generates a Boolean value. This filter can help avoid problems where a developer assigns the `"true"` or `"false"` (quoted) value to a variable rather than the `true` or `false` (unquoted) value to a variable.
`default`	Provides a value if the variable is undefined.
`dict2items`	Converts a dictionary to a list.
`from_json`	Indicates that the data in the variable comes from the JSON format.
`from_yaml`	Indicates that the data in the variable comes from the YAML format.
`map`	Generates a list of values from array items that contain a specific attribute key.
`password_hash`	Transforms a text string to a hashed password.
`regex_replace`	Removes or replaces a text pattern.
`selectattr`	Selects items from an array that match a particular value.
`split`	Generates an array that is delimited by the pattern used in the `split` filter expression.
`to_json`	Converts data into formatted JSON. Use the `to_nice_json` filter to convert data into nicely formatted JSON.
`to_yaml`	Converts data into formatted YAML. Use the `to_nice_yaml` filter to convert data into nicely formatted YAML.

Several of the filters in the previous table are available as part of the `ansible-core` RPM package. If the `ansible-core` package is installed on your local machine, then you can list all the files in that package and search for a specific plug-in.

```
[user@host ~]$ rpm -ql ansible-core | grep to_nice_yaml
/usr/lib/python3.9/site-packages/ansible/plugins/filter/to_nice_yaml.yml
```

After you identify the location of one plug-in, you can find additional plug-ins in the same directory:

```
[user@host ~]$ ls /usr/lib/python3.9/site-packages/ansible/plugins/filter/
__init__.py       flatten.yml         random.yml                to_nice_json.yml
__pycache__       from_json.yml       realpath.yml              to_nice_yaml.yml
b64decode.yml     from_yaml.yml       regex_escape.yml          to_uuid.yml
b64encode.yml     from_yaml_all.yml   regex_findall.yml         to_yaml.yml
basename.yml      hash.yml            regex_replace.yml         type_debug.yml
bool.yml          human_readable.yml  regex_search.yml          union.yml
checksum.yml      human_to_bytes.yml  rekey_on_member.yml       unique.yml
combinations.yml  intersect.yml       relpath.yml               unvault.yml
combine.yml       items2dict.yml      root.yml                  urldecode.yml
comment.yml       log.yml             sha1.yml                  urls.py
core.py           mandatory.yml       shuffle.yml               urlsplit.py
dict2items.yml    mathstuff.py        split.yml                 vault.yml
difference.yml    md5.yml             splitext.yml              win_basename.yml
dirname.yml       password_hash.yml   strftime.yml              win_dirname.yml
encryption.py     path_join.yml       subelements.yml           win_splitdrive.yml
expanduser.yml    permutations.yml    symmetric_difference.yml  zip.yml
expandvars.yml    pow.yml             ternary.yml               zip_longest.yml
extract.yml       product.yml         to_datetime.yml
fileglob.yml      quote.yml           to_json.yml
```

The other filters from the table are Jinja2 built-in filters. You can find more information about both types of filters using the links in the References section.

References

Filters - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

Template Designer Documentation: Filters
https://jinja.palletsprojects.com/en/3.0.x/templates/#builtin-filters

Using the ipaddr Filter - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/utils/docsite/filters_ipaddr.html

Ansible.Utils - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/utils/index.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Using Filters to Parse Command Output

Use filters to process the output of commands.

Outcomes

- Use options to modify the output of the `ansible.utils.ipaddr` filter.
- Use filters to extract data gathered by querying the automation controller API.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

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

Instructions

Use the provided playbooks to explore options to the `ansible.utils.ipaddr` filter and to query information about automation controller. These playbooks use the `ansible.builtin.debug` module to apply filters and return output.

1. Open the `/home/student/manage-filters` directory in VS Code and view the `filter_data.yml` playbook.

   1.1. Open VS Code and navigate to **File** > **Open Folder**.

   1.2. Navigate to **Home** > **manage-filters** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Click the `filter_data.yml` playbook. The `possible_options` variable defines the options that this playbook accepts for the `ansible.utils.ipaddr` filter. The top of the playbook displays sample commands with each of these options.

```
---
# Sample commands:
# ansible-navigator run filter_data.yml -e option=host
# ansible-navigator run filter_data.yml -e option=net
# ansible-navigator run filter_data.yml -e option=private
# ansible-navigator run filter_data.yml -e option=public
- name: Play for the "Filtering Data" section of the lecture
  hosts: localhost
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  gather_facts: false
  vars:
    samples: ❶
      - 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
    possible_options:
      - host    # Return valid hosts and use a CIDR prefix
      - net     # Return valid networks and use a CIDR prefix
      - private # Return hosts and networks in the private range of addresses
      - public  # Return hosts and networks in the public range of addresses
  tasks:
    - name: Fail if the 'option' variable is not defined or is invalid ❷
      ansible.builtin.assert:
        that:
          - option is defined
          - option in possible_options
        fail_msg: >-
          The 'option' variable must be defined and have one of the following
          values {{ possible_options }}.

    - name: Display the 'samples' variable ❸
      ansible.builtin.debug:
        var: samples

    - name: Use the "ansible.utils.ipaddr('{{ option }}')" filter expression ❹
      ansible.builtin.debug:
        var: samples | ansible.utils.ipaddr(option)
```

❶ The `samples` variable includes IPv4 and IPv6 hosts and networks. If desired, then you can add hosts and networks to the list.

❷ This playbook assumes that you pass the `option` variable, such as `-e option=host`. In this task, the `ansible.builtin.assert` module checks that the `option` variable has been defined and that the value of the variable matches one of four possible values.

❸ The playbook displays the `samples` variable so that you can compare the original value with the value returned by the filter.

❹ This task displays the result of using a filter expression on the `samples` variable.

2. Run the `filter_data.yml` playbook and set the value of the `option` variable to one of the choices in the `possible_options` variable.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/manage-filters` directory in a GNOME terminal:

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

2.2. Use the `host` option to the `ansible.utils.ipaddr` filter so that the filter only returns list items that are hosts. Notice that the filter ensures that each host contains a CIDR prefix.

```
[student@workstation manage-filters]$ ansible-navigator run filter_data.yml \
-e option=host

...output omitted...
TASK [Use the "ansible.utils.ipaddr('host')" filter expression] ****************
ok: [localhost] => {
    "samples | ansible.utils.ipaddr(option)": [
        "192.168.2.1/32",
        "172.24.10.0/32",
        "ff02::1/128",
        "::1/128",
        "2001::1/64"
    ]
}
...output omitted...
```

2.3. *(Optional)* Run the `filter_data.yml` playbook to explore the `net`, `private`, and `public` options to the `ansible.utils.ipaddr` filter. The top of the `filter_data.yml` playbook provides sample commands for you to run.

3. Run the `manipulate.yml` playbook to explore options to the `ansible.utils.ipaddr` filter that manipulate or extract components of a host address. The playbook sends a single host address to the `ansible.utils.ipaddr` filter and expects that you set the value of the `option` variable to one of the choices in the `possible_options` variable.

3.1. Click the `manipulate.yml` playbook in VS Code. This playbook uses options to the `ansible.utils.ipaddr` filter to return a component of the `sample_address` variable.

```
---
# Sample commands:
# ansible-navigator run manipulate.yml -e option=address
# ansible-navigator run manipulate.yml -e option=netmask
# ansible-navigator run manipulate.yml -e option=prefix
# ansible-navigator run manipulate.yml -e option=revdns
# ansible-navigator run manipulate.yml -e option=network
# ansible-navigator run manipulate.yml -e option=broadcast
- name: Tasks for the "Manipulating IP Addresses" section of the lecture
  hosts: localhost
  gather_facts: false
  vars:
    sample_address: 192.0.2.1/24
    possible_options:
      - address   # Return the address without a subnet mask or prefix
      - netmask   # Return the variable length subnet mask for the address
      - prefix    # Return the CIDR prefix for the address
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
      - revdns    # Return the DNS pointer record for the address
      - network   # Return the network address for the network
      - broadcast # Return the broadcast address for the network
  tasks:
    - name: Fail if the 'option' variable is not defined or is invalid
      ansible.builtin.assert:
        that:
          - option is defined
          - option in possible_options
        fail_msg: >-
          The 'option' variable must be defined and have one of the following
          values {{ possible_options }}.

    - name: Display the 'sample_address' variable
      ansible.builtin.debug:
        var: sample_address

    - name: Use the "ansible.utils.ipaddr('{{ option }}')" filter expression
      ansible.builtin.debug:
        var: sample_address | ansible.utils.ipaddr(option)
```

3.2. Run the `manipulate.yml` playbook to extract the address part of the `sample_address` variable:

```
[student@workstation manage-filters]$ ansible-navigator run manipulate.yml \
-e option=address

...output omitted...
TASK [Display the 'sample_address' variable] ***********************************
ok: [localhost] => {
    "sample_address": "192.0.2.1/24"
}

TASK [Use the "ansible.utils.ipaddr('address')" filter expression] *************
ok: [localhost] => {
    "sample_address | ansible.utils.ipaddr(option)": "192.0.2.1"
}
...output omitted...
```

3.3. *(Optional)* Run the `manipulate.yml` playbook to explore the `netmask`, `prefix`, `revdns`, `network`, and `broadcast` options to the `ansible.utils.ipaddr` filter. The top of the `manipulate.yml` playbook provides sample commands for you to run.

4. Run the `controller.yml` playbook to query the automation controller API. Rather than providing you with a finished filter expression, the following substeps have you incrementally adjust filters in the playbook to display information about existing credentials.

4.1. In VS Code, click the `controller.yml` playbook to display the playbook:

```
---
- name: Query automation controller API
  hosts: controller.lab.example.com
  gather_facts: false
```

```
  tasks:
    - name: Include controller_auth.yml ❶
      ansible.builtin.include_vars: vars/controller_auth.yml

    - name: Query for credentials ❷
      ansible.builtin.uri:
        url: https://{{ controller['host'] }}/api/v2/credentials/
        method: GET
        headers:
          Authorization: >-
            Basic {{ controller['username_password'] | string | b64encode }}
      register: query_credentials

    - name: Show query results ❸
      ansible.builtin.debug:
        var: query_credentials['json']['results']
```

❶ The first task includes a variable file that contains authentication information for the automation controller instance in the lab environment. Although not performed in this exercise, you might consider using Ansible Vault to encrypt files that contain sensitive information.

❷ The second task queries the automation controller API for information about credentials and saves the results of the query in the `query_credentials` variable.

❸ The third task displays the results of the query in JSON format. The `query_credentials['json']['results']` variable is an array that contains a list of existing credentials.

4.2. Run the `controller.yml` playbook. The playbook generates several lines of output. Scroll back through the output until you see the start of the output for the `Show query results` task.

The specific credential displayed in this example is not important and your output might be different. The important thing to notice is that there are different kinds of credentials and each credential has a name and a unique identification number (among other things).

```
[student@workstation manage-filters]$ ansible-navigator run controller.yml

...output omitted...
TASK [Show query results] ******************************************************
ok: [controller.lab.example.com] => {
    "query_credentials['json']['results']": [
        {
            "cloud": false,
            "created": "2023-04-27T19:27:43.561109Z",
            "credential_type": 18,
            "description": "",
            "id": 2,
            "inputs": {
                "url": "https://galaxy.ansible.com/"
            },
            "kind": "galaxy_api_token",
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
            "kubernetes": false,
            "managed": true,
            "modified": "2023-04-27T19:27:43.561121Z",
            "name": "Ansible Galaxy",
...output omitted...
```

4.3. Edit the `controller.yml` playbook so that the last task displays the names of the credentials. You can do this by using the `map` filter. Ensure that the last task contains the following content and then save the file:

```
...output omitted...
    - name: Show query results
      ansible.builtin.debug:
        var: query_credentials['json']['results'] | map(attribute='name')
```

4.4. Run the `controller.yml` playbook to display the credential names. Your playbook output might be slightly different.

```
[student@workstation manage-filters]$ ansible-navigator run controller.yml

...output omitted...
TASK [Show query results] *******************************************************
ok: [controller.lab.example.com] => {
    "query_credentials['json']['results'] | map(attribute='name')": [
        "Ansible Galaxy",
        "Automation Hub Community Repository",
        "Automation Hub Container Registry",
        "Automation Hub Published Repository",
        "Automation Hub RH Certified Repository",
        "Automation Hub Validated Repository",
        "Cisco IOS XE Auth",
        "Default Execution Environment Registry Credential",
        "Demo Credential",
        "Git Projects",
        "Juniper Junos Auth"
    ]
}
...output omitted...
```

4.5. Based on the previous output, you might want to display details for the `Automation Hub Container Registry` credential. You can do this by using the `selectattr` filter. Edit the `controller.yml` playbook. Ensure that the last task contains the following content and then save the file:

```
...output omitted...
    - name: Show query results
      ansible.builtin.debug:
        var: >-
          query_credentials['json']['results'] |
          selectattr('name', '==', 'Automation Hub Container Registry')
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

Although not required, this task breaks up the variable into multiple lines. On the `var` option line, the hyphen after the greater-than (`>`) sign indicates that a newline character (`\n`) is not added to the end of the variable.

4.6. Run the `controller.yml` playbook to display information about the `Automation Hub Container Registry` credential. Your playbook output might be slightly different.

```
[student@workstation manage-filters]$ ansible-navigator run controller.yml

...output omitted...
TASK [Show query results] ******************************************************
ok: [controller.lab.example.com] => {
    "query_credentials['json']['results'] | selectattr('name', '==', 'Automation
 Hub Container Registry')": [
        {
            "cloud": false,
            "created": "2023-04-27T19:27:55.024754Z",
            "credential_type": 17,
            "description": "",
            "id": 8,
            "inputs": {
                "host": "hub.lab.example.com",
                "password": "$encrypted$",
                "username": "admin",
                "verify_ssl": true
            },
            "kind": "registry",
...output omitted...
```

4.7. Although automation controller stores a lot of information about each credential, only a small subset of that information might interest you. Based on the previous output, extract the value of the unique identification number of the credential. You can do this by adding the `map` filter after the `selectattr` filter.

Ensure that the last task contains the following content and then save the file:

```
...output omitted...
    - name: Show query results
      ansible.builtin.debug:
        var: >-
          query_credentials['json']['results'] |
          selectattr('name', '==', 'Automation Hub Container Registry') |
          map(attribute='id')
```

4.8. Run the `controller.yml` playbook to extract the identification number for the `Automation Hub Container Registry` credential. The identification number of your credential might be different from the following output:

```
[student@workstation manage-filters]$ ansible-navigator run controller.yml

...output omitted...
TASK [Show query results] ******************************************************
ok: [controller.lab.example.com] => {
    "query_credentials['json']['results'] |  selectattr('name', '==', 'Automation
 Hub Container Registry') | map(attribute='id')": [
        8
    ]
}
...output omitted...
```

4.9. Because the previous `map` filter only creates one array item, you can add another filter to select that item. Based on the previous output, edit the `controller.yml` playbook to add the `first` filter to the end of the previous expression.

Ensure that the last task contains the following content and then save the file:

```
...output omitted...
    - name: Show query results
      ansible.builtin.debug:
        var: >-
          query_credentials['json']['results'] |
          selectattr('name', '==', 'Automation Hub Container Registry') |
          map(attribute='id') |
          first
```

4.10. Run the `controller.yml` playbook to extract the previous identification number without the array:

```
[student@workstation manage-filters]$ ansible-navigator run controller.yml

...output omitted...
TASK [Show query results] ******************************************************
ok: [controller.lab.example.com] => {
    "query_credentials['json']['results'] | selectattr('name', '==', 'Automation
 Hub Container Registry') | map(attribute='id') | first": "8"
}
...output omitted...
```

5. Close the `/home/student/manage-filters` directory in VS Code and return to the `/home/student` directory from the CLI if you are using the GNOME terminal.

5.1. Click **File** > **Close Folder** to close the `/home/student/manage-filters` directory.

5.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation manage-filters]$ cd
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Finish

On the `workstation` machine, 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-filters
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Managing Variables and Facts

Write and run an Ansible Playbook that uses variables and facts, and which gets values for its variables from a survey when run in automation controller.

Outcomes

- Configure variable files for playbooks.
- Use filters to format output.
- Create a survey to prompt for variables used by a 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 installs the required software, downloads an automation execution environment, and creates the required resources for the exercise.

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

Instructions

This lab contains two parts. In the first part of the lab, you work with files on the `workstation` machine and push your changes to remote Git repositories. In the second part of the lab, you synchronize your changes with two automation controller projects and then add a survey to an existing job template.

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. Your automation controller instance contains the `Inventories` project resource, which uses the `git@git.lab.example.com:student/inventories` Git repository. The `Common` inventory resource uses the `inventory` file in that Git repository to populate automation controller host resources.

   Rather than defining variables in a single inventory file, you have decided to move group variables into their own files.

   Clone the `git@git.lab.example.com:student/inventories` Git repository into the `/home/student/git-repos` directory and then check out the `exercise` branch.

2. Create the necessary directories and then create the following files. Move the connection variables from the `/home/student/git-repos/inventories/inventory` file into the corresponding variable file for each inventory group and ensure that the variables use YAML format. When finished, the `/home/student/git-repos/inventories/inventory` file should no longer contain variables.

   - `/home/student/git-repos/inventories/group_vars/ios/connection.yml`
   - `/home/student/git-repos/inventories/group_vars/junos/connection.yml`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3. Define the following logging variables for the `ios` group in the `/home/student/git-repos/inventories/group_vars/ios/syslog_config.yml` file. Create the file and any directories as needed.

Variable	Value
`severity_level`	`warnings`
`facility`	`local5`
`log_host_1`	`192.168.10.10/24`
`log_host_2`	`192.168.11.10/24`

4. Commit and push all the changes that you made in the `/home/student/git-repos/inventories` directory to the remote Git repository. Add a descriptive commit message, such as `Move variables to separate files.`

5. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/inventories` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

6. The `Syslog` automation controller project resource uses the `git@git.lab.example.com:student/syslog` Git repository. Clone that Git repository into the `/home/student/git-repos` directory and then check out the `exercise` branch. Update the `syslog.yml` playbook in that repository so that it uses the `severity_level`, `facility`, `log_host_1`, and `log_host_2` variables, which you defined in the `/home/student/git-repos/inventories/group_vars/ios/syslog_config.yml` file.

Important

Use an option to the `ansible.utils.ipaddr` filter to extract the IP addresses of the `log_host_1` and `log_host_2` variables. For example, if the `log_host_1` variable has a value of `192.168.0.5/24`, then the filter should return the `192.168.0.5` IP address.

7. Verify that the `syslog.yml` playbook runs successfully.

Important

The automation controller `Cisco IOX XE Auth` machine credential defines the `student` username and authenticates using an SSH private key.

Add the `-u student` and `-k` options when running the `ansible-navigator` command. When prompted for the SSH password, type `student` and press `Enter`.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

8. Update the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook.

   - The playbook must target managed nodes in the `ios` inventory group.
   - The playbook must gather configuration facts about the managed nodes. Your playbook does this correctly if it defines the `ansible_facts['net_config']` fact.
   - Add a task that uses the `ansible.builtin.debug` module to display any line that matches the `^logging .*$` pattern in the `ansible_facts['net_config']` fact.

> **Important**
> You can use the `regex_findall` filter to search for the specified pattern. Add the `multiline=true` option to the filter.

9. Run the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook to display the current logging configuration of the managed nodes in the `ios` inventory group. Add the `-u student` and `-k` options when running the `ansible-navigator` command. When prompted, use `student` for the password.

   If the playbook succeeds, then add, commit, and push the changes that you made to the `syslog.yml` and `check_syslog_config.yml` files to the remote Git repository. Add a descriptive commit message, such as `Use variables and show configuration.`

10. From the automation controller web UI, update the `Inventories` and `Syslog` projects so that the projects contain the most recent Git repository changes. For the `Common` inventory, synchronize its source and then verify that the `ios` host group uses the variables that you defined in the `syslog_config.yml` file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

11. Although the inventory variables defined for the `ios` group provide one method for configuring logging settings, you want to make it easier to change those settings when someone runs the `Configure IOS syslog` job template.

Add a survey to the `Configure IOS syslog` job template with the following questions:

Survey Question 1

Field	Value
Question	`Primary syslog destination host`
Answer variable name	`log_host_1`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	18

Survey Question 2

Field	Value
Question	`Secondary syslog destination host`
Answer variable name	`log_host_2`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	18

Survey Question 3

Field	Value
Question	`Logging facility`
Answer variable name	`facility`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	6

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Survey Question 4

Field	Value
Question	`Logging severity level`
Answer variable name	`severity_level`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	13

12. Enable the survey for the `Configure IOS syslog` job template.

13. Launch the `Configure IOS syslog` job template and use the following answers to the survey questions. After the job completes, run the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook to display the updated logging configuration.

    - **Primary syslog destination host**: `192.168.80.10/24`
    - **Secondary syslog destination host**: `192.168.80.11/24`
    - **Logging facility**: `local2`
    - **Logging severity level**: `errors`

14. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/syslog` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

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

As the `student` user on the `workstation` machine, 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
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Managing Variables and Facts

Write and run an Ansible Playbook that uses variables and facts, and which gets values for its variables from a survey when run in automation controller.

Outcomes

- Configure variable files for playbooks.
- Use filters to format output.
- Create a survey to prompt for variables used by a 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 installs the required software, downloads an automation execution environment, and creates the required resources for the exercise.

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

Instructions

This lab contains two parts. In the first part of the lab, you work with files on the `workstation` machine and push your changes to remote Git repositories. In the second part of the lab, you synchronize your changes with two automation controller projects and then add a survey to an existing job template.

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. Your automation controller instance contains the `Inventories` project resource, which uses the `git@git.lab.example.com:student/inventories` Git repository. The `Common` inventory resource uses the `inventory` file in that Git repository to populate automation controller host resources.

   Rather than defining variables in a single inventory file, you have decided to move group variables into their own files.

   Clone the `git@git.lab.example.com:student/inventories` Git repository into the `/home/student/git-repos` directory and then check out the `exercise` branch.

   1.1. Use either VS Code or the `git clone` command to clone the `git@git.lab.example.com:student/inventories` Git repository into the `/home/student/git-repos` directory:

```
[student@workstation ~]$ cd ~/git-repos/
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/inventories
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1.2. Use either VS Code or the `git checkout exercise` command to check out the `exercise` branch:

```
[student@workstation git-repos]$ cd inventories
[student@workstation inventories]$ git checkout exercise
Switched to branch 'exercise'
```

2. Create the necessary directories and then create the following files. Move the connection variables from the `/home/student/git-repos/inventories/inventory` file into the corresponding variable file for each inventory group and ensure that the variables use YAML format. When finished, the `/home/student/git-repos/inventories/inventory` file should no longer contain variables.

- `/home/student/git-repos/inventories/group_vars/ios/connection.yml`
- `/home/student/git-repos/inventories/group_vars/junos/connection.yml`

2.1. Open the `/home/student/git-repos/inventories` directory in VS Code. Click **File** > **Open Folder**.

Navigate to **Home** > **git-repos** > **inventories** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

2.2. Click the inventory file. The file contains the following content. Notice that the file defines variables for the `ios` and `junos` groups:

```
[ios]
iosxe1.lab.example.com
iosxe2.lab.example.com

[ios:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios

[junos]
junos1.lab.example.com
junos2.lab.example.com

[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
```

2.3. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/git-repos/inventories` directory in a GNOME terminal. Create the `group_vars/ios` and `group_vars/junos` directories to hold the variables files for the managed nodes. Use separate `mkdir` commands, or use the following `mkdir` command. If desired, use the `-v` option to display the four directories created by the command:

```
[student@workstation inventories]$ mkdir -pv group_vars/{ios,junos}
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/ios'
mkdir: created directory 'group_vars/junos'
```

2.4. Create the `group_vars/ios/connection.yml` file and define the two variables currently found in the `inventory` file for the `ios` group:

```
---
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
```

2.5. Create the `group_vars/junos/connection.yml` file and define the two variables currently found in the `inventory` file for the `junos` group:

```
---
ansible_connection: ansible.netcommon.netconf
ansible_network_os: junipernetworks.junos.junos
```

2.6. Update the `inventory` file so that it does not contain variables. The updated file contains the following content:

```
[ios]
iosxe1.lab.example.com
iosxe2.lab.example.com

[junos]
junos1.lab.example.com
junos2.lab.example.com
```

3. Define the following logging variables for the `ios` group in the `/home/student/git-repos/inventories/group_vars/ios/syslog_config.yml` file. Create the file and any directories as needed.

Variable	Value
`severity_level`	`warnings`
`facility`	`local5`
`log_host_1`	`192.168.10.10/24`
`log_host_2`	`192.168.11.10/24`

3.1. Create the `group_vars/ios/syslog_config.yml` file with the following content:

```
---
severity_level: warnings
facility: local5
log_host_1: 192.168.10.10/24
log_host_2: 192.168.11.10/24
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4. Commit and push all the changes that you made in the `/home/student/git-repos/inventories` directory to the remote Git repository. Add a descriptive commit message, such as `Move variables to separate files.`

4.1. Use VS Code or the `git add` command to add the new and updated files and directories:

```
[student@workstation inventories]$ git add inventory group_vars
```

4.2. Use VS Code or the `git status` command to display the list of files that you staged for the next commit:

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

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   group_vars/ios/connection.yml
        new file:   group_vars/ios/syslog_config.yml
        new file:   group_vars/junos/connection.yml
        modified:   inventory
```

4.3. Use VS Code or the `git commit` command to commit the files. Use a descriptive commit message.

```
[student@workstation inventories]$ git commit \
-m 'Move variables to separate files'
[exercise ce28988] Move variables to separate files
 4 files changed, 11 insertions(+), 8 deletions(-)
 create mode 100644 group_vars/ios/connection.yml
 create mode 100644 group_vars/ios/syslog_config.yml
 create mode 100644 group_vars/junos/connection.yml
```

4.4. Use VS Code or the `git push` command to update the remote Git repository.

```
[student@workstation inventories]$ git push -u origin exercise
...output omitted...
To git.lab.example.com:student/inventories
   ef6f142..ce28988  exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

5. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/inventories` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

```
[student@workstation inventories]$ cd
```

6. The `Syslog` automation controller project resource uses the `git@git.lab.example.com:student/syslog` Git repository. Clone that Git repository into the `/home/student/git-repos` directory and then check out the `exercise` branch. Update the `syslog.yml` playbook in that repository so that it uses the `severity_level`,

`facility`, `log_host_1`, and `log_host_2` variables, which you defined in the `/home/student/git-repos/inventories/group_vars/ios/syslog_config.yml` file.

> **Important**
> Use an option to the `ansible.utils.ipaddr` filter to extract the IP addresses of the `log_host_1` and `log_host_2` variables. For example, if the `log_host_1` variable has a value of `192.168.0.5/24`, then the filter should return the `192.168.0.5` IP address.

6.1. Use either VS Code or the `git clone` command to clone the `git@git.lab.example.com:student/syslog` Git repository into the `/home/student/git-repos` directory:

```
[student@workstation ~]$ cd ~/git-repos/
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/syslog
```

6.2. Use either VS Code or the `git checkout exercise` command to check out the `exercise` branch:

```
[student@workstation git-repos]$ cd syslog
[student@workstation syslog]$ git checkout exercise
Switched to branch 'exercise'
```

6.3. Open the `/home/student/git-repos/syslog` directory in VS Code. Click **File** > **Open Folder**.

Navigate to **Home** > **git-repos** > **syslog** and click **Open**.

6.4. Update the `syslog.yml` playbook so that it uses variables. Replace the specific values in the playbook with their associated variable names. Use the `ansible.utils.ipaddr('address')` filter to extract the IP address from the `log_host_1` and `log_host_2` variables. The updated playbook uses the following highlighted lines:

```
---
- name: Confgure syslog on IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Remove syslog configuration
      cisco.ios.ios_logging_global:
        state: deleted

    - name: Configure syslog
      cisco.ios.ios_logging_global:
        config:
          trap: "{{ severity_level }}"
          facility: "{{ facility }}"
          hosts:
            - host: "{{ log_host_1 | ansible.utils.ipaddr('address') }}"
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
            - host: "{{ log_host_2 | ansible.utils.ipaddr('address') }}"
          logging_on: enable
        state: replaced
```

7. Verify that the `syslog.yml` playbook runs successfully.

> **Important**
> The automation controller `Cisco IOX XE Auth` machine credential defines the `student` username and authenticates using an SSH private key.
>
> Add the `-u student` and `-k` options when running the `ansible-navigator` command. When prompted for the SSH password, type `student` and press `Enter`.

7.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/git-repos/syslog` directory in a GNOME terminal. Run the `syslog.yml` playbook and use the `-u student` and `-k` options. When prompted for the SSH password, type `student` and press `Enter`.

```
[student@workstation syslog]$ ansible-navigator run syslog.yml -u student -k
SSH password: student

PLAY [Confgure syslog on IOS managed nodes] ************************************

TASK [Remove syslog configuration] *********************************************
ok: [iosxe1.lab.example.com]
ok: [iosxe2.lab.example.com]

TASK [Configure syslog] ********************************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=2    changed=1    unreachable=0    failed=0   ...
```

8. Update the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook.

- The playbook must target managed nodes in the `ios` inventory group.
- The playbook must gather configuration facts about the managed nodes. Your playbook does this correctly if it defines the `ansible_facts['net_config']` fact.
- Add a task that uses the `ansible.builtin.debug` module to display any line that matches the `^logging .*$` pattern in the `ansible_facts['net_config']` fact.

> **Important**
> You can use the `regex_findall` filter to search for the specified pattern. Add the `multiline=true` option to the filter.

8.1. Update the `check_syslog_config.yml` playbook to target managed nodes in the `ios` inventory group.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Display existing logging settings
  hosts: ios
...output omitted...
```

8.2. Add a task to gather configuration facts. Your task might look similar to the following and should use either the `config` or `all` subset:

```
---
- name: Display existing logging settings
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather configuration facts
      cisco.ios.ios_facts:
        gather_subset:
          - config
...output omitted...
```

8.3. Update the `Show logging configuration` task so that it uses the `regex_findall` filter. The completed playbook should have content similar to the following:

```
---
- name: Display existing logging settings
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather configuration facts
      cisco.ios.ios_facts:
        gather_subset:
          - config

    - name: Show logging configuration
      ansible.builtin.debug:
        var: >-
          ansible_facts['net_config'] |
          regex_findall('^logging .*$', multiline=true)
```

9. Run the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook to display the current logging configuration of the managed nodes in the `ios` inventory group. Add the `-u student` and `-k` options when running the `ansible-navigator` command. When prompted, use `student` for the password.

If the playbook succeeds, then add, commit, and push the changes that you made to the `syslog.yml` and `check_syslog_config.yml` files to the remote Git repository. Add a descriptive commit message, such as `Use variables and show configuration.`

9.1. Run the `check_syslog_config.yml` playbook and use the `-u student` and `-k` options. When prompted for the SSH password, type `student` and press `Enter`.

The logging settings use the values of the variables defined in the `/home/student/git-config/inventories/group_vars/ios/syslog_config.yml` file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation syslog]$ ansible-navigator run check_syslog_config.yml \
-u student -k
SSH password: student

PLAY [Display existing logging configuration] **********************************

TASK [Gather configuration facts] **********************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

TASK [Show logging configuration] **********************************************
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^logging .*$', multiline=true)":
 [
        "logging trap warnings",
        "logging facility local5",
        "logging host 192.168.10.10",
        "logging host 192.168.11.10"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^logging .*$', multiline=true)":
 [
        "logging trap warnings",
        "logging facility local5",
        "logging host 192.168.10.10",
        "logging host 192.168.11.10"
    ]
}

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=2    changed=0    unreachable=0    failed=0   ...
```

9.2. Use VS Code or the `git add` command to add the updated files:

```
[student@workstation syslog]$ git add syslog.yml check_syslog_config.yml
```

9.3. Use VS Code or the `git commit` command to commit the files. Use a descriptive commit message.

```
[student@workstation syslog]$ git commit \
-m 'Use variables and show configuration'
[exercise f769bdd] Use variables and show configuration
 2 files changed, 19 insertions(+), 20 deletions(-)
 rewrite check_syslog_config.yml (73%)
```

9.4. Use VS Code or the `git push` command to update the remote Git repository:

```
[student@workstation syslog]$ git push -u origin exercise
...output omitted...
To git.lab.example.com:student/syslog
   17e7962..f769bdd  exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

10. From the automation controller web UI, update the `Inventories` and `Syslog` projects so that the projects contain the most recent Git repository changes. For the `Common` inventory, synchronize its source and then verify that the `ios` host group uses the variables that you defined in the `syslog_config.yml` file.

10.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

10.2. Synchronize the `Inventories` project.

Navigate to **Resources** > **Projects** and then click the **Sync Project** icon for the **Inventories** project. The project status changes from **Pending**, to **Running**, to **Successful**.

10.3. Synchronize the `Syslog` project.

Navigate to **Resources** > **Projects** and then click the **Sync Project** icon for the **Syslog** project. The project status changes from **Pending**, to **Running**, to **Successful**.

10.4. Synchronize the `Common Inventories` source for the `Common` inventory.

Navigate to **Resources** > **Inventories** and then click **Common**. Click the **Sources** tab and then click the **Start sync process** icon for the **Common Inventories** source. The source status changes from **Pending**, to **Running**, to **Successful**.

10.5. View the populated variables for the `ios` group.

Click the **Groups** tab and then click **ios**. The **Variables** field displays the following variables in JSON format:

```
{
  "ansible_connection": "ansible.netcommon.network_cli",
  "ansible_network_os": "cisco.ios.ios",
  "facility": "local5",
  "log_host_1": "192.168.10.10/24",
  "log_host_2": "192.168.11.10/24",
  "severity_level": "warnings"
}
```

11. Although the inventory variables defined for the `ios` group provide one method for configuring logging settings, you want to make it easier to change those settings when someone runs the `Configure IOS syslog` job template.

Add a survey to the `Configure IOS syslog` job template with the following questions:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Survey Question 1

Field	Value
Question	`Primary syslog destination host`
Answer variable name	`log_host_1`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	18

Survey Question 2

Field	Value
Question	`Secondary syslog destination host`
Answer variable name	`log_host_2`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	18

Survey Question 3

Field	Value
Question	`Logging facility`
Answer variable name	`facility`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	6

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Survey Question 4

Field	Value
Question	`Logging severity level`
Answer variable name	`severity_level`
Answer type	`Text`
Required	*(not selected)*
Minimum length	1
Maximum length	13

11.1. Navigate to **Resources** > **Templates** and then click the **Configure IOS syslog** link. Click the **Survey** tab.

11.2. Click **Add** and then use the settings in the "Survey Question 1" table to create the `Primary syslog destination host` survey question. Click **Save**.

11.3. Click **Add** and then use the settings in the "Survey Question 2" table to create the `Secondary syslog destination host` survey question. Click **Save**.

11.4. Click **Add** and then use the settings in the "Survey Question 3" table to create the `Logging facility` survey question. Click **Save**.

11.5. Click **Add** and then use the settings in the "Survey Question 4" table to create the `Logging severity level` survey question. Click **Save**.

12. Enable the survey for the `Configure IOS syslog` job template.

12.1. Navigate to **Resources** > **Templates** and click the **Configure IOS syslog** link. Click the **Survey** tab.

12.2. Click **Survey Disabled** and verify that the label changes to **Survey Enabled**.

13. Launch the `Configure IOS syslog` job template and use the following answers to the survey questions. After the job completes, run the `/home/student/git-repos/syslog/check_syslog_config.yml` playbook to display the updated logging configuration.

- **Primary syslog destination host**: `192.168.80.10/24`
- **Secondary syslog destination host**: `192.168.80.11/24`
- **Logging facility**: `local2`
- **Logging severity level**: `errors`

13.1. Navigate to **Resources** > **Templates** and click the **Launch Template** icon for the **Configure IOS syslog** job template.

The automation controller web UI opens a dialog box with the job template survey.

13.2. Answer the survey questions using the following table and then click **Next** to preview the settings for the job template.

Survey Question	Answer
Primary syslog destination host	`192.168.80.10/24`
Secondary syslog destination host	`192.168.80.11/24`
Logging facility	`local2`
Logging severity level	`errors`

13.3. Click **Launch** to launch the job template and use the variables defined by the survey. The `Configure IOS syslog` job displays a status of **Successful**.

13.4. Switch back to the **Terminal** tab in VS Code, or ensure that you are in the `/home/student/git-repos/syslog` directory in a GNOME terminal.

Run the `check_syslog_config.yml` playbook. The playbook output displays the values that you entered using the job template survey.

```
[student@workstation syslog]$ ansible-navigator run check_syslog_config.yml \
-u student -k
SSH password: student

...output omitted...
TASK [Show logging configuration] ***********************************************
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^logging .*$', multiline=true)":
 [
        "logging trap errors",
        "logging facility local2",
        "logging host 192.168.80.10",
        "logging host 192.168.80.11"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['net_config'] | regex_findall('^logging .*$', multiline=true)":
 [
        "logging trap errors",
        "logging facility local2",
        "logging host 192.168.80.10",
        "logging host 192.168.80.11"
    ]
}
...output omitted...
```

14. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/syslog` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

```
[student@workstation syslog]$ cd
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

As the `student` user on the `workstation` machine, 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
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- Ansible variables help you reuse values across files in an Ansible project.
- You can take advantage of inventory groups and the `group_vars/` and `host_vars/` directories to improve project maintainability.
- Connection variables, such as `ansible_connection` and `ansible_network_os`, control how Ansible connects to a managed node.
- The `register` keyword captures the output of a task in a variable.
- Ansible facts are information about managed nodes that are stored in variables.
- If the `gather_facts` keyword is set to `true`, then Ansible tries to automatically detect the network platform for each managed node and gather a minimal set of facts. You can use `*_facts` resource modules to gather facts for your managed network nodes. For example, use the `cisco.ios.ios_facts` module for Cisco IOS or IOS XE managed nodes, or the `junipernetworks.junos.junos_facts` module for Juniper Junos managed nodes.
- Job template surveys help users to launch jobs with custom variable settings.
- You can use filters in Jinja2 expressions and templates to format, transform, and parse data. You can apply multiple filters sequentially in the same expression.
- The `ansible.utils.ipaddr` filter can process network address data into various output formats.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 5

Implementing Task Control

Goal

Manage task control and task errors in Ansible Playbooks.

Objectives

- Use loops to write efficient tasks and use conditions to control when to run tasks.
- Control what happens when a task fails, and what conditions cause a task to fail.
- Create a workflow job template in automation controller that launches a branching chain of jobs based on the success or failure of each job in the workflow sequence.

Sections

- Writing Loops and Conditional Tasks (and Guided Exercise)
- Handling Task Failure (and Guided Exercise)
- Building a Workflow Job Template (and Guided Exercise)

Lab

- Implementing Task Control

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Writing Loops and Conditional Tasks

Objectives

- Use loops to write efficient tasks and use conditions to control when to run tasks.

Task Iteration with Loops

You can use loops to avoid writing multiple tasks that use the same module. For example, instead of writing five tasks to ensure that five users exist, you can write one task that iterates over a list of five users to ensure that they all exist.

To iterate a task over a set of items, you can use the `loop` keyword. You can configure loops to repeat a task using each item in a list, the contents of each of the files in a list, a generated sequence of numbers, or using more complicated structures.

This section covers simple loops that iterate over a list of items. Consult the References section for more advanced looping scenarios.

Simple Loops

A simple loop iterates a task over a list of items. The `loop` keyword is added to the task, and takes as a value the list of items over which the task should be iterated. The loop variable `item` holds the value used during each iteration.

Consider the following example, which uses two tasks to ensure that two hosts are reachable:

```
- name: Ping host (hosta.lab.example.com)
  cisco.ios.ios_ping:
    dest: hosta.lab.example.com
    state: present

- name: Ping host (hostb.lab.example.com)
  cisco.ios.ios_ping:
    dest: hostb.lab.example.com
    state: present
```

These two tasks can be rewritten to use a simple loop so that only one task is needed to ensure that both hosts are reachable:

```
- name: Ping hosts
  cisco.ios.ios_ping:
    dest: "{{ item }}"
    state: present
  loop:
    - hosta.lab.example.com
    - hostb.lab.example.com
```

The loop can use a list provided by a variable. In the following example, the task attempts to reach each host defined by the `host_list` variable:

```
- name: Ping hosts
  vars:
    host_list:
      - hosta.lab.example.com
      - hostb.lab.example.com
      - hostc.lab.example.com
      - hostd.lab.example.com
  cisco.ios.ios_ping:
    dest: "{{ item }}"
    state: present
  loop: "{{ host_list }}"
```

Loops over a List of Dictionaries

The loop list does not need to be a list of simple values.

In the following example, each item in the list is a dictionary. Each dictionary contains three keys: `name`, `hash_type`, and `hashed_password`. You can retrieve the value of each key in the current `item` loop variable with the `item['name']`, `item['hash_type']`, and `item['hashed_password']` variables.

```
- name: Users exist with the correct password
  no_log: true
  cisco.ios.ios_user:
    name: "{{ item['name'] }}"
    hashed_password:
      type: "{{ item['hash_type'] }}"
      value: "{{ item['hashed_password'] }}"
    update_password: always
    state: present
  loop:
    - name: ansible
      hash_type: 5
      hashed_password: "$IjvDh1wt$O6ceAhpuYesiPJh.s2weQ1"
    - name: netops
      hash_type: 5
      hashed_password: "$/bqwREWG$0zUXcphlYOll0anjs/tEs1"
```

The outcome of the preceding task is that the `ansible` user exists with the `$IjvDh1wt$O6ceAhpuYesiPJh.s2weQ1` MD5 hashed password, and that the `netops` user exists with the `$/bqwREWG$0zUXcphlYOll0anjs/tEs1` MD5 hashed password.

Using Register Variables with Loops

The `register` keyword can also capture the output of a task that loops. The following example shows the structure of the `register` variable from a task that loops:

```
---
- name: Ping hosts from IOS managed nodes
  hosts: iosxe1.lab.example.com
  gather_facts: false
  vars:
    host_list:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
      - hosta.lab.example.com
      - hostb.lab.example.com
  tasks:
    - name: Ping hosts
      cisco.ios.ios_ping:
        dest: "{{ item }}"
      loop: "{{ host_list }}"
      register: ping_results ❶

    - name: Show ping_results
      ansible.builtin.debug:
        var: ping_results ❷
```

❶ The `ping_results` variable is registered.

❷ The contents of the `ping_results` variable are displayed to the screen.

Running the preceding playbook produces the following output:

```
TASK [Ping hosts] ************************************************************
ok: [iosxe1.lab.example.com] => (item=hosta.lab.example.com)
ok: [iosxe1.lab.example.com] => (item=hostb.lab.example.com)

TASK [Show ping_results] *****************************************************
ok: [iosxe1.lab.example.com] => {
    "ping_results": { ❶
        "changed": false,
        "msg": "All items completed",
        "results": [ ❷
            { ❸
                "ansible_loop_var": "item",
                "changed": false,
                "commands": "ping ip hosta.lab.example.com",
                "failed": false,
                "invocation": {
                    "module_args": {
                        "afi": "ip",
                        "count": null,
                        "dest": "hosta.lab.example.com",
                        "df_bit": false,
                        "egress": null,
                        "ingress": null,
                        "source": null,
                        "state": "present",
                        "timeout": null,
                        "vrf": null
                    }
                },
                "item": "hosta.lab.example.com",
                "packet_loss": "0%",
                "packets_rx": 5,
                "packets_tx": 5,
                "rtt": {
                    "avg": 2,
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
                    "max": 3,
                    "min": 2
                }
            },
...output omitted...
```

1. The `ping_results` variable is composed of key-value pairs.
2. The `results` key is a list that contains the results from the previous task.
3. Each item in the `ping_results['results']` list contains metadata keys, such as `commands`, `failed`, `item`, and `packet_loss` keys.

The previous playbook output displayed all the content for the `ping_results` variable. The next playbook extracts and displays the round-trip time for each item in the `ping_results['results']` list:

```
---
- name: Ping hosts from IOS managed nodes
  hosts: iosxe1.lab.example.com
  gather_facts: false
  vars:
    host_list:
      - hosta.lab.example.com
      - hostb.lab.example.com
  tasks:
    - name: Ping hosts
      cisco.ios.ios_ping:
        dest: "{{ item }}"
      loop: "{{ host_list }}"
      register: ping_results

    - name: Show ping_results round-trip times
      ansible.builtin.debug:
        var: ping_results['results'] | map(attribute='rtt')
```

The preceding playbook produces the following output:

```
...output omitted...
TASK [Show ping_results round-trip times] ************************************
ok: [iosxe1.lab.example.com] => {
    "ping_results['results'] | map(attribute='rtt')": [
        {
            "avg": 2,
            "max": 3,
            "min": 2
        },
        {
            "avg": 2,
            "max": 3,
            "min": 1
        }
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    ]
}
...output omitted...
```

Running Tasks Conditionally

Ansible can use *conditionals* to run tasks or plays when certain conditions are met. Playbook variables, registered variables, and Ansible facts can all be tested with conditionals. Operators to compare strings, numeric data, and Boolean values are available.

The following scenarios illustrate the use of conditions in Ansible.

- Run tasks on managed nodes that match a specific value for the `ansible_network_os` variable.
- Stop a playbook run when a variable is not defined or does not have a valid value.
- Define a hard limit in a variable (for example, `min_memory`) and compare it to the available memory on a managed node.
- Capture the output of a command and evaluate the output to determine whether a task completed before taking further action. For example, if a program fails, then a batch is skipped.

Conditional Task Syntax

The `when` statement is used to run a task conditionally. It takes as a value the condition to test. If the condition is met, the task runs. If the condition is not met, the task is skipped.

One of the simplest conditions that can be tested is whether a Boolean variable is true or false.

Note

The YAML 1.1 standard allows you to specify Boolean values in multiple ways, such as `true`, `True`, `yes`, and `1` (for true) and `false`, `False`, `no`, and `0` (for false).

The YAML 1.2 standard specifies that you can only use `true` or `false` to set Boolean values.

Although Ansible YAML files are based on the YAML 1.1 standard, you might see gradual standardization toward using only `true` or `false` for Boolean values in playbooks and other Ansible files.

For more information on the change to Boolean handling in conditions in Ansible Core 2.12 and later, see https://docs.ansible.com/ansible/latest/porting_guides/porting_guide_5.html#deprecated.

Starting with Ansible Core 2.12, strings are always treated by `when` conditionals as `true` Booleans if they contain any content. For example, setting the value of a variable to `"false"` (quoted) evaluates to the `true` Boolean value when used by a `when` conditional.

When using true/false conditions, you must make sure that your variable is treated by Ansible as a Boolean and not as a string. You can do this by passing the variable to the `bool` filter.

```
# The value of the my_bool variable is a string.
# Because the string has a value, the string evaluates to true.
# The condition is not met and the task is skipped.
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
- name: The task is skipped
  vars:
    my_bool: "false"
  ansible.builtin.debug:
    msg: The task runs.
  when: my_bool == false

# The value of the my_bool variable is a string.
# The bool filter converts the string to a boolean.
# The condition is met and the task runs.
- name: The task runs
  vars:
    my_bool: "false"
  ansible.builtin.debug:
    msg: The task runs.
  when: my_bool | bool == false
```

Important

Observe the indentation of the `when` statement. Because the `when` statement is not a module variable, it must be placed outside the module by being indented at the top level of the task.

A task is a YAML dictionary, and the `when` statement is one more key in the task, just like the task's name and the module it uses. A common convention places any `when` keyword that might be present after the task's name and the module (and module arguments).

The following table shows some operations that you can use when working with conditionals:

Example Conditionals

Operation	Example
Equal (value is a string)	`ansible_facts['net_version'] == "17.06.02"`
Not Equal (value is a string)	`ansible_facts['net_version'] != "17.06.02"`
Equal (value is numeric)	`max_memory == 512`
Not equal (value is numeric)	`max_memory != 512`
Less than	`min_memory < 128`
Greater than	`min_memory > 0`
Less than or equal to	`min_memory <= 256`
Greater than or equal to	`min_memory >= 512`
Variable exists	`my_var is defined`
Variable does not exist	`my_var is not defined`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Operation	Example
Boolean variable is `true`. The values of `1`, `True`, or `yes` evaluate to `true`.	`my_bool`
	`my_bool | bool == true`
Boolean variable is `false`. The values of `0`, `False`, or `no` evaluate to `false`.	`not my_bool`
	`my_bool | bool == false`
First variable's value exists as a value in second variable's list	`facility in facility_choices`

The playbook in the following example illustrates how the last entry in the preceding table works:

```
---
- name: Demonstrate the "in" keyword
  hosts: ios
  gather_facts: false
  vars:
    facility_choices: >- ❶
      [auth, cron, daemon, kern, local0, local1, local2, local3, local4,
      local5, local6, local7, lpr, mail, news, sys10, sys11, sys12, sys13,
      sys14, sys9, syslog, user, uucp]
  tasks:
    - name: Fail if 'facility' is not defined
      ansible.builtin.assert:
        that: facility is defined ❷
        fail_msg: >-
          The 'facility' variable must be defined such as: -e facility=local5

    - name: Validate 'facility' choice
      ansible.builtin.assert:
        that: facility in facility_choices ❸
        fail_msg: >-
          The value of the 'facility' variable ({{ facility }}) is not valid.
          Use one of the following values: {{ facility_choices }}
```

❶ The `facility_choices` variable contains a list of facilities copied from documentation for the `cisco.ios.ios_logging_global` module.

❷ The first task verifies that the user provided a value for the `facility` variable.

❸ The second task verifies that the value of the `facility` variable is one of the items in the `facility_choices` variable list. If the value of the `facility` variable is an item in the `facility_choices` variable list, then the condition passes and the playbook continues, otherwise the condition fails and the task displays a failure message.

Although not shown in this example, the playbook might proceed to configure logging settings by using the `cisco.ios.ios_logging_global` module.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Testing Multiple Conditions

A `when` statement can evaluate multiple conditions by combining the conditions with the `and` or `or` keywords. The following examples show how to evaluate multiple conditionals.

- With the `or` keyword, the conditional statement is met if either condition evaluates to true:

```
when: ansible_network_os == "cisco.ios.ios" or ansible_network_os ==
 "cisco.iosxr.iosxr"
```

- With the `and` keyword, both conditions have to be true for the entire conditional statement to be met:

```
when: ansible_facts['net_model'] == "C8000V" and ansible_facts['net_version'] ==
 "17.06.02"
```

- The `when` keyword also supports using a list to describe a list of conditions. Using a list improves readability, and when a list is provided to the `when` keyword, all the conditions have to be true for the entire conditional statement to be met:

```
when:
  - ansible_facts['net_model'] == "C8000V"
  - ansible_facts['net_version'] == "17.06.02"
```

- You can express more complex conditional statements by grouping conditions with parentheses. Using parentheses ensures that the conditional statements are correctly interpreted.

  The following example uses facts gathered from managed nodes and uses the greater-than symbol (`>`) so that the long condition can be split over multiple lines in the playbook:

```
when: >
  (ansible_network_os == "cisco.ios.ios" and
    ansible_facts['net_model'] == "C8000V")
  or
  (ansible_network_os == "cisco.iosxr.iosxr" and
  ansible_facts['net_model'] == "ASR9000")
```

References

Loops - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

Tests - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_tests.html

Conditionals - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html

Ansible 5 Porting Guide: Deprecated - Ansible Documentation
https://docs.ansible.com/ansible/latest/porting_guides/porting_guide_5.html#deprecated

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring and Verifying Specific Device Settings

Write a playbook containing tasks to configure specific settings on specific devices using conditions and loops, and verify the device state before and after configuration.

Outcomes

- Create a playbook that configures interfaces by looping over variable list items.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start task-loops
```

Instructions

1. Open the `/home/student/task-loops` directory in VS Code and then create the `group_vars/ios/interfaces.yml` file to store interface information about the Cisco IOS managed nodes.

   1.1. Open VS Code and click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **task-loops** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Create the `group_vars/ios/interfaces.yml` file with the following content:

```
---
interfaces:
  iosxe1.lab.example.com:
    - name: Loopback0
      description: Managed by Ansible
      enabled: true
      ipv4: "10.10.10.10/32"

    - name: GigabitEthernet2
      description: Managed by Ansible
      enabled: true
      ipv4: "172.25.150.20/24"
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  iosxe2.lab.example.com:
    - name: Loopback0
      description: Managed by Ansible
      enabled: true
      ipv4: "10.10.20.10/32"

    - name: GigabitEthernet2
      description: Managed by Ansible
      enabled: true
      ipv4: "172.25.150.21/24"
```

Important

This step intentionally omits configuring the `GigabitEthernet1` interfaces. Configuring the `GigabitEthernet1` interfaces incorrectly might result in you being unable to connect to the `iosxe1.lab.example.com` or `iosxe2.lab.example.com` managed nodes. If this happens, then you might need to recreate your lab environment.

2. Create the `configure_interfaces.yml` playbook in the `/home/student/task-loops` directory. This playbook loops through the interfaces defined in the `group_vars/ios/interfaces.yml` file.

2.1. Create the `configure_interfaces.yml` playbook and target managed nodes in the `ios` inventory group. Add a task to gather interface facts and a second task that displays the content of the `ansible_facts['net_interfaces']` variable:

```
---
- name: Configure Cisco IOS interfaces
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather interface facts (pre)
      cisco.ios.ios_facts:
        gather_subset:
          - interfaces

    - name: Show ansible_facts['net_interfaces'] (pre)
      ansible.builtin.debug:
        var: ansible_facts['net_interfaces']
```

2.2. Add a task to the end of the play that uses the `cisco.ios.ios_interfaces` module to configure basic interface information. This task must ensure that an interface with the specified name exists. The task might add a description to the interface and either enables or disables the interface.

```
...output omitted...

    - name: Configure basic interface information
      when: (1)
        - interfaces is defined
        - interfaces[inventory_hostname] is defined
      cisco.ios.ios_interfaces:
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        config:
          - name: "{{ item['name'] }}" ❷
            description: "{{ item['description'] | default(omit) }}" ❸
            enabled: "{{ item['enabled'] | default(false) }}" ❹
        state: merged
      loop: "{{ interfaces[inventory_hostname] }}" ❺
```

❶ The playbook implicitly includes variables in the `group_vars/ios` directory. Because the `loop` keyword in this task uses the `interfaces[inventory_hostname]` variable, both the `interfaces` and the `interfaces[inventory_hostname]` variables must be defined or Ansible skips running the task. For example, the `interfaces['iosxe1.lab.example.com']` variable must be defined or Ansible skips running the task for the `iosxe1.lab.example.com` managed node.

❷ The list item must define the `name` variable.

❸ If the list item does not define the `description` variable, then the `default` filter does not specify a default value. Because of this `default` filter, Ansible does not generate an error if the `description` variable is not defined.

❹ If the list item does not define the `enabled` variable, then the `default` filter provides the `false` Boolean value.

❺ The task loops through elements of the `interfaces` variable and uses the `inventory_hostname` variable to identify each dictionary.

2.3. Add a task to the end of the play that uses the `cisco.ios.ios_l3_interfaces` module to configure IPv4 addresses for the defined interfaces. In addition to the conditions in the previous task, Ansible only runs this second task if the list item defines the `ipv4` variable:

```
...output omitted...

    - name: Configure L3 interface information
      when:
        - interfaces is defined
        - interfaces[inventory_hostname] is defined
        - item['ipv4'] is defined
      cisco.ios.ios_l3_interfaces:
        config:
          - name: "{{ item['name'] }}"
            ipv4:
              - address: "{{ item['ipv4'] }}"
        state: merged
      loop: "{{ interfaces[inventory_hostname] }}"
```

2.4. Add a task to the end of the play that pauses the play for five seconds. This pause allows time for the interface to come up before querying for interface facts again. You only need to run the task once rather that once for each managed node.

```
...output omitted...
    - name: Pause 5 seconds
      ansible.builtin.pause:
        seconds: 5
      run_once: true
```

2.5. Add two tasks to the end of the play to gather interface facts and display the content of the `ansible_facts['net_interfaces']` variable. If you copy and paste the first two task in the play, then change `pre` to `post` for the final two tasks.

The entire playbook consists of the following content:

```
---
- name: Configure Cisco IOS interfaces
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather interface facts (pre)
      cisco.ios.ios_facts:
        gather_subset:
          - interfaces

    - name: Show ansible_facts['net_interfaces'] (pre)
      ansible.builtin.debug:
        var: ansible_facts['net_interfaces']

    - name: Configure basic interface information
      when:
        - interfaces is defined
        - interfaces[inventory_hostname] is defined
      cisco.ios.ios_interfaces:
        config:
          - name: "{{ item['name'] }}"
            description: "{{ item['description'] | default(omit) }}"
            enabled: "{{ item['enabled'] | default(false) }}"
        state: merged
      loop: "{{ interfaces[inventory_hostname] }}"

    - name: Configure L3 interface information
      when:
        - interfaces is defined
        - interfaces[inventory_hostname] is defined
        - item['ipv4'] is defined
      cisco.ios.ios_l3_interfaces:
        config:
          - name: "{{ item['name'] }}"
            ipv4:
              - address: "{{ item['ipv4'] }}"
        state: merged
      loop: "{{ interfaces[inventory_hostname] }}"

    - name: Pause 5 seconds
      ansible.builtin.pause:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        seconds: 5
      run_once: true

    - name: Gather interface facts (post)
      cisco.ios.ios_facts:
        gather_subset:
          - interfaces

    - name: Show ansible_facts['net_interfaces'] (post)
      ansible.builtin.debug:
        var: ansible_facts['net_interfaces']
```

3. Run the `configure_interfaces.yml` playbook.

3.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/task-loops` directory in a GNOME terminal.

3.2. Run the playbook:

```
[student@workstation task-loops]$ ansible-navigator run configure_interfaces.yml
...output omitted...
```

3.3. Scroll back through the playbook output and notice the following in the `Show ansible_facts['net_interfaces'] (pre)` task for each managed node.

- The `GigabitEthernet1` and `GigabitEthernet2` interfaces do not have descriptions.
- The `GigabitEthernet1` interface already has an IPv4 addresses.
- The `GigabitEthernet2` interface does not have an IPv4 address and displays the `administratively down` operational status.
- The `Loopback0` interface does not exist.

Note

The `ansible-navigator.yml` file in this project enables playbook artifacts. If desired, you can review the output of the playbook using the `ansible-navigator replay` command. You might select the artifact file name using tab completion. You might also consider using the `-m interactive` option so that you can selectively view the output of specific tasks.

For example:

```
[student@workstation task-loops]$ ansible-navigator replay -m interactive \
configure_interfaces-artifact-2023-08-14T13\:58\:15.922542+00\:00.json
```

3.4. Scroll back through the playbook output and notice that the `Configure basic interface information` task performed the following tasks:

- Added the `Loopback0` interface to both managed nodes
- Added the `Managed by Ansible` description to each `Loopback0` and `GigabitEthernet2` interface

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Did not add a description to the `GigabitEthernet1` interfaces because the `interfaces.yml` file does not define the `GigabitEthernet1` interfaces
- Enabled the `GigabitEthernet2` interface on both managed nodes

3.5. Scroll back through the playbook output and notice that the `Configure L3 interface information` task performed the following tasks:

- Added IPv4 addresses to each `Loopback0` interface
- Added IPv4 addresses to each `GigabitEthernet2` interface

3.6. Scroll back through the playbook output and notice the following in the `Show ansible_facts['net_interfaces'] (post)` task for each managed node.

- You can see the added descriptions for the `Loopback0` and `GigabitEthernet2` interfaces.
- You can see the added IPv4 addresses for the `Loopback0` and `GigabitEthernet2` interfaces.
- You can see that the `GigabitEthernet2` interface displays the `up` operational status.

Important

If either `GigabitEthernet2` interface displays the `down` operational status, then the playbook probably did not wait long enough before querying for interface information. Run the `query_interfaces.yml` playbook to display interface facts without making any changes. Each `GigabitEthernet2` interface should display the `up` operational status.

```
[student@workstation task-loops]$ ansible-navigator run query_interfaces.yml
...output omitted...
```

4. Click **File** > **Close Folder** in VS Code to close the `/home/student/task-loops` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory:

```
[student@workstation task-loops]$ cd
```

Finish

On the `workstation` machine, 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-loops
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Handling Task Failure

Objectives

- Control what happens when a task fails, and what conditions cause a task to fail.

Managing Task Errors in Plays

Ansible evaluates the return code of each task to determine whether the task succeeded or failed. Normally, when a task fails, Ansible immediately skips all subsequent tasks.

Sometimes you might want to have play execution continue even if a task fails. For example, you might expect that a particular task could fail, and you might want to recover by conditionally running some other task. A number of Ansible features can be used to manage task errors.

Ignoring Task Failure

By default, if a task fails, the play is aborted. However, this behavior can be overridden by ignoring failed tasks. You can use the `ignore_errors` keyword in a task to accomplish this.

The following example shows how to use `ignore_errors` in a task to continue playbook execution on the managed node even if the task fails. If the virtual LAN (VLAN) with ID 150 does not exist, then the `cisco.ios.ios_command` module fails, but having `ignore_errors` set to `true` enables execution to continue.

```
tasks:
  - name: Displaying VLAN id 150 information
    cisco.ios.ios_command:
      commands:
        - show vlan id 150
    ignore_errors: true
```

Specifying Task Failure Conditions

You can use the `failed_when` keyword on a task to specify the conditions that indicate a failed task. This is often used with command modules that might successfully execute a command on the managed node, but where the command output indicates a failure.

For example, you can execute a command on the managed node that outputs a message that you can consider an error, and then use that message to define the failed state for the task. The following example shows how you can use the `failed_when` keyword in a task:

```
tasks:
  - name: Displaying configured VLANs
    cisco.ios.ios_command:
      commands: show vlans
    register: vlan_result
    failed_when: "'No Virtual LANs configured' in vlan_result['stdout'][0]"
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Forcing a Task Failure with a Custom Message

You can use the `ansible.builtin.fail` module to force a task failure. You can also use that module to provide a custom failure message for the task.

For example, you can modify the previous example as follows, where the tasks produce the failure and send a customized message when you run the playbook:

```
tasks:
  - name: Displaying configured VLANs
    cisco.ios.ios_command:
      commands: show vlans
    register: vlan_result

  - name: Reporting command failure
    ansible.builtin.fail:
      msg: "Missing VLAN configuration"
    when: "'No Virtual LANs configured' in vlan_result['stdout'][0]"
```

Note

Using the `when: true` condition with the `ansible.builtin.fail` module causes the playbook to terminate immediately at the point where the task is defined.

Controlling If a Task Reports That It Made a Change

When a task makes a change to a managed node, it reports a task result of `changed`. However, sometimes tasks are performed outside of the managed node and you might not want this change to be reported on the managed node.

You can use the `changed_when` keyword to control how a task reports that it has changed something on the managed node. If you set the value of the `changed_when` keyword to `false`, then the task never reports a `changed` state.

For example, the `Copying the vlan configuration to a file` task in the following example gathers information from the managed node, but does not change any configuration on the managed node:

```
tasks:
  - name: Gather IOS interface facts
    cisco.ios.ios_facts:
      gather_network_resources: vlans

  - name: Copying the vlan configuration to a file
    ansible.builtin.copy:
      content: "{{ ansible_network_resources['vlans'] | to_nice_yaml }}"
      dest: "{{ inventory_hostname }}_vlan.yml"
      mode: 0600
```

When you run the playbook for the first time, the task reports a `changed` result for each managed node against which the task runs:

```
TASK [Copying the vlan configuration to a file] ********************************
changed: [iosxe1]
changed: [iosxe2]
changed: [iosxe3]
```

The following example shows how to use the `changed_when` keyword in a task to never report the `changed` result:

```
tasks:
  - name: Gather IOS interface facts
    cisco.ios.ios_facts:
      gather_network_resources: vlans

  - name: Copying the vlan configuration to a file
    ansible.builtin.copy:
      content: "{{ ansible_network_resources['vlans'] | to_nice_yaml }}"
      dest: "{{ inventory_hostname }}_vlan.yml"
      mode: 0600
    changed_when: false
```

Note

The task where you added the `changed_when` keyword with the `false` value can still have the `ok`, `failed`, or `skipped` result when you run the playbook.

Error Handling by Using Ansible Blocks

In playbooks, you can use *blocks* to group tasks that should be executed as a single unit. It is a good practice to add a name to each block to document the purpose of the block.

Use the `block` keyword to create a block and then add tasks to the `block` section. You can use the `when` keyword to apply a condition to the entire block rather than to individual tasks.

```
tasks:
  - name: Configure DNS for Juniper Junos managed nodes
    when: ansible_network_os == 'junipernetworks.junos.junos'
    block:
      - name: Enable netconf service on port 830
        vars:
          ansible_connection: ansible.netcommon.network_cli
        junipernetworks.junos.junos_netconf:
          netconf_port: 830
          state: present

      - name: Configure DNS settings
        junipernetworks.junos.junos_system:
          domain_search:
            - lab.example.com
            - example.com
          name_servers:
            - 172.25.250.220
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Blocks also enable error handling using the `block` keyword in combination with the `rescue` and `always` keywords for the grouped tasks. The `rescue` and `always` keywords have the same indentation in your playbook as the `block` keyword.

If any task in a `block` section fails, then the playbook runs tasks in the `rescue` section. After running tasks in the `block` section, and possibly tasks in the `rescue` section, the playbook runs tasks in the `always` section.

To summarize:

`block`
: Defines the main tasks to run.

`rescue`
: Defines the tasks to run if any task in the `block` section fails.

`always`
: Defines the tasks that always run independently of the success or failure of tasks defined in the `block` or `rescue` sections.

> **Note**
> If you define variables or `when` conditions for a block, then those settings apply to all sections of the block.

As an example of using blocks for error handling, suppose that you have a group of Cisco IOS managed nodes that were not fully configured to use the `172.25.250.220` DNS server. You want to gather a list of misconfigured managed nodes and then configure them to use the `172.25.250.220` DNS sever.

The following example show how to accomplish this by using blocks:

```
tasks:
  - name: Verify DNS for IOS managed nodes
    block:
      - name: Verifying initial DNS configuration (1)
        cisco.ios.ios_command:
          commands: show ip name-server
        register: ios_dns
        failed_when: ios_dns['stdout'][0] != '172.25.250.220' (2)

    rescue:
      - name: Add the misconfigured server to the list (3)
        ansible.builtin.lineinfile:
          path: misconfigured_servers.txt
          line: "{{ inventory_hostname }}"
          create: yes

      - name: Configure DNS settings in failed managed nodes (4)
        cisco.ios.ios_system:
          name_servers: 172.25.250.220

    always: (5)
      - name: Verifying DNS configuration
        cisco.ios.ios_command:
```

```
            commands: show ip name-server
          register: final_dns

        - name: Displaying final DNS configuration
          ansible.builtin.debug:
            var: final_dns['stdout']
```

1. Start by reviewing the expected settings. Each task in the `block` section runs unless a task fails.
2. The `failed_when` keyword is used in the block to specify the task failure condition.
3. The first task in the `rescue` section saves the `inventory_hostname` variable for the managed node in the `misconfigured_servers.txt` text file. This task only runs for the misconfigured managed nodes.
4. The second task in the `rescue` section configures the `172.25.250.220` address as the DNS server in the misconfigured managed nodes. This task only runs for the misconfigured managed nodes.
5. Tasks in the `always` section run regardless of whether any task in the `block` section fails. In this example, the tasks in the `always` section query the current DNS configuration for the managed nodes targeted by the playbook and then display the results of the query.

References

Error Handling in Playbooks - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

Error Handling - Blocks - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_blocks.html#blocks-error-handling

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Recovering From a Failed Configuration

Attempt to configure devices, detect errors, and automatically recover from errors by restoring the original device state and verifying that it has been restored.

Outcomes

- Modify a playbook that configures the Network Time Protocol (NTP) on Juniper Junos and Cisco IOS managed nodes.
- Use blocks to handle the errors found in the NTP managed nodes' settings.
- Specify failure conditions in the block tasks.
- Use the `rescue` section of the block to restore settings on managed nodes that contain an erroneous configuration, and use the `always` section to verify that the initial configuration was restored.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available.

This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start task-failure
```

Instructions

The `lab` command provides the playbooks necessary to complete this exercise.

You must ensure that the correct NTP servers are configured on the managed nodes specified in the `inventory` file.

The following table lists the NTP servers that are present in the configuration on some managed nodes:

Existing NTP Servers

NTP server	Description
`172.25.23.240`	First NTP server
`172.25.23.230`	Second NTP server

The following table lists the NTP servers that should be configured on the managed nodes. These should be the only NTP servers configured, and any other NTP servers should be removed from the configuration.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

New NTP Servers

NTP server	Description
`172.25.254.254`	First NTP server
`172.25.250.220`	Second NTP server

1. Open the `/home/student/task-failure` directory in VS Code and perform the following tasks:

   - Review the inventory file.
   - Review the `ntp_configuration.yml` playbook file and identify the tasks that configure NTP on your managed nodes.
   - Locate the variables that define the NTP servers.
   - Update the values of the NTP server variables to use the IP addresses for the new NTP servers.

   1.1. Open VS Code and then click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **task-failure** and then click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Click the `inventory` file to open it.

   Note that there are two groups of managed nodes in your inventory. The `junos` group corresponds to the Juniper Junos managed nodes. The `ios` group corresponds to the Cisco IOS managed nodes.

```
[junos]
junos[1:2].lab.example.com

[ios]
iosxe[1:2].lab.example.com
```

   1.4. Click the `ntp_configuration.yml` playbook file to open it. Identify the tasks that configure NTP on your managed nodes.

   Notice that the playbook runs against all the managed nodes in the inventory using conditions to establish when to run a task on a managed node.

```
...output omitted...

    - name: Configure NTP on Juniper Junos managed nodes
      junipernetworks.junos.junos_ntp_global:
        config:
          servers:
            - server: "{{ ntp_server_1 }}"
            - server: "{{ ntp_server_2 }}"
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        state: merged
      when: ansible_network_os == "junipernetworks.junos.junos"

    - name: Configure NTP on IOS managed nodes
      cisco.ios.ios_ntp_global:
        config:
          servers:
            - server: "{{ ntp_server_1 }}"
            - server: "{{ ntp_server_2 }}"
        state: merged
      when: ansible_network_os == "cisco.ios.ios"

...output omitted...
```

1.5. The `group_vars/all/ntp.yml` file has the variables used in the playbook to define the NTP servers. Edit the `group_vars/all/ntp.yml` file and change the values of the NTP servers to the IP addresses from the "New NTP Servers" table:

```
---
ntp_server_1: 172.25.254.254
ntp_server_2: 172.25.250.220
```

2. Click the `ntp_configuration.yml` playbook file and analyze the tasks that create backups of the managed node configurations. In case something goes wrong, you have recovery backups.

2.1. View the task that backs up the Juniper Junos managed nodes:

```
...output omitted...
    - name: Enable the NETCONF service
      vars:
        ansible_connection: ansible.netcommon.network_cli
      junipernetworks.junos.junos_netconf:
        listens_on: 830
        state: present
      when: ansible_network_os == "junipernetworks.junos.junos"

    - name: Back up the Juniper Junos configurations
      junipernetworks.junos.junos_config:
        backup: true
        backup_options:
          filename: "{{ inventory_hostname }}.txt"
      when: ansible_network_os == "junipernetworks.junos.junos"

...output omitted...
```

2.2. View the task that backs up the IOS managed nodes:

```
...output omitted...
    - name: Back up the Juniper Junos configurations
      junipernetworks.junos.junos_config:
        backup: true
        backup_options:
```

```
            filename: "{{ inventory_hostname }}.txt"
        when: ansible_network_os == "junipernetworks.junos.junos"

      - name: Back up the IOS configurations
        cisco.ios.ios_config:
          backup: true
          backup_options:
            filename: "{{ inventory_hostname }}.txt"
        when: ansible_network_os == "cisco.ios.ios"


...output omitted...
```

Note

By default, a directory named `backup` is created and the configuration backups are stored in that directory.

3. Run the `playbooks/backup_configs.yml` playbook to create an initial backup of the managed node configurations.

   3.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/task-failure` directory in a GNOME terminal, and then run the `playbooks/backup_configs.yml` playbook:

```
[student@workstation task-failure]$ ansible-navigator run \
playbooks/backup_configs.yml
...output omitted...
```

Note

Each time you run the `ntp_configuration.yml` playbook, it makes a backup of the configuration found on the managed nodes. Because the playbook changes the NTP configuration, when you run the playbook again the backup is overwritten with the new configuration backup.

This behavior is desired for the final version of your playbook. During its development, however, you must keep a copy of the first backup of your managed nodes.

   3.2. Copy the first backup of your managed nodes from the `backup` directory into the current project directory:

```
[student@workstation task-failure]$ cp -v playbooks/backup/*.txt .
'playbooks/backup/iosxe1.lab.example.com.txt' -> './iosxe1.lab.example.com.txt'
'playbooks/backup/iosxe2.lab.example.com.txt' -> './iosxe2.lab.example.com.txt'
'playbooks/backup/junos1.lab.example.com.txt' -> './junos1.lab.example.com.txt'
'playbooks/backup/junos2.lab.example.com.txt' -> './junos2.lab.example.com.txt'
```

4. Run the `ntp_configuration.yml` playbook to back up the configurations and configure NTP on the managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

After you run the `ntp_configuration.yml` playbook, run the `playbooks/junos_ntp_verify.yml` and `playbooks/ios_ntp_verify.yml` playbooks to verify the NTP configuration on the managed nodes.

4.1. Run the `ntp_configuration.yml` playbook by using the `ansible-navigator run` command.

The playbook runs successfully, first creating a backup of the managed nodes and then configuring the managed nodes to use the new NTP servers:

```
[student@workstation task-failure]$ ansible-navigator run \
ntp_configuration.yml

PLAY [Configure NTP on managed nodes] *********************************************

TASK [Enable the NETCONF service] **************************************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
ok: [junos1.lab.example.com]
ok: [junos2.lab.example.com]

TASK [Back up the Juniper Junos configurations] ***********************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
changed: [junos2.lab.example.com]
changed: [junos1.lab.example.com]

TASK [Back up the IOS configurations] *********************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
changed: [iosxe2.lab.example.com]
changed: [iosxe1.lab.example.com]

TASK [Configure NTP on Juniper Junos managed nodes] ******************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

TASK [Configure NTP on IOS managed nodes] *****************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
changed: [iosxe2.lab.example.com]
changed: [iosxe1.lab.example.com]

PLAY RECAP *************************************************************************
iosxe1.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0    ...
iosxe2.lab.example.com     : ok=2    changed=2    unreachable=0    failed=0    ...
junos1.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0    ...
junos2.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0    ...
```

4.2. Run the `playbooks/junos_ntp_verify.yml` playbook to verify the current NTP configuration on the Juniper Junos managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note that the resulting verification is not what was expected. The `junos1.lab.example.com` managed node does not have the desired configuration:

```
[student@workstation task-failure]$ ansible-navigator run \
playbooks/junos_ntp_verify.yml
...output omitted...
TASK [Display NTP configuration] ************************************************
ok: [junos1.lab.example.com] => {
    "junos_ntp['stdout_lines']][0]": [
            "server 172.25.23.240;",
            "server 172.25.23.230;",
            "server 172.25.254.254;",
            "server 172.25.250.220;"

    ]
}
ok: [junos2.lab.example.com] => {
    "junos_ntp['stdout_lines']][0]": [
            "server 172.25.254.254;",
            "server 172.25.250.220;"
    ]
}
...output omitted...
```

If the `ntp_configuration.yml` playbook runs on managed nodes without any NTP server configuration, then the managed nodes are configured as expected. If the `ntp_configuration.yml` playbook runs on managed nodes with an existing NTP server configuration, then the new NTP servers are added to the previous NTP server configuration.

4.3. Run the `playbooks/ios_ntp_verify.yml` playbook to verify the current NTP configuration on the IOS managed nodes.

As with the `junos1.lab.example.com` managed node, the NTP configuration of the `iosxe2.lab.example.com` managed node does not have the desired configuration:

```
[student@workstation task-failure]$ ansible-navigator run \
playbooks/ios_ntp_verify.yml
...output omitted...

TASK [Display NTP configuration] ************************************************
ok: [iosxe1.lab.example.com] => {
    "ios_ntp['stdout_lines']][0]": [
            "ntp server 172.25.254.254",
            "ntp server 172.25.250.220"
    ]
}
ok: [iosxe2.lab.example.com] => {
    "ios_ntp['stdout_lines']][0]": [
            "ntp server 172.25.23.240",
            "ntp server 172.25.23.230",
            "ntp server 172.25.254.254",
            "ntp server 172.25.250.220"
```

```
    ]
}
...output omitted...
```

5. Restore the initial configuration of your managed nodes using the backups that you previously created. The `junos_recover.yml` and `ios_recover.yml` playbooks in the `playbooks` directory enable you to restore the backups to your managed nodes.

   5.1. Run the `playbooks/junos_recover.yml` playbook to restore the Juniper Junos backup configuration:

```
[student@workstation task-failure]$ ansible-navigator run \
playbooks/junos_recover.yml
...output omitted...
```

   5.2. Run the `playbooks/ios_recover.yml` playbook to restore the Cisco IOS backup configuration:

```
[student@workstation task-failure]$ ansible-navigator run \
playbooks/ios_recover.yml
...output omitted...
```

6. Create another playbook by copying the `ntp_configuration.yml` file to the `ntp_configuration_blocks.yml` file. Modify the new `ntp_configuration_blocks.yml` playbook to use Ansible blocks by performing the following tasks:

   - Create a `block` section that nests the task that configures NTP on the Juniper Junos managed nodes. Create another `block` section that nests the task that configures the NTP service on the IOS managed nodes.
   - Add a task to each of the blocks to verify that the previous NTP servers are no longer configured on the managed nodes. Use the `failed_when` failure condition for this purpose.
   - As a good practice, name the blocks.
   - For each block, move the condition for the `ansible_network_os` variable to apply to all the tasks in the block.

> **Note**
> For your convenience, you can find the already completed `ntp_configuration_blocks.yml` playbook file under the `playbooks` directory.

   6.1. Copy the `ntp_configuration.yml` file to the `ntp_configuration_blocks.yml` file:

```
[student@workstation task-failure]$ cp ntp_configuration.yml \
ntp_configuration_blocks.yml
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6.2. Modify the `ntp_configuration_blocks.yml` playbook to add the required block sections.

The `block` section for the Juniper Junos managed nodes must contain the following content:

```
...output omitted...
      when: ansible_network_os == "cisco.ios.ios"


    - name: Reconfiguring Juniper managed nodes
      block:
        - name: Configure NTP on Juniper Junos managed nodes
          junipernetworks.junos.junos_ntp_global:
            config:
              servers:
                - server: "{{ ntp_server_1 }}"
                - server: "{{ ntp_server_2 }}"
            state: merged

        - name: Verify previous NTP configuration no longer exists
          junipernetworks.junos.junos_command:
            commands:
              - show configuration system ntp
          register: ntp_servers
          failed_when: >
            ('server 172.25.23.240;' in ntp_servers['stdout_lines'][0])
            or
            ('server 172.25.23.230;' in ntp_servers['stdout_lines'][0])

      when: ansible_network_os == "junipernetworks.junos.junos"

    - name: Configure NTP on IOS managed nodes
...output omitted...
```

Pay attention to the fact that the line with the `when` condition must have the same indentation as the `block` keyword to apply to all the tasks in the block.

6.3. The `block` section for the IOS managed nodes must have the following content:

```
...output omitted...
      when: ansible_network_os == "junipernetworks.junos.junos"

    - name: Reconfiguring IOS managed nodes
      block:
        - name: Configure NTP on IOS managed nodes
          cisco.ios.ios_ntp_global:
            config:
              servers:
                - server: "{{ ntp_server_1 }}"
                - server: "{{ ntp_server_2 }}"
            state: merged

        - name: Verify previous NTP configuration no longer exists
          cisco.ios.ios_command:
```

```
            commands:
              - show run | include ntp
          register: ntp_servers
          failed_when: >
            ('ntp server 172.25.23.240' in ntp_servers['stdout_lines'][0])
            or
            ('ntp server 172.25.23.230' in ntp_servers['stdout_lines'][0])

      when: ansible_network_os == "cisco.ios.ios"
```

7. Create a `rescue` section for the `Reconfiguring Juniper managed nodes` and `Reconfiguring IOS managed nodes` blocks in your playbook.

   - If Ansible detects errors when the playbook attempts to configure NTP, the corresponding backup must be restored on the managed nodes.
   - The `rescue` section must be placed after the `block` section and before the `when` condition for the `ansible_network_os` variable.

> **Note**
> Use the path to the first backup of your managed nodes for the task that restores its configuration. In a later step you modify this path to use the backup created by the playbook.

7.1. The `rescue` section in your playbook must contain the following content:

```
...output omitted...
            ('server 172.25.23.230;' in ntp_servers['stdout_lines'][0])

      rescue:
        - name: Remove all existing configuration
          junipernetworks.junos.junos_ntp_global:
            config:
            state: deleted

        - name: Restore the Juniper Junos configurations
          junipernetworks.junos.junos_config:
            update: replace
            src: "{{ inventory_hostname }}.txt"

      when: ansible_network_os == "junipernetworks.junos.junos"
...output omitted...
```

7.2. The `rescue` section for the block related to IOS managed nodes must contain the following content:

```
...output omitted...

            ('ntp server 172.25.23.230' in ntp_servers['stdout_lines'][0])

      rescue:
        - name: Remove all existing configuration
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
          cisco.ios.ios_ntp_global:
            state: deleted

        - name: Restore the IOS configurations
          cisco.ios.ios_config:
            src: "{{ inventory_hostname }}.txt"

      when: ansible_network_os == "cisco.ios.ios"
```

Important

Due to the way certain network platforms handle restoring configuration, some `*_config` modules merge the backup configuration with the active configuration on the target device instead of overwriting it.

For examples of how to restore network configurations, see https://github.com/network-automation/toolkit/tree/master/roles/restore.

8. Create an `always` section for each of the blocks in the `ntp_configuration.yml` playbook.

- The tasks in the `always` section of the blocks must verify and display how the NTP configurations look after running the block and recovery tasks against the managed nodes.
- The `always` section must be placed after the `rescue` section in each of the blocks and before the `when` condition for the `ansible_network_os` variable.

8.1. Create an `always` section for the block related to the Juniper Junos managed nodes.

The `always` section must contain the following content:

```
...output omitted...

      always:
        - name: Verifying NTP configuration
          junipernetworks.junos.junos_command:
            commands:
              - show configuration system ntp
          register: junos_ntp

        - name: Display NTP configuration
          ansible.builtin.debug:
            var: junos_ntp['stdout_lines'][0]

      when: ansible_network_os == "junipernetworks.junos.junos"

...output omitted...
```

8.2. Create an `always` section for the block related to the Cisco IOS managed nodes.

The `always` section must contain the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
...output omitted...

      always:
        - name: Verifying NTP configuration
          cisco.ios.ios_command:
            commands:
              - show run | include ntp
          register: ios_ntp

        - name: Display NTP configuration
          ansible.builtin.debug:
            var: ios_ntp['stdout_lines'][0]

      when: ansible_network_os == "cisco.ios.ios"
```

9. Run the `ntp_configuration_blocks.yml` playbook.

If Ansible generates an error when running any of the tasks in the `block` section, then Ansible runs the tasks in the `rescue` section to restore the managed nodes to their original state. The tasks in each `always` section verify the current NTP configuration on the managed nodes.

When the `ntp_configuration_blocks.yml` playbook runs as expected, modify it so that the restore tasks in the `rescue` section of the Ansible blocks use the path to the backups created by the playbook instead of the first backup of your managed nodes.

9.1. Run the `ntp_configuration_blocks.yml` playbook by using the `ansible-navigator run` command:

```
[student@workstation task-failure]$ ansible-navigator run \
ntp_configuration_blocks.yml
```

The `ntp_configuration_blocks.yml` playbook creates a back up of the current NTP configuration for the Juniper Junos and IOS managed nodes:

```
...output omitted...

TASK [Back up the Juniper Junos configurations] **********************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
ok: [junos2.lab.example.com]
ok: [junos1.lab.example.com]

TASK [Back up the IOS configurations] ********************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
changed: [iosxe2.lab.example.com]
changed: [iosxe1.lab.example.com]

...output omitted...
```

The tasks in the first block in your playbook start running against the Juniper Junos managed nodes. The playbook configures the NTP service and then verifies that the previous NTP configuration no longer exists.

The failure condition in the `Verify previous NTP configuration no longer exists` task makes the task fail for the `junos1.lab.example.com` managed node:

```
...output omitted...

TASK [Configure NTP on Juniper managed nodes] ************************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
changed: [junos1.lab.example.com]
changed: [junos2.lab.example.com]

TASK [Verify previous NTP configuration no longer exists] ************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
fatal: [junos1.lab.example.com]: FAILED! => {"changed":
 false, "failed_when_result": true, "stdout": ["server 172.25.23.240;\nserver
 172.25.23.230;\nserver 172.25.254.254;\nserver 172.25.250.220;"], "stdout_lines":
 [["server 172.25.23.240;", "server 172.25.23.230;", "server 172.25.254.254;",
 "server 172.25.250.220;"]]}
ok: [junos2.lab.example.com]

...output omitted...
```

The `ntp_configuration_blocks.yml` playbook uses the backup of the `junos1.lab.example.com` managed node to restore its original configuration. This rescue task only runs against the `junos1.lab.example.com` managed node.

Note

If no tasks fail in the section under the `block` keyword, then this rescue task does not run.

```
...output omitted...

TASK [Restore the Juniper Junos configurations] **********************************
changed: [junos1.lab.example.com]

...output omitted...
```

The `ntp_configuration_blocks.yml` playbook displays the NTP configuration for the Juniper Junos managed nodes:

```
...output omitted...

TASK [Display NTP configuration] *************************************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
ok: [junos1.lab.example.com] => {
    "junos_ntp['stdout_lines'][0]": [
            "server 172.25.23.240;",
            "server 172.25.23.230;"
    ]
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
}
ok: [junos2.lab.example.com] => {
    "junos_ntp['stdout_lines']['0']": [
            "server 172.25.254.254;",
            "server 172.25.250.220;"
    ]
}

...output omitted...
```

The tasks in the second block in your playbook start running against the IOS managed nodes. The playbook configures the NTP service and then verifies that the previous NTP configuration no longer exists.

The failure condition in the `Verify previous NTP configuration no longer exists` task makes the task fail for the `iosxe2.lab.example.com` managed node:

```
...output omitted...

TASK [Configure NTP on IOS managed nodes] ****************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

TASK [Verify previous NTP configuration no longer exists] ************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
ok: [iosxe1.lab.example.com]
fatal: [iosxe2.lab.example.com]: FAILED! => {"changed":
 false, "failed_when_result": true, "stdout": ["ntp server 172.25.23.240\nntp
 server 172.25.250.220\nntp server 172.25.23.230\nntp server 172.25.254.254"],
 "stdout_lines": [["ntp server 172.25.23.240", "ntp server 172.25.250.220", "ntp
 server 172.25.23.230", "ntp server 172.25.254.254"]]}

...output omitted...
```

The `ntp_configuration_blocks.yml` playbook uses the backup of the `iosxe2.lab.example.com` managed node to restore its original configuration. This rescue task only runs against the `iosxe2.lab.example.com` managed node.

```
...output omitted...

TASK [Restore the IOS configurations] ********************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines
 should be similar to how they appear if present in the running
configuration on device including the indentation
changed: [iosxe2.lab.example.com]

...output omitted...
```

The `ntp_configuration_blocks.yml` playbook displays the NTP configuration for the IOS managed nodes:

```
...output omitted...

TASK [Display NTP configuration] *************************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
ok: [iosxe2.lab.example.com] => {
    "ios_ntp['stdout_lines']][0]": [
            "ntp server 172.25.23.240",
            "ntp server 172.25.23.230"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "ios_ntp['stdout_lines']][0]": [
            "ntp server 172.25.254.254",
            "ntp server 172.25.250.220"
    ]
}

...output omitted...
```

9.2. Edit the `ntp_configuration_blocks.yml` playbook. The path for the backup in the `Restore the Juniper Junos configurations` and `Restore the IOS configuration` tasks in the `rescue` section of the blocks must use the backup file created by the `ntp_configuration_blocks.yml` playbook.

- The `rescue` section for the block related to Juniper Junos managed nodes must contain the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    rescue:
      - name: Remove all existing configuration
        junipernetworks.junos.junos_ntp_global:
          config:
          state: deleted

      - name: Restore the Juniper Junos configurations
        junipernetworks.junos.junos_config:
          src: "backup/{{ inventory_hostname }}.txt"
```

- The `rescue` section for the block related to Cisco IOS managed nodes must contain the following content:

```
    rescue:
      - name: Remove all existing configuration
        cisco.ios.ios_ntp_global:
          state: deleted

      - name: Restore the IOS configurations
        cisco.ios.ios_config:
          src: "backup/{{ inventory_hostname }}.txt"
```

9.3. Run the `ntp_configuration_blocks.yml` playbook again and verify that the playbook works as expected:

```
[student@workstation task-failure]$ ansible-navigator run \
ntp_configuration_blocks.yml
...output omitted...


TASK [Display NTP configuration] ************************************************
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
ok: [junos1.lab.example.com] => {
    "junos_ntp['stdout_lines']][0]": [
            "server 172.25.23.240;",
            "server 172.25.23.230;"
    ]
}
ok: [junos2.lab.example.com] => {
    "junos_ntp['stdout_lines']][0]": [
            "server 172.25.254.254;",
            "server 172.25.250.220;"
    ]
}

...output omitted...

TASK [Display NTP configuration] ************************************************
skipping: [junos1.lab.example.com]
skipping: [junos2.lab.example.com]
ok: [iosxe2.lab.example.com] => {
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    "ios_ntp['stdout_lines'][0]": [
            "ntp server 172.25.23.240",
            "ntp server 172.25.23.230"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "ios_ntp['stdout_lines'][0]": [
            "ntp server 172.25.254.254",
            "ntp server 172.25.250.220"
    ]
}

...output omitted...
```

Note that after executing the playbook the managed nodes are correctly configured with only the new NTP servers.

10. Close the `/home/student/task-failure` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

10.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/task-failure` directory.

10.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation task-failure]$ cd
```

Finish

On the `workstation` machine, 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-failure
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Building a Workflow Job Template

Objectives

- Create a workflow job template in automation controller that launches a branching chain of jobs based on the success or failure of each job in the workflow sequence.

Workflow Job Templates

Job templates run single Ansible Playbooks as jobs. As the use of Red Hat Ansible grows in an organization, so does the number of Ansible Playbooks it has. Each playbook typically performs a set of tasks associated with a certain function.

Instead of writing one large playbook to automate a complex operation, you might want to run several playbooks in sequence. For example, your goal might be to configure NTP on your network devices. This process might first require testing connectivity to your network devices. You might then use a separate playbook to back up the current configurations. Finally, you might use another playbook to configure NTP. In other words, there might be a particular workflow that you need to follow for the process to succeed.

You could manage this in automation controller by manually launching multiple jobs in sequence. But, you have to execute the jobs in the correct order, as defined by your workflow, for everything to work correctly. For example, you might have the following workflow:

1. Run the connectivity jobs first.
2. Run the backup jobs only if the connectivity jobs completed successfully.
3. Run the configuration change jobs only if the connectivity and backup jobs completed successfully.

Finally, if one of these playbooks fails, you might need to run other playbooks to recover from the failure. For example, you might need to restore your network devices using the backup that was created if the playbook used to configure NTP fails.

Automation controller supports *workflow job templates* to make it easier to automate complex operations. A workflow job template connects multiple job templates into a single workflow. When launched, a workflow job template launches a job using the first job template. Depending on whether that job succeeds or fails, the workflow job template determines which job template to launch next. This allows you to launch a sequence of jobs, and to automatically take recovery steps if a job fails.

You can start workflow job templates in many ways:

- Manually, from the automation controller web UI
- As a scheduled job
- By an external program using the automation controller API

Workflow job templates do not only run job templates serially. Using the graphical workflow editor, workflow job templates can chain together multiple job templates and run different job templates depending on whether the previous one succeeded or failed.

Creating Workflow Job Templates

You need to create a workflow job template before associating a workflow with it.

You create a workflow job template much the same way you create a job template:

1. Navigate to **Resources** > **Templates** and then click **Add** > **Add workflow template**.

2. At a minimum, enter a name for the workflow template and then save the template.

After creating a workflow job template, you can define an associated workflow by using the workflow visualizer.

Using the Workflow Visualizer

After you have created a workflow job template, the workflow visualizer becomes active on the **Visualizer** tab of the workflow job template. The workflow visualizer is a graphical interface for defining the node types to incorporate in a workflow, as well as the decision tree structure that chains the job templates together.

The workflow visualizer contains a single **Start** button that represents the starting point for the workflow. Click **Start** to start editing the workflow. The workflow visualizer displays a list of node types that you can use for the first step of the workflow. These node types are described in the following table. When added to the workflow, each node is identified by a symbol in the lower left of the node.

Node type	Purpose	Symbol
Approval	Request manual approval before workflow continues.	||
Inventory Source Sync	Synchronize inventory from a specified source.	I
Job Template	Run a job template.	JT
Project Sync	Synchronize a specified project.	P
Management Job	Run one of the management jobs.	M
Workflow Job Template	Run another workflow job template.	W

Being able to add job templates and other resources ensures that, for example, project and inventory resources are synchronized before the workflow job uses job templates that depend on those resources.

You can select the desired node type, the specific resource, and then save it to add it as the first node in the workflow.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

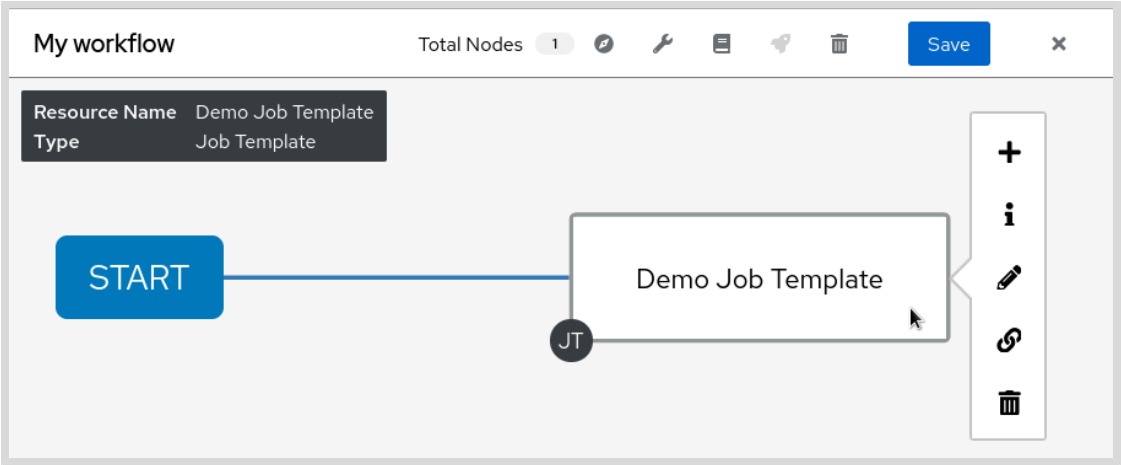


Figure 5.1: Adding a job template node and hovering over it

After you add a workflow node, you can hover over it to open a column of icons. You can use these icons to add another node, view node details, edit the node, link to an available node, or delete the node.

When you add subsequent nodes, the **Run** page opens, prompting for the conditions under which the node should execute. You can choose one of three relationships between the new node and the preceding node:

Run condition	Node relationship
On Success	The node resource is executed upon successful completion of the actions associated with the previous node.
On Failure	The node resource is executed upon failure of the actions associated with the previous node.
Always	The node resource is executed regardless of the outcome of the actions associated with the previous node.

Select the desired condition and then click **Next** to select the new node type and the specific resource. Click **Save** to add the new node.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

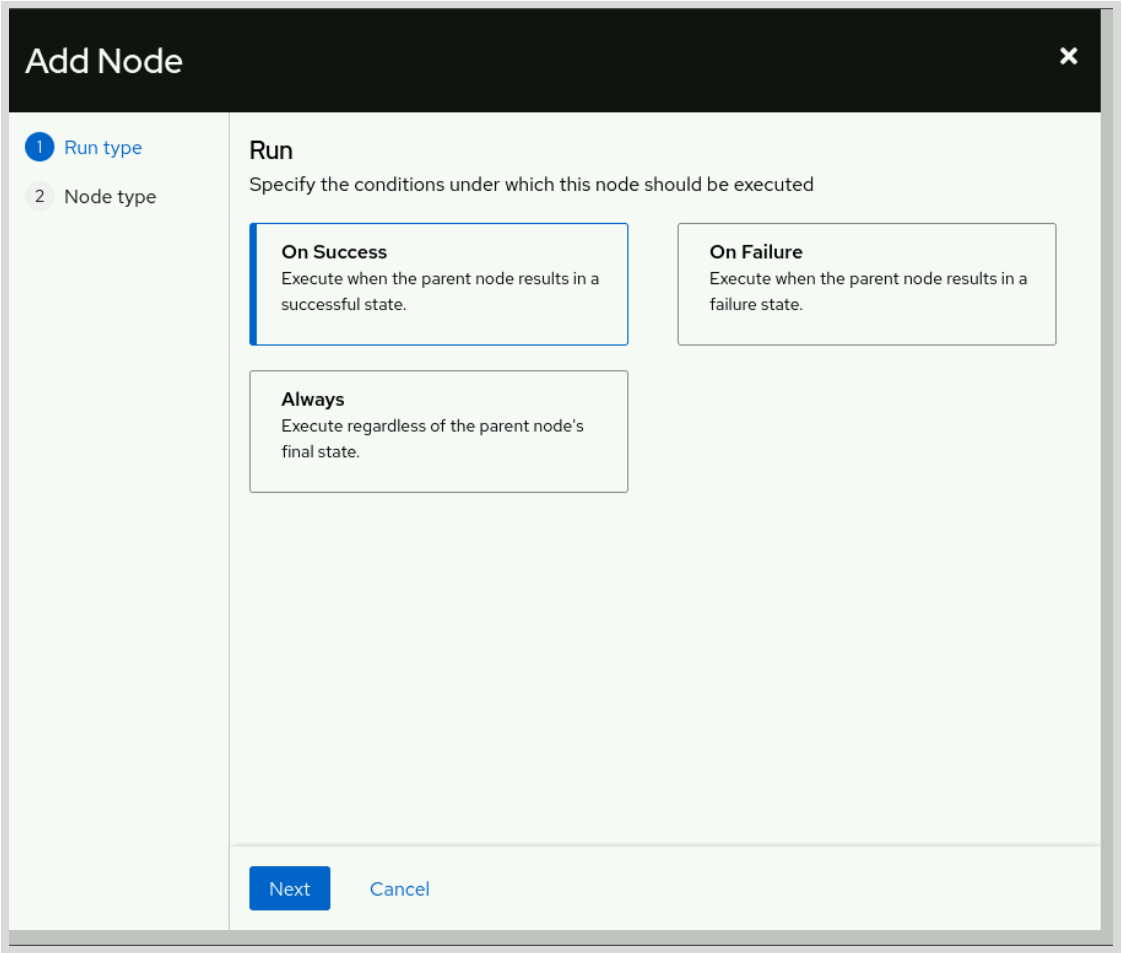


Figure 5.2: Adding an "On Success" node to the workflow

A node can have more than one child node. For example, you can add a child node to a parent node using an **On Success** association type. You can add a second child node to the same parent using an **On Failure** association type. This creates a branch in the workflow structure such that one course of action is taken upon the success of an action, and a different course is taken upon failure.

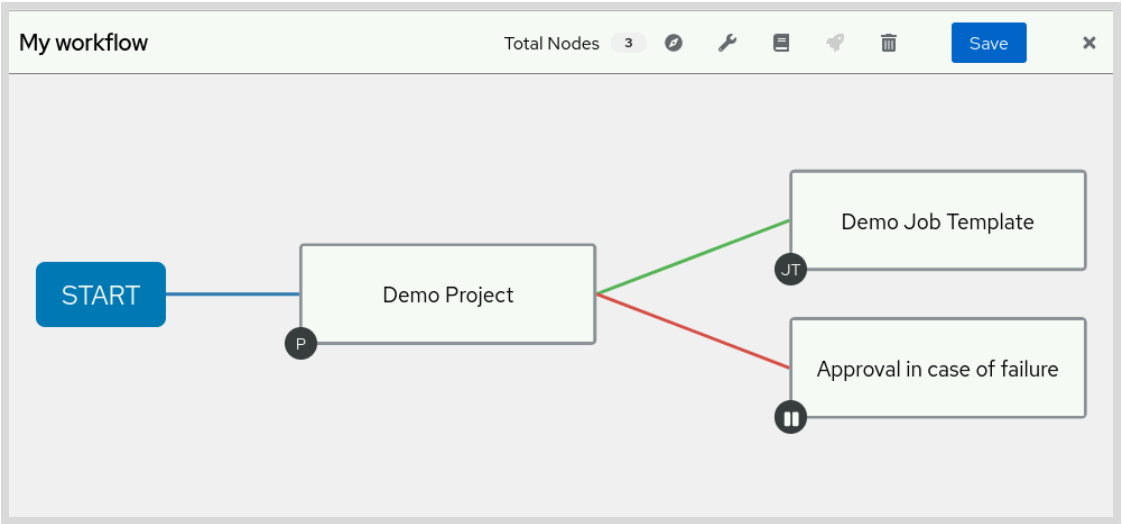


Figure 5.3: Adding multiple child nodes to the workflow

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

As you add nodes to a workflow, differently colored lines connecting the nodes in the workflow editor indicate the relationships between parent and child nodes:

- A green line indicates an **On Success** relationship.
- A red line indicates an **On Failure** relationship.
- A blue line indicates an **Always** relationship.

In addition, if you hover your mouse pointer over a line, then the web UI displays the relationship type as text in the upper left corner of the visualizer.

> **Important**
> You can change the type of relationship between two nodes in the visualizer by hovering over the line connecting the two nodes, and then clicking the pencil icon to edit the relationship. You can also hover over the line connecting two nodes and click the plus icon to add a new node between the two nodes.

After you create the entire decision tree structure of the workflow in the workflow editor, click **Save** to save the workflow.

The new workflow job template displays on the same page as the job templates but with a different type (**Workflow Job Template**) and with an icon that directs you to the workflow visualizer.

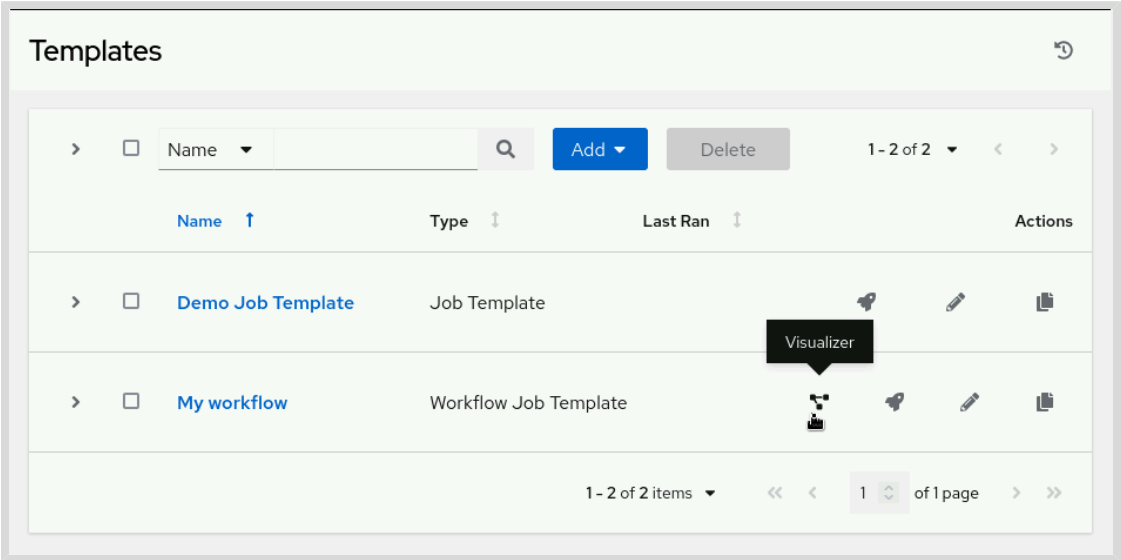


Figure 5.4: Visualizer icon for a workflow job template

Adding Multiple Nodes with the Same Relationship

You can have a node that has the same relationship with several subsequent nodes. For example, one node can be configured with **Always** relationships to several nodes.

In the following example, automation controller always runs the two job template nodes after running the project synchronization job for the `Demo Project` node. Because the `Check Cisco Devices` and `Check Juniper Devices` nodes are separate job templates, automation controller runs the jobs in parallel.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

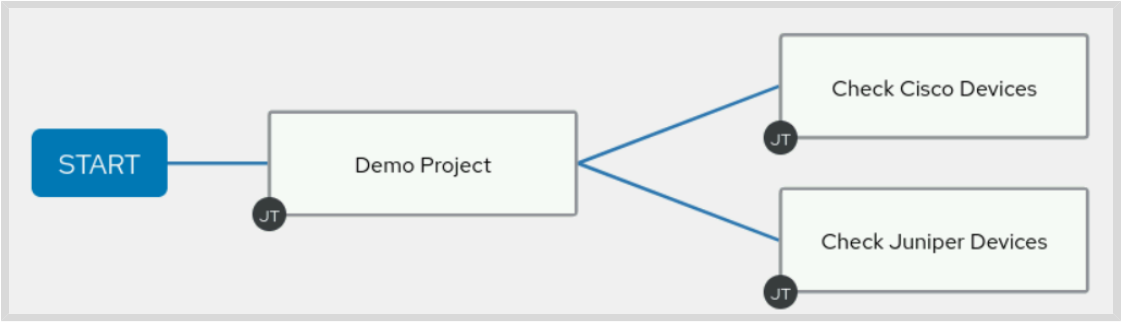


Figure 5.5: Multiple nodes with the same relationship to the preceding node

Creating Convergent Nodes

Nodes can have multiple parent nodes. For example, you can have a node that runs if *any* parent node succeeds, or only if *all* parent nodes succeed.

If you have a configuration that uses a multiple node relationship (as described in the previous section), then you can configure convergent nodes with the following steps:

- Hover over any of the parent nodes and add a new node.
- Hover over one of the other parent nodes and click the chain icon.
- Connect the dashed gray arrow that follows your mouse pointer to the node you just added and specify when the node should run (always, on success, or on failure). That node now has two parent nodes, and is a *convergent node*.

If you edit the convergent node, on its **Node type** page there is a field labeled **Convergence**. This controls the conditions that cause the node to run.

- `Any` is the default setting. After all parent nodes complete, if any parent node triggers the convergent node, then automation controller runs the job for the convergent node.
- `All` requires that all parent nodes trigger the convergent node, based on their relationship, for the convergent node to run.

In the following example from the visualizer, the convergent node `Apply configuration` only runs if both of its parent nodes report success.

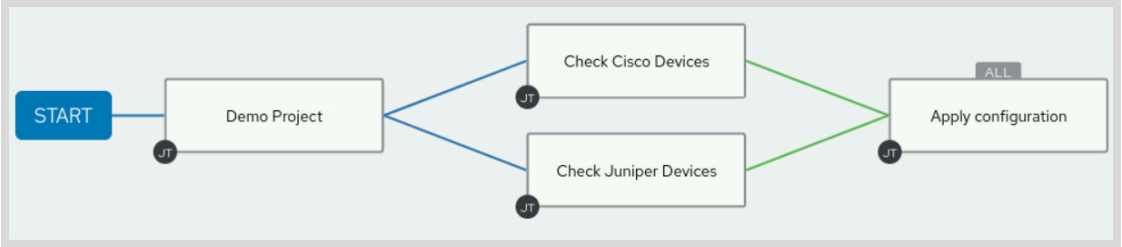


Figure 5.6: Convergent node with multiple parent nodes

Workflow Job Template Surveys

Workflow job templates have access to many of the features that have been discussed for job templates. Like job templates, you can add surveys to workflow job templates so that users can interactively set extra variables.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note
When you add a survey to a workflow job template, the resulting extra variables are accessible by every job executed by the workflow.

Launching Workflow Jobs

The procedure to launch a workflow job template is similar to how you launch a job template:

1. Log in to automation controller.
2. Navigate to **Resources** > **Templates** to list the available templates.
3. Click the **Launch Template** icon for the desired workflow job template.

Evaluating Workflow Job Execution

After you launch a workflow job template, the automation controller web UI redirects you to the **Output** tab for this workflow job template, and displays the progress and result in the workflow visualizer.

The workflow job page consists of two tabs. The **Details** tab displays details of the parameters for the workflow job template. The **Output** tab displays the progress of the job through the steps in the workflow.

As each step in the workflow job completes, the web UI indicates whether the step succeeded or failed. If the step succeeds, then the web UI outlines the node in green and adds a box to the node in which the upper half of the box is colored green. If the step fails, then the web UI outlines the node in red and adds a box to the node in which the lower half of the box is colored red. Within each successful or failed node, the web UI displays the amount of time it took to complete the step. Nodes that are not run are outlined in gray.

Progress from one node to another is represented in the same way as it is in the visualizer; by colored lines indicating the decision responsible for the progression:

- Green indicates that the next node runs if the previous node succeeds.
- Red indicates that the next node runs if the previous node fails.
- Blue indicates that the next node always runs.

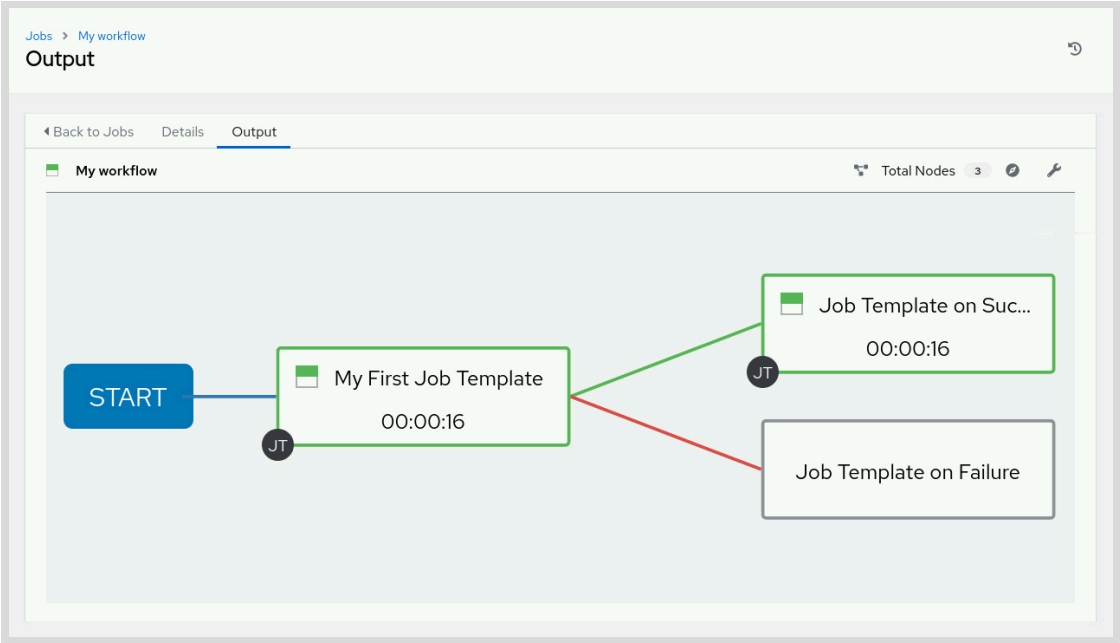


Figure 5.7: Workflow job progress

You can examine the live output of the running workflow job or review the output after the job completes. In a workflow diagram that represents a currently running job or completed job, you can click each node to display the results and standard output for the job run.

References

Automation Controller User Guide - Workflows
https://docs.ansible.com/automation-controller/latest/html/userguide/workflows.html

Automation Controller User Guide - Workflow Job Templates
https://docs.ansible.com/automation-controller/latest/html/userguide/workflow_templates.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Back up, Configure, Recover from Errors, and Restore from Backups

Create a workflow job template consisting of a branching chain of jobs and launch it using the web UI of automation controller.

Outcomes

- Create a workflow job template.
- Run a job template when the previously run job template succeeds.
- Run a job template when the previously run job template fails.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available.

This command also creates the project, inventory, and credentials in automation controller so that you can complete the exercise.

```
[student@workstation ~]$ lab start task-workflow
```

Instructions

1. Change to the `~/task-workflow` directory. Run the `ios_backup.yml` playbook.

   1.1. In a terminal, change to the `~/task-workflow` directory:

```
[student@workstation ~]$ cd ~/task-workflow
```

   1.2. Run the `ios_backup.yml` playbook:

```
[student@workstation task-workflow]$ ansible-navigator run ios_backup.yml

PLAY [Backup IOS XE configurations] ******************************************

TASK [Backup IOS XE configurations] ******************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *******************************************************************
iosxe1.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=1    changed=1    unreachable=0    failed=0 ...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password. Create a workflow job template named `IOS - Add and Verify NTP` that uses the `Common` inventory.

2.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

2.2. Navigate to **Resources** > **Templates** and then click **Add** > **Add workflow template**. On the next page, add the following details and then click **Save**.

Field	Value
Name	`IOS - Add and Verify NTP`
Organization	`Default`
Inventory	Common

3. Add a `Project Sync` node named `Synchronize Project` to synchronize the `IOS_NTP` project.

3.1. On the workflow visualizer page, click **Start**.

3.2. On the **Add Node** page, select **Project Sync** in the **Node Type** list.

3.3. Select the **IOS_NTP** project, scroll down to the **Node Alias** field, and enter `Synchronize Project` in that field.

3.4. Click **Save**. On the workflow visualizer page under `Synchronize Project`, notice the **P** symbol that indicates it is a project sync node.

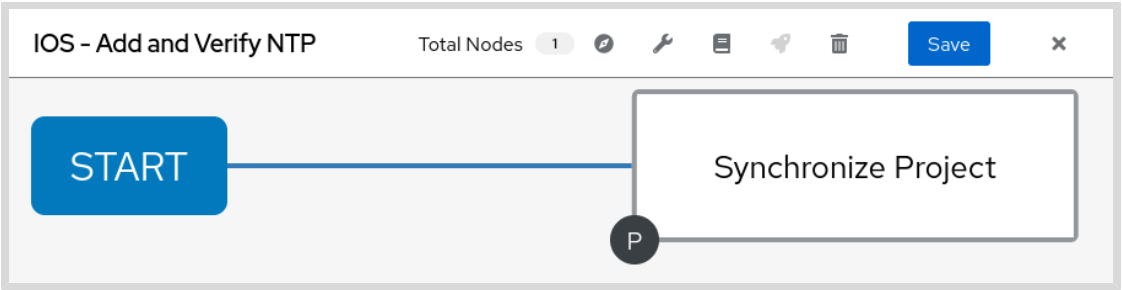


Figure 5.8: Project synchronization node

4. Add a `Job Template` node that launches the `IOS - Add NTP Configuration` job template to configure NTP if the `Synchronize Project` node succeeds.

4.1. In the workflow visualizer, hover over the **Synchronize Project** node and click the **+** icon to add a new node.

4.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

4.3. Select the **IOS - Add NTP Configuration** job template, scroll down to the **Node Alias** field, and enter `Configure NTP` in that field.

4.4. Click **Save**. Notice the **JT** symbol, which indicates it is a job template node.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

You might need to increase the display resolution on the `workstation` machine, decrease the web browser zoom, or both, in order to see the entire visualizer display screen.

If your mouse has a scroll wheel, then you can use the scroll wheel to zoom in and out of the visualizer.

5. Add a job template node that launches the `IOS - Verify NTP` job template to verify that NTP is enabled if the `Configure NTP` node succeeds.

   5.1. In the workflow visualizer, hover over the **Configure NTP** node and then click the **+** icon to add a new node.

   5.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

   5.3. Select the **IOS - Verify NTP** job template, scroll down to the **Node Alias** field, and enter `Verify NTP` in that field.

   5.4. Click **Save**.

6. Add a job template node that launches the `IOS - Revert NTP Configuration` job template to revert the NTP configuration if the `Configure NTP` node *fails*.

   6.1. In the workflow visualizer, hover over the **Configure NTP** node and then click the **+** icon to add a new node.

   6.2. On the **Run type** page, select the **On Failure** condition and then click **Next**.

   6.3. Select the **IOS - Revert NTP Configuration** job template, scroll down to the **Node Alias** field, and enter `Revert NTP Configuration` in that field.

   6.4. Click **Save**.

   6.5. Click **Save** again in the upper-right corner to save this workflow template.

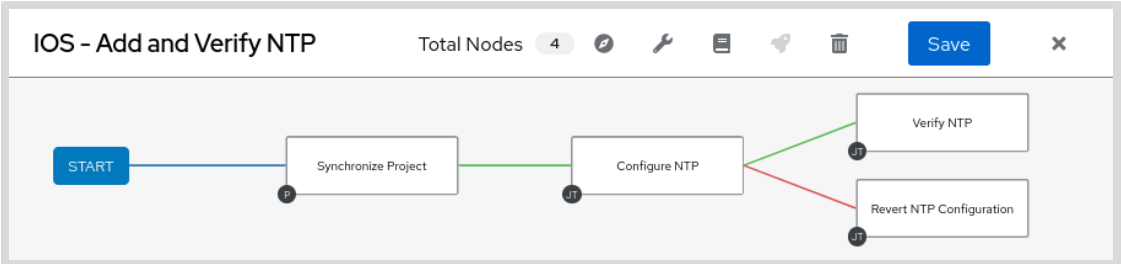


Figure 5.9: Workflow job template

7. Launch the workflow job template and observe the result. The `Configure NTP` node fails and triggers the `Revert NTP Configuration` node.

   7.1. Navigate to **Resources** > **Templates**.

   7.2. Click the **Launch Template** icon for the **IOS - Add and Verify NTP** workflow job template.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

7.3. Observe that the **Configure NTP** node fails and triggers the **Revert NTP Configuration** node.

7.4. Click the **Configure NTP** node and then click the **Output** tab to determine the cause of the failure.

The output indicates that the `cisco.ios.ios_ntp_globla` module is misspelled in the playbook.

8. Fix the issue with the `ntp.yml` playbook.

8.1. Create the `~/git-repos` directory and then change to that directory. Clone the git repository into the `~/git-repos` directory:

```
[student@workstation task-workflow]$ mkdir -p ~/git-repos
[student@workstation task-workflow]$ cd ~/git-repos
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/ntp
Cloning into 'ntp'...
...output omitted...
[student@workstation git-repos]$ cd ntp
```

8.2. Edit the `ntp.yml` file and fix the mistake in the playbook by changing `cisco.ios.ios_ntp_globla` to `cisco.ios.ios_ntp_global` as the module name. The modified playbook must consist of the following content:

```
---
- name: Configure NTP on IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure NTP
      cisco.ios.ios_ntp_global:
        config:
          logging: true
          source: GigabitEthernet0/1
          servers:
            - server: 10.10.100.1
              version: 2
            - server: 10.10.200.1
              version: 2
        state: overridden
```

8.3. Add, commit, and push the `ntp.yml` file:

```
[student@workstation ntp]$ git add ntp.yml
[student@workstation ntp]$ git commit -m "Fix typo in playbook"
[student@workstation ntp]$ git push
```

9. Launch the template through the automation controller web UI and verify that it now runs successfully. The `Configure NTP` node completes successfully and launches the `Verify NTP` node.

9.1. Navigate to **Resources** > **Templates**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

9.2. Click the **Launch Template** icon for the **IOS - Add and Verify NTP** workflow job template.

9.3. The first step updates the project with the changes you have committed. Observe that this time the **Configure NTP** job template completes successfully and as a result the **Verify NTP** job template is launched.

10. If you updated the `ntp.yml` playbook using VS Code, then click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/ntp` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory:

```
[student@workstation ntp]$ cd
```

Finish

On the `workstation` machine, 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-workflow
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Implementing Task Control

Create a new workflow job template and modify the playbooks used to resolve issues, improve efficiency, and handle errors.

Outcomes

- Create a workflow job template with "On Success" and "On Failure" nodes.
- Use loops to write efficient tasks.
- Use conditions to control where tasks run.
- Use blocks with `rescue` and `always` sections to handle playbook errors.

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 Git repositories, creates some automation controller resources, and enables the Network Configuration Protocol (NETCONF) system service on the Juniper Junos managed node.

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

Instructions

This exercise provides you with playbooks to configure interface settings on a Cisco IOS XE and a Juniper Junos managed node in your environment. The playbooks have associated job templates created in automation controller for you to use to create a workflow job template. The playbooks do not work and you must fix them. After updating the playbooks, verify that the workflow job template runs successfully.

1. Navigate to `https://controller.lab.example.com` to access the automation controller web UI. Log in as `admin` using `redhat` as the password.
2. Create a workflow job template called `Configure Interfaces` with the following information. In the following steps, you create nodes for the workflow job template.

Workflow Template

Field	Value
Name	`Configure Interfaces`
Description	`Configure interfaces on network devices`
Organization	`Default`
Inventory	`Common`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3. Start the workflow for the **Configure Interfaces** workflow job template by synchronizing the **Interfaces** project. Create the first node using the following information:

Workflow Node #1

Field	Value
Node Type	`Project Sync`
Name	`Interfaces`
Node Alias	`Sync Project`

4. Create a node that runs if the **Sync Project** node succeeds. This new node must launch the **Add Interfaces** job template. Create the second node using the following information:

Workflow Node #2

Field	Value
Node Type	`Job Template`
Name	`Add Interfaces`

5. Create a node that runs if the **Add Interfaces** node succeeds. This new node must launch the **Ping Interfaces** job template. Create the third node using the following information:

Workflow Node #3

Field	Value
Node Type	`Job Template`
Name	`Ping Interfaces`

6. Create a node that runs if the **Add Interfaces** node *fails*. This new node must launch the **Revert Changes** job template. Create the fourth node using the following information:

Workflow Node #4

Field	Value
Node Type	`Job Template`
Name	`Revert Changes`

After this step, the completed workflow job template should appear as follows:

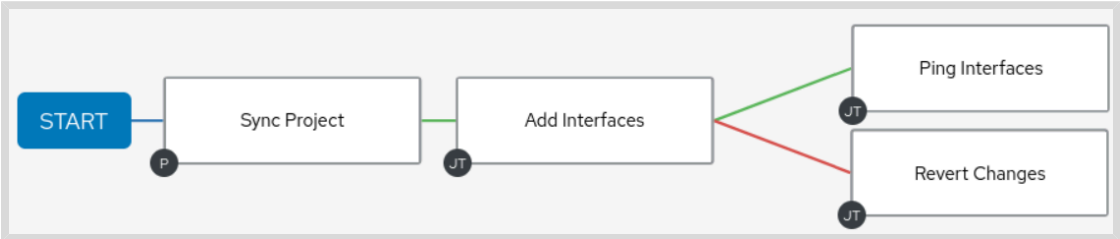


Figure 5.12: Completed workflow job template

7. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node fails, which causes the **Revert Changes** node to run.

8. Repair the issues in the `add_interfaces.yml` playbook that caused the **Add Interfaces** node in your workflow job to fail. Clone the `git@git.lab.example.com:student/interfaces` Git repository into the `/home/student/git-repos` directory.

   Change to the `exercise` branch and then edit the tasks in the `add_interfaces.yml` playbook. Any task using a module from the `cisco.ios` collection should only run on IOS devices, and any task using a module from the `junipernetworks.junos` collection should only run on Junos devices.

> **Note**
> You can view connection variables for the `ios` group in the `~/git-repos/inventories/group_vars/ios/connection.yml` file and connection variables for the `junos` group in the `~/git-repos/inventories/group_vars/junos/connection.yml` file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

9. The existing `add_interfaces.yml` playbook only adds one interface to each managed node. Extend the functionality of the `add_interfaces.yml` playbook by using loops.

   Add a loop to the `Create loopback interface` task that iterates over the interface names in the `~/git-repos/inventories/group_vars/ios/interfaces.yml` file.

   Add a second loop to the `Assign IP to loopback interface` task to iterate over the addresses in the `~/git-repos/inventories/group_vars/ios/interfaces.yml` file.

   Add a third loop to the `Add IP addresses` task to iterate over the addresses in the `~/git-repos/inventories/group_vars/junos/interfaces.yml` file.

> **Note**
> Each task in the playbook correctly references the `interfaces` variable in the `~/git-repos/inventories/group_vars/junos/interfaces.yml` file. However, the existing tasks only reference the first list item for each managed node.

   Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository.

10. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node succeeds but the **Ping Interfaces** node fails.

11. Repair the issue in the `ping_check.yml` playbook that caused the **Ping Interfaces** node in your workflow job to fail.

    Edit the tasks in the `ping_check.yml` playbook in the `exercise` branch of the repository.

    Tasks in the `IOS tasks` block should only run on IOS devices, the `Show ios_ping` task should only run if the `Ping IOS hosts` task fails, and the `Report IOS host status` task should always run. Similarly, the tasks in the `Junos tasks` block should only run on Junos devices, the `Show junos_ping` task should only run if the `Ping Junos hosts` task fails, and the `Report Junos host status` task should always run.

    Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository. When finished, either close the opened directory in VS Code or return to the `/home/student` directory from the CLI.

12. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node and the **Ping Interfaces** node both succeed.

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

As the `student` user on the `workstation` machine, 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
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Implementing Task Control

Create a new workflow job template and modify the playbooks used to resolve issues, improve efficiency, and handle errors.

Outcomes

- Create a workflow job template with "On Success" and "On Failure" nodes.
- Use loops to write efficient tasks.
- Use conditions to control where tasks run.
- Use blocks with `rescue` and `always` sections to handle playbook errors.

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 Git repositories, creates some automation controller resources, and enables the Network Configuration Protocol (NETCONF) system service on the Juniper Junos managed node.

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

Instructions

This exercise provides you with playbooks to configure interface settings on a Cisco IOS XE and a Juniper Junos managed node in your environment. The playbooks have associated job templates created in automation controller for you to use to create a workflow job template. The playbooks do not work and you must fix them. After updating the playbooks, verify that the workflow job template runs successfully.

1. Navigate to `https://controller.lab.example.com` to access the automation controller web UI. Log in as `admin` using `redhat` as the password.
2. Create a workflow job template called `Configure Interfaces` with the following information. In the following steps, you create nodes for the workflow job template.

Workflow Template

Field	Value
Name	`Configure Interfaces`
Description	`Configure interfaces on network devices`
Organization	`Default`
Inventory	`Common`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add workflow template**.

2.2. Create a workflow template using information from the "Workflow Template" table. When finished, click **Save**.

3. Start the workflow for the **Configure Interfaces** workflow job template by synchronizing the **Interfaces** project. Create the first node using the following information:

Workflow Node #1

Field	Value
Node Type	`Project Sync`
Name	`Interfaces`
Node Alias	`Sync Project`

3.1. Using the workflow visualizer for the **Configure Interfaces** workflow job template, click **Start**. Create the first node using information from the "Workflow Node #1" table. Select the **Interfaces** project and specify the node alias. When finished, click **Save**.

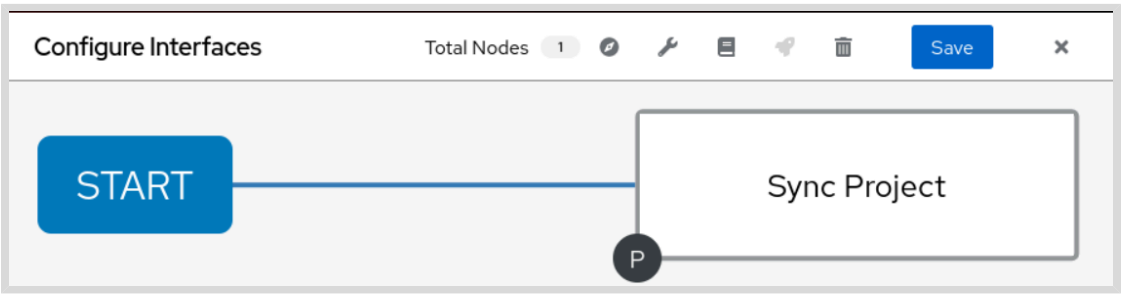


Figure 5.10: Synchronizing a project as the first step in the workflow

> **Note**
> The **P** symbol under the **Sync Project** node in the workflow visualizer indicates it is a project sync node.
>
> The blue line connecting **Start** and the **Sync Project** node indicates that this step is always performed.

4. Create a node that runs if the **Sync Project** node succeeds. This new node must launch the **Add Interfaces** job template. Create the second node using the following information:

Workflow Node #2

Field	Value
Node Type	`Job Template`
Name	`Add Interfaces`

4.1. Using the workflow visualizer for the **Configure Interfaces** workflow job template, hover over the **Sync Project** node and then click **Add a new node**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4.2. Select **On Success** and then click **Next**.

4.3. Create the second node using information from the "Workflow Node #2" table. Select the **Add Interfaces** job template and then click **Save**.

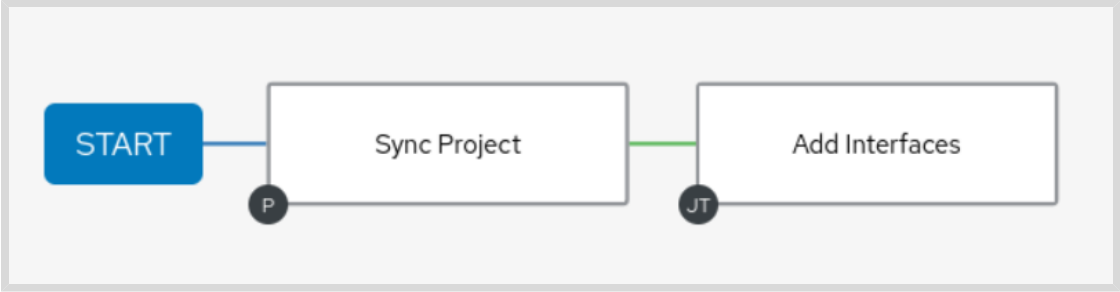


Figure 5.11: Adding a job template node to the workflow

> **Note**
> The **JT** symbol under the **Add Interfaces** node in the workflow visualizer indicates it is a job template node.
>
> The green line connecting the **Sync Project** node and the **Add Interfaces** node indicates that this step is performed only on success.

5. Create a node that runs if the **Add Interfaces** node succeeds. This new node must launch the **Ping Interfaces** job template. Create the third node using the following information:

Workflow Node #3

Field	Value
Node Type	`Job Template`
Name	`Ping Interfaces`

5.1. Using the workflow visualizer for the **Configure Interfaces** workflow job template, hover over the **Add Interfaces** node and then click **Add a new node**.

5.2. Select **On Success** and then click **Next**.

5.3. Create the third node using information from the "Workflow Node #3" table. Select the **Ping Interfaces** job template and then click **Save**.

6. Create a node that runs if the **Add Interfaces** node *fails*. This new node must launch the **Revert Changes** job template. Create the fourth node using the following information:

Workflow Node #4

Field	Value
Node Type	`Job Template`
Name	`Revert Changes`

After this step, the completed workflow job template should appear as follows:

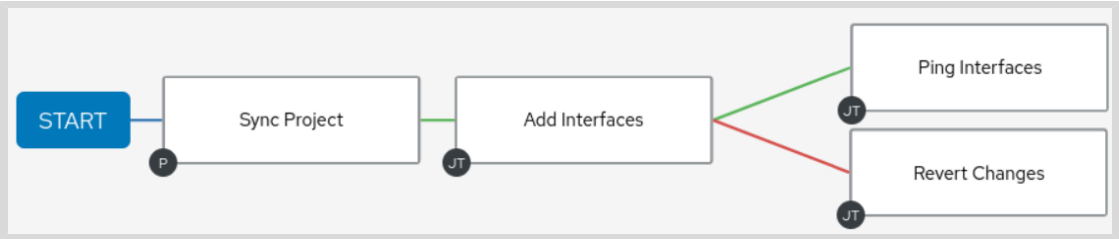


Figure 5.12: Completed workflow job template

6.1. Using the workflow visualizer for the **Configure Interfaces** workflow job template, hover over the **Add Interfaces** node and then click **Add a new node**.

6.2. Select **On Failure** and then click **Next**.

6.3. Create the fourth node using information from the "Workflow Node #4" table. Select the **Revert Changes** job template and then click **Save**.

The red line connecting the **Add Interfaces** node and the **Revert Changes** node indicates that this step is performed only on failure.

6.4. Click **Save** to exit the workflow visualizer.

7. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node fails, which causes the **Revert Changes** node to run.

7.1. Navigate to **Resources** > **Templates** and then click the **Launch Template** icon for the **Configure Interfaces** workflow job template.

7.2. Monitor the workflow and notice that the **Add Interfaces** node fails, which causes the **Revert Changes** node to run successfully.

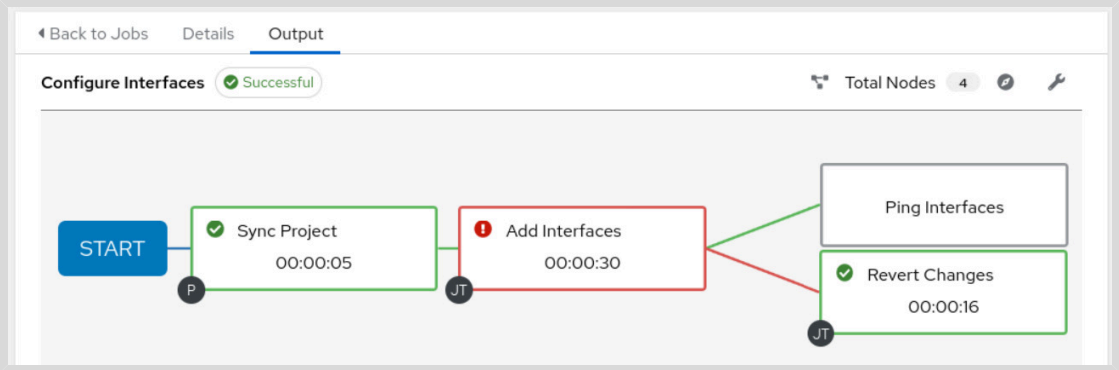


Figure 5.13: On Failure node runs successfully

> **Note**
> The **Add Interfaces** node fails but the **Revert Changes** node succeeds, which results in a successful run of the **Configure Interfaces** workflow.

8. Repair the issues in the `add_interfaces.yml` playbook that caused the **Add Interfaces** node in your workflow job to fail. Clone the `git@git.lab.example.com:student/interfaces` Git repository into the `/home/student/git-repos` directory.

Change to the `exercise` branch and then edit the tasks in the `add_interfaces.yml` playbook. Any task using a module from the `cisco.ios` collection should only run on IOS

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

devices, and any task using a module from the `junipernetworks.junos` collection should only run on Junos devices.

> **Note**
> You can view connection variables for the `ios` group in the `~/git-repos/inventories/group_vars/ios/connection.yml` file and connection variables for the `junos` group in the `~/git-repos/inventories/group_vars/junos/connection.yml` file.

8.1. Use either VS Code or the `git clone` command to clone the `git@git.lab.example.com:student/interfaces` Git repository into the `/home/student/git-repos` directory:

```
[student@workstation ~]$ cd ~/git-repos/
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/interfaces
```

8.2. Use either VS Code or the `git checkout exercise` command to switch to the `exercise` branch:

```
[student@workstation git-repos]$ cd interfaces
[student@workstation interfaces]$ git checkout exercise
Switched to branch 'exercise'
Your branch is up to date with 'origin/exercise'.
```

8.3. Edit the `add_interfaces.yml` playbook to add the `when: ansible_network_os == "cisco.ios.ios"` case statement to the tasks that use the `cisco.ios` modules:

```
...output omitted...
    - name: Create loopback interface
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_interfaces:
...output omitted...
    - name: Assign IP to loopback interface
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_l3_interfaces:
```

8.4. Edit the `add_interfaces.yml` playbook to add the `when: ansible_network_os == "junipernetworks.junos.junos"` case statement to the task that uses the `junipernetworks.junos` module:

```
...output omitted...
    - name: Add IP addresses
      when: ansible_network_os == 'junipernetworks.junos.junos'
      junipernetworks.junos.junos_l3_interfaces:
...output omitted...
```

9. The existing `add_interfaces.yml` playbook only adds one interface to each managed node. Extend the functionality of the `add_interfaces.yml` playbook by using loops.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Add a loop to the `Create loopback interface` task that iterates over the interface names in the `~/git-repos/inventories/group_vars/ios/interfaces.yml` file.

Add a second loop to the `Assign IP to loopback interface` task to iterate over the addresses in the `~/git-repos/inventories/group_vars/ios/interfaces.yml` file.

Add a third loop to the `Add IP addresses` task to iterate over the addresses in the `~/git-repos/inventories/group_vars/junos/interfaces.yml` file.

> **Note**
> Each task in the playbook correctly references the `interfaces` variable in the `~/git-repos/inventories/group_vars/junos/interfaces.yml` file. However, the existing tasks only reference the first list item for each managed node.

Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository.

9.1. Inspect the `~/git-repos/inventories/group_vars/ios/interfaces.yml` file:

```
---
interfaces:
  iosxe1.lab.example.com:
    - name: Loopback100
      address: 10.10.10.100/32
      enabled: true
    - name: Loopback101
      address: 10.10.10.101/32
      enabled: true
    - name: Loopback102
      address: 10.10.10.102/32
      enabled: true
    - name: Loopback103
      address: 10.10.10.103/32
      enabled: true
```

9.2. Update the `~/git-repos/interfaces/add_interfaces.yml` playbook to add a loop to the `Create loopback interfaces` task. Update the interface name, enabled state, and description parameters. The updated task contains the following content:

```
    - name: Create loopback interfaces
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_interfaces:
        config:
          - name: "{{ item['name'] }}"
            enabled: "{{ item['enabled'] }}"
            description: >-
              {{ item['description'] |
              default(omit) }}
        state: replaced
      loop: "{{ interfaces[inventory_hostname] }}"
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

9.3. Add a loop to the `Assign IP to loopback interface` task and then update the interface name and address parameters. The updated task contains the following content:

```
  - name: Assign IP to loopback interface
    when: ansible_network_os == 'cisco.ios.ios'
    cisco.ios.ios_l3_interfaces:
      config:
        - name: "{{ item['name'] }}"
          ipv4:
            - address: "{{ item['address'] }}"
      state: replaced
    loop: "{{ interfaces[inventory_hostname] }}"
```

9.4. Inspect the `~/git-repos/inventories/group_vars/junos/interfaces.yml` file:

```
---
interfaces:
  junos1.lab.example.com:
    - name: fxp0
      address: 172.25.250.205/24
    - name: fxp0
      address: 172.25.250.206/24
    - name: fxp0
      address: 172.25.250.207/24
    - name: fxp0
      address: 172.25.250.208/24
```

9.5. Add a loop to the `Add IP addresses` task and then update the interface name and address parameters. The updated task contains the following content:

```
  - name: Add IP addresses
    when: ansible_network_os == 'junipernetworks.junos.junos'
    junipernetworks.junos.junos_l3_interfaces:
      config:
        - name: "{{ item['name'] }}"
          ipv4:
            - address: "{{ item['address'] }}"
    loop: "{{ interfaces[inventory_hostname] }}"
```

9.6. The final playbook should contain the following content:

```
---
- name: Configure loopback interfaces
  hosts:
    - iosxe1.lab.example.com
    - junos1.lab.example.com
  gather_facts: false
  tasks:
    - name: Create loopback interface
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_interfaces:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        config:
          - name: "{{ item['name'] }}"
            enabled: "{{ item['enabled'] }}"
            description: >-
              {{ item['description'] |
              default(omit) }}
        state: replaced
      loop: "{{ interfaces[inventory_hostname] }}"

    - name: Assign IP to loopback interface
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_l3_interfaces:
        config:
          - name: "{{ item['name'] }}"
            ipv4:
              - address: "{{ item['address'] }}"
        state: replaced
      loop: "{{ interfaces[inventory_hostname] }}"

    - name: Add IP addresses
      when: ansible_network_os == 'junipernetworks.junos.junos'
      junipernetworks.junos.junos_l3_interfaces:
        config:
          - name: "{{ item['name'] }}"
            ipv4:
              - address: "{{ item['address'] }}"
      loop: "{{ interfaces[inventory_hostname] }}"
```

9.7. Use either VS Code or the CLI to add, commit, and push the changes to the remote repository. Use a descriptive commit message, such as `Improving add_interfaces.yml playbook`. If you choose to use the CLI, then use the `git add`, `git commit`, and `git push` commands:

```
[student@workstation interfaces]$ git add add_interfaces.yml
[student@workstation interfaces]$ git commit -m \
"Improving add_interfaces.yml playbook"
[exercise 96b349d] Improving add_interfaces.yml playbook
...output omitted...
[student@workstation interfaces]$ git push -u origin exercise
...output omitted...
To git.lab.example.com:student/interfaces.git
   6563170..d4a96cb  exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

10. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node succeeds but the **Ping Interfaces** node fails.

10.1. Navigate to **Resources** > **Templates** and then click the **Launch Template** icon for the **Configure Interfaces** workflow job template.

10.2. Monitor the workflow and notice that the **Add Interfaces** node succeeds and the **Ping Interfaces** node fails, which results in a failed run of the **Configure Interfaces** workflow.

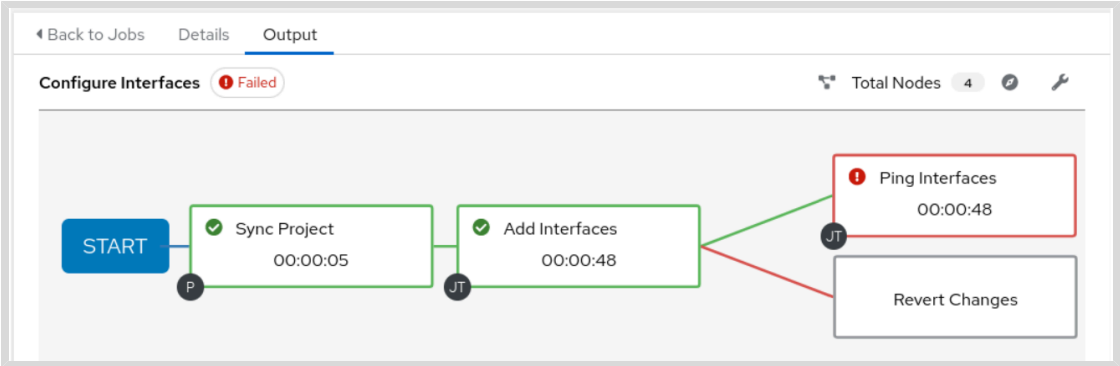


Figure 5.14: Ping Interfaces node fails

11. Repair the issue in the `ping_check.yml` playbook that caused the **Ping Interfaces** node in your workflow job to fail.

Edit the tasks in the `ping_check.yml` playbook in the `exercise` branch of the repository.

Tasks in the `IOS tasks` block should only run on IOS devices, the `Show ios_ping` task should only run if the `Ping IOS hosts` task fails, and the `Report IOS host status` task should always run. Similarly, the tasks in the `Junos tasks` block should only run on Junos devices, the `Show junos_ping` task should only run if the `Ping Junos hosts` task fails, and the `Report Junos host status` task should always run.

Stage and commit the modified files and use a descriptive commit message. Push your changes to the remote Git repository. When finished, either close the opened directory in VS Code or return to the `/home/student` directory from the CLI.

11.1. Add a `when` condition statement to the `IOS tasks` block so that the tasks only run on hosts where the `ansible_network_os` variable is set to the `cisco.ios.ios` value. The updated block contains the following content:

```
...output omitted...
  tasks:
    - name: IOS tasks
      when: ansible_network_os == 'cisco.ios.ios'
      block:
        - name: Ping IOS hosts
          cisco.ios.ios_ping:
            dest: "{{ item }}"
...output omitted...
```

11.2. Move the `Show ios_ping` task into a `rescue` block for the `IOS tasks` block:

```
...output omitted...
    - name: IOS tasks
      when: ansible_network_os == 'cisco.ios.ios'
      block:
        - name: Ping IOS hosts
          cisco.ios.ios_ping:
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
          register: ios_ping

        - name: Report IOS host status
          ansible.builtin.debug:
            msg: >-
              "The host {{ item['item'] }} had
              {{ item['packet_loss'] }} packet loss."
          loop: "{{ ios_ping['results'] | flatten(1) }}"

      rescue:
        - name: Show ios_ping
          ansible.builtin.debug:
            var: ios_ping['results'] | selectattr('failed', '==', true)
...output omitted...
```

11.3. Move the `Report IOS host status` task into an `always` block for the `IOS tasks` block:

```
...output omitted...
    - name: IOS tasks
      when: ansible_network_os == 'cisco.ios.ios'
      block:
        - name: Ping IOS hosts
          cisco.ios.ios_ping:
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
          register: ios_ping

      rescue:
        - name: Show ios_ping
          ansible.builtin.debug:
            var: ios_ping['results'] | selectattr('failed', '==', true)

      always:
        - name: Report IOS host status
          ansible.builtin.debug:
            msg: >-
              "The host {{ item['item'] }} had
              {{ item['packet_loss'] }} packet loss."
          loop: "{{ ios_ping['results'] | flatten(1) }}"
...output omitted...
```

11.4. Add a `when` condition statement to the `Junos tasks` block so the tasks only run on hosts where the `ansible_network_os` variable is set to the `junipernetworks.junos.junos` value. The updated block contains the following content:

```
...output omitted...
  tasks:
    - name: Junos tasks
      when: ansible_network_os == 'junipernetworks.junos.junos'
```

```
      block:
        - name: Ping Junos hosts
          junipernetworks.junos.junos_ping:
            dest: "{{ item }}"
...output omitted...
```

11.5. Move the `Show junos_ping` task into a `rescue` block for the `Junos tasks` block:

```
...output omitted...
    - name: Junos tasks
      when: ansible_network_os == 'junipernetworks.junos.junos'
      block:
        - name: Ping Junos hosts
          vars:
            ansible_connection: ansible.netcommon.network_cli
          junipernetworks.junos.junos_ping:
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
          register: junos_ping

        - name: Report Junos host status
          ansible.builtin.debug:
            msg: >-
              "The host {{ item['item'] }} had
              {{ item['packet_loss'] }} packet loss."
          loop: "{{ junos_ping['results'] | flatten(1) }}"

      rescue:
        - name: Show junos_ping
          ansible.builtin.debug:
            var: junos_ping['results'] | selectattr('failed', '==', true)
...output omitted...
```

11.6. Move the `Report Junos host status` task into an `always` block for the `Junos tasks` block:

```
...output omitted...
    - name: Junos tasks
      when: ansible_network_os == 'junipernetworks.junos.junos'
      block:
        - name: Ping Junos hosts
          vars:
            ansible_connection: ansible.netcommon.network_cli
          junipernetworks.junos.junos_ping:
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
          register: junos_ping
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    rescue:
      - name: Show junos_ping
        ansible.builtin.debug:
          var: junos_ping['results'] | selectattr('failed', '==', true)

    always:
      - name: Report Junos host status
        ansible.builtin.debug:
          msg: >-
            "The host {{ item['item'] }} had
            {{ item['packet_loss'] }} packet loss."
        loop: "{{ junos_ping['results'] | flatten(1) }}"
```

11.7. The final playbook should contain the following content:

```
---
- name: Ping new IP addresses
  hosts:
    - iosxe1.lab.example.com
    - junos1.lab.example.com
  gather_facts: false
  tasks:
    - name: IOS tasks
      when: ansible_network_os == 'cisco.ios.ios'
      block:
        - name: Ping IOS hosts
          cisco.ios.ios_ping:
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
          register: ios_ping

      rescue:
        - name: Show ios_ping
          ansible.builtin.debug:
            var: ios_ping['results'] | selectattr('failed', '==', true)

      always:
        - name: Report IOS host status
          ansible.builtin.debug:
            msg: >-
              "The host {{ item['item'] }} had
              {{ item['packet_loss'] }} packet loss."
          loop: "{{ ios_ping['results'] | flatten(1) }}"

    - name: Junos tasks
      when: ansible_network_os == 'junipernetworks.junos.junos'
      block:
        - name: Ping Junos hosts
          vars:
            ansible_connection: ansible.netcommon.network_cli
          junipernetworks.junos.junos_ping:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
            dest: "{{ item }}"
          loop: >-
            {{ interfaces[inventory_hostname] |
            map(attribute='address') |
            ansible.utils.ipaddr('address') }}
          register: junos_ping

      rescue:
        - name: Show junos_ping
          ansible.builtin.debug:
            var: junos_ping['results'] | selectattr('failed', '==', true)

      always:
        - name: Report Junos host status
          ansible.builtin.debug:
            msg: >-
              "The host {{ item['item'] }} had
              {{ item['packet_loss'] }} packet loss."
          loop: "{{ junos_ping['results'] | flatten(1) }}"
```

11.8. Use either VS Code or the CLI to add, commit, and push the changes to the remote repository. Use a descriptive commit message, such as `Improving ping_check.yml playbook`. If you choose to use the CLI, then use the `git add`, `git commit`, and `git push` commands:

```
[student@workstation interfaces]$ git add ping_check.yml
[student@workstation interfaces]$ git commit -m \
"Improving ping_check.yml playbook"
[exercise 96b349d] Improving ping_check.yml playbook
...output omitted...
[student@workstation interfaces]$ git push -u origin exercise
...output omitted...
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

11.9. If you completed this step and the previous steps using VS Code, then click **File** > **Close Folder** to close the `/home/student/git-repos/interfaces` directory. If you completed this step and the previous steps using the CLI, then run the `cd` command to return to the `/home/student` directory:

```
[student@workstation interfaces]$ cd
```

12. Run the **Configure Interfaces** workflow job template. Notice that the **Add Interfaces** node and the **Ping Interfaces** node both succeed.

12.1. Navigate to **Resources** > **Templates** and then click the **Launch Template** icon for the **Configure Interfaces** workflow job template.

12.2. Monitor the workflow and notice that the **Add Interfaces** node and the **Ping Interfaces** node both succeed.

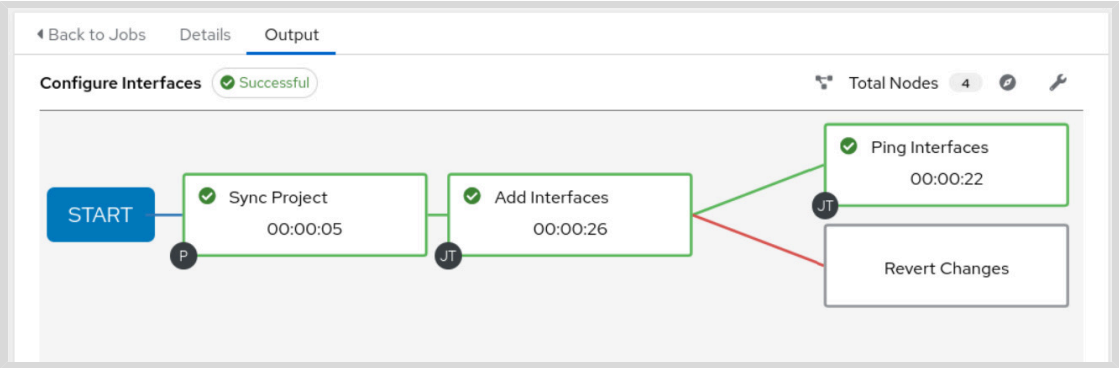


Figure 5.15: Workflow template runs successfully

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 task-review
```

Finish

As the `student` user on the `workstation` machine, 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
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- You can use *loops* in playbooks to iterate over a set of values, such as a simple list of strings, or a list of dictionaries.
- You can use *conditions* to determine when to execute tasks or blocks.
- You can handle errors in playbook tasks by ignoring task failures, marking tasks as failed under certain conditions, or overriding the behavior that causes a task to be marked as changed.
- You can use *blocks* to group tasks as a unit, executing other tasks depending on whether all the tasks in the block succeed.
- Workflow job templates connect multiple job templates into a single workflow. You can use workflow job templates to automate complex operations, including synchronizing projects, synchronizing inventories, and launching multiple job templates.
- The workflow visualizer in the automation controller is a graphical interface that helps you to design the workflow of a workflow job template. For example, you can specify if a node should always run, if a node should run if the previous node succeeded, or if a node should run if the previous node failed.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 6

Simplifying Playbooks with Roles and Ansible Content Collections

Goal

Use Ansible Roles and Ansible Content Collections to develop playbooks more quickly and to reuse Ansible code.

Objectives

- Manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.
- Describe the purpose of an Ansible Role, its structure, and how roles are used in playbooks.
- Create a role in a playbook's project directory and run it as part of one of the plays in the playbook.
- Select and retrieve roles from external sources such as Git repositories or Ansible Galaxy, and use them in your playbooks.
- Obtain a set of related roles, supplementary modules, and other content from an Ansible Content Collection and use them in a playbook.

Sections

- Including and Importing Files (and Guided Exercise)
- Ansible Role Structure (and Quiz)
- Creating Roles (and Guided Exercise)
- Deploying Roles from External Content Sources (and Guided Exercise)
- Getting Roles and Modules from Ansible Content Collections (and Guided Exercise)

Lab

- Simplifying Playbooks with Roles and Ansible Content Collections

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Including and Importing Files

Objectives

- Manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.

Managing Large Playbooks

When a playbook gets long or complex, you can divide it up into smaller files to make it easier to manage. You can combine multiple playbooks into a main playbook, or insert lists of tasks from a file into a play.

A file that contains only a list of tasks is called a *task file*. The following is an example of the content of a task file:

```
---
- name: Add Cisco IOS VLAN configuration
  cisco.ios.ios_vlans:
    config: "{{ vlans }}"
    state: merged

- name: Verify Cisco IOS VLAN configuration
  cisco.ios.ios_command:
    commands: show vlans
  register: vlan_data

- name: Display configured VLANs
  ansible.builtin.debug:
    var: vlan_data
```

Splitting a playbook into smaller files can make it easier to reuse plays or sequences of tasks in different projects.

Importing or Including Files

Ansible supports two methods for bringing content into a playbook. You can *include* content, or you can *import* content.

Importing content is a *static* operation. Ansible preprocesses imported content when the playbook is initially parsed, before the run starts.

Including content is a *dynamic* operation. Ansible processes included content while the playbook is running, as content is reached.

Importing Playbooks

The `ansible.builtin.import_playbook` module enables you to import external files containing lists of plays into a playbook. In other words, you can have a main playbook that imports

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

one or more additional playbooks. If you import multiple playbooks, then they are imported and run in order.

The following is a simple example of a main playbook that imports two additional playbooks:

```
---
- name: Enable NETCONF service
  ansible.builtin.import_playbook: juniper_netconf.yml

- name: Configure SNMP settings
  ansible.builtin.import_playbook: juniper_snmp.yml
```

Notice that you must specify the imported playbook in the same line as the module.

You can also interleave plays in your main playbook with imported playbooks. In the following example, the `Back up the IOS configuration` play runs first, followed by the plays imported from the `ios_vlans.yml` playbook.

```
---
- name: Back up the IOS configuration
  hosts: ios
  gather_facts: false
  tasks:
    - name: Back up the current configuration
      cisco.ios.ios_config:
        backup: true
        backup_options:
          filename: "{{ inventory_hostname }}.txt"

- name: Import IOS VLANs playbook
  ansible.builtin.import_playbook: ios_vlans.yml
```

Importing Task Files

You can import a task file into a play inside a playbook by using the `ansible.builtin.import_tasks` module. When you import a task file, the tasks in that file are directly inserted when the playbook is parsed.

In the following example, instead of writing a list of tasks in the playbook, the task named `Import Juniper Junos VLANs tasks` imports the tasks from the `junos_vlan.yml` task file:

```
---
- name: Juniper Junos VLANs configuration
  hosts: junos
  gather_facts: false
  tasks:
  - name: Import Juniper Junos VLANs tasks
    ansible.builtin.import_tasks:
      file: junos_vlan.yml
```

To import a task file, you can use the `file` parameter of the `ansible.builtin.import_tasks` module to indicate the name of the task file, or you can give the name of the task file in the same line as the module:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Juniper Junos VLANs configuration
  hosts: junos
  gather_facts: false
  tasks:
  - name: Import Juniper Junos VLANs tasks
    ansible.builtin.import_tasks: junos_vlan.yml
```

Because the `ansible.builtin.import_tasks` module statically imports the tasks when the playbook is parsed, you must consider the following with respect to loops and conditionals:

Using loops in imported tasks
You cannot use loops with the `ansible.builtin.import_tasks` module.

Using conditionals in imported tasks
Conditional statements set on the import, such as `when`, are applied to each of the tasks that are imported. In other words, if you use a conditional on a task that imports content, then each task in the imported content performs that conditional verification before it runs.

In the following example, all the tasks in the `junos_security.yml` task file run against the Juniper Junos managed nodes, and all the tasks in the `ios_security.yml` task file run against the Cisco IOS managed nodes:

```
---
- name: Apply security settings on the managed nodes
  hosts: all
  gather_facts: false
  tasks:
  - name: Import Juniper Junos security settings tasks
    ansible.builtin.import_tasks: junos_security.yml
    when: ansible_network_os == "junipernetworks.junos.junos"

  - name: Import Cisco IOS security settings tasks
    ansible.builtin.import_tasks: ios_security.yml
    when: ansible_network_os == "cisco.ios.ios"
```

Including Task Files

You can also dynamically include a task file into a play inside a playbook by using the `ansible.builtin.include_tasks` module. To include a task file, you can use the `file` parameter to indicate the name of the file with the task list, or you can give the name of this file in the same line as the module.

The following example uses the `ansible.builtin.include_tasks` module to include a task file named `ios_ntp.yml` in the playbook:

```
---
- name: Cisco IOS NTP configuration
  hosts: ios
  gather_facts: false
  tasks:
  - name: Include Cisco IOS NTP tasks
    ansible.builtin.include_tasks: ios_ntp.yml
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Because the `ansible.builtin.include_tasks` module does not process content in the playbook until the play is running and that part of the play is reached, you must consider the following with respect to loops and conditionals:

Using loops in included tasks

Unlike the `ansible.builtin.import_tasks` module, you can use loops with the `ansible.builtin.include_tasks` module. The loop applies to each task in the included content.

In the following example, the task file named `ios_add_users.yml` contains the list of tasks to perform to create users in the Cisco IOS managed nodes. The list of users is given using the loop.

```
---
- name: Cisco IOS users
  hosts: ios
  gather_facts: false
  tasks:
  - name: Add users to Cisco IOS managed nodes
    ansible.builtin.include_tasks: ios_add_users.yml
    loop: "{{ expected_users }}"
```

Using conditionals in included tasks

Conditional statements, such as `when` set on the `ansible.builtin.include_tasks` module, determine whether the tasks are included in the play at all. The condition is applied only to the included task itself and not to any other tasks within the included file.

In other words, if you put a conditional on a task that includes content, then the conditional determines whether the content is included or not. If the condition is met, then all the tasks that are included run normally.

In the following example, if the `vlans` variable is defined, then the tasks in the `ios_vlans.yml` task file are run. The condition is no longer verified for tasks in the `ios_vlans.yml` task file:

```
---
- name: Cisco IOS VLANs configuration
  hosts: ios
  gather_facts: false
  tasks:
  - name: Apply conditional to imported task
    ansible.builtin.include_tasks: ios_vlans.yml
    when: vlans is defined
```

Use Cases for Task Files

Consider the following examples where it might be useful to manage sets of tasks as external files separate from the playbook:

- If new managed nodes require complete configuration, then network administrators could create various sets of tasks for creating users and setting up their access, configuring the login banner on the managed nodes, hardening the managed nodes, installing security updates, configuring DNS, Simple Network Management Protocol (SNMP), logging, VLANs, NTP settings, and so on. Each of these sets of tasks could be managed through a separate self-contained task file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- If different groups of network administrators are in charge of the managed nodes, then each group can write their own task file, which can then be reviewed and integrated by the network manager.
- If a managed node requires a particular configuration, then it can be integrated as a set of tasks that are executed based on a conditional. In other words, including the tasks only if specific criteria are met.
- If a group of managed nodes needs to run a particular task or set of tasks, then the tasks might only be run on a managed node if it is part of that specific group.

Working with External Plays and Tasks

The incorporation of plays or tasks from external files into playbooks using the Ansible import and include features enhances the ability to reuse tasks and playbooks across an Ansible environment.

Managing Task Files

You can create a dedicated directory for task files, and save all task files in that directory. Your playbook can then include or import task files from that directory.

For example, suppose that you have the following task file named `ios_verification.yml` that verifies the NTP configuration in the Cisco IOS managed nodes, and then displays the existing NTP configuration:

```
---
- name: Verifying NTP configuration
  cisco.ios.ios_command:
    commands:
      - show run | include ntp
  register: ios_value

- name: Display NTP configuration
  ansible.builtin.debug:
    var: ios_value['stdout_lines']
```

If the previous task file was saved in a `tasks` directory, then you might incorporate the tasks as follows:

```
...output omitted...

    - name: Import task file
      ansible.builtin.import_tasks: tasks/ios_verification.yml

...output omitted...
```

This enables you to construct a complex playbook and makes it easy to manage its structure and components.

Defining Variables

To maximize the possibility of reuse, plays and tasks should be as generic as possible. Use variables to parameterize play and task elements to expand the application of plays and tasks.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For example, if you parameterize the configuration to be verified from the previous example, then the `ios_verification.yml` task file can also be used for the verification of other configurations, rather than being useful only for verifying the NTP configuration:

```
- name: Verifying {{ ios_service }} configuration
  cisco.ios.ios_command:
    commands: "{{ ios_commands }}"
  register: ios_value

- name: Display {{ ios_service }} configuration
  ansible.builtin.debug:
    var: ios_value['stdout_lines']
```

When incorporating the task file in your playbook, you can define the variables to use for the task execution as follows:

```
...output omitted...

    - name: Import task file and set variables
      ansible.builtin.import_tasks: tasks/ios_verification.yml
      vars:
        ios_service: NTP
        ios_commands:
          - show run | include ntp

...output omitted...
```

The following example shows how to use the same task file to verify DNS on the managed nodes:

```
...output omitted...

    - name: Import task file and set variables
      ansible.builtin.import_tasks: tasks/ios_verification.yml
      vars:
        ios_service: DNS
        ios_commands:
          - show ip name-server
          - show ip dns view | include list

...output omitted...
```

You can use the same technique to make playbooks more reusable. When incorporating a playbook into another playbook, pass the variables to use for the play execution as follows:

```
- name: Import play file and set the variable
  ansible.builtin.import_playbook: play.yml
  vars:
    managed_nodes: junos

...output omitted...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

Re-using Ansible Artifacts - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse.html

`ansible.builtin.import_playbook` Module - Import a Playbook - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/import_playbook_module.html

`ansible.builtin.import_tasks` Module - Import a Task List - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/import_tasks_module.html

`ansible.builtin.include_tasks` Module - Dynamically Include a Task List - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/include_tasks_module.html

Conditionals with Imports - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html#conditionals-with-imports

Conditionals with Includes - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html#conditionals-with-includes

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Configuring a Multi-vendor Network

Include and import playbooks and tasks for different network vendors in a top-level Ansible Playbook.

Outcomes

- Create a playbook that uses conditionals to select the appropriate task file.
- Create task files to encourage reusing code.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start simplify-files
```

Instructions

1. Open the `/home/student/simplify-files` directory in VS Code and create the `backup.yml` playbook to create backup files of the managed nodes in the `junos` and `ios` inventory groups.

   1.1. Open VS Code and then click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **simplify-files** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Create the `backup.yml` file and target the managed nodes in the `junos` and `ios` inventory groups. Do not gather facts for the managed nodes. Add the following lines to the playbook:

```
---
- name: Back up all managed network nodes
  hosts:
    - junos
    - ios
  gather_facts: false (1)
  tasks:
```

(1) Fact gathering fails for the Juniper Junos managed nodes unless the NETCONF service is running on those nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1.4. Add a block to the bottom of the play that targets the Juniper Junos managed nodes. Add two tasks to include the specified task files:

```
---
- name: Back up all managed network nodes
  hosts:
    - junos
    - ios
  gather_facts: false
  tasks:
    - name: Juniper Junos block
      when: ansible_network_os == 'junipernetworks.junos.junos'
      block:
        - name: Enable NETCONF
          ansible.builtin.include_tasks: tasks/junos_netconf_enable.yml (1)

        - name: Create Junos backup files
          ansible.builtin.include_tasks: tasks/junos_backup_create.yml (2)
```

(1) Most tasks for the Juniper Junos managed nodes require that the NETCONF service is running. This task file enables the NETCONF service.

(2) This task file performs the necessary steps to create the backup files.

2. Create the `tasks` directory and the task files specified in the `backup.yml` playbook.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/simplify-files` directory in a GNOME terminal:

```
[student@workstation ~] cd ~/simplify-files
```

2.2. Use the `mkdir` command to create the `tasks` directory:

```
[student@workstation simplify-files]$ mkdir tasks
```

2.3. Create the `tasks/junos_netconf_enable.yml` task file with the following content:

```
---
- name: Enable NETCONF service
  vars:
    ansible_connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
    netconf_port: 830
    state: present
```

2.4. Create the `tasks/junos_backup_create.yml` task file with the following content:

```
---
- name: Create the backup file
  junipernetworks.junos.junos_config: (1)
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    backup: true
  register: create_file

- name: Show create_file
  ansible.builtin.debug:
    var: create_file ❷
```

❶ This task uses the `junipernetworks.junos.junos_config` module with the default values for the `backup_options` parameter. These values include the directory where the module stores the backups and the name of each backup file.

❷ If you register the output of the `Create the backup file` task, then you can display the registered variable using the `ansible.builtin.debug` module. Ultimately, this task file must create symbolic links that point to the latest backup file for each managed node.

3. Run the `backup.yml` playbook. Using the information displayed in the `Show create_file` task, update the `tasks/junos_backp_create.yml` task file to create a symbolic link that points to the latest backup file.

3.1. Run the `backup.yml` playbook and examine the output of the `Show create_file` task:

```
[student@workstation simplify-files]$ ansible-navigator run backup.yml
...output omitted...
TASK [Show create_file] ******************************************************
ok: [junos1.lab.example.com] => {
    "create_file": {
        "backup_path": "/home/student/simplify-files/backup/
junos1.lab.example.com_config.2023-08-16@14:12:46", ❶
        "changed": true,
        "date": "2023-08-16",
        "failed": false,
        "filename": "junos1.lab.example.com_config.2023-08-16@14:12:46", ❷
        "shortname": "/home/student/simplify-files/backup/
junos1.lab.example.com_config", ❸
        "time": "14:12:46"
    }
}
ok: [junos2.lab.example.com] => {
    "create_file": {
        "backup_path": "/home/student/simplify-files/backup/
junos2.lab.example.com_config.2023-08-16@14:12:46",
        "changed": true,
        "date": "2023-08-16",
        "failed": false,
        "filename": "junos2.lab.example.com_config.2023-08-16@14:12:46",
        "shortname": "/home/student/simplify-files/backup/
junos2.lab.example.com_config",
        "time": "14:12:46"
    }
}
...output omitted...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1. The value of the `create_file['backup_path']` variable contains the absolute path to the created backup file.
2. The value of the `create_file['filename']` variable contains the name of the created backup file.
3. The value of the `create_file['shortname']` variable matches the value of the `create_file['backup_path']` variable without the date and time stamp.

3.2. Update the `tasks/junos_backup_create.yml` task file. Add a task that creates a symbolic link to the most recent backup file. You might also update the `Show create_file` task so that the task is skipped unless you run the playbook in verbose mode:

```
---
- name: Create the backup file
  junipernetworks.junos.junos_config:
    backup: true
  register: create_file

- name: Show create_file
  ansible.builtin.debug:
    var: create_file
    verbosity: 1 (1)

- name: Create or update the 'latest' link
  ansible.builtin.file:
    path: "{{ create_file['shortname'] }}.latest" (2)
    src: "{{ create_file['filename'] }}" (3)
    state: link (4)
    force: true (5)
```

1. Skip this task unless you run the playbook with at least one `-v` option.
2. Create the symbolic link by appending `.latest` to the value of the `create_file['shortname']` variable.
3. The task creates a symbolic link that uses a *relative path* to a file in the same directory as the link. *(Optional)* Use the value of the `create_file['backup_path']` variable to create a symbolic link that uses an *absolute path*.
4. Create a symbolic link.
5. Because this example uses a relative path, you must force the creation of the symbolic link.

3.3. Run the `backup.yml` playbook:

```
[student@workstation simplify-files]$ ansible-navigator run backup.yml
...output omitted...
TASK [Create or update the 'latest' link] ************************************
changed: [junos2.lab.example.com]
changed: [junos1.lab.example.com]
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=0    changed=0    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=0    changed=0    unreachable=0    failed=0   ...
junos1.lab.example.com     : ok=6    changed=2    unreachable=0    failed=0   ...
junos2.lab.example.com     : ok=6    changed=2    unreachable=0    failed=0   ...
```

3.4. Use the `tree` command to verify the creation of the symbolic links. The date and time stamps displayed in this example differ from your output.

```
[student@workstation simplify-files]$ tree backup
backup
├── junos1.lab.example.com_config.2023-08-16@14:12:46
├── junos1.lab.example.com_config.2023-08-16@14:16:16
├── junos1.lab.example.com_config.latest ->
 junos1.lab.example.com_config.2023-08-16@14:16:16
├── junos2.lab.example.com_config.2023-08-16@14:12:46
├── junos2.lab.example.com_config.2023-08-16@14:16:17
└── junos2.lab.example.com_config.latest ->
 junos2.lab.example.com_config.2023-08-16@14:16:17

0 directories, 6 files
```

4. Update the `backup.yml` playbook so that it creates backup files for the Cisco IOS managed nodes.

4.1. Add a block to the bottom of the play that targets the Cisco IOS managed nodes. Add a task to include the `tasks/ios_backup_create.yml` task file.

The finished playbook contains the following content:

```
---
- name: Back up all managed network nodes
  hosts:
    - junos
    - ios
  gather_facts: false
  tasks:
    - name: Juniper Junos block
      when: ansible_network_os == 'junipernetworks.junos.junos'
      block:
        - name: Enable NETCONF
          ansible.builtin.include_tasks: tasks/junos_netconf_enable.yml

        - name: Create Junos backup files
          ansible.builtin.include_tasks: tasks/junos_backup_create.yml

    - name: Cicso IOS block
      when: ansible_network_os == 'cisco.ios.ios'
      block: (1)
        - name: Create IOS backup files
          ansible.builtin.include_tasks: tasks/ios_backup_create.yml (2)
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1. Although this playbook only has one task for the Cisco IOS managed nodes, you might still use a block in case you decide to add more tasks.

2. This task file is almost the same as the `tasks/junos_backup_create.yml` task file.

> **Note**
> Because the playbook does not perform loops or pass variables to the task files, you could use the `ansible.builtin.import_tasks` module instead of the `ansible.builtin.include_tasks` module.

4.2. Use VS Code or a GNOME terminal to copy the `tasks/junos_backup_create.yml` task file to the `tasks/ios_backup_create.yml` task file:

```
[student@workstation simplify-files]$ cp tasks/junos_backup_create.yml \
tasks/ios_backup_create.yml
```

4.3. Update the `tasks/ios_backup_create.yml` task file so that it uses the `cisco.ios.ios_config` module instead of the `junipernetworks.junos.junos_config` module. The updated task file contains the following content:

```
---
- name: Create the backup file
  cisco.ios.ios_config:
    backup: true
  register: create_file

- name: Show create_file
  ansible.builtin.debug:
    var: create_file
    verbosity: 1

- name: Create or update the 'latest' link
  ansible.builtin.file:
    path: "{{ create_file['shortname'] }}.latest"
    src: "{{ create_file['filename'] }}"
    state: link
    force: true
```

5. Run the `backup.yml` playbook and verify that the playbook creates symbolic links for both the Juniper Junos and Cisco IOS managed nodes.

5.1. Run the `backup.yml` playbook:

```
[student@workstation simplify-files]$ ansible-navigator run backup.yml
...output omitted...
```

5.2. Use the `tree` command to verify the creation of the symbolic links. The date and time stamps displayed in this example differ from your output. Notice that even

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

though the Junos managed nodes have three backup files each, the symbolic links point to the most recent files.

```
[student@workstation simplify-files]$ tree backup
backup
├── iosxe1.lab.example.com_config.2023-08-16@14:21:56
├── iosxe1.lab.example.com_config.latest ->
 iosxe1.lab.example.com_config.2023-08-16@14:21:56
├── iosxe2.lab.example.com_config.2023-08-16@14:21:56
├── iosxe2.lab.example.com_config.latest ->
 iosxe2.lab.example.com_config.2023-08-16@14:21:56
├── junos1.lab.example.com_config.2023-08-16@14:12:46
├── junos1.lab.example.com_config.2023-08-16@14:16:16
├── junos1.lab.example.com_config.2023-08-16@14:21:51
├── junos1.lab.example.com_config.latest ->
 junos1.lab.example.com_config.2023-08-16@14:21:51
├── junos2.lab.example.com_config.2023-08-16@14:12:46
├── junos2.lab.example.com_config.2023-08-16@14:16:17
├── junos2.lab.example.com_config.2023-08-16@14:21:51
└── junos2.lab.example.com_config.latest ->
 junos2.lab.example.com_config.2023-08-16@14:21:51

0 directories, 12 files
```

6. Close the `/home/student/simplify-files` directory in VS Code, or run the `cd` command in a GNOME terminal to return to the `student` home directory.

```
[student@workstation simplify-files]$ cd
```

Finish

On the `workstation` machine, 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 simplify-files
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Ansible Role Structure

Objectives

- Describe the purpose of an Ansible Role, its structure, and how roles are used in playbooks.

Structuring Ansible Playbooks with Roles

As you develop more playbooks, you are likely to discover that you have many opportunities to reuse code from playbooks that you wrote previously.

For example, you might create a play to configure a network managed node, then reuse that same play to configure other managed nodes with different configuration settings. That play might be long and complex, with many included or imported files and tasks to manage various situations. Copying all that code into another playbook might take a lot of time.

Ansible Roles make it easy to reuse Ansible code. You can package all the tasks, variables, files, templates, and other resources into a standardized directory structure. You can copy a role from project to project by copying the role directory and subdirectories, and then calling that role within a play.

A well-written role can take variables passed from the playbook. These variables can adjust the behavior of the role by setting hostnames, IP addresses, usernames, or other specific details. You can also ensure that reasonable default values are set for the variables in the role.

Ansible Roles provide the following benefits:

- Roles group content together, enabling easy sharing of code with others.
- Roles make larger projects more manageable.
- Roles can be developed in parallel by different users.

In addition to writing, using, reusing, and sharing your own roles, you can obtain roles from sources such as the following:

- Distribution packages, such as Ansible Content Collections
- The hosted automation hub (`https://console.redhat.com/ansible/automation-hub`)
- A private automation hub
- The community Ansible Galaxy website

The Ansible Role Structure

An Ansible Role is defined by a standardized structure of subdirectories and files.

The top level directory defines the name of the role itself. Files are organized into subdirectories that are named according to each file's purpose in the role, such as `tasks`.

The following `tree` command displays the directory structure of the `user.example` role:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Ansible Role Subdirectories

Subdirectory	Function
`defaults`	The `main.yml` file in this directory contains the default values of role variables that can be overwritten when the role is used. These variables have low precedence and are intended to be changed and customized in plays.
`files`	This directory contains static files that are referenced by role tasks.
`handlers`	The `main.yml` file in this directory contains the role handler definitions. Handlers are not covered in this course.
`meta`	The `main.yml` file in this directory contains information about the role, including author, license, platforms, and optional role dependencies.
`tasks`	The `main.yml` file in this directory contains the role task definitions.
`templates`	This directory contains Jinja2 templates that are referenced by role tasks.
`tests`	This directory can contain an inventory and `test.yml` playbook that can be used to test the role.
`vars`	The `main.yml` file in this directory defines the role variable values. These variables are used for internal purposes within the role. These variables have high precedence and are not intended to be changed when used in a playbook.

Not every role has all the directories in the preceding table.

Defining Variables and Defaults

Role variables are defined by creating a `vars/main.yml` file with key-value pairs. These variables are referenced in role task files like any other variable: `{{ VAR_NAME }}`. These variables have a

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

high precedence and cannot be overridden by inventory variables. These variables are used within the role itself, not by playbooks that reference the role.

Default variables enable you to set default values for variables that can be used in a play within the role. These variables are defined by creating a `defaults/main.yml` file with key-value pairs in the role directory hierarchy.

Default variable values can be overridden by any other variable, including inventory variables. These variables are intended to provide you with a way to customize how the role functions.

Define a specific variable in either `vars/main.yml` or `defaults/main.yml`, but not in both places. Use default variables in `defaults/main.yml` when you intend that the variable values might be overridden. Variables defined in `vars/main.yml` override variables defined in `defaults/main.yml`.

Important
Roles should not contain any hard-coded data, secrets, passwords, private keys, or similar information. Roles should be generic, reusable, and freely shareable.

Using Ansible Roles in a Play

There are several ways to call roles in a play. The two primary methods are:

- Including or importing them in your `tasks` list
- Creating a `roles` list that runs specific roles before your play's tasks

The first method is newer and offers more flexibility, but the second method is still commonly used.

Including and Importing Roles as Tasks

Use the `ansible.builtin.import_role` module in a task to **statically import** a role. Use the `ansible.builtin.include_role` module in a task to **dynamically include** a role.

The following play demonstrates how you can import a role by using a task with the `ansible.builtin.import_role` module. The example play runs the task called `A normal task` first, and then imports the `role2` role:

```
- name: Run a role as a task
  hosts: ios
  tasks:
    - name: A normal task
      ansible.builtin.debug:
        msg: 'first task'
    - name: A task to import role2 here
      ansible.builtin.import_role:
        name: role2
```

With the `ansible.builtin.import_role` module, Ansible treats the role as a static import and parses it during initial playbook processing.

In the preceding example, when the playbook is parsed, Ansible performs the following tasks:

- If `roles/role2/tasks/main.yml` exists, Ansible adds the tasks in that file to the play.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- If `roles/role2/defaults/main.yml` exists, Ansible adds the default variables in that file to the play.
- If `roles/role2/vars/main.yml` exists, Ansible adds the variables in that file to the play (possibly overriding values from role default variables due to precedence).

> **Important**
> Because `ansible.builtin.import_role` is processed when the playbook is parsed, the variables in the role's `default` and `vars` directories are exposed to all the tasks and roles in the play, and can be accessed by tasks and roles that precede it in the play (even though the role has not run yet).

You can also set variables for the role when you call the task, in the same way that you can set task variables:

```
- name: Run a role as a task
  hosts: ios
  tasks:
    - name: A task to include role2 here
      ansible.builtin.import_role:
        name: role2
      vars:
        var1: val1
        var2: val2
```

The `ansible.builtin.include_role` module works in a similar way, but it dynamically includes the role when the playbook is running instead of statically importing it when the playbook is initially parsed.

One key difference between the two modules is how they handle task-level keywords, conditionals, and loops:

- `ansible.builtin.import_role` applies task conditionals and loops to each of the tasks being imported.
- `ansible.builtin.include_role` applies task conditionals and loops to the statement that determines whether the role is included or not.

In addition, when you include a role, its role variables and default variables are *not* exposed to the rest of the play, unlike when you import a role.

Using a Roles Section in a Play

Another way you can call roles in a play is to list them in a `roles` section. The `roles` section is similar to the `tasks` section, except instead of having a list of tasks, it has a list of roles.

In the following example play, the `role1` role runs first, followed by the `role2` role:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: A play that only has roles
  hosts: ios
  roles:
    - role: role1
    - role: role2
```

For each specified role, the role's tasks, variables, and dependencies are imported into the play in the order in which they are listed.

When you use a `roles` section to import roles into a play, the roles run before any tasks that you define for that play. Roles are run before tasks whether the `roles` section is listed before or after the `tasks` section.

```
- name: Roles always run first
  hosts: ios
  tasks:
    - name: A task
      ansible.builtin.debug:
        msg: "This task runs after the role."
  roles:
    - role: role1
```

Because roles run first, it generally makes sense to list the `roles` section before the `tasks` section, if you must have both.

Important

A `tasks` section is not required in a play, especially if you have a `roles` section. In fact, it is generally a good practice to avoid using both `roles` and `tasks` sections in a play at the same time to avoid confusion about the order in which roles and tasks run.

If you must have a `tasks` section and a `roles` section, it is better to create tasks that use `ansible.builtin.import_roles` and `ansible.builtin.include_roles` to run at the correct points in the play's execution.

The following example sets values for the `var1` and `var2` variables, which are associated with the `role2` role. If the `var1` and `var2` variables are defined in the `defaults` or `vars` directories for the `role2` role, then the variables are overridden when the `role2` role is used.

```
---
- name: A play that runs the second role with variables
  hosts: ios
  roles:
    - role: role1
    - role: role2
      var1: val1
      var2: val2
```

Another equivalent playbook syntax that you might see in this case is as follows:

```
---
- name: A play that runs the second role with variables
  hosts: ios
  roles:
    - role: role1
    - { role: role2, var1: val1, var2: val2 }
```

This can be harder to read in some situations, even though it is more compact.

Important

Ansible looks for duplicate role lines in the `roles` section. If two roles are listed with exactly the same parameters, the role only runs once.

For example, the following `roles` section only runs `role1` once:

```
roles:
  - { role: role1, service: "httpd" }
  - { role: role2, var1: true }
  - { role: role1, service: "httpd" }
```

To run the same role a second time, it must have different parameters defined:

```
roles:
  - { role: role1, service: "httpd" }
  - { role: role2, var1: true }
  - { role: role1, service: "syslogd" }
```

Special Tasks Sections

Two special task sections, `pre_tasks` and `post_tasks`, are occasionally used with `roles` sections. The `pre_tasks` section is a list of tasks, similar to `tasks`, but these tasks run before any of the roles in the `roles` section.

Plays also support a `post_tasks` keyword. These tasks run after the play's tasks.

The following play shows an example with `pre_tasks`, `roles`, `tasks`, and `post_tasks`. It is unusual that a play would contain all four sections.

```
- name: Play to illustrate order of execution
  hosts: ios
  pre_tasks:
    - name: This task runs first
      ansible.builtin.debug:
        msg: This task is in pre_tasks
      notify: my handler
      changed_when: true
  roles:
    - role: role1
  tasks:
    - name: This task runs after the roles
      ansible.builtin.debug:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        msg: This task is in tasks
      notify: my handler
      changed_when: true
  post_tasks:
    - name: This task runs last
      ansible.builtin.debug:
        msg: This task is in post_tasks
      notify: my handler
      changed_when: true
```

Note

In general, if you think you need `pre_tasks` and `post_tasks` sections in your play because you are using `roles`, consider importing the roles as tasks and including only a `tasks` section. Alternatively, it might be simpler to have multiple plays in your playbook.

References

Roles - Ansible Documentation

https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Quiz

Ansible Role Structure

Choose the correct answers to the following questions:

1. **Which one of the following statements best describes Ansible Roles?**
   a. Configuration settings that enable specific users to run Ansible Playbooks.
   b. A long and complex Ansible Playbook.
   c. Collections of YAML task files and supporting items arranged in a specific structure.
   d. A core engine feature of Ansible that includes or imports files.

2. **Which one of the following subdirectories is not part of the structure of an Ansible Role?**
   a. `defaults/`
   b. `files/`
   c. `playbooks/`
   d. `meta/`
   e. `vars/`

3. **Which one of the following Ansible Role subdirectories contains variables with low precedence?**
   a. `defaults/`
   b. `vars/`
   c. `meta/`
   d. `files/`

4. **Which one of the following Ansible Role subdirectories contains variables that cannot be overridden by inventory variables?**
   a. `defaults/`
   b. `vars/`
   c. `meta/`
   d. `files/`

5. **Which one of the following statements related to Ansible Roles is false?**
   a. Ansible Roles make larger projects more manageable.
   b. Ansible Roles can be developed in parallel by different network developers.
   c. Ansible Roles must store passwords and private keys to make it more agile.
   d. Ansible Roles group content together, enabling easy sharing of code.

For use by Abdul Majeed Khan Yamkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. What are three ways you can use an Ansible Role in a play? (Choose three.)

a. By importing the Ansible Role using the `ansible.builtin.import_role` module.

b. By including the Ansible Role using the `ansible.builtin.include_role` module.

c. By using a `roles` section in the play with the Ansible Role in the list of Ansible Roles.

d. By including the Ansible Role in the `pre_roles` or `post_roles` section of the play.

Solution

Ansible Role Structure

Choose the correct answers to the following questions:

1. Which one of the following statements best describes Ansible Roles?

a. Configuration settings that enable specific users to run Ansible Playbooks.
b. A long and complex Ansible Playbook.
c. Collections of YAML task files and supporting items arranged in a specific structure.
d. A core engine feature of Ansible that includes or imports files.

2. Which one of the following subdirectories is not part of the structure of an Ansible Role?

a. `defaults/`
b. `files/`
c. `playbooks/`
d. `meta/`
e. `vars/`

3. Which one of the following Ansible Role subdirectories contains variables with low precedence?

a. `defaults/`
b. `vars/`
c. `meta/`
d. `files/`

4. Which one of the following Ansible Role subdirectories contains variables that cannot be overridden by inventory variables?

a. `defaults/`
b. `vars/`
c. `meta/`
d. `files/`

5. Which one of the following statements related to Ansible Roles is false?

a. Ansible Roles make larger projects more manageable.
b. Ansible Roles can be developed in parallel by different network developers.
c. Ansible Roles must store passwords and private keys to make it more agile.
d. Ansible Roles group content together, enabling easy sharing of code.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. What are three ways you can use an Ansible Role in a play? (Choose three.)

a. By importing the Ansible Role using the `ansible.builtin.import_role` module.

b. By including the Ansible Role using the `ansible.builtin.include_role` module.

c. By using a `roles` section in the play with the Ansible Role in the list of Ansible Roles.

d. By including the Ansible Role in the `pre_roles` or `post_roles` section of the play.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Creating Roles

Objectives

- Create a role in a playbook's project directory and run it as part of one of the plays in the playbook.

The Ansible Role Creation Process

Creating roles in Ansible does not require any special development tools. Creating and using an Ansible Role is a three-step process:

1. Create the role directory structure.
2. Define the role content.
3. Use the role in a playbook.

Creating the Role Directory Structure

Ansible looks for roles in a subdirectory called `roles` in the project directory containing your Ansible Playbook. Each role has its own directory with a standardized directory structure. This structure enables you to store roles with the playbook and other supporting files.

For example, the following directory structure contains the files within the `layer3_interface` role:

```
[user@host ~]$ tree roles/layer3_interface/
roles/layer3_interface/
├── README.md
├── default
│   └── main.yml
├── meta
│   └── main.yml
├── files
└── tasks
    ├── ios.yml
    ├── junos.yml
    └── main.yml

4 directories, 6 files
```

The `README.md` file provides a basic human-readable description of the role, documentation, examples of how to use it, and any non-Ansible role requirements.

The `defaults` subdirectory contains a `main.yml` file that defines the default values for the variables used in the role.

The `meta` subdirectory contains a `main.yml` file that specifies information about the author, license, compatibility, and dependencies for the role.

If a subdirectory exists but is empty, such as `files` in this example, then it is ignored. You can omit the subdirectory if the role does not use it.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The `main.yml` file in the `tasks` subdirectory contains the role's task definitions. In this example, the `tasks` subdirectory also contains specific tasks for Cisco IOS and Juniper Junos managed nodes in the `ios.yml` and `junos.yml` files.

Creating an Ansible Role Structure

You can create all the subdirectories and files needed for a new Ansible Role in VS Code, or by using standard Linux commands. Alternatively, you can also use the `ansible-galaxy` command.

Use the following procedure to create a role structure using the `ansible-galaxy` command:

1. Change to the `roles` directory in your project:

```
[user@host project]$ cd roles
[user@host roles]$
```

2. Run the `ansible-galaxy init` command, specifying the role name as an argument to create the role:

```
[user@host roles]$ ansible-galaxy init vlans_configuration
- Role vlans_configuration was created successfully
```

The following screen capture illustrates the directory structure created for the `vlans_configuration` role as displayed in VS Code:

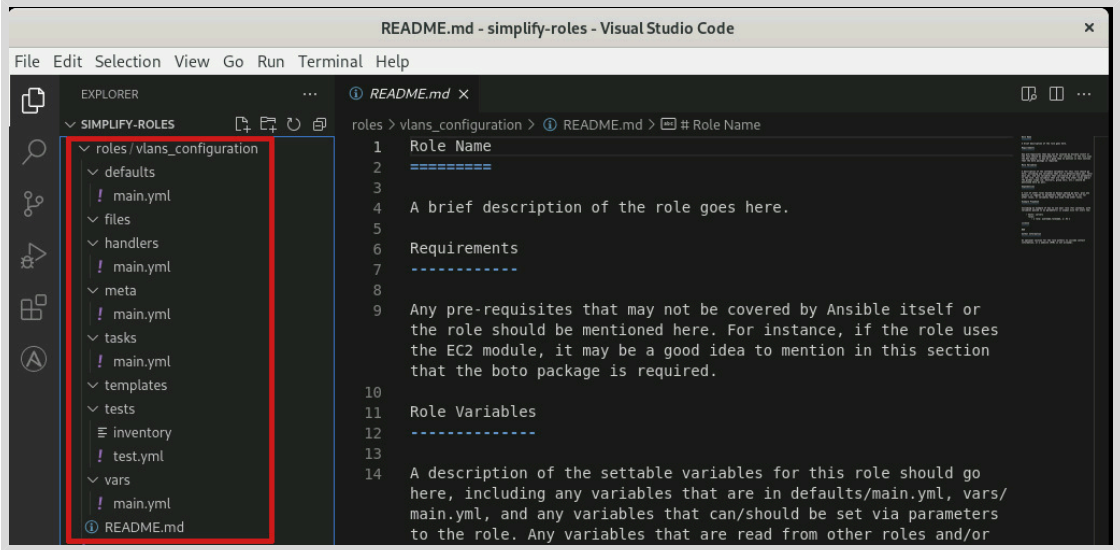


Figure 6.1: Skeleton for the `vlans_configuration` Role

Defining the Role Content

After creating the directory structure, you need to write the content of the role. A good place to start is the *`ROLENAME`*`/tasks/main.yml` task file. This file contains the main list of tasks that the role runs.

As an example, suppose that you are creating an Ansible Role to configure Layer 3 (L3) interfaces for Cisco IOS and Juniper Junos managed nodes. You can create the following `tasks/main.yml` task file within the role:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Load the right configuration task
  ansible.builtin.include_tasks: "{{ ansible_network_os }}.yml"
```

For Cisco IOS managed nodes, the `ansible_network_os` variable has the `cisco.ios.ios` value. For Juniper Junos managed nodes, the variable has the `junipernetworks.junos.junos` value. Depending on the value of that variable, the role configures the L3 interface for Cisco IOS managed nodes or for Juniper Junos managed nodes. Because the `cisco.ios.ios.yml` and `junipernetworks.junos.junos.yml` task files are defined within a role task, the files are retrieved from the role's `tasks` subdirectory.

You can create the `tasks/cisco.ios.ios.yml` task file with the following content:

```
---
- name: Configure L3 interface for Cisco IOS managed node
  cisco.ios.ios_l3_interfaces:
    config: "{{ l3_interface }}"
```

Similarly, you can create the `tasks/junipernetworks.junos.junos.yml` task file as follows:

```
---
- name: Configure L3 interface for Juniper Junos managed node
  junipernetworks.junos.junos_l3_interfaces:
    config: "{{ l3_interface }}"
```

Note

Notice that you can easily grow your role to any supported platform. For example, if you want to add Arista EOS managed nodes as another option to configure, then it would be enough to add the `tasks/arista.eos.eos.yml` task file with the following content:

```
---
- name: Configure L3 interface for Arista EOS managed node
  arista.eos.eos_l3_interfaces:
    config: "{{ l3_interface }}"
```

You can find the list of supported platforms at: `https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html`.

You can use the `defaults/main.yml` file to set the default value for the `l3_interface` variable. The value defined in the `defaults` subdirectory can be overwritten by defining the variable in the play that loads the role.

Defining Role Dependencies

You can define role dependencies in the `meta/main.yml` file in the role directory hierarchy. Role dependencies enable a role to include other roles as dependencies.

For example, a role that performs some configuration on your managed nodes might depend on a role that backs up their initial configuration.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

A `meta/main.yml` file might also have a top-level `galaxy_info` key that has a dictionary of other attributes that specify the author, purpose, license, and metadata such as platforms supported for managed nodes.

The following is a sample `meta/main.yml` file with dependencies on the `node_backups` and `node_reload` roles:

```
---
galaxy_info:
  author: Student
  description: Network managed nodes configuration
  company: Company, Ltd.
  license: GPL-2.0
  min_ansible_version: 2.1

dependencies:
  - role: node_backup
    path: backups
  - role: node_reload
```

By default, if multiple roles have dependencies on another role, and that role is called by different roles in the play multiple times with the same attributes, then the role only runs the first time it is called. You can override this behavior by setting the `allow_duplicates` variable to `yes` in your role's `meta/main.yml` file.

Important
You should limit your role's dependencies on other roles. Dependencies make it harder to maintain your role, especially if your role has many complex dependencies.

Using the Role in a Playbook

When the role structure has been created and its content defined, you can use the role in a playbook.

You can use roles in playbooks in several ways. You can use the `roles` keyword in a play to run one or more roles before any tasks in the section defined by the `tasks` keyword. If you need to run some tasks before running the roles, then you can place those tasks in the section defined by the `pre_tasks` keyword.

For even more control of the order in which a role runs, use the `ansible.builtin.include_role` or the `ansible.builtin.import_role` module. By using these modules, you can run a role in a specific location in the play.

For example, the following playbook includes the role named `layer3_interface`, which configures the L3 interface on your managed nodes. Because the `layer3_interface` role is defined within a role task, the role is retrieved from the `roles` subdirectory in the project directory containing your playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Configure L3 interface on managed nodes
  hosts: all
  gather_facts: false
  tasks:
    - name: Load layer3_interface role
      ansible.builtin.include_role:
        name: layer3_interface
```

Changing Role Behavior with Variables

A well-written Ansible Role uses default variables to alter the role's behavior to match a related configuration scenario. Roles that use variables are more generic and reusable in various contexts.

Keep in mind that the value of any variable defined in a role's `defaults` subdirectory is overwritten if the same variable is defined in any of the following places:

- In an inventory file, either as a host variable or a group variable
- In a YAML file under the `group_vars` or `host_vars` directories of a playbook project
- As a variable nested in the `vars` keyword of a play
- As a variable when including the role in `roles` keyword of a play

Suppose you have created a role named `banner` that configures the `login` and `motd` banners on your managed nodes. The role uses the `banner_text` and the `banner_type` variables to specify the text and the type of banner to configure. You can define these variables in the `main.yml` file within the `defaults` or `var` subdirectories of the role:

```
---
banner_text: Default message
banner_type: login
```

The following playbook uses the `banner` role with a different value for the `banner_text` and `banner_type` variables:

```
---
- name: Configure banners on managed nodes
  hosts: all
  gather_facts: false
  vars:
    - banner_text: This managed node is managed by Ansible
    - banner_type: motd
  roles:
    - role: banner
```

If the role variables were defined in the `defaults` subdirectory, then the `banner_text` and `banner_type` variables defined in the playbook replace the variable values. If the role variables were defined in the `var` subdirectory, then the `banner_text` and `banner_type` variables defined in the playbook do not replace their value.

The following example also shows how to use the `banner` role with a different value for the `banner_text` and `banner_type` role variables:

```
---
- name: Configure banners on managed nodes
  hosts: all
  gather_facts: false
  roles:
    - role: banner
      banner_text: This managed node is managed by Ansible
      banner_type: motd
```

This time the value specified for the `banner_text` and `banner_type` role variables replace the variable reference regardless of being defined in the role's `vars` or `defaults` directory.

Important

Variable precedence can be confusing when working with role variables in a play. It is important to understand and take this into account to keep your play from accidentally changing the internal functioning of the role.

- Most other variables override a role's default variables: inventory variables, play variables, inline role parameters, and so on.
- Fewer variables can override variables defined in a role's `vars` directory. Facts, variables loaded with `include_vars`, registered variables, and role parameters can override these variables. Inventory variables and play variables cannot.
- Variables declared inline as role parameters have high precedence; they can also override variables defined in a role's `vars` directory.
- If a role parameter has the same name as a variable set in play `vars` section, a role's `vars` section, or an inventory or playbook variable, the role parameter overrides the other variable.

Recommended Practices for Role Development

Ansible Roles enable you to break down playbooks into multiple files, resulting in reusable code. To maximize the effectiveness of newly developed roles, consider implementing the following recommended practices into your role development:

- Maintain each role in its own version control repository. Ansible works well with Git-based repositories.
- Use variables to configure roles so that you can reuse the role to perform similar tasks in similar circumstances.
- Avoid storing sensitive information in a role, such as passwords or SSH keys. Configure role variables that contain sensitive values when called in a play with default values that are not sensitive.
- Use the `ansible-galaxy init` command to create your role structure, and then remove any unnecessary files and directories.
- Create and maintain a `README.md` file and a `meta/main.yml` file to document the role's purpose, author, and usage.
- Keep your role focused on a specific purpose or function. Instead of making one role do many things, write multiple roles.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Reuse roles often.
- Resist creating new roles for edge configurations. If an existing role accomplishes most of the required configuration, refactor the existing role to integrate the new configuration scenario.
- Use integration and regression testing techniques to ensure that the role provides the required new functionality and does not cause problems for existing playbooks.

A longer unofficial list of good practices to follow when you write a role is available at `https://redhat-cop.github.io/automation-good-practices/#_roles_good_practices_for_ansible`.

References

Using Roles - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#using-roles

Using Role Dependencies - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html#using-role-dependencies

Defining Variables in Included Files and Roles - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#defining-variables-in-included-files-and-roles

Roles Good Practices for Ansible
https://redhat-cop.github.io/automation-good-practices/#_roles_good_practices_for_ansible

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Creating a Network Automation Role

Create an Ansible Role that can back up the configuration of managed network nodes made by different vendors.

Outcomes

- Create a role that backs up configuration files for Junos and IOS managed network nodes, and then stores those backup files in a separate location.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start simplify-roles
```

Instructions

1. Open VS Code and switch to the **Terminal** tab, or open a GNOME terminal, and create the directory structure for a role named `net_backup`. The role must include the `meta`, `tasks`, and `vars` directories.

   Run the following commands to create the role and display the initial role structure and files:

```
[student@workstation ~]$ cd /home/student/simplify-roles
[student@workstation simplify-roles]$ mkdir roles; cd roles
[student@workstation roles]$ ansible-galaxy init net_backup
- Role net_backup was created successfully
[student@workstation roles]$ rm -rvf \
net_backup/{files,defaults,templates,handlers,tests}
...output omitted...
[student@workstation roles]$ tree net_backup/
net_backup/
├── README.md
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
└── vars
    └── main.yml

3 directories, 4 files
```

2. Open the `/home/student/simplify-roles` directory in VS Code.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Create the `roles/net_backup/tasks/junos_netconf_enable.yml` task file to enable NETCONF on the Junos managed nodes.
- Create the `roles/net_backup/tasks/backup_junos.yml` task file to back up the configurations on the Junos managed nodes.
- Create the `roles/net_backup/tasks/backup_ios.yml` task file to back up the configurations on the IOS managed nodes.

2.1. Open VS Code and then click **File** > **Open Folder**.

2.2. Navigate to **Home** > **simplify-roles** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

2.3. Create a task file named `roles/net_backup/tasks/junos_netconf_enable.yml` to enable NETCONF on the Junos managed nodes.

The completed file must consist of the following content:

```
---
- name: Enable the netconf service on port 830
  vars:
    ansible_connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
    netconf_port: 830
    state: present
```

2.4. Create a task file named `roles/net_backup/tasks/backup_junos.yml` to back up the configurations on the Junos managed nodes.

The completed file must consist of the following content:

```
---
- name: Backup Junos configurations
  junipernetworks.junos.junos_config:
    backup: true
    backup_options:
      dir_path: "{{ backup_path }}"
      filename: "{{ inventory_hostname }}.cfg"
```

2.5. Create a task file `roles/net_backup/tasks/backup_ios.yml` to back up the configurations on the IOS managed nodes.

The completed file must consist of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Backup IOS XE configurations
  cisco.ios.ios_config:
    backup: true
    backup_options:
      dir_path: "{{ backup_path }}"
      filename: "{{ inventory_hostname }}.cfg"
```

3. Create a task file named `roles/net_backup/tasks/junos.yml`. Add two tasks to the file to include the `junos_netconf_enable.yml` and `backup_junos.yml` task files.

   The completed file must consist of the following content:

```
---
- name: Enable NETCONF
  ansible.builtin.include_tasks:
    file: junos_netconf_enable.yml

- name: Create Junos backup files
  ansible.builtin.include_tasks:
    file: backup_junos.yml
```

4. Create the `roles/net_backup/tasks/ios.yml` file. Add a task to the file that includes the `tasks/backup_ios.yml` task file.

   The completed file must consist of the following content:

```
---
- name: Create IOS backup files
  ansible.builtin.include_tasks:
    file: tasks/backup_ios.yml
```

5. Define the role variable called `backup_path` in the `roles/net_backup/vars/main.yml` file. Set the `backup_path` variable to the `backups` value.

   The completed file must consist of the following content:

```
---
# vars file for net_backup
backup_path: backups
```

6. Edit the `roles/net_backup/tasks/main.yml` task file to dynamically include the `junos.yml` and `ios.yml` task files.

   Use the `tree` command to display the final role structure and files.

   6.1. Edit the `roles/net_backup/tasks/main.yml` task file to dynamically include the `junos.yml` and `ios.yml` task files.

   The completed file must consist of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
# tasks file for net_backup
- name: Backup network device configs
  ansible.builtin.include_tasks:
    file: "{{ ansible_network_os.split('.')[2] }}.yml"
```

6.2. Use the `tree` command to display the final role structure and files:

```
[student@workstation simplify-roles]$ tree roles/net_backup
roles/net_backup
├── README.md
├── meta
│   └── main.yml
├── tasks
│   ├── backup_ios.yml
│   ├── backup_junos.yml
│   ├── ios.yml
│   ├── junos.yml
│   ├── junos_netconf_enable.yml
│   └── main.yml
└── vars
    └── main.yml

3 directories, 9 files
```

7. Edit the `roles/net_backup/meta/main.yml` file to document your role. Add your name, a role description, and, optionally, your company name:

```
galaxy_info:
  author: your name
  description: your role description
  company: your company (optional)
...output omitted...
```

Note

Normally, you should also modify the `README.md` file to document your role.

That step is skipped in this exercise to save time.

8. Create a playbook named `net_backup.yml` that uses the `net_backup` role, and then run the playbook.

8.1. Create a playbook in the `/home/student/simplify-roles` directory named `net_backup.yml` that uses the `net_backup` role. Add the following content to the playbook:

```
---
- name: Back up managed network node configurations
  hosts:
    - junos
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - ios
  gather_facts: false
  tasks:
    - name: Backup network managed node configurations
      ansible.builtin.include_role:
        name: net_backup
      when: ansible_network_os is defined
```

8.2. Run the `net_backup.yml` playbook:

```
[student@workstation simplify-roles]$ ansible-navigator run net_backup.yml
...output omitted...
```

9. Display the contents of some files generated by the playbook to verify that it was successful, or review them in VS Code.

```
[student@workstation simplify-roles]$ head backups/iosxe1.lab.example.com.cfg
...output omitted...
[student@workstation simplify-roles]$ head backups/junos2.lab.example.com.cfg
...output omitted...
```

10. Click **File** > **Close Folder** in VS Code to close the `/home/student/simplify-roles` directory, or run the `cd` command in a GNOME terminal to return to the `student` home directory:

```
[student@workstation simplify-roles]$ cd
```

Finish

On the `workstation` machine, 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 simplify-roles
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Deploying Roles from External Content Sources

Objectives

- Select and retrieve roles from external sources such as Git repositories or Ansible Galaxy, and use them in your playbooks.

External Content Sources

If you use roles in your Ansible Playbooks, then you might want to use a centrally managed location to store and manage them.

Using a centralized location for roles helps to ensure that your projects have the current version of the role, and that each project benefits from updates and bug fixes done by other project teams that share the role.

For example, if the role is created by your organization, that role might be stored in a Git repository managed by your organization, or as a `.tar` file.

Using the Ansible Galaxy Library

The open source community maintains some roles through the Ansible Galaxy website at `https://galaxy.ansible.com`. Ansible Galaxy is a public library of Ansible content written by various Ansible administrators and users, and is publicly available.

The Ansible Galaxy library contains thousands of Ansible roles, and it has a searchable database that helps you identify roles that might help you accomplish an administrative task.

Warning
The roles in the Ansible Galaxy library can be useful, but they are not supported by Red Hat, nor does Red Hat audit or review this code. Volunteers in the Ansible community maintain this site. Use these roles at your own risk.

Searching for Roles

You can search for roles on the `https://galaxy.ansible.com` website. You can also use the `ansible-galaxy search` subcommand to search Ansible Galaxy for roles.

Use the `--author`, `--platforms`, and `--galaxy-tags` options to narrow the search results.

The following example displays the names of roles that include "vpn", and are available for the Enterprise Linux (EL) platform:

```
[user@host]$ ansible-galaxy search vpn --platforms EL

Found 70 roles matching your search:

 Name                                          Description
 ----                                          -----------
```

```
 abelfodil.ansible_role_wireguard            Installs Wireguard incl. systemd i
 adamruzicka.wireguard                       Configure WireGuard VPN
 ansyble.webtier                             Webtier deployment
 aprt5pr.ocserv                              OpenConnect SSL VPN Server (ocserv)
 arillso.wireguard                           Installation and configuration of
 asm0dey.ansible_role_openvpn                OpenVPN playbook for CentOS/Fedora
...output omitted...
```

Installing Roles

Use the `ansible-galaxy` command to download, install, and manage roles, whether they are from the Ansible Galaxy library or from some other source.

You can also use the `ansible-galaxy` command to search for, display information about, install, list, remove, or initialize roles.

By default, Ansible looks for installed roles in the following locations, ordered from the lowest to highest precedence:

- In the same directory as your playbook
- In the path specified by the `roles_path` variable
- In your project `roles` directory

Important

It is increasingly common for roles to be packaged as Ansible Content Collections and offered by their authors through the following methods:

- In Red Hat Certified or Validated Ansible Content Collections from the Red Hat Ansible automation hub at `https://console.redhat.com`, or through a private automation hub
- As private content packaged from a private automation hub
- By the community from Ansible Galaxy at `https://galaxy.ansible.com`

This section only covers how to get roles that are not packaged into an Ansible Content Collection.

Using a Role Requirements File

If your playbook requires that specific roles be installed, then you can create a `roles/requirements.yml` file in the project directory that specifies which roles are needed.

The `roles/requirements.yml` file acts as a dependency manifest for the playbook project that enables playbooks to be developed and tested separately from any supporting roles.

For example, you could have a role in a public repository on a Git server at `https://git.example.com/someauthor/netrole`. A simple `requirements.yml` file to install the `netrole` role might contain the following content:

```
---
- src: https://git.example.com/someauthor/netrole
  scm: git
  version: "1.5.0"
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The `src` attribute specifies the source of the role, in this case the location of the role repository on your Git server. You can also use SSH key-based authentication by specifying something like `git@git.example.com:someauthor/netrole` as provided by your Git repository.

The `scm` attribute indicates that this role is from a Git repository.

The `version` attribute is optional, and specifies the version of the role to install, in this case `1.5.0`. If a role is stored in a Git repository, then the version can be the name of a branch, a tag, or a Git commit hash. If you do not specify a version, the command uses the latest commit on the default branch.

Important

Specify the version of the role in your `requirements.yml` file, especially for playbooks in production.

If you do not specify a version, then you get the latest version available when you install the role.

If the upstream author makes changes to the role that are incompatible with your playbook, then the changes might cause an automation failure or other problems.

If you have a `.tar` file that contains a role, then you can use the `roles/requirements.yml` file to install that role.

```
---
# from a role .tar file, given a URL;
#   supports 'http', 'https', or 'file' protocols
- src: file:///opt/local/roles/myrole.tar
  name: myrole

- src: https://www.example.com/role-archive/someauthor.otherrole.tar
  name: someauthor.otherrole
```

Note

Red Hat recommends storing your roles in a version control system such as Git.

If a recent change to a role causes problems, then using version control enables you to roll back to a previous, stable version of the role.

To install roles using a role requirements file, run the `ansible-galaxy role install` command from within your project directory.

Include the following options:

- `-r roles/requirements.yml` to specify the location of your requirements file
- `-p roles` to install the role into a subdirectory of the `roles` directory

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

If you do not specify the `-p roles` option, then `ansible-galaxy` uses the first directory defined by the `roles_path` variable to determine where to install the role. This defaults to the user's `~/.ansible/roles` directory, which is outside the project directory and unavailable to the execution environment if you use the `ansible-navigator` command to run your playbooks.

One way to avoid the need to specify `-p roles` is to apply the following setting in the `defaults` section of your project's `ansible.cfg` file:

```
[defaults]
roles_path = roles
```

```
[user@host project]$ ansible-galaxy role install -r roles/requirements.yml \
-p roles
Starting galaxy role install process
- downloading role from https://git.example.com/someauthor/netrole
- extracting netrole to /home/user/project/roles/netrole
- netrole (1.5.0) was installed successfully
```

Important

If you use the `ansible-navigator run` command to run your playbook, then install all required roles before running the command.

If you use automation controller to run your playbook, then you do not need to manually install roles because automation controller automatically downloads all roles specified in your `roles/requirements.yml` file when it runs your playbook.

Creating a Role Requirements File

The following example shows how to configure a requirements file that uses various remote sources:

```
[user@host project]$ cat roles/requirements.yml
---
# from Ansible Galaxy, using the latest version
- src: someauthor.netrole (1)

# from Ansible Galaxy, overriding the name and using a specific version
- src: someauthor.netrole
  version: "1.5.0" (2)
  name: netrole

# from any Git based repository, using HTTPS
- src: https://github.com/someauthor/ansible-role-otherrole.git
  scm: git (3)
  version: main
  name: otherrole
```

```
# from a role .tar file, given a URL;
#   supports 'http', 'https', or 'file' protocols
- src: file:///opt/local/roles/myrole.tar ❹
  name: myrole ❺
```

❶ The `src` keyword specifies the Ansible Galaxy role name. If the role is not hosted on Ansible Galaxy, then the `src` keyword indicates the role's URL.

❷ The `version` keyword specifies the role version. The `version` keyword can be any value that corresponds to a branch, tag, or commit hash from the role's software repository.

❸ If the role is hosted in a source control repository, the `scm` attribute is required.

❹ If the role is hosted on Ansible Galaxy or as a `.tar` file, the `scm` keyword is omitted.

❺ The `name` keyword is used to override the local name of the role.

Managing Downloaded Roles

The `ansible-galaxy` command can also manage local roles, such as those roles found in the `roles` directory of a playbook project.

The `ansible-galaxy list` subcommand lists the local roles.

```
[user@host project]$ ansible-galaxy list
# /home/user/project/roles
- netrole, 1.5.0
- otherrole, main
- myrole, (unknown version)
...output omitted...
```

You can remove a role with the `ansible-galaxy remove` subcommand.

```
[user@host ~]$ ansible-galaxy remove myrole
- successfully removed myrole
```

References

ansible-galaxy - Ansible Documentation
https://docs.ansible.com/ansible/latest/cli/ansible-galaxy.html

Red Hat Hybrid Cloud Console | Ansible Automation Platform Dashboard
https://console.redhat.com/ansible/ansible-dashboard

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Running a Playbook that Requires a Role

Create a `requirements.yml` file in a project to download and install an Ansible Role, and then use automation controller to automatically get that role and run a playbook that uses it.

Outcomes

- Create a requirements file to specify role dependencies for a playbook in a Git repository.
- Create a project in automation controller that points to the Git repository.
- Launch a job template that automatically retrieves the playbook role dependencies from the requirements 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 also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start simplify-external
```

Instructions

1. Verify that the Message of the Day (MOTD) banner is not configured on the `iosxe1.lab.example.com` managed node.

   1.1. Connect to the `iosxe1.lab.example.com` managed node as the `student` user:

```
[student@workstation ~]$ ssh student@iosxe1.lab.example.com


iosxe1.lab.example.com#
```

   1.2. Return to the `workstation` machine:

```
iosxe1.lab.example.com#exit
Connection to iosxe1.lab.example.com closed by remote host.
Connection to iosxe1.lab.example.com closed.
```

2. From the automation controller web UI, create a project that points to the `simplify-external` Git project.

   Use the information in the following table to create the project:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Field	Value
Name	`Cisco IOS MOTD`
Organization	`Default`
Source Control Type	`Git`
Source Control URL	`git@git.lab.example.com:student/simplify-external`
Source Control Credential	`Git Projects`
Options	`Delete`
	`Allow Branch Override`

2.1. Navigate to `https://controller.lab.example.com` and log in as `admin` using `redhat` as the password.

2.2. Navigate to **Resources** > **Projects** and then click **Add**.

2.3. Create the project with settings from the preceding table and then click **Save**.

Automation controller immediately attempts to synchronize the project. The **Last Job Status** field updates to indicate that the project synchronization succeeded.

> **Important**
> If synchronization fails, ensure that the project specifies the correct source control URL and credentials and then attempt the project synchronization again.

3. Create a job template to run the `ios_motd.yml` playbook from the `Cisco IOS MOTD` automation controller project. Launch the job template and review the job output.

Use the information in the following table to create the job template:

Field	Value
Name	`Cisco IOS MOTD banner`
Description	`Configure a MOTD banner on Cisco IOS managed nodes`
Inventory	`Common`
Project	`Cisco IOS MOTD`
Playbook	`ios_motd.yml`
Credentials	`Cisco IOS XE Auth`

3.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**.

3.2. Create the job template with settings from the preceding table and then click **Save**.

3.3. Click **Launch** to run the job template.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.4. Navigate to **Views** > **Jobs** and look for a numbered job with `Cisco IOS MOTD banner` in the name. The job displays a status of **Failed**.

3.5. Click the name of the `Cisco IOS MOTD banner` job to review the job output. The `ios_motd.yml` playbook fails in the `Configure the MOTD banner` task:

```
...output omitted...
TASK [Configure the MOTD banner] ************************************************
ERROR! the role 'banner' was not found in /runner/project/roles:/runner/
requirements_roles:/home/runner/.ansible/roles:/usr/share/ansible/roles:/etc/
ansible/roles:/runner/project

The error appears to be in '/runner/project/ios_motd.yml': line 12, column 15, but
 may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

      ansible.builtin.include_role:
        name: banner
              ^ here
...output omitted...
```

The task fails because the playbook includes the `banner` role, but the project does not indicate where to find that role.

4. Clone the `git@git.lab.example.com:student/simplify-external` Git repository into the `/home/student/git-repos` directory. Create a branch named `exercise` and check out that branch.

4.1. On the `workstation` machine, change to the `/home/student/git-repos` directory and then use the `git clone` command to clone the `git@git.lab.example.com:student/simplify-external` Git repository:

```
[student@workstation ~]$ cd ~/git-repos/
[student@workstation git-repos]$ git clone \
git@git.lab.example.com:student/simplify-external
Cloning into 'simplify-external'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

4.2. Change to the `simplify-external` directory, and then create a branch named `exercise` and check it out:

```
[student@workstation git-repos]$ cd simplify-external
[student@workstation simplify-external]$ git checkout -b exercise
Switched to a new branch 'exercise'
```

5. Open the `/home/student/git-repos/simplify-external` directory in VS Code. Examine the `ios_motd.yml` playbook. Notice that the playbook uses the `banner` role. For this task to succeed, the `banner` role must be installed.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

5.1. Open VS Code and click **File** > **Open Folder**.

5.2. Navigate to **git-repos** > **simplify-external** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

5.3. Examine the `ios_motd.yml` playbook. The playbook uses the `ansible.builtin.include_role` module to include the `banner` role.

```
---
- name: Add a MOTD banner on Cisco IOS managed nodes
  hosts: ios
  gather_facts: false

  tasks:
    - name: Configure the MOTD banner
      vars:
        banner_text: Managed by Ansible
        banner_type: motd
      ansible.builtin.include_role:
        name: banner
```

Notice that the playbook is the only file in the Git repository and the playbook does not specify where to find the `banner` role.

6. Create a role requirements file in your project directory that downloads and installs the `banner` role from the `git@git.lab.example.com:student/banner` Git repository. The `banner` role configures banners on Cisco IOS and Juniper Junos managed nodes.

After you create the role requirements file, use either VS Code or the CLI to add, commit, and push the changes to the remote repository.

6.1. Switch to the **Terminal** tab in VS Code, or return to your GNOME terminal. Use the `mkdir` command to create the `roles` directory in your project:

```
[student@workstation simplify-external]$ mkdir roles
```

6.2. In VS Code, create the `roles/requirements.yml` file with the following content:

```
---
- src: git@git.lab.example.com:student/banner
  scm: git
  version: main
```

The `src` attribute specifies the location of the `banner` role. The `scm` attribute specifies Git as the source code management tool. The `version` attribute specifies the use of the `main` branch of the specified Git repository.

6.3. Add, commit, and push the changes to the remote repository. Use a descriptive commit message, such as `Add requirements file`.

If you choose to use the CLI, then use the `git add`, `git commit`, and `git push` commands from your GNOME terminal:

```
[student@workstation simplify-external]$ git add roles
[student@workstation simplify-external]$ git commit -m "Add requirements file"
 1 file changed, 4 insertions(+)
 create mode 100644 roles/requirements.yml
[student@workstation simplify-external]$ git push -u origin exercise
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 6 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 422 bytes | 422.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for exercise, visit:
remote:   https://git.lab.example.com/student/simplify-external/-/merge_requests/
new?merge_request%5Bsource_branch%5D=exercise
remote:
To git.lab.example.com:student/simplify-external
 * [new branch]      exercise -> exercise
Branch 'exercise' set up to track remote branch 'exercise' from 'origin'.
```

7. From the automation controller web UI, update the `Cisco IOS MOTD` project resource so that the project uses the `exercise` branch you created with the requirements file.

   7.1. Navigate to **Resources** > **Projects** and then click the **Edit Project** icon for the **Cisco IOS MOTD** project.

   7.2. Enter `exercise` in the **Source Control Branch/Tag/Commit** field and then click **Save**. Automation controller immediately attempts to synchronize the project. The **Last Job Status** field updates to indicate that the project synchronization succeeded.

   7.3. *(Optional)* Click **Successful** in the **Last Job Status** field to review the job output.

Figure 6.2: Project synchronization includes the requirements file

The `Cisco IOS MOTD` project now includes the `requirements.yml` file specifying the role dependencies for the `ios_motd.yml` playbook.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

8. Launch the `Cisco IOS MOTD banner` job template and verify that the resulting job succeeds. Verify that MOTD banner is now configured on the `iosxe1.lab.example.com` managed node.

8.1. Navigate to **Resources** > **Templates** and then click the **Launch Template** icon for the **Cisco IOS MOTD banner** job template. The job displays the following output:

```
PLAY [Add a MOTD banner on Cisco IOS managed nodes] ****************************

TASK [Configure the MOTD banner] ***********************************************

TASK [banner : Configure managed node banner] **********************************
included: /runner/requirements_roles/banner/tasks/cisco.ios.ios.yml for
 iosxe1.lab.example.com, iosxe2.lab.example.com

TASK [banner : Create a Cisco IOS banner] **************************************
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=2   changed=1   unreachable=0   failed=0      ...
iosxe2.lab.example.com     : ok=2   changed=1   unreachable=0   failed=0      ...
```

8.2. On the `workstation` machine, connect to the `iosxe1.lab.example.com` managed node as the `student` user:

```
[student@workstation simplify-external]$ ssh student@iosxe1.lab.example.com

Managed by Ansible



iosxe1.lab.example.com#
```

8.3. Return to the `workstation` machine:

```
iosxe1.lab.example.com#exit
Connection to iosxe1.lab.example.com closed by remote host.
Connection to iosxe1.lab.example.com closed.
```

9. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/simplify-external` directory. On the `workstation` machine, run the `cd` command to return to the `/home/student` directory:

```
[student@workstation simplify-external]$ cd
```

Finish

On the `workstation` machine, 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 simplify-external
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Getting Roles and Modules from Ansible Content Collections

Objectives

- Obtain a set of related roles, supplementary modules, and other content from an Ansible Content Collection and use them in a playbook.

Ansible Content Collections

When Ansible was first developed, all the modules that it used were included as part of the core software package. As the number of modules increased, it became harder for the upstream project to manage all these modules. Every module required a unique name and synchronizing module updates with updates to the core Ansible code.

With *Ansible Content Collections*, Ansible code updates are separated from updates to modules and plug-ins. An Ansible Content Collection provides a set of related modules, roles, and other plug-ins that you can use in your playbooks. This approach enables vendors and developers to maintain and distribute their collections at their own pace, independently of Ansible releases.

For example:

- The `ansible.netcommon` content collection provides modules and plug-ins that you can use to help automate the management of network, security, and cloud devices.
- The `cisco.ios` content collection, supported and maintained by Cisco, provides modules and plug-ins that manage Cisco IOS network appliances.
- The `network.ospf` content collection provides a role that enables you to manage Open Shortest Path First (OSPF) resources and perform Shortest Path First (SPF) health checks on devices.

Ansible Content Collections also provide flexibility. You can install only the content you need instead of installing all supported modules.

You can also select a specific version of a collection (possibly an earlier or later one) or choose between a version of a collection supported by Red Hat or vendors, or one provided by the community.

Ansible 2.9 and later support Ansible Content Collections. 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 2.3 provides automation execution environments based on Ansible Core 2.14 that inherit this feature.

> **Note**
> You can develop your own collections to provide custom roles and modules to your teams. You can learn more about how to create Ansible Content Collections in the course *Developing Advanced Automation with Red Hat Ansible Automation Platform (DO374)*.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Namespaces for Ansible Content Collections

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 collections.

The namespace is the first part of a collection name. For example, all the collections in the `network` namespace have names such as `network.ospf`, `network.bgp`, and `network.vpn`. Likewise, Ansible Content Collections that Red Hat directly maintains and supports might use the `redhat` namespace, and have names such as `redhat.rhv`, `redhat.satellite`, and `redhat.insights`.

Selecting Sources of Ansible Content Collections

Regardless of whether you are using `ansible-navigator` with the minimal automation execution environment or `ansible-playbook` on bare metal Ansible Core, you always have at least one Ansible Content Collection available to you: `ansible.builtin`.

In addition, your automation execution environment might have additional collections built into it. For example, the default automation execution environment used by Red Hat Ansible Automation Platform 2.3, `ee-supported-rhel8`, contains 30 collections.

If you need additional Ansible Content Collections, you can add them to the `collections` subdirectory of your Ansible project. You might obtain Ansible Content Collections from several sources:

Automation Hub

Automation hub is a service provided by Red Hat to distribute Red Hat Certified Ansible Content Collections that are supported by Red Hat and ecosystem partners. As an end customer, you can open a support ticket with Red Hat for these Ansible Content Collections to be addressed by Red Hat and the ecosystem partner.

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 browse these collections.

Private automation hub

Your organization might have its own on-site private automation hub, and might also use that hub to distribute its own Ansible Content Collections. Private automation hub is included with Red Hat Ansible Automation Platform.

Ansible Galaxy

Ansible Galaxy is a community-supported website that hosts Ansible Content Collections that have been submitted by various Ansible developers and users. Ansible Galaxy is a public library that provides no formal support guarantees and is not curated by Red Hat. For example, the `community.network`, `community.asa`, and `community.ciscosmb` collections are all available from that platform.

Use the Ansible Galaxy web UI at `https://galaxy.ansible.com/` to search for collections.

Third-party Git repository or archive file

You can also download Ansible Content Collections from a Git repository or a local or remote `.tar` file, similar to how you can download roles.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Installing Ansible Content Collections

Before your playbooks can use content from an Ansible Content Collection, you must ensure that the collection is available in your automation execution environment.

The Ansible configuration `collections_paths` setting specifies a colon-separated list of paths on the system where Ansible looks for installed collections.

You can set this directive in the `ansible.cfg` configuration file.

Important

The following example shows the `collections_paths` setting in an `ansible.cfg` file.

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

If you remove the `/usr/share/ansible/collections` path from the `collections_paths` setting, then the `ansible-navigator` command cannot find the Ansible Content Collections provided inside the automation execution environment.

The following example uses the `ansible-galaxy collection install` command to download and install the `network.ospf` Ansible Content Collection. The `-p collections` option installs the collection in the local `collections` subdirectory.

```
[user@controlnode ~]$ ansible-galaxy collection install network.ospf \
-p collections
```

Important

You must specify the `-p collections` option or the `ansible-galaxy` command installs the collection based on your current `collections_paths` setting, or into your `~/.ansible/collections/` directory on the control node by default. The `ansible-navigator` command does not load the `~/.ansible/collections/` directory into the automation execution environment.

The `ansible-galaxy` command can also install a collection from a local or a remote `.tar` file, or a Git repository. A Git repository must have a valid `galaxy.yml` or `MANIFEST.json` file that provides metadata about the collection, such as its namespace and version number. For example, see the `ansible.netcommon` collection at `https://github.com/ansible-collections/ansible.netcommon`.

```
[user@controlnode ~]$ ansible-galaxy collection install \
/tmp/ansible-netcommon-5.1.0.tar.gz -p collections
...output omitted...
[user@controlnode ~]$ ansible-galaxy collection install \
http://www.example.com/network-bgp-2.0.0.tar.gz -p collections
...output omitted...
[user@controlnode ~]$ ansible-galaxy collection install \
git@git.example.com:organization/repo_name.git -p collections
```

Installing Ansible Content Collections with a Requirements File

If your Ansible project needs additional Ansible Content Collections, you can create a `collections/requirements.yml` file in the project directory that lists all the collections that the project requires. Automation controller detects this file and automatically installs the specified collections before running your playbooks.

A requirements file for Ansible Content Collections is a YAML file that consists of a `collections` dictionary key that has the list of collections to install as its value. Each list item can also specify the particular version of the collection to install, as shown in the following example:

```
---
collections: (1)
  - name: network.ospf (2)

  - name: network.base
    version: 2.0.0 (3)

  - name: /tmp/ansible-netcommon-5.1.0.tar.gz (4)

  - name: http://www.example.com/network-bgp-2.0.0.tar.gz (5)

  - name: git+https://github.com:redhat-cop/network.vpn.git (6)
    version: main
```

1. Specifies the list of collections that the Ansible project requires.
2. Installs the `network.ospf` Ansible Content Collection from the first available source. The next part of this section covers how to configure sources.
3. Installs version `2.0.0` of the `network.base` Ansible Content Collection from the first available source. It is a good practice to specify the version whenever you can.
4. Installs an Ansible Content Collection from the specified local `.tar.gz` file.
5. Installs an Ansible Content Collection from the `.tar.gz` file at the specified remote URL.
6. Installs the `network.vpn` Ansible Content Collection from a public Git repository, selecting the version in the `main` branch. The version directive can be a branch, tag, or commit hash. The `ansible-galaxy` command can then use the `collections/requirements.yml` file to install the required collections. Specify the requirements file with the `--requirements-file` (or the `-r`) option, and use the `-p collections` option to install the Ansible Content Collection into the `collections` subdirectory.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[user@controlnode ~]$ ansible-galaxy collection install \
-r collections/requirements.yml -p collections
```

Configuring Ansible Content Collection Sources

By default, the `ansible-galaxy` command uses Ansible Galaxy at `https://galaxy.ansible.com/` to download Ansible Content Collections. You might not want to use this source, preferring automation hub or your own private automation hub. Alternatively, you might want to try automation hub first, and then try Ansible Galaxy.

You can configure the sources that `ansible-galaxy` uses to get Ansible Content Collections in your Ansible project's `ansible.cfg` file. The relevant parts of that file might look like the following example:

```
...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. The list all the repositories that the `ansible-galaxy` command must use, in order. For each name you specify, add a `[galaxy_server.name]` section to this file that provides the connection parameters.

   This example configures the `ansible-galaxy` command to get Ansible Content Collections from automation hub first (`[galaxy_server.automation_hub]`). If a collection is not available from that source, then the `ansible-galaxy` command tries to get the collection from the Ansible Galaxy website (`[galaxy_server.galaxy]`).

2. The URL to access the repository.

3. The URL for authentication.

4. To access automation hub, you need an authentication token associated with your account. Use the automation hub web UI at `https://console.redhat.com/ansible/automation-hub/token/` to generate that token. For more details on that process, refer to the links in the References section.

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

However, you might not want to expose your credentials in the `ansible.cfg` file because the file could potentially get committed when using version control.

It is preferable to remove the authentication parameters from the `ansible.cfg` file and define them in environment variables, as shown in the following example:

```
export ANSIBLE_GALAXY_SERVER_<SERVER_ID>_<KEY>=value
```

`server_id`
: Server identifier in uppercase. The server identifier is the name you used in the `server_list` parameter and in the name of the `[galaxy_server.server_id]` section.

`key`
: Name of the parameter in uppercase.

The following example provides the `token` parameter as an environment variable:

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

Using Resources from Ansible Content Collections

After you install an Ansible Content Collection in your Ansible project, you can use it with that project's Ansible Playbooks.

You can use the `ansible-navigator collections -m interactive` command in your Ansible project directory to list all the collections that are installed in your automation execution environment. This includes the Ansible Content Collections in your project's `collections` subdirectory.

You can select the line number of the collection in the interactive mode of `ansible-navigator` to view its contents. Then, you can select the line number of a module, role, or other plug-in to see its documentation. You can also use other tools, such as `ansible-navigator doc`, with the fully qualified collection name of a module to view that module's documentation.

The following playbook invokes the `cli_config` module from the `ansible.netcommon` collection for a task.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Update the configuration on junos1.lab.example.com
  hosts: junos1.lab.example.com

  tasks:
    - name: Ensure the proper configuration is in place
      ansible.netcommon.cli_config:
        replace: /var/home/ansible/junos01.cfg
```

The following playbook uses the `run` role from the `network.ospf` collection.

```
- name: Perform health checks
  hosts: ios
  gather_facts: false

  tasks:
  - name: OSPF Manager
    ansible.builtin.include_role:
      name: network.ospf.run
    vars:
      actions:
        - name: health_check
          vars:
            details: true
            checks:
              - name: all_neighbors_up
              - name: all_neighbors_down
              - name: min_neighbors_up
                min_count: 1
              - name: ospf_status_summary
```

References

Ansible Automation Platform Certified Content
https://access.redhat.com/articles/3642632

Ansible Certified Content Frequently Asked Questions
https://access.redhat.com/articles/4916901

Using collections - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/collections_using.html

Galaxy User Guide - Ansible Documentation
https://docs.ansible.com/ansible/latest/galaxy/user_guide.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Installing and Using an Ansible Content Collection

Install an Ansible Content Collection and use a role or module from it in a playbook.

Outcomes

- Configure various collection sources on a system to install collections.
- Install collections from a private automation hub using a requirements file.
- Create a playbook that uses a role provided by a 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 also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start simplify-collections
```

Instructions

1. Browse the Ansible Content Collections available from the classroom's private automation hub. Access the details for the validated `network.ospf` collection.

   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 `network.ospf` collection is a validated collection, select **Validated** from the **Filter by repository** menu.

   1.3. Search for the collection by entering `ospf` in the search field. When the search completes, click the **ospf** tile.

   1.4. Notice that the UI displays an `ansible-galaxy collection install network.ospf` command that you can use to install the collection. Click the **Documentation** tab. The `Readme` file provides information on how to use the `run` role included in the collection. The playbooks in this exercise use examples from the `Readme` file.

2. Open the `/home/student/simplify-collections` directory in VS Code. Configure the Ansible project in that directory to use collections installed in the `collections` subdirectory.

   2.1. Open VS Code and click **File** > **Open Folder**.

   2.2. Navigate to **Home** > **simplify-collections** and click **Open**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note
If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

2.3. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/simplify-collections` directory in a GNOME terminal:

```
[student@workstation ~] cd ~/simplify-collections
```

Important
You need to run the `ansible-galaxy` commands in this exercise from the `/home/student/simplify-collections` 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.

2.4. Use the `mkdir` command to create the `collections` directory:

```
[student@workstation simplify-collections]$ mkdir collections
```

2.5. Create the `ansible.cfg` file with the following content:

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

This `collections_paths` line indicates that collections are found in the project `collections` directory and the `/usr/share/ansible/collections` directory (where collections exist within an automation execution environment).

3. Configure the Ansible project in the `/home/student/simplify-collections` directory so that the `ansible-galaxy` command retrieves collections from the classroom's private automation hub. Enable access to the `rh-certified` and `validated` repositories.

3.1. 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.

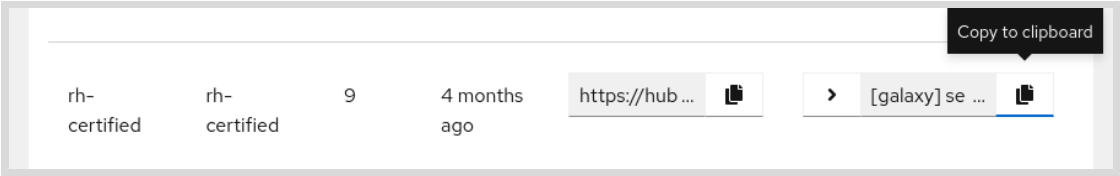


Figure 6.3: Copy settings for the Red Hat certified repository

3.2. Update the `ansible.cfg` file and add the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file contains the following content:

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

[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>
```

3.3. 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 **validated** row.

Do not close your web browser window.

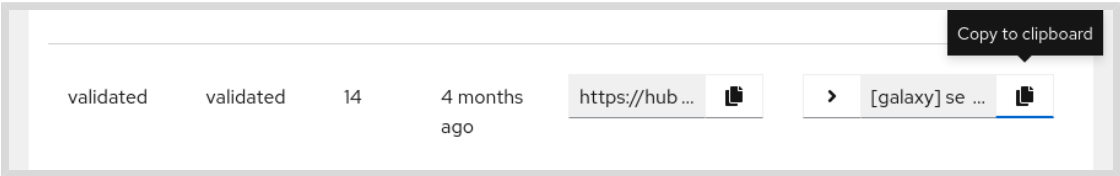


Figure 6.4: Copy settings for the validated repository

3.4. Update the `ansible.cfg` file to append the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file now contains the following content:

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

[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 = validated_repo

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

3.5. Update the `ansible.cfg` file so that it only contains one `[galaxy]` section. Update the first `[galaxy]` section so that it lists both the `rh-certified_repo` and the `validated_repo` repositories. Remove the second `[galaxy]` section. The updated `ansible.cfg` file now contains the following content:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

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

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

3.6. Return to the private automation hub web UI. Navigate to **Collections** > **API token management** and then click **Load token**. Copy the API token.

> **Important**
> Loading a new token invalidates any of your previous tokens.

3.7. Using the copied token, update both `token` lines in the `ansible.cfg` file. Your token is different from the token displayed in this example. Save and close the file when done:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

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

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

4. Create a `requirements.yml` file in the project `collections` directory and then install the required collections.

4.1. Create the `/home/student/simplify-collections/collections/requirements.yml` file. Add the following content to require the `1.0.0` version

of the `network.ospf` collection and the `2.0.0` version of the `network.base` collection:

```
---
collections:
  - name: network.ospf
    version: 1.0.0
  - name: network.base
    version: 2.0.0
```

4.2. In the `/home/student/simplify-collections` directory, use the `ansible-galaxy` command to install the required collections. Add the `-r` option to specify the location of the `collections/requirements.yml` file and add the `--no-deps` option so that the command does not install dependencies. The dependent collections are already available within the `ee-supported-rhel8` automation execution environment.

```
[student@workstation simplify-collections]$ ansible-galaxy collection install \
-r collections/requirements.yml --no-deps
...output omitted...
network.ospf:1.0.0 was installed successfully
...output omitted...
network.base:2.0.0 was installed successfully
```

4.3. List the installed collections:

```
[student@workstation simplify-collections]$ ansible-galaxy collection list

# /usr/share/ansible/collections/ansible_collections
Collection                Version
------------------------- -------
redhat.rhel_system_roles 1.20.1

# /home/student/simplify-collections/collections/ansible_collections
Collection        Version
----------------- -------
network.base      2.0.0
network.ospf      1.0.0
```

Important

The output in the `/usr/share/ansible/collections/ansible_collections` section displays collections installed on the local system, which is different from collections installed in an automation execution environment.

To see collections available to automation content navigator from the project directory, run the following command:

```
[student@workstation simplify-collections]$ ansible-navigator collections \
-m interactive
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

5. View the provided `/home/student/simplify-collections/ospf_deploy.yml` playbook and then run the playbook to verify that it runs without errors.

5.1. View the `ospf_deploy.yml` playbook. This playbook configures Open Shortest Path First (OSPF) on the managed network nodes in the `ios` inventory group. The `network.ospf.run` role uses inventory variables defined in the `inventory/hosts_vars` directory.

```
---
- name: Deploy OSPF Configuration
  hosts: ios
  gather_facts: false
  tasks:
  - name: OSPF Manager (deploy)
    ansible.builtin.include_role:
      name: network.ospf.run
    vars:
      actions:
        - name: deploy
      inventory_directory: ./
```

5.2. Run the `ospf_deploy.yml` playbook. The playbook succeeds and the play summary indicates a change to both the `iosxe1.lab.example.com` and `iosxe2.lab.example.com` managed nodes.

```
[student@workstation simplify-collections]$ ansible-navigator run ospf_deploy.yml
...output omitted...
PLAY RECAP *********************************************************************
iosxe1.lab.example.com     : ok=19   changed=1    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=19   changed=1    unreachable=0    failed=0   ...
```

5.3. View the `ospf_gather.yml` playbook. This playbook uses the `network.ospf.run` role to display the OSPF configuration for managed network nodes in the `ios` inventory group.

```
---
- name: Gather OSPF Configuration
  hosts: ios
  gather_facts: false
  tasks:
  - name: OSPF Manager (gather)
    ansible.builtin.include_role:
      name: network.ospf.run
    vars:
      actions:
        - name: gather
      inventory_directory: ./
```

5.4. Run the `ospf_gather.yml` playbook to display the OSPF configuration.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

It might take a minute or two for the OSPF configuration changes to start and converge. If you do not see the expected output, then wait for a minute or two and try again.

```
[student@workstation simplify-collections]$ ansible-navigator run \
ospf_gather.yml
...output omitted...
TASK [network.base.resource_manager : Resource Facts] *************************
ok: [iosxe1.lab.example.com] => {
    "msg": {
        "ansible_connection": "ansible.netcommon.network_cli",
        "ansible_network_os": "cisco.ios.ios",
        "changed": false,
        "failed": false,
        "gathered": {
            "processes": [
                {
                    "network": [
                        {
                            "address": "172.25.250.0",
                            "area": "0",
                            "wildcard_bits": "0.0.0.255"
                        }
                    ],
                    "process_id": 110,
                    "router_id": "1.1.1.1"
                }
            ]
        },
        "resource_module_name": "cisco.ios.ios_ospfv2"
    }
}
ok: [iosxe2.lab.example.com] => {
    "msg": {
        "ansible_connection": "ansible.netcommon.network_cli",
        "ansible_network_os": "cisco.ios.ios",
        "changed": false,
        "failed": false,
        "gathered": {
            "processes": [
                {
                    "network": [
                        {
                            "address": "172.25.250.0",
                            "area": "0",
                            "wildcard_bits": "0.0.0.255"
                        }
                    ],
                    "process_id": 110,
                    "router_id": "2.2.2.2"
```

```
                }
            ]
        },
        "resource_module_name": "cisco.ios.ios_ospfv2"
    }
}
...output omitted...
```

6. Close the `/home/student/simplify-collections` directory in VS Code, or run the `cd` command in a GNOME terminal to return to the `student` home directory:

```
[student@workstation simplify-collections]$ cd
```

Finish

On the `workstation` machine, 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 simplify-collections
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Simplifying Playbooks with Roles and Ansible Content Collections

Create Ansible Roles that use variables and tasks.

Outcomes

- Create a playbook that includes tasks from other files and imports another playbook.
- Configure a project to use various collection sources and install collections.
- Create a playbook that uses a role provided by a 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 installs the required software, downloads an automation execution environment, and creates the required resources for the exercise.

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

Instructions

1. Create a playbook called `monitoring.yml` in the `/home/student/simplify-review` directory.
   - The playbook must include the tasks in the `tasks/juniper_netconf_enable.yml` file to enable NETCONF only on managed nodes in the `junos` inventory group.
   - The playbook must include the tasks in the `tasks/juniper_snmp.yml` file to configure the Simple Network Management Protocol (SNMP) on Juniper devices. These tasks must target managed nodes in the `junos` inventory group.
   - The playbook must also include the tasks in the `cisco_snmp.yml` file to configure SNMP on Cisco devices. These tasks must target managed nodes in the `ios` inventory group.
   - Likewise, the playbook must also include the tasks in the `arista_snmp.yml` file to configure SNMP on Arista devices. These tasks must target managed nodes in the `eos` inventory group.
   - Finally the playbook must import the `syslog.yml` playbook.
2. Run the `monitoring.yml` playbook, and then run the `verify_monitoring.yml` playbook to verify that the configuration set by the `monitoring.yml` playbook was successful.
3. Configure the Ansible project in the `/home/student/simplify-review` directory so that the project searches for collections in the project `collections` directory and the `/usr/share/ansible/collections` directory. Configure the Ansible project so that the

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

`ansible-galaxy` command retrieves collections from the classroom's private automation hub. Enable access to the `rh-certified` and `validated` repositories.

You can access the classroom's private automation hub at `https://hub.lab.example.com`. Log in as `student` using `redhat123` as the password.

4. Create a `requirements.yml` file in the project `collections` directory to install the `2.0.0` version of the `network.base` collection. When done, install the required collection. Add the `-n` (or `--no-deps`) option to the `ansible-galaxy` command so that the command does not install dependencies specified by the `network.base` collection.

> **Important**
> In the lab environment, if you do not add the `-n` (or `--no-deps`) option to the `ansible-galaxy` command, then the command installs versions of the `ansible.netcommon` and `ansible.utils` collections that do not function correctly with the `ansible.base.resource_manager` role.

5. Create a playbook called `resource_modules.yml` in the `/home/student/simplify-review` directory. The playbook must target managed nodes in the `ios` inventory group. The playbook must have one task that includes the `resource_manager` role from the `network.base` collection. Define the `action` variable for the task and assign `list` as the value of the variable.

> **Note**
> The classroom private automation hub contains documentation for the `network.base` collection. You might use this documentation as a resource for creating the playbook.

6. Run the `resource_modules.yml` playbook to display resource modules for the managed nodes. The playbook runs successfully if you see output for each managed node that includes module resources, such as:

```
...output omitted...
        "modules": [
            "acl_interfaces",
            "acls",
            "bgp_address_family",
            "bgp_global",
...output omitted...
```

7. Close the `/home/student/simplify-review` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

   7.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/simplify-review` directory.

   7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation simplify-review]$ cd
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 simplify-review
```

Finish

As the `student` user on the `workstation` machine, 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 simplify-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Simplifying Playbooks with Roles and Ansible Content Collections

Create Ansible Roles that use variables and tasks.

Outcomes

- Create a playbook that includes tasks from other files and imports another playbook.
- Configure a project to use various collection sources and install collections.
- Create a playbook that uses a role provided by a 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 installs the required software, downloads an automation execution environment, and creates the required resources for the exercise.

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

Instructions

1. Create a playbook called `monitoring.yml` in the `/home/student/simplify-review` directory.

   - The playbook must include the tasks in the `tasks/juniper_netconf_enable.yml` file to enable NETCONF only on managed nodes in the `junos` inventory group.
   - The playbook must include the tasks in the `tasks/juniper_snmp.yml` file to configure the Simple Network Management Protocol (SNMP) on Juniper devices. These tasks must target managed nodes in the `junos` inventory group.
   - The playbook must also include the tasks in the `cisco_snmp.yml` file to configure SNMP on Cisco devices. These tasks must target managed nodes in the `ios` inventory group.
   - Likewise, the playbook must also include the tasks in the `arista_snmp.yml` file to configure SNMP on Arista devices. These tasks must target managed nodes in the `eos` inventory group.
   - Finally the playbook must import the `syslog.yml` playbook.

   1.1. Open VS Code and click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **simplify-review** and click **Open**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note
If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

1.3. Create a playbook in the `/home/student/simplify-review` directory called `monitoring.yml` that targets managed nodes in the `junos`, `ios`, and `eos` inventory groups. Add the following content to the playbook:

```
---
- name: Configure monitoring on managed network nodes
  hosts:
    - junos
    - ios
    - eos
  gather_facts: false
  tasks:
```

1.4. Add a block that runs tasks on managed nodes in the `junos` inventory group. In the Junos block, include the tasks from the `tasks/juniper_netconf_enable.yml` and `tasks/juniper_snmp.yml` files. Add the following content to the playbook:

```
...output omitted...
    - name: Juniper Junos block
      when: inventory_hostname in groups['junos']
      block:
        - name: Enable NETCONF
          ansible.builtin.include_tasks: tasks/juniper_netconf_enable.yml

        - name: Configure Juniper SNMP settings
          ansible.builtin.include_tasks: tasks/juniper_snmp.yml
```

1.5. Add a block that runs tasks on managed nodes in the `ios` inventory group. In the Cisco block, include the tasks from the `tasks/cisco_snmp.yml` file. Add the following content to the playbook:

```
...output omitted...
    - name: Cisco IOS block
      when: inventory_hostname in groups['ios']
      block:
        - name: Configure Cisco SNMP settings
          ansible.builtin.include_tasks: tasks/cisco_snmp.yml
```

1.6. Add a block that runs tasks on managed nodes in the `eos` inventory group. In the Arista block, include the tasks from the `tasks/arista_snmp.yml` file. Add the following content to the playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
...output omitted...
    - name: Arista EOS block
      when: inventory_hostname in groups['eos']
      block:
        - name: Configure Arista SNMP settings
          ansible.builtin.include_tasks: tasks/arista_snmp.yml
```

1.7. Add a play to import the `syslog.yml` playbook:

```
...output omitted...
- name: Configure syslog
  ansible.builtin.import_playbook: syslog.yml
```

1.8. The final playbook should consist of the following content:

```
---
- name: Configure monitoring on managed network nodes
  hosts:
    - junos
    - ios
    - eos
  gather_facts: false
  tasks:
    - name: Juniper Junos block
      when: inventory_hostname in groups['junos']
      block:
        - name: Enable NETCONF
          ansible.builtin.include_tasks: tasks/juniper_netconf_enable.yml

        - name: Configure Juniper SNMP settings
          ansible.builtin.include_tasks: tasks/juniper_snmp.yml

    - name: Cisco IOS block
      when: inventory_hostname in groups['ios']
      block:
        - name: Configure Cisco SNMP settings
          ansible.builtin.include_tasks: tasks/cisco_snmp.yml

    - name: Arista EOS block
      when: inventory_hostname in groups['eos']
      block:
        - name: Configure Arista SNMP settings
          ansible.builtin.include_tasks: tasks/arista_snmp.yml

- name: Configure syslog
  ansible.builtin.import_playbook: syslog.yml
```

1.9. *(Optional)* The previous playbook demonstrates one possible solution. Instead of using `when` conditions, you might create separate plays for each inventory group. That playbook might consist of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Configure monitoring on Juniper managed network nodes
  hosts: junos
  gather_facts: false
  tasks:
    - name: Enable NETCONF
      ansible.builtin.include_tasks: tasks/juniper_netconf_enable.yml

    - name: Configure Juniper SNMP settings
      ansible.builtin.include_tasks: tasks/juniper_snmp.yml

- name: Configure monitoring on Cisco managed network nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Configure SNMP settings
      ansible.builtin.include_tasks: tasks/cisco_snmp.yml

- name: Configure monitoring on Arisa managed network nodes
  hosts: eos
  gather_facts: false
  tasks:
    - name: Configure Arista SNMP settings
      ansible.builtin.include_tasks: tasks/arista_snmp.yml

- name: Configure syslog
  ansible.builtin.import_playbook: syslog.yml
```

2. Run the `monitoring.yml` playbook, and then run the `verify_monitoring.yml` playbook to verify that the configuration set by the `monitoring.yml` playbook was successful.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/simplify-review` directory in a GNOME terminal:

```
[student@workstation ~] cd ~/simplify-review
```

2.2. Run the `monitoring.yml` playbook to configure SNMP on all managed nodes and to configure logging on managed nodes in the `ios` inventory group:

```
[student@workstation simplify-review]$ ansible-navigator run monitoring.yml
...output omitted...
PLAY RECAP *********************************************************************
arista1.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0 ...
arista2.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0 ...
iosxe1.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0 ...
iosxe2.lab.example.com     : ok=3    changed=2    unreachable=0    failed=0 ...
junos1.lab.example.com     : ok=4    changed=2    unreachable=0    failed=0 ...
junos2.lab.example.com     : ok=4    changed=2    unreachable=0    failed=0 ...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.3. Run the `verify_monitoring.yml` playbook. The playbook displays a message for each managed node to indicate if SNMP is configured correctly, and for each managed node in the `ios` inventory group to indicate if logging is configured correctly.

```
[student@workstation simplify-review]$ ansible-navigator run verify_monitoring.yml

PLAY [Verify Junos SNMP was configured properly] ******************************
...output omitted...
TASK [Verify the correct contact info was configured] *************************
ok: [junos1.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'junos1.lab.example.com'."
}
ok: [junos2.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'junos2.lab.example.com'."
}

PLAY [Verify IOS SNMP was configured properly] *******************************
...output omitted...
TASK [Verify the correct contact info was configured] *************************
ok: [iosxe2.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'iosxe2.lab.example.com'."
}
ok: [iosxe1.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'iosxe1.lab.example.com'."
}

PLAY [Verify EOS SNMP was configured properly] ******************************
...output omitted...
TASK [Verify the correct contact info was configured] *************************
ok: [arista1.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'arista1.lab.example.com'."
}
ok: [arista2.lab.example.com] => {
    "changed": false,
    "msg": "SNMP is configured properly on 'arista2.lab.example.com'."
}

PLAY [Verify IOS syslog was configured properly] ******************************
...output omitted...
TASK [Verify the correct syslog server logging level was configured] **********
ok: [iosxe2.lab.example.com] => {
    "changed": false,
    "msg": "Logging is configured properly on 'iosxe2.lab.example.com'."
}
ok: [iosxe1.lab.example.com] => {
    "changed": false,
    "msg": "Logging is configured properly on 'iosxe1.lab.example.com'."
}
...output omitted...
```

```
PLAY RECAP *********************************************************************
arista1.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0   ...
arista2.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0   ...
iosxe1.lab.example.com     : ok=5    changed=0    unreachable=0    failed=0   ...
iosxe2.lab.example.com     : ok=5    changed=0    unreachable=0    failed=0   ...
junos1.lab.example.com     : ok=3    changed=0    unreachable=0    failed=0   ...
junos2.lab.example.com     : ok=3    changed=0    unreachable=0    failed=0   ...
localhost                  : ok=2    changed=0    unreachable=0    failed=0   ...
```

3. Configure the Ansible project in the `/home/student/simplify-review` directory so that the project searches for collections in the project `collections` directory and the `/usr/share/ansible/collections` directory. Configure the Ansible project so that the `ansible-galaxy` command retrieves collections from the classroom's private automation hub. Enable access to the `rh-certified` and `validated` repositories.

You can access the classroom's private automation hub at `https://hub.lab.example.com`. Log in as `student` using `redhat123` as the password.

3.1. Create the `/home/student/simplify-review/ansible.cfg` file and set the `collections_paths` variable:

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

3.2. 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.

3.3. 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.

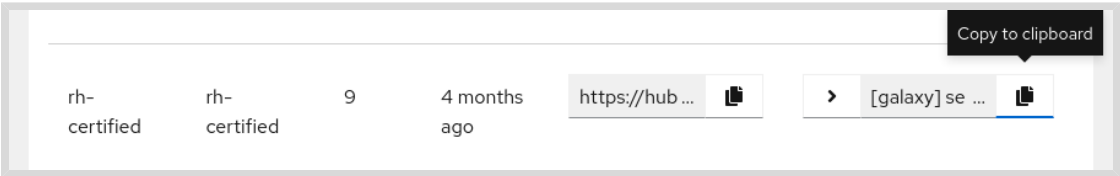


Figure 6.5: Copying settings for the Red Hat certified repository

3.4. Update the `ansible.cfg` file and add the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file contains the following content:

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

[galaxy]
server_list = rh-certified_repo
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

3.5. 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 **validated** row. Do not close your web browser window.

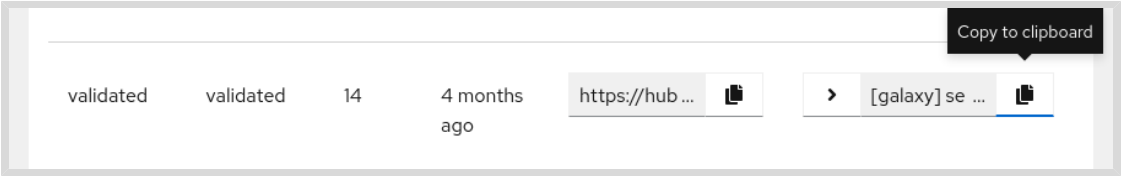


Figure 6.6: Copying settings for the validated repository

3.6. Update the `ansible.cfg` file to append the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file now contains the following content:

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

[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 = validated_repo

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

3.7. Update the `ansible.cfg` file so that it only contains one `[galaxy]` section. Update the first `[galaxy]` section so that it lists both the `rh-certified_repo` and the `validated_repo` repositories. Remove the second `[galaxy]` section. The updated `ansible.cfg` file now contains the following content:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

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

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

3.8. Return to the private automation hub web UI. Navigate to **Collections** > **API token management** and then click **Load token**. Copy the API token.

> **Important**
> Loading a new token invalidates any of your previous tokens.

3.9. Using the copied token, update both token lines in the `ansible.cfg` file. Your token is different from the token displayed in this example. Save and close the file when done:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=19abc11a37723c1d1ace762b9d00221d2b15dad0

[galaxy_server.validated_repo]
url=https://hub.lab.example.com/api/galaxy/content/validated/
token=19abc11a37723c1d1ace762b9d00221d2b15dad0
```

4. Create a `requirements.yml` file in the project `collections` directory to install the `2.0.0` version of the `network.base` collection. When done, install the required collection. Add the `-n` (or `--no-deps`) option to the `ansible-galaxy` command so that the command does not install dependencies specified by the `network.base` collection.

> **Important**
> In the lab environment, if you do not add the `-n` (or `--no-deps`) option to the `ansible-galaxy` command, then the command installs versions of the `ansible.netcommon` and `ansible.utils` collections that do not function correctly with the `ansible.base.resource_manager` role.

4.1. Create the `/home/student/simplify-review/collections/requirements.yml` file. Add the following content to require the `2.0.0` version of the `network.base` collection:

```
---
collections:
  - name: network.base
    version: 2.0.0
```

4.2. In the `/home/student/simplify-review` directory, use the `ansible-galaxy` command to install the required collections. Add the `-r` option to specify the location of the `collections/requirements.yml` file and the `--no-deps` option to prevent installing dependencies.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation simplify-review]$ ansible-galaxy collection install \
-r collections/requirements.yml --no-deps
...output omitted...
network.base:2.0.0 was installed successfully
```

4.3. List the installed collections:

```
[student@workstation simplify-review]$ ansible-galaxy collection list

# /home/student/simplify-review/collections/ansible_collections
Collection        Version
----------------- -------
network.base      2.0.0

# /usr/share/ansible/collections/ansible_collections
Collection                Version
------------------------- -------
redhat.rhel_system_roles 1.20.1
```

Important

The output in the `/usr/share/ansible/collections/ansible_collections` section displays collections installed on the local system, which is different from collections installed in an automation execution environment.

To see collections available to automation content navigator from the project directory, run the following command:

```
[student@workstation simplify-review]$ ansible-navigator collections \
-m interactive
```

5. Create a playbook called `resource_modules.yml` in the `/home/student/simplify-review` directory. The playbook must target managed nodes in the `ios` inventory group. The playbook must have one task that includes the `resource_manager` role from the `network.base` collection. Define the `action` variable for the task and assign `list` as the value of the variable.

Note

The classroom private automation hub contains documentation for the `network.base` collection. You might use this documentation as a resource for creating the playbook.

5.1. Create a playbook in the `/home/student/simplify-review` directory called `resource_modules.yml` that targets managed nodes in the `ios` inventory group. Add the following contents to the playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: List supported resource modules
  hosts: ios
  gather_facts: false
  tasks:
```

5.2. Add a task to the playbook that includes the `resource_manager` role from the `network.base` collection:

```
...output omitted...
    - name: Network Resource Manager
      ansible.builtin.include_role:
        name: network.base.resource_manager
```

5.3. Define the variable for the role in the task:

```
...output omitted...
    - name: Network Resource Manager
      ansible.builtin.include_role:
        name: network.base.resource_manager
      vars:
        action: list
```

5.4. The final playbook should consist of the following content:

```
---
- name: List supported resource modules
  hosts: ios
  gather_facts: false
  tasks:
    - name: Network Resource Manager
      ansible.builtin.include_role:
        name: network.base.resource_manager
      vars:
        action: list
```

6. Run the `resource_modules.yml` playbook to display resource modules for the managed nodes. The playbook runs successfully if you see output for each managed node that includes module resources, such as:

```
...output omitted...
        "modules": [
            "acl_interfaces",
            "acls",
            "bgp_address_family",
            "bgp_global",
...output omitted...
```

6.1. Run the `resource_modules.yml` playbook. Each managed node should display the following module resources:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation simplify-review]$ ansible-navigator run resource_modules.yml
...output omitted...
        "modules": [
            "acl_interfaces",
            "acls",
            "bgp_address_family",
            "bgp_global",
            "hostname",
            "interfaces",
            "l2_interfaces",
            "l3_interfaces",
            "lacp",
            "lacp_interfaces",
            "lag_interfaces",
            "lldp_global",
            "lldp_interfaces",
            "logging_global",
            "ntp_global",
            "ospf_interfaces",
            "ospfv2",
            "ospfv3",
            "prefix_lists",
            "route_maps",
            "snmp_server",
            "static_routes",
            "vlans"
        ]
...output omitted...
```

7. Close the `/home/student/simplify-review` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

   7.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/simplify-review` directory.

   7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation simplify-review]$ cd
```

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 simplify-review
```

Finish

As the `student` user on the `workstation` machine, 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 simplify-review
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- You can use the `ansible.builtin.import_playbook` module to incorporate external play files into playbooks.
- You can use the `ansible.builtin.include_tasks` or `ansible.builtin.import_tasks` modules to incorporate external task files into playbooks.
- You can use the `ansible.builtin.include_role` or `ansible.builtin.import_role` modules to incorporate roles into playbooks.
- When you *include* content, Ansible processes it dynamically as content is reached. When you *import* content, Ansible processes it before the run starts.
- *Ansible Content Collections* provide a set of related modules, roles, and other plug-ins that you can use in your playbooks.
- Ansible Roles and Ansible Content Collections help you to reuse and share Ansible code.
- You can use the `ansible-galaxy` command to find, download, and manage Ansible Roles and Ansible Content Collections.
- Red Hat provides Red Hat Certified Ansible Content Collections through a cloud based service, *automation hub*. Red Hat and its ecosystem partners support these Ansible Content Collections.
- You can use an on-premise service, *private automation hub*, to distribute Red Hat Certified Ansible Content Collections or your own Ansible Content Collections.
- You can create a `requirements.yml` file in the project directory that lists all the roles and collections that your Ansible project requires. Automation controller detects this file and automatically installs the specified roles or collections before running your playbooks.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 7

Automating Network Administration Tasks

Goal

Automate common network administration tasks, discussing recommended practices and approaches to cross-vendor automation.

Objectives

- Configure connectivity to managed network devices, gather device information, and generate a dynamic report that documents the state of your network infrastructure.
- Generate new device configurations using Jinja2 templates, and use templates to manage existing devices.
- Simplify tasks by using vendor-independent network resource modules and the `ansible.netcommon` Ansible Content Collection.
- Perform rolling network configuration updates, and automatically recover from errors.

Sections

- Gaining Infrastructure Awareness (and Guided Exercise)
- Generating Configuration Settings from Jinja2 Templates (and Guided Exercise)
- Simplifying Tasks with Platform-independent Modules (and Guided Exercise)
- Performing Rolling Configuration Updates (and Guided Exercise)

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Gaining Infrastructure Awareness

Objectives

- Configure connectivity to managed network devices, gather device information, and generate a dynamic report that documents the state of your network infrastructure.

Collect Information about Your Current Environment

You can collect information about your current environment in multiple ways. Depending on your goal, you might use one or more of the following methods:

- Collect facts about your managed nodes using vendor-specific `*_facts` modules, such as the `cisco.ios.ios_facts`, `junipernetworks.junos.junos_facts`, and `arista.eos.eos_facts` modules.
- Create backup files of the configuration of your managed nodes using vendor-specific `*_config` modules, such as the `cisco.ios.ios_config`, `junipernetworks.junos.junos_config`, and `arista.eos.eos_config` modules.
- Save configuration information about your managed nodes using the `network.base.resource_manager` role.

Collecting and Using Facts to Gain Infrastructure Awareness

You might use the `*_facts` modules in plays where you need to access current configuration settings for your managed nodes, but where you do not necessarily need to save this information. To collect a full set of facts, disable fact gathering at the play level with `gather_facts: false`, and then add a task such as the following:

```
- name: Collect facts
  cisco.ios.ios_facts:
    gather_subset:
      - all
```

If a task gathered configuration facts, then you might use the following task to display those facts:

```
- name: Display ansible_facts['net_config']
  ansible.builtin.debug:
    var: ansible_facts['net_config']
```

You might use filters to process the `ansible_facts['net_config']` variable, such as the `regex_findall` filter to search for a pattern, or the `split` filter to convert the variable into a list where each line is a list item:

```
- name: Display SNMP configuration
  ansible.builtin.debug:
    var: ansible_facts['net_config'] | regex_findall('^snmp.*$', multiline=true)

- name: Display configuration as a list
```

```
    vars:
      separator: "\n"
    ansible.builtin.debug:
      var: ansible_facts['net_config'] | split(separator)
```

To make it easier to use a variable in another playbook task, you might use the `ansible.builtin.set_fact` module. The following example uses that module to set the `snmp_config_lines` variable. Another task could then use the `snmp_config_lines` variable. In this example, the `ansible.builtin.debug` module displays the `snmp_config_lines` variable if the variable contains at least one list item:

```
- name: Set snmp_config_lines
  ansible.builtin.set_fact:
    snmp_config_lines: >-
      {{ ansible_facts['net_config'] |
      regex_findall('^snmp.*$', multiline=true) }}

- name: Display snmp_config_lines
  when: snmp_config_lines | length > 0
  ansible.builtin.debug:
    var: snmp_config_lines
```

Creating and Using Backup Files to Gain Infrastructure Awareness

Usually, the content of a backup file matches the content in the `ansible_facts['net_config']` variable. When that is true, you might find it easier to use gathered configuration facts rather than creating and processing backup files.

For Juniper Junos managed nodes, the content of a backup file does not match the content in the `ansible_facts['net_config']` variable, and the content of that variable can be difficult to process. The `junipernetworks.junos.junos_config` module provides the `backup_format` option, which is not found in other `*_config` modules. Changing the format of backup files to something different from the default `set` format might help with processing the backup files. If you create backup files using the JSON format, then you can process those backup files using the `from_json` filter.

The following example creates a backup file in JSON format for each Juniper Junos managed node:

```
- name: Create a backup file
  junipernetworks.junos.junos_config:
    backup: true
    backup_options:
      filename: "{{ inventory_hostname }}.json"
      backup_format: json
```

If you decide to process the content of backup files, then you might use the `ansible.builtin.slurp` module. This module works for both local and remote files.

Using the `ansible.builtin.slurp` module typically involves two steps:

1. Use the `ansible.builtin.slurp` module to read file content. Add the `register` keyword to the task so that the task captures the file content.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2. Decode the captured content so that you can use the content in another task. You might use the `ansible.builtin.debug` module to display the output of applied filters and then use the `ansible.builtin.set_fact` module after verifying that the applied filters have the desired output.

```
- name: Slurp backup file
  ansible.builtin.slurp:
    src: /backups/junos1.lab.example.com.json (1)
  register: slurped_file

- name: Display decoded slurped_file['content']
  ansible.builtin.debug:
    var: slurped_file['content'] | b64decode | from_json (2)

- name: Set json_content
  ansible.builtin.set_fact: (3)
    json_content: >
      {{ slurped_file['content'] | b64decode | from_json }}

- name: Show json_content['configuration']['interfaces']
  ansible.builtin.debug:
    var: >-
      json_content['configuration']['interfaces'] (4)
```

1. This line specifies the path to a backup file.
2. This line uses the `content` key in the slurped file is Base64 encoded. Use the `b64decode` filter to decode the value of the `content` key. Skip using the `from_json` filter if the file is not a JSON file. For most backup files, this content matches the content of the `ansible_facts['net_config']` variable.
3. This task uses the `ansible.builtin.set_fact` module to make it easier for additional tasks to work with the data.
4. This line displays configured interfaces and is specific to Juniper Junos managed nodes. By using a backup file in JSON format, you can display configuration information, such as the interfaces on the managed node. Additionally, you might use the `json_content['configuration']['routing-options']` variable to display routing options or the `json_content['configuration']['snmp']` variable to display the Simple Network Management Protocol (SNMP) configuration.

For backup files that do not use JSON format, you might use filters such as the `regex_findall`, `regex_search`, or `split` filters for additional data processing.

Saving Configuration Settings Using the Resource Manager Role

You can use the `resource_manager` role from the `network.base` collection to generate YAML files that contain information about the infrastructure of your managed nodes. The following example defines the `action` variable with the `persist` value. This action causes the role to save infrastructure information about your managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
- name: Network Resource Manager
  ansible.builtin.include_role:
    name: network.base.resource_manager
  vars:
    action: persist
    inventory_directory: ./
```

Important

The `persist` action expects to find an inventory file named `inventory.yaml` in the directory defined by the `inventory_directory` variable. By default, the `inventory_directory` variable has the `./inventory` value.

If the inventory file in your Ansible project uses the INI format, then you can redirect the output of the `ansible-navigator inventory` command to create a YAML version of the inventory file:

```
[user@host project]$ ansible-navigator inventory -i inventory --list \
--yaml > inventory.yaml
```

The previous command generates a file with content similar to the following example:

```
all:
  children:
    ios:
      hosts:
        iosxe1.lab.example.com:
          ansible_become: true
          ansible_become_method: enable
          ansible_connection: ansible.netcommon.network_cli
          ansible_network_os: cisco.ios.ios
          ansible_ssh_private_key_file: ~/.ssh/lab_rsa
          ansible_user: student
        iosxe2.lab.example.com:
          ansible_become: true
          ansible_become_method: enable
          ansible_connection: ansible.netcommon.network_cli
          ansible_network_os: cisco.ios.ios
          ansible_ssh_private_key_file: ~/.ssh/lab_rsa
          ansible_user: student
    ungrouped: {}
```

If your Ansible project already defines host variables in the `host_vars` directory or group variables in the `group_vars` directory, then you can delete the variable information from the `inventory.yaml` file. After removing the variables, the previous example consists of the following content:

```
all:
  children:
    ios:
      hosts:
        iosxe1.lab.example.com:
        iosxe2.lab.example.com:
    ungrouped: {}
```

After creating an `inventory.yaml` file, update your `ansible-navigator.yml` file or your `ansible.cfg` file to point to the new inventory file. You can then delete the previous inventory file.

When you use the `persist` action of the `network.base.resource_manager` role, the play creates the `host_vars` directory in the directory specified by the `inventory_directory` variable. In the `host_vars` directory, the play creates a directory for each managed node and then creates YAML files for the collected resources. For the `iosxe1.lab.example.com` managed node, you might see the following files:

```
[user@host project]$ tree host_vars/iosxe1.lab.example.com/
host_vars/iosxe1.lab.example.com/
├── acls.yaml
├── hostname.yaml
├── interfaces.yaml
├── l2_interfaces.yaml
├── l3_interfaces.yaml
├── lacp.yaml
├── lacp_interfaces.yaml
├── ospf_interfaces.yaml
├── static_routes.yaml
└── vlans.yaml
```

After you create these files, you can use the content in these files as regular host variables. Based on the content of the `l3_interfaces.yaml` file, you might use the following task to display the IPv4 addresses for the `GigabitEthernet1` interface:

```
- name: Show IPv4 addresses for the GigabitEthernet1 interface
  ansible.builtin.debug:
    var: >-
      hostvars[inventory_hostname]['l3_interfaces'] |
      selectattr('name', '==', 'GigabitEthernet1') |
      map(attribute='ipv4') |
      flatten |
      map(attribute='address')
```

This task might result in the following output:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
TASK [Show the IPv4 address for the GigabitEthernet1 interface] ****************
ok: [iosxe1.lab.example.com] => {
    "hostvars[inventory_hostname]['l3_interfaces'] | selectattr('name',
 '==', 'GigabitEthernet1') | map(attribute='ipv4') | flatten |
 map(attribute='address')": [
        "172.25.250.20/24"
    ]
}
```

As with the other methods for gaining infrastructure awareness, you might use the `ansible.builtin.set_fact` module to make it easier for additional tasks to work with specific pieces of data.

Generating Dynamic Reports

The preceding section discussed methods for collecting information about your current environment and how you might extract pieces of information for further use. Now that you can collect and extract information about your infrastructure, you can generate reports.

You might integrate the extracted infrastructure information into existing reports, or you might create new reports. You can decide on what type of report to create and what information should be included in the report. Although you might create dynamic reports that use simple text files, you might also create interactive reports that you publish to a web server.

> **Note**
> Creating a dynamic report for a web server requires additional knowledge related to items such as HTML, XML, CSS, and graphic design.

The following screen capture shows an interactive web page with Layer 1, Layer 2, and Layer 3 information for a couple of network devices. Users can click various web page elements to either reveal or hide information about each network device. This is just one example and you might choose to do something completely different.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

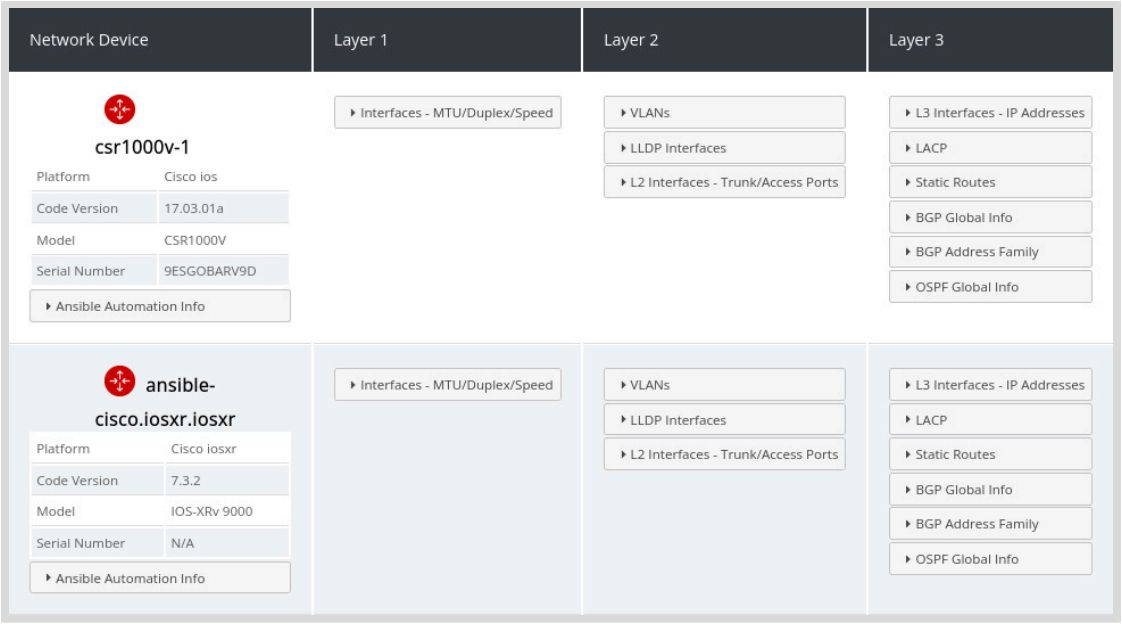


Figure 7.1: Example web report

Creating a dynamic web report might require creating multiple content collections or roles so that the report displays the desired information in the desired format. The `network.toolkit` collection available from the `https://github.com/network-automation/toolkit` Git repository provides a good starting point.

References

Network Base Validated Content
https://console.redhat.com/ansible/automation-hub/repo/validated/network/base/docs/?version=2.0.0

Red Hat Ansible Automation Platform - Network Tool Kit
https://github.com/network-automation/toolkit

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Gaining Infrastructure Awareness

Gather device information and generate a dynamic documentation report.

Outcomes

- Create a playbook that gathers data from Cisco IOS and Arista EOS managed nodes.
- Use an Ansible Role in a Git repository that creates a dynamic web report to display the data collected by the playbook.
- Configure the managed nodes and verify that the information is updated in the dynamic web report.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start network-infra
```

Instructions

1. Open the `/home/student/network-infra` directory in VS Code and review the `dynamic_report.yml` playbook.

   1.1. Open VS Code and click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **network-infra** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Click the `dynamic_report.yml` playbook. The playbook consists of one play named `Play to build a report` that uses a role named `report`. The `start` script configured the `workstation` machine as a web server. Because the report is built on this local web server, the play runs on the `workstation` machine.

```
---
- name: Play to build a report
  hosts: workstation
  gather_facts: false
  tasks:
    - name: Build the report
      ansible.builtin.include_role:
        name: report
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2. The `report` role used in the `dynamic_report.yml` playbook is in the `git@git.lab.example.com:student/report` Git repository. Create a role requirements file in your project directory and then use the `ansible-galaxy` command to download and install the `report` role.

2.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/network-infra` directory in a GNOME terminal:

```
[student@workstation ~]$ cd ~/network-infra
[student@workstation network-infra]$
```

2.2. Use the `mkdir` command to create the `roles` directory in your project:

```
[student@workstation network-infra]$ mkdir roles
```

2.3. In VS Code, create the `roles/requirements.yml` file with the following content:

```
---
- src: git@git.lab.example.com:student/report
  scm: git
  version: main
```

2.4. Use the `ansible-galaxy` command to download and install the `report` role into the `roles` directory of your project:

```
[student@workstation network-infra]$ ansible-galaxy install \
-r roles/requirements.yml -p roles
Starting galaxy role install process
- extracting report to /home/student/network-infra/roles/report
- report (main) was installed successfully
```

2.5. Use the `ansible-galaxy` command to list the installed roles. Verify that the `report` role is listed:

```
[student@workstation network-infra]$ ansible-galaxy list -p roles
# /home/student/network-infra/roles
- report, main
# /usr/share/ansible/roles
...output omitted...
```

3. Edit the `dynamic_report.yml` playbook. Add two plays before the `Play to build a report` play that gather a full set of facts from the Cisco IOS and Arista EOS managed nodes.

3.1. Add the following play as the first play in the `dynamic_report.yml` playbook to gather all the facts from the `ios` group of managed nodes. Use the `cisco.ios.ios_facts` module with the `gather_subset` parameter in the task of the play:

```
---
- name: Gather facts from Cisco IOS managed nodes
  hosts: ios
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all

- name: Play to build a report
  hosts: workstation
  gather_facts: false
  tasks:
    - name: Build the report
      ansible.builtin.include_role:
        name: report
```

3.2. Add another play to the `dynamic_report.yml` playbook between the two existing plays. This play must gather all the facts from the `eos` group of managed nodes. Use the `arista.eos.eos_facts` module with the `gather_subset` parameter in the task of the play:

```
---
- name: Gather facts from Cisco IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all

- name: Gather facts from Arista EOS managed nodes
  hosts: eos
  gather_facts: false
  tasks:
    - name: Gather EOS facts
      arista.eos.eos_facts:
        gather_subset:
          - all

- name: Play to build a report
  hosts: workstation
  gather_facts: false
  tasks:
    - name: Build the report
      ansible.builtin.include_role:
        name: report
```

4. Run the `dynamic_report.yml` playbook to gather all the facts from the managed nodes and to create the report. You can find the report at `http://localhost:8080`. The `Managed Node` column in the generated report shows a set of facts gathered from the managed nodes. Verify the collected data for the `arista1.lab.example.com` managed nodes.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4.1. Use the `ansible-navigator` command to run the `dynamic_report.yml` playbook:

```
[student@workstation network-infra]$ ansible-navigator run dynamic_report.yml

PLAY [Gather facts from Cisco IOS managed nodes] *********************************

TASK [Gather IOS facts] ***********************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]

PLAY [Gather facts from Arista EOS managed nodes] ********************************

TASK [Gather EOS facts] ***********************************************************
ok: [arista2.lab.example.com]
ok: [arista1.lab.example.com]

PLAY [Play to build a report] *****************************************************

TASK [Build the report] ***********************************************************

TASK [report : Copy CSS style] ****************************************************
changed: [workstation]

TASK [report : Copy CSS images] ***************************************************
changed: [workstation]

TASK [report : Create HTML report] ************************************************
changed: [workstation]

PLAY RECAP ************************************************************************
arista1.lab.example.com   : ok=1     changed=0     unreachable=0     failed=0 ...
arista2.lab.example.com   : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe1.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe2.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
workstation               : ok=3     changed=3     unreachable=0     failed=0 ...
```

4.2. Navigate to `http://localhost:8080`. The report shows relevant information for your network infrastructure.

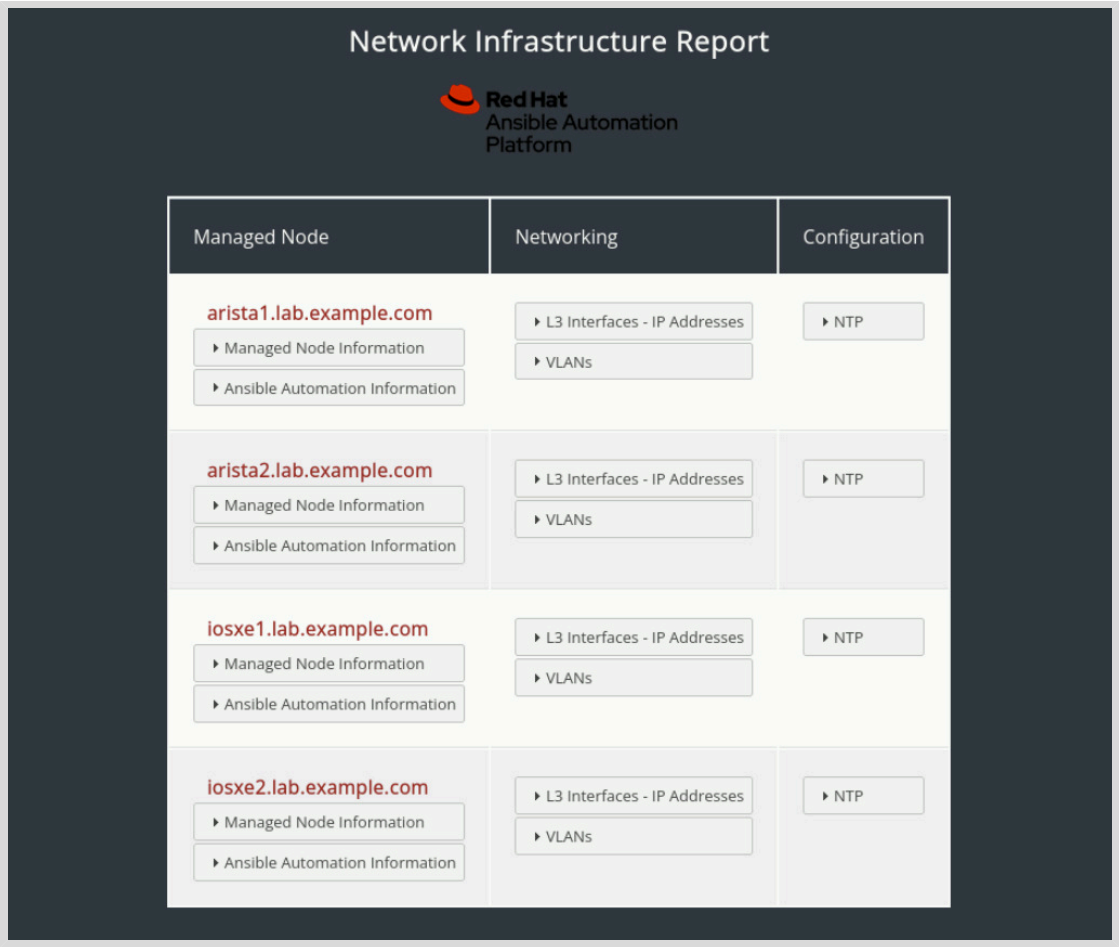


Figure 7.2: Example web report

4.3. Locate the `arista1.lab.example.com` managed node. Click **Managed Node Information** and then click **Ansible Automation Information**.

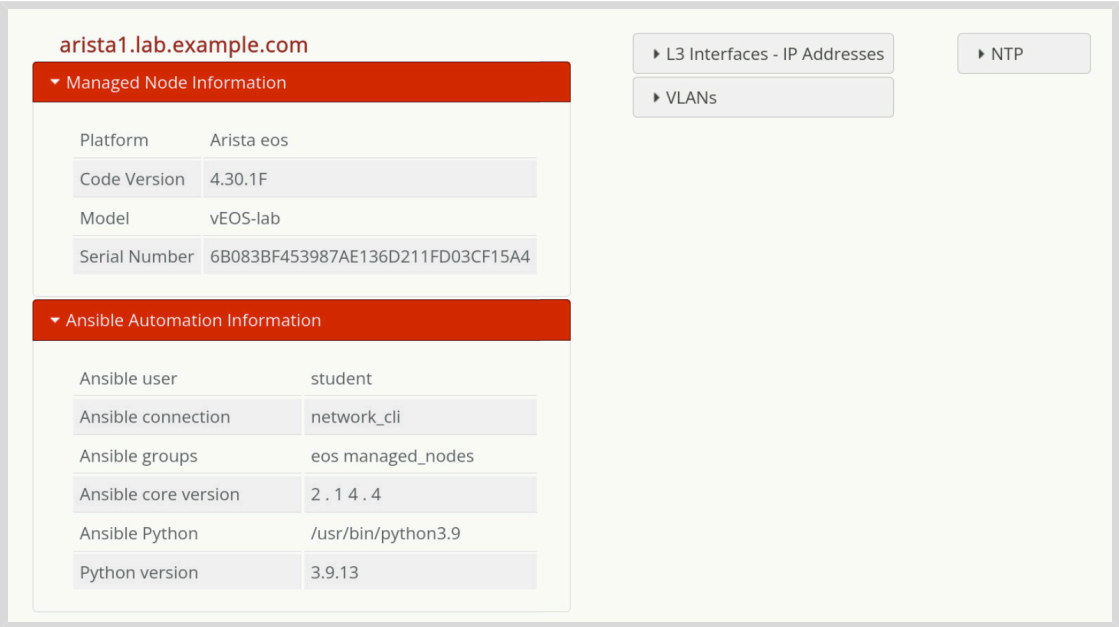


Figure 7.3: Gathered facts from the `arista1.lab.example.com` managed node

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The report shows the collected information from the `arista1.lab.example.com` managed node.

> **Note**
> The serial number for your managed node could be different from the one displayed here.

4.4. *Optional.* Click **Managed Node Information** and **Ansible Automation Information** for any of the other managed nodes.

5. The report also displays gathered information related to the configuration of Layer 3 (L3) interfaces, VLANs, and Network Time Protocol (NTP) on the managed nodes.

Edit the `dynamic_report.yml` playbook to also gather the subset of resource facts related to L3 and virtual LAN (VLAN) configurations. Use the `gather_network_resources` parameter with the `l3_interfaces` and `vlans` values.

Run the `configure.yml` playbook to configure L3, VLANs, and NTP on your managed nodes. When finished, run the `dynamic_report.yml` playbook again to verify the configured values in the report.

5.1. Edit the `dynamic_report.yml` playbook so that the first two plays gather the L3 and VLAN resources from the managed nodes:

```
---
- name: Gather facts from Cisco IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all
        gather_network_resources:
          - vlans
          - l3_interfaces

- name: Gather facts from Arista EOS managed nodes
  hosts: eos
  gather_facts: false
  tasks:
    - name: Gather EOS facts
      arista.eos.eos_facts:
        gather_subset:
          - all
        gather_network_resources:
          - vlans
          - l3_interfaces

...output omitted...
```

5.2. Use the `ansible-navigator run` command to run the `configure.yml` playbook.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The `configure.yml` playbook imports the `configure_l3.yml`, `configure_vlans.yml`, and `configure_ntp.yml` playbooks.

The `configure_l3.yml` and the `configure_ntp.yml` playbooks run against the Cisco IOS and Arista EOS managed nodes. The `configure_vlans.yml` playbook runs only against the Arista EOS managed nodes.

```
[student@workstation network-infra]$ ansible-navigator run configure.yml

...output omitted...

PLAY RECAP *********************************************************************
arista1.lab.example.com    : ok=6     changed=6     unreachable=0     failed=0 ...
arista2.lab.example.com    : ok=6     changed=6     unreachable=0     failed=0 ...
iosxe1.lab.example.com     : ok=4     changed=4     unreachable=0     failed=0 ...
iosxe2.lab.example.com     : ok=4     changed=4     unreachable=0     failed=0 ...
```

Note

The playbooks configure the Cisco IOS and Arista EOS managed nodes based on the variable files in the `group_vars` directory.

5.3. Use the `ansible-navigator` command to run the `dynamic_report.yml` playbook again:

```
[student@workstation network-infra]$ ansible-navigator run dynamic_report.yml

...output omitted...

PLAY RECAP *********************************************************************
arista1.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
arista2.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe1.lab.example.com     : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe2.lab.example.com     : ok=1     changed=0     unreachable=0     failed=0 ...
workstation                : ok=3     changed=1     unreachable=0     failed=0 ...
```

5.4. Return to the generated report at `http://localhost:8080` and reload the web page. Click **L3 interfaces - IP Addresses**, **VLANs**, and **NTP** for the `arista2.lab.example.com` and `iosxe1.lab.example.com` managed nodes.

Notice the following points:

- Even though NTP was configured on the managed nodes, the report does not display NTP information because NTP facts were not gathered in the `Gather IOS facts` and `Gather EOS facts` tasks in the `dynamic_report.yml` playbook.
- Even though VLANs were not configured on the Cisco IOS managed nodes, the report displays VLAN information because VLAN facts were gathered in the `Gather IOS facts` and `Gather EOS facts` tasks in the `dynamic_report.yml` playbook. This results in the report displaying the legacy reserved VLAN information.

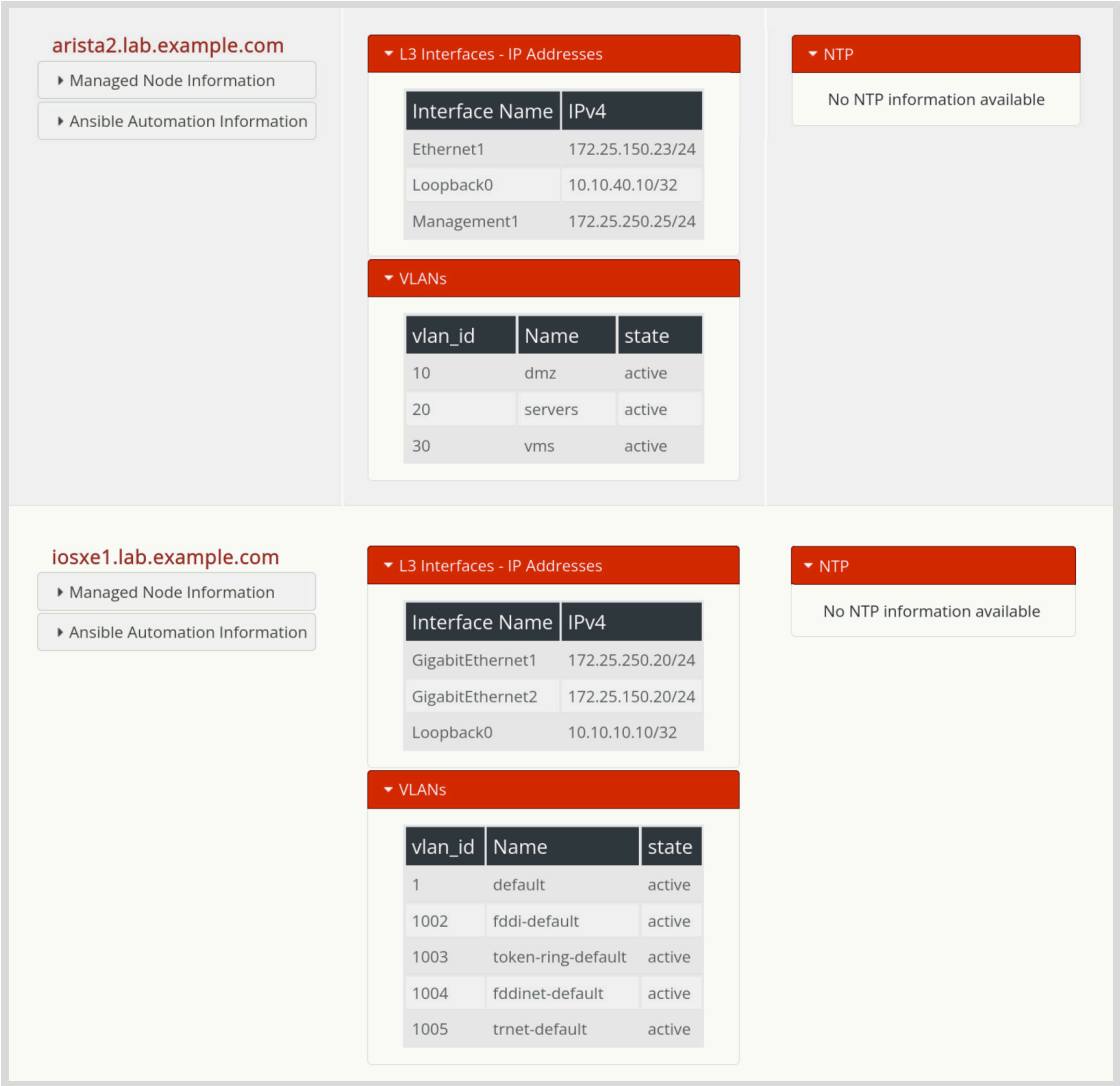


Figure 7.4: L3 interfaces and VLANs facts from the managed nodes

6. Edit the `dynamic_report.yml` playbook to also gather the subset of resource facts related to NTP configurations, and then run the `dynamic_report.yml` playbook again to verify that the NTP configuration is displayed in the report. Use the `gather_network_resources` parameter with the `ntp_global` value.

   6.1. Edit the `dynamic_report.yml` playbook so that the first two plays also gather the NTP resources from the managed nodes:

```
---
- name: Gather facts from Cisco IOS managed nodes
  hosts: ios
  gather_facts: false
  tasks:
    - name: Gather IOS facts
      cisco.ios.ios_facts:
        gather_subset:
          - all
        gather_network_resources:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
          - vlans
          - l3_interfaces
          - ntp_global

- name: Gather facts from Arista EOS managed nodes
  hosts: eos
  gather_facts: false
  tasks:
    - name: Gather EOS facts
      arista.eos.eos_facts:
        gather_subset:
          - all
        gather_network_resources:
          - vlans
          - l3_interfaces
          - ntp_global

...output omitted...
```

6.2. Use the `ansible-navigator` command to run the `dynamic_report.yml` playbook:

```
[student@workstation network-infra]$ ansible-navigator run dynamic_report.yml

...output omitted...

PLAY RECAP *********************************************************************
arista1.lab.example.com   : ok=1     changed=0     unreachable=0     failed=0 ...
arista2.lab.example.com   : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe1.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
iosxe2.lab.example.com    : ok=1     changed=0     unreachable=0     failed=0 ...
workstation               : ok=3     changed=1     unreachable=0     failed=0 ...
```

6.3. Return to the generated report at `http://localhost:8080` and reload the web page. Click **NTP** for the `arista2.lab.example.com` and `iosxe1.lab.example.com` managed nodes.

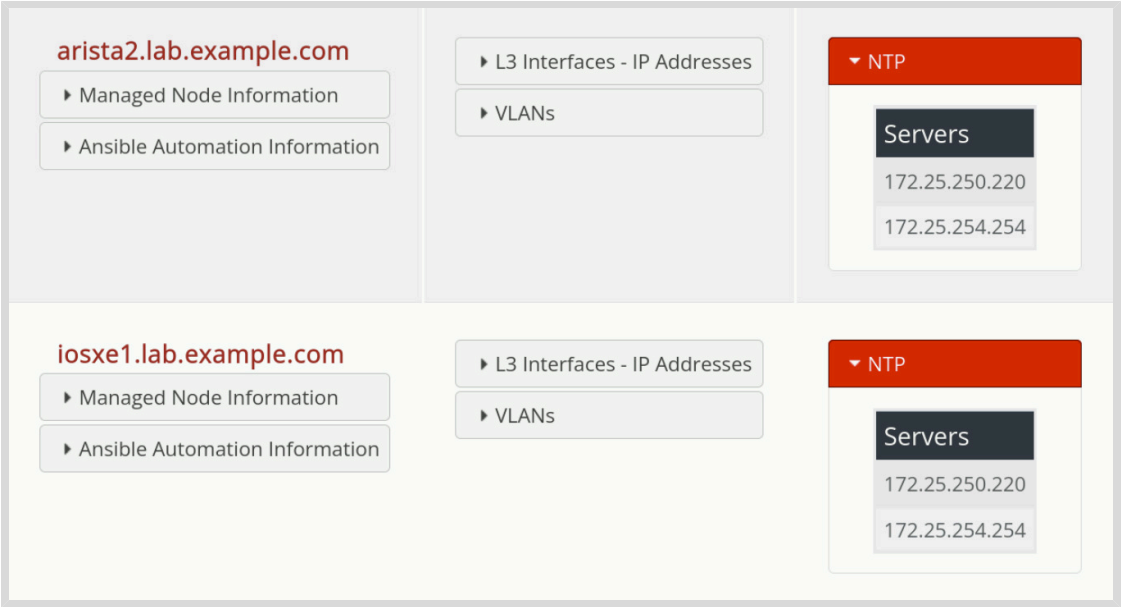


Figure 7.5: NTP configuration on managed nodes

The report displays the NTP configuration from the managed nodes.

6.4. Click **NTP** again for the `arista2.lab.example.com` and `iosxe1.lab.example.com` managed nodes in the report to hide the displayed information.

7. Close the `/home/student/network-infra` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

7.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/network-infra` directory.

7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation network-infra]$ cd
```

Finish

On the `workstation` machine, 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 network-infra
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Generating Configuration Settings from Jinja2 Templates

Objectives

- Generate new device configurations using Jinja2 templates, and use templates to manage existing devices.

Templating Files

The `arista.eos` and `cisco.ios` Ansible Content Collections provide a number of modules that you can use to modify existing configurations. These include modules to configure interfaces and access lists, among others.

An alternative way to deploy network device configurations is to *template* them. With this method, you can write a template configuration file that is automatically customized for the managed network device when the configuration is deployed. Templates typically use Ansible variables and facts.

Introduction to Jinja2

Ansible uses the Jinja2 templating system for template files. Ansible also uses Jinja2 syntax to reference variables in playbooks.

Using Delimiters

Variables and logic expressions are placed between tags, or *delimiters*. When a Jinja2 template is evaluated, the expression `{{ EXPR }}` is replaced with the results of that expression or variable. Jinja2 templates use the `{% EXPR %}` syntax for special control structures or logic that loops over Jinja2 code or perform tests. You can use the `{# COMMENT #}` syntax to enclose comments that should not appear in the final file.

The following example illustrates the use of the `{# COMMENT #}` syntax in the first line of the Jinja2 template. The variable reference in the second line is replaced with the value of the system facts being referenced.

```
{# Obtain the hostname from facts #}
hostname {{ ansible_facts['net_hostname'] }}
```

Building a Jinja2 Template

A Jinja2 template is composed of multiple elements: data, variables, and expressions. Variables and expressions are replaced with their values when the Jinja2 template is rendered. The variables used in the template can be specified in any location where you can specify variables. These variables can include inventory variables, variables defined in included or imported files, and variables defined in the `vars` section of a play, block, or task. You can also use gathered facts as variables in a template.

Template files are most commonly kept in the `templates` directory of the project for your playbook, and typically are assigned a `.j2` file extension to make it clear that they are Jinja2 template files.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

A file containing a Jinja2 template does not need to have any specific file extension (for example, `.j2`). However, providing such a file extension might make it easier for you to remember that it is a template file.

The following example shows how to create a template for an Arista managed node with variables and facts retrieved by Ansible from the managed network device.

```
{# Hostname from facts #} ❶
hostname {{ ansible_facts['net_hostname'] }} ❷
ip name-server vrf default 172.25.250.220
!
spanning-tree mode mstp
!
system l1
   unsupported speed action error
   unsupported error-correction action error
!
interface Ethernet1
!
interface Loopback100
   description Loopback100 Interface
   ip address 10.10.10.100/32
!
{# Use Loopback101 info from variable #}
interface {{ interfaces[inventory_hostname][1]['name'] }} ❸
   description {{ interfaces[inventory_hostname][1]['name'] }} Interface
   ip address {{ interfaces[inventory_hostname][1]['address'] }}
!
interface Management1
   ip address 172.25.250.24/24
!
```

❶ This line is a comment, which is not included in the final file.

❷ In this line, the hostname is obtained from the `ansible_facts['net_hostname']` fact.

❸ In this line, the name of the interface is obtained from a variable.

Deploying Jinja2 Templates

Jinja2 templates provide a powerful way to customize configuration files to be deployed on managed nodes. After you create a Jinja2 template, you can use the respective `*_config` module for the network OS to deploy the customized configuration to the managed node.

To deploy the network device configuration from a template to an Arista managed node, use the following syntax. The value associated with the `src` key specifies the source Jinja2 template.

```
- name: Deploy configuration template on EOS
  arista.eos.eos_config:
    src: eos_configuration.j2
    save_when: changed
```

For use by Abdul Majeed Khan Yamkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The following example shows how to deploy a network device configuration from a template on Cisco IOS:

```
- name: Deploy configuration template on IOS
  cisco.ios.ios_config:
    src: ios_configuration.j2
    save_when: changed
```

Important

The previous examples use the `save_when` option to persistently save changes. You might decide to omit this option and only save the running configuration to the startup configuration after you have validated the changes.

Control Structures

You can use Jinja2 control structures in template files to reduce repetitive typing, to enter entries for each managed node in a play dynamically, or conditionally insert text into a file.

Using Loops

Jinja2 uses `for` statements to provide looping functionality. In the following example, the `interfaces` variable has a list of values. The `interface` variable is replaced with all the values in the `interfaces` variable, one value per line.

```
{% for interface in interfaces %}
      {{ interface }}
{% endfor %}
```

The following example template uses a `for` statement and a conditional to run through all the values in the `interfaces` variable, replacing `myinterface` with each value, unless the value is `Loopback100`.

```
{# for statement #}
{% for myinterface in interfaces if not myinterface == "Loopback100" %}
Interface number {{ loop.index }} - {{ myinterface }}
{% endfor %}
```

The `loop.index` variable expands to the index number that the loop is currently on. It has a value of 1 the first time the loop executes, and it increments by 1 through each iteration.

As another example, this template also uses a `for` statement. It assumes a `mynodes` variable that contains a list of managed nodes to be managed has been defined by the inventory being used. If you put the following `for` statement in a Jinja2 template, all nodes in the `mynodes` group from the inventory would be listed in the resulting file:

```
{% for mynode in groups['mynodes'] %}
{{ mynode }}
{% endfor %}
```

For a more practical example, you can use this example to configure access lists on an Arista managed node from a variable file. Assume that you have the following playbook:

```
---
- name: Configure access lists
  hosts: arista01
  vars_files:
    - vars.yml
  gather_facts: false
  tasks:
    - name: Deploy configuration
      arista.eos.eos_config:
        src: templates/acls.j2
        save_when: changed
```

Also assume that the `vars.yml` file contains the following content:

```
---
acls:
  - id: 10
    address: 172.25.251.20
  - id: 20
    address: 172.25.251.21
  - id: 30
    address: 172.25.251.22
  - id: 40
    address: 172.25.251.23
```

The following lines in the `templates/acls.j2` template file configure all the access lists using a `for` statement:

```
{# for statement to deploy access lists #}
{% for acl in acls %}
   {{ acl['id'] }} permit ip host {{ acl['address'] }} any log
{% endfor %}
```

The preceding example results in the following lines in the managed node configuration:

```
   10 permit ip host 172.25.251.20 any log
   20 permit ip host 172.25.251.21 any log
   30 permit ip host 172.25.251.22 any log
   40 permit ip host 172.25.251.23 any log
```

Using Conditionals

Jinja2 uses `if` statements to provide conditional control. This allows you to put a line in a deployed file if certain conditions are met.

In the following example, the value of the `result` variable is placed in the deployed file only if the value of the `finished` variable is `true`.

```
{% if finished %}
{{ result }}
{% endif %}
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

References

`arista.eos.eos_config` - Manage Arista EOS Configuration Sections - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/arista/eos/eos_config_module.html

`cisco.ios.ios_config` - Module to Manage Configuration Sections - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

Templating (Jinja2) - Ansible Documentation
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_templating.html

Using Filters to Manipulate Data - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

Template Designer Documentation
https://jinja.palletsprojects.com/en/latest/templates/

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Generating Configuration Settings from Jinja2 Templates

Generate device configurations using Jinja2 templates, and use templates to manage existing devices.

Outcomes

- Create Jinja2 templates from existing configuration backups by using variables and facts.
- Create a playbook to apply the configuration templates on managed network nodes.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start network-templates
```

Instructions

You want to use configuration backups as a template to deploy the same configuration across multiple managed network nodes, without the need to manually edit the configuration for each individual node. To do this, you need to inspect an existing configuration file and replace node-specific values with variables and facts.

1. Open the `/home/student/network-templates` directory in VS Code. Examine the host variables and group variables for the managed Arista EOS nodes.

   1.1. Open VS Code and click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **network-templates** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Inspect the `interfaces` variable in the `inventory/host_vars/arista1.lab.example.com/interfaces.yml` file:

```
---
interfaces:
  - name: Loopback100
    address: 10.10.10.100/32
  - name: Loopback101
    address: 10.10.10.101/32
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

1.4. Inspect the `interfaces` variable in the `inventory/host_vars/arista2.lab.example.com/interfaces.yml` file:

```
---
interfaces:
  - name: Loopback100
    address: 10.10.10.102/32
  - name: Loopback101
    address: 10.10.10.103/32
```

1.5. Inspect the `acls` variable in the `inventory/group_vars/all/acls.yml` file:

```
---
acls:
  - id: 10
    host: 172.25.251.20
  - id: 20
    host: 172.25.251.21
  - id: 30
    host: 172.25.251.22
  - id: 40
    host: 172.25.251.23
```

2. Inspect the `eos_config.cfg` configuration backup file and make a copy of this file called `templates/eos_config.j2`. Convert the `templates/eos_config.j2` file into a Jinja2 template by replacing the node-specific information with expressions using variables and facts.

2.1. Inspect the `eos_config.cfg` file. Note all the instances of the hostname, interfaces, and access lists.

```
! Command: show running-config
! device: arista-host01 (vEOS-lab, EOS-4.30.1F)
!
hostname arista-host01
!
interface Loopback100
   description Loopback100 Interface
   ip address 10.0.30.2/32
!
interface Loopback101
   description Loopback101 Interface
   ip address 10.0.31.2/32
!
ip access-list test
   10 permit ip host 172.25.202.7 any log
   20 permit ip host 172.25.210.7 any log
!
```

2.2. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/network-templates` directory in a GNOME terminal. Use the `cp` command to create a copy of the `eos_config.cfg` configuration backup as the `templates/eos_config.j2` template:

```
[student@workstation ~]$ cd ~/network-templates
[student@workstation network-templates]$ cp -v eos_config.cfg \
templates/eos_config.j2
'eos_config.cfg' -> 'templates/eos_config.j2'
```

2.3. The hostname is different for every managed Arista EOS node. Replace any instance of the hostname in the `templates/eos_config.j2` template with the `hostname` variable:

```
...output omitted...
! device: {{ hostname }} (vEOS-lab, EOS-4.30.1F)
!
hostname {{ hostname }}
...output omitted...
```

2.4. Every managed Arista EOS node has a different IP address, and the interface names might not always be the same. Use a `for` statement to replace any instances of interface names and associated IP addresses in the `templates/eos_config.j2` template using the host variables for the Arista EOS hosts. Include a comment to state the purpose of the `for` statement.

```
...output omitted...
!
{# loop through each interface #}
{% for INTERFACE in interfaces %}
interface {{ INTERFACE['name'] }}
   description {{ INTERFACE['name'] }} Interface
   ip address {{ INTERFACE['address'] }}
!
{% endfor %}
...output omitted...
```

Important

This Jinja2 template matches the format of the existing `eos_config.cfg` file where the interface description uses the name of the interface. If the `interfaces` variable includes a description for each interface, then the `description` line might be written as `description {{ INTERFACES['description'] }}` or even `description {{ INTERFACES['description'] | default(INTERFACES['name'] + ' Interface') }}`.

2.5. In this example, the access lists are the same across all managed nodes. Use a `for` statement to replace the Access Control Entries in the `templates/eos_config.j2` template using the group variables for the `all` group. Include a comment to state the purpose of the `for` statement.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
...output omitted...
ip access-list templates-project
{# loop through each access list #}
{% for ACL in acls %}
   {{ ACL['id'] }} permit ip host {{ ACL['host'] }} any log
{% endfor %}
!
end
```

Note

In this example all managed network nodes use the same access list configuration. The access lists could be hard coded into the template file in this instance, but it is still a good idea to use variables. Later in this exercise the same access list variables are used in a template to configure Cisco IOS devices. Using the group variables for the access lists prevents the need to edit hard-coded access lists across multiple template files in a multi-vendor environment.

2.6. The final `templates/eos_config.j2` template must consist of the following content:

```
! Command: show running-config
! device: {{ hostname }} (vEOS-lab, EOS-4.30.1F)
!
hostname {{ hostname }}
!
{# loop through each interface #}
{% for INTERFACE in interfaces %}
interface {{ INTERFACE['name'] }}
   description {{ INTERFACE['name'] }} Interface
   ip address {{ INTERFACE['address'] }}
!
{% endfor %}
ip access-list templates-project
{# loop through each access list #}
{% for ACL in acls %}
   {{ ACL['id'] }} permit ip host {{ ACL['host'] }} any log
{% endfor %}
!
end
```

3. Examine the host variables for the managed Cisco IOS nodes.

3.1. Inspect the `interfaces` variable in the `inventory/host_vars/iosxe1.lab.example.com/interfaces.yml` file:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
interfaces:
  - name: Loopback100
    address: 10.10.10.104
    netmask: 255.255.255.255
  - name: Loopback101
    address: 10.10.10.105
    netmask: 255.255.255.255
```

3.2. Inspect the `interfaces` variable in the `inventory/host_vars/iosxe2.lab.example.com/interfaces.yml` file:

```
---
interfaces:
  - name: Loopback100
    address: 10.10.10.106
    netmask: 255.255.255.255
  - name: Loopback101
    address: 10.10.10.107
    netmask: 255.255.255.255
```

4. Inspect the `ios_config.cfg` configuration backup file and make a copy of this file called `templates/ios_config.j2`. Convert the `templates/ios_config.j2` file into a Jinja2 template by replacing the node-specific information with expressions using variables and facts.

4.1. Inspect the `ios_config.cfg` file. Note all the instances of the hostname, interfaces, and access lists.

```
hostname iosxe-host01
!
interface Loopback100
 description Loopback100 Interface
 ip address 10.0.32.2 255.255.255.255
!
interface Loopback101
 description Loopback101 Interface
 ip address 10.0.33.2 255.255.255.255
!
ip access-list extended templates-project
 10 permit ip host 172.25.203.7 any log
 20 permit ip host 172.25.211.7 any log
!
end
```

4.2. Use the `cp` command to create a copy of the `ios_config.cfg` configuration backup as the `templates/ios_config.j2` template:

```
[student@workstation network-templates]$ cp -v ios_config.cfg \
templates/ios_config.j2
'ios_config.cfg' -> 'templates/ios_config.j2'
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4.3. The hostname is different for every managed Cisco IOS node. Replace any instances of the hostname in the `templates/ios_config.j2` template with the `hostname` variable.

```
...output omitted...
hostname {{ hostname }}
...output omitted...
```

4.4. Every managed Cisco IOS node has a different IP address, and the interface names might not always be the same. Use a `for` statement to replace any instances of interface names and associated IP addresses in the `templates/ios_config.j2` template using the host variables for the Cisco IOS hosts. Include a comment to state the purpose of the `for` statement.

```
...output omitted...
!
{# loop through each interface #}
{% for INTERFACE in interfaces %}
interface {{ INTERFACE['name'] }}
 description {{ INTERFACE['name'] }} Interface
 ip address {{ INTERFACE['address'] }} {{ INTERFACE['netmask'] }}
!
{% endfor %}
...output omitted...
```

4.5. In this example, the access lists are the same across all managed nodes. Use a `for` statement to replace the Access Control Entries in the `templates/ios_config.j2` template using the group variables for the `all` group. Include a comment to state the purpose of the `for` statement.

```
...output omitted...
ip access-list extended templates-project
{# loop through each access list #}
{% for ACL in acls %}
 {{ ACL['id'] }} permit ip host {{ ACL['host'] }} any log
{% endfor %}
!
end
```

4.6. The final `templates/ios_config.j2` template must consist of the following content:

```
hostname {{ hostname }}
!
{# loop through each interface #}
{% for INTERFACE in interfaces %}
interface {{ INTERFACE['name'] }}
 description {{ INTERFACE['name'] }} Interface
 ip address {{ INTERFACE['address'] }} {{ INTERFACE['netmask'] }}
!
{% endfor %}
ip access-list extended templates-project
{# loop through each access list #}
```

```
{% for ACL in acls %}
 {{ ACL['id'] }} permit ip host {{ ACL['host'] }} any log
{% endfor %}
!
end
```

5. Create the `generate_files.yml` playbook to generate the configuration files from the templates.

5.1. Begin the playbook with a play that gathers facts on the `eos` and `ios` groups:

```
---
- name: Gather minimal facts
  hosts:
    - eos
    - ios
  gather_facts: true
  tasks: []
```

5.2. Create a second play that generates the configuration files from the templates on the `workstation.lab.example.com` host:

```
...output omitted...
- name: Generate configuration files from templates
  hosts: workstation.lab.example.com
  become: true
  become_user: student
  gather_facts: false
  vars:
    config_dir: "{{ playbook_dir }}/configs"
  tasks:
```

5.3. Create a task in the second play that creates the `config` directory in the `network-templates` project directory:

```
...output omitted...
    - name: Create the '{{ config_dir }}' directory
      ansible.builtin.file:
        path: "{{ config_dir }}"
        state: directory
```

5.4. Create a second task in the second play that generates the configuration files for the managed Arista nodes:

```
...output omitted...
    - name: Generate Arista configuration files
      vars:
        hostname: "{{ hostvars[item]['ansible_facts']['net_hostname'] }}"
        interfaces: "{{ hostvars[item]['interfaces'] }}"
        acls: "{{ hostvars[item]['acls'] }}"
      ansible.builtin.template:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        src: templates/eos_config.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
      loop: "{{ groups['eos'] }}"
```

5.5. Create a third task in the second play that generates the configuration files for the managed Cisco nodes. The completed `generate_files.yml` playbook must consist of the following content:

```
---
- name: Gather minimal facts
  hosts:
    - eos
    - ios
  gather_facts: true
  tasks: []

- name: Generate configuration files from templates
  hosts: workstation.lab.example.com
  become: true
  become_user: student
  gather_facts: false
  vars:
    config_dir: "{{ playbook_dir }}/configs"
  tasks:
    - name: Create the '{{ config_dir }}' directory
      ansible.builtin.file:
        path: "{{ config_dir }}"
        state: directory

    - name: Generate Arista configuration files
      vars:
        hostname: "{{ hostvars[item]['ansible_facts']['net_hostname'] }}"
        interfaces: "{{ hostvars[item]['interfaces'] }}"
        acls: "{{ hostvars[item]['acls'] }}"
      ansible.builtin.template:
        src: templates/eos_config.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
      loop: "{{ groups['eos'] }}"

    - name: Generate Cisco configuration files
      vars:
        hostname: "{{ hostvars[item]['ansible_facts']['net_hostname'] }}"
        interfaces: "{{ hostvars[item]['interfaces'] }}"
        acls: "{{ hostvars[item]['acls'] }}"
      ansible.builtin.template:
        src: templates/ios_config.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
      loop: "{{ groups['ios'] }}"
```

6. Run the `generate_files.yml` playbook to generate the configuration files from the templates.

6.1. Use the `ansible-navigator run` command to run the `generate_files.yml` playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation network-templates]$ ansible-navigator run generate_files.yml

PLAY [Gather minimal facts] ****************************************************

TASK [Gathering Facts] *********************************************************
ok: [iosxe2.lab.example.com]
ok: [iosxe1.lab.example.com]
ok: [arista1.lab.example.com]
ok: [arista2.lab.example.com]

PLAY [Generate configuration files from templates] ******************************

TASK [Create the '/home/student/Dev/network-templates/configs' directory] ********
changed: [workstation.lab.example.com]

TASK [Generate Arista configuration files] **************************************
changed: [workstation.lab.example.com] => (item=arista1.lab.example.com)
changed: [workstation.lab.example.com] => (item=arista2.lab.example.com)

TASK [Generate Cisco configuration files] ***************************************
changed: [workstation.lab.example.com] => (item=iosxe1.lab.example.com)
changed: [workstation.lab.example.com] => (item=iosxe2.lab.example.com)

PLAY RECAP *********************************************************************
arista1.lab.example.com      : ok=1    changed=0    unreachable=0    failed=0 ...
arista2.lab.example.com      : ok=1    changed=0    unreachable=0    failed=0 ...
iosxe1.lab.example.com       : ok=1    changed=0    unreachable=0    failed=0 ...
iosxe2.lab.example.com       : ok=1    changed=0    unreachable=0    failed=0 ...
workstation.lab.example.com  : ok=3    changed=3    unreachable=0    failed=0 ...
```

6.2. Examine the `configs/arista1.lab.example.com.cfg` file:

```
! Command: show running-config
! device: arista1.lab.example.com (vEOS-lab, EOS-4.30.1F)
!
hostname arista1.lab.example.com
!
interface Loopback100
   description Loopback100 Interface
   ip address 10.10.10.100/32
!
interface Loopback101
   description Loopback101 Interface
   ip address 10.10.10.101/32
!
ip access-list templates-project
   10 permit ip host 172.25.251.20 any log
   20 permit ip host 172.25.251.21 any log
   30 permit ip host 172.25.251.22 any log
   40 permit ip host 172.25.251.23 any log
!
end
```

6.3. Examine the `configs/iosxe1.lab.example.com.cfg` file:

```
hostname iosxe1.lab.example.com
!
interface Loopback100
 description Loopback100 Interface
 ip address 10.10.10.104 255.255.255.255
!
interface Loopback101
 description Loopback101 Interface
 ip address 10.10.10.105 255.255.255.255
!
ip access-list extended templates-project
 10 permit ip host 172.25.251.20 any log
 20 permit ip host 172.25.251.21 any log
 30 permit ip host 172.25.251.22 any log
 40 permit ip host 172.25.251.23 any log
!
end
```

7. Create the `deploy_config.yml` playbook that uses the appropriate `*_config` modules to deploy the configuration files to the managed nodes. Match the appropriate configuration file to the managed nodes with the correct corresponding network OS.

7.1. Create the `deploy_config.yml` playbook with the following content:

```
---
- name: Configure access lists
  hosts:
    - eos
    - ios
  gather_facts: false
  tasks:
    - name: Deploy Arista configuration
      when: ansible_network_os == 'arista.eos.eos'
      arista.eos.eos_config:
        src: configs/{{ inventory_hostname }}.cfg

    - name: Deploy Cisco configuration
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_config:
        src: configs/{{ inventory_hostname }}.cfg
```

8. Run the `deploy_config.yml` playbook to configure the managed network nodes.

8.1. Use the `ansible-navigator run` command to run the `deploy_config.yml` playbook:

```
[student@workstation network-templates]$ ansible-navigator run deploy_config.yml

PLAY [Configure access lists] ****************************************************

TASK [Deploy Arista configuration] ***********************************************
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
skipping: [iosxe1.lab.example.com]
skipping: [iosxe2.lab.example.com]
[WARNING]: To ensure idempotency and correct diff the input configuration lines
should be similar to how they appear if present in the running configuration on
device including the indentation
changed: [arista2.lab.example.com]
changed: [arista1.lab.example.com]

TASK [Deploy Cisco configuration] ************************************************
skipping: [arista1.lab.example.com]
skipping: [arista2.lab.example.com]
changed: [iosxe1.lab.example.com]
changed: [iosxe2.lab.example.com]

PLAY RECAP ***********************************************************************
arista1.lab.example.com   : ok=1     changed=1     unreachable=0     failed=0 ...
arista2.lab.example.com   : ok=1     changed=1     unreachable=0     failed=0 ...
iosxe1.lab.example.com    : ok=1     changed=1     unreachable=0     failed=0 ...
iosxe2.lab.example.com    : ok=1     changed=1     unreachable=0     failed=0 ...
```

9. Run the `show_run.yml` playbook to view the running configurations of the devices. Examine the output of the playbook to verify the templates were applied.

9.1. Use the `ansible-navigator run` command to run the `show_run.yml` playbook. Note that the output of the running configuration for each device reflects the `hostname`, `interfaces`, and `acls` variables used in the Jinja2 templates.

```
[student@workstation network-templates]$ ansible-navigator run show_run.yml
...output omitted...
ok: [arista1.lab.example.com] => {
...output omitted...
                "! device: arista1.lab.example.com (vEOS-lab, EOS-4.30.1F)",
...output omitted...
                "hostname arista1.lab.example.com",
...output omitted...
                "interface Loopback100",
                "   description Loopback100 Interface",
                "   ip address 10.10.10.100/32",
                "!",
                "interface Loopback101",
                "   description Loopback101 Interface",
                "   ip address 10.10.10.101/32",
...output omitted...
                "ip access-list templates-project",
                "   10 permit ip host 172.25.251.20 any log",
                "   20 permit ip host 172.25.251.21 any log",
                "   30 permit ip host 172.25.251.22 any log",
                "   40 permit ip host 172.25.251.23 any log",
...output omitted...
ok: [iosxe1.lab.example.com] => {
...output omitted...
                "hostname iosxe1.lab.example.com",
...output omitted...
                "interface Loopback100",
```

```
                 " description Loopback100 Interface",
                 " ip address 10.10.10.104 255.255.255.255",
                 "!",
                 "interface Loopback101",
                 " description Loopback101 Interface",
                 " ip address 10.10.10.105 255.255.255.255",
...output omitted...
                 "ip access-list extended templates-project",
                 " 10 permit ip host 172.25.251.20 any log",
                 " 20 permit ip host 172.25.251.21 any log",
                 " 30 permit ip host 172.25.251.22 any log",
                 " 40 permit ip host 172.25.251.23 any log",
...output omitted...
```

10. Close the `/home/student/network-templates` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

10.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/network-templates` directory.

10.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation network-templates]$ cd
```

Finish

On the `workstation` machine, 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 network-templates
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Simplifying Tasks with Platform-independent Modules

Objectives

- Simplify tasks by using vendor-independent network resource modules and the `ansible.netcommon` Ansible Content Collection.

Platform Independence with Network Resource Modules

Network platforms typically separate their configuration data into subsections, or *resources*.

You can use network resource modules to read and configure these resources, such as VLANs, interfaces, ACLs, and others. The goal of network resource modules is to provide a consistent experience across network devices running different network platforms.

For example, you can use the following network resource modules to configure Layer 3 interfaces on various network platforms. Notice that the `l3_interfaces` part in the name of the resource module is the same in each case.

- `cisco.ios.ios_l3_interfaces`
- `arista.eos.eos_l3_interfaces`
- `junipernetworks.junos.junos_l3_interfaces`

In the same way, you can use the following network resource modules to configure ACLs on different network platforms.

- `cisco.ios.ios_acls`
- `arista.eos.eos_acls`
- `junipernetworks.junos.junos_acls`

Network resource modules enable you to standardize your automation files and create configurations that you can apply consistently across network operating systems.

Network Resource Module States

Use network resource modules by setting the module state. Resource modules support the following states:

merged
: Ansible merges the current device configuration with the configuration provided in the task.

replaced
: Ansible replaces the current device configuration subsection with the configuration provided in the task.

overridden
: Ansible overrides the current device configuration with the configuration provided in the task. This means that the entire resource is deleted (for example, all L3 interfaces), and then the new configuration is added.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Warning
Use caution with the `overridden` state as you could remove your access to the device.

deleted
Ansible deletes the current device configuration subsection and restores any default settings.

gathered
Ansible displays the resource details gathered from the network device. Ansible displays these details in the `gathered` key in the task output.

rendered
Ansible renders the provided configuration in the task in the native format for the network platform. Ansible displays this rendered configuration in the `rendered` key in the task output.

Note
The `rendered` state does not communicate with or change the network device and can be used offline.

parsed
Ansible parses the configuration subsection data from the `running_config` parameter into Ansible structured data. View the collection documentation for the particular network platform to determine how to generate the correct configuration data to pass to the `running_config` parameter. Ansible displays this data in the `parsed` key in the task output.

Note
The `parsed` state does not communicate with or change the network device and can be used offline.

For more details on resource modules and states, visit `https://docs.ansible.com/ansible/latest/network/user_guide/network_resource_modules.html`.

Resource Module Return Values

Network resource modules return a data structure that can be registered into a variable, or displayed in the task output.

Resource modules all support the same return values, regardless of the network OS. This makes it easy to automate consistently across network platforms.

Resource modules support the following return values:

before
The configuration, displayed as structured data, before the module runs.

after
The configuration, displayed as structured data, after the module runs.

commands
The set of commands pushed to the network device.

You can use the return values to verify if and how the device configuration was changed.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Platform-independent Modules

The `ansible.netcommon` collection provides modules that can be used to automate network devices without having to specify a particular network platform. This enables you to manage or configure devices from different vendors without having to use a separate module for each different type of device.

The following table lists the modules available in the `ansible.netcommon` collection:

Module	Description
`cli_command`	Run a CLI command.
`cli_config`	Push text-based configuration to network devices.
`grpc_config`	Append `configs` option to an existing configuration in a gRPC-enabled devices.
`grpc_get`	Fetch configuration/state data from gRPC-enabled target hosts.
`net_get`	Copy a file from a network device to automation controller.
`net_ping`	Use `ping` from a network device to ensure communication to a remote destination.
`net_put`	Copy a file from automation controller to a network device.
`netconf_config`	Enable the user to send a configuration XML file to a NETCONF device, and detects if there was a configuration change.
`netconf_get`	Fetch configuration/state data from NETCONF-enabled network devices.
`netconf_rpc`	Execute operations on NETCONF-enabled network devices.
`network_resource`	Manage resource modules.
`restconf_config`	Create, update, read, or delete configuration data on RESTCONF-enabled devices.
`restconf_get`	Fetch configuration/state data from RESTCONF-enabled devices.
`telnet`	Execute a Telnet command.

Sending Commands to Managed Network Nodes

You can use platform-specific `*_command` modules to send commands to managed network nodes that have a command-line interface, and then process the results of those commands.

If you use a separate module for each different network platform, then you would need three separate tasks to run the same type of command, as in the following example:

```
---
- name: Run Cisco IOS command
  cisco.ios.ios_command:
    commands: "show ip interface"
  when: ansible_network_os == 'cisco.ios.ios'
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
- name: Run Junos command
  junipernetworks.junos.junos_command:
    commands: "show interface terse"
  when: ansible_network_os == 'junipernetworks.junos.junos'

- name: Run Arista EOS command
  arista.eos.eos_command:
    commands: "show ip interface"
  when: ansible_network_os == 'arista.eos.eos'
```

You can simplify such playbooks by using the platform-independent `ansible.netcommon.cli_command` module, as in the following example:

```
---
- name: Run a command across multiple network platforms
  hosts: ios,junos,eos
  gather_facts: false
  vars:
    ansible_connection: ansible.netcommon.network_cli
  tasks:
    - name: Run command and display results
      block:
        - name: Run a command
          ansible.netcommon.cli_command:
            command: "{{ some_command }}"
          register: result

        - name: Display result
          ansible.builtin.debug:
            var: result['stdout_lines']
```

Store the `some_command` variable in files under the `group_vars` directory, with the actual commands to run on each network platform.

For example, the following variable file holds the `some_command` variable for the Arista EOS platform:

```
---
some_command: "show ip interface brief"
```

In the following example, the variable file holds the `some_command` variable for the Juniper Junos platform:

```
---
some_command: "show interface terse"
```

Simultaneously Configuring Multiple Network Platforms

You can use the `ansible.netcommon.cli_config` module to configure managed network nodes, back up existing configurations, or look for configuration differences.

The following example configures NTP on Cisco IOS, Cisco Junos, and Arista EOS platforms:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Configure IOS, Junos, and EOS platforms
  hosts: ios,junos,eos
  gather_facts: false
  vars:
    ansible_connection: ansible.netcommon.network_cli
  tasks:
    - name: Configure NTP settings
      ansible.netcommon.cli_config:
         config: "{{ config_ntp }}"
      notify:
        - Save configuration

  handlers:
    - name: Save configuration
      ansible.netcommon.cli_command:
        command: "{{ save_config }}"
```

You store the `config_ntp` and `save_config` variables in files under the `group_vars` directory, with the actual commands to run on each network platform.

For example, the following variable file holds the `config_ntp` and `save_config` variable for the Arista EOS platform:

```
---
config_ntp: |
  ntp server 10.10.10.10 prefer
  ntp server 10.10.10.11
save_config: "copy running-config startup-config"
```

In the following example, the variable file holds the `config_ntp` and `save_config` variables for the Juniper Junos platform:

```
---
config_ntp: |
  set system ntp server 10.10.10.10 prefer
  set system ntp server 10.10.10.11
save_config: "commit"
```

The `ansible.netcommon.cli_config` module can also back up configurations across network platforms. The following example backs up the configurations on Cisco IOS, Juniper Junos, and Arista EOS platforms:

```
---
- name: Backup configurations on IOS, Junos, and EOS
  hosts: ios,junos,eos
  gather_facts: false
  tasks:
    - name: Backup configurations
      vars:
        ansible_connection: ansible.netcommon.network_cli
      ansible.netcommon.cli_config:
```

```
          backup: true
          backup_options:
            filename: "{{ inventory_hostname }}.cfg"
            dir_path: ./backup
```

Parsing Configuration and Operational State Data

Parsing is the process of converting unstructured or semistructured text into structured data. You can use parsed data for many purposes, including creating dynamic documentation or to automate troubleshooting steps.

Network resource modules and fact modules parse the data returned from managed network nodes when running Ansible tasks. You can also use the `ansible.utils.cli_parse` module to parse data.

Parsing Configuration Data

Network configurations are stored in different formats on different network platforms. You might want to store configurations from multiple network platforms in a vendor-neutral format for configuration management, migration, or troubleshooting purposes.

You can use the resource module `parsed` state to parse and reformat sections of configuration data. You can then store the configuration data in a version control system or import it into a configuration management system.

For example, the following playbook parses an IOS Layer 3 interface configuration stored in the `l3int.cfg` file, and then saves the transformed configuration data to the `parsed_l3int.cfg` file. The `l3int.cfg` file is generated by running the `show running-config | section ^interface` command on the IOS device and then saving the output to the file:

```
---
- name: Parse L3 interface configuration and save the data to a file
  hosts: ios1.example.com
  gather_facts: false
  tasks:
    - name: Parse L3 interface configuration
      cisco.ios.ios_l3_interfaces:
        running_config: "{{ lookup('file', 'l3int.cfg') }}"
        state: parsed
      register: ios_l3int

    - name: Save configuration data
      ansible.builtin.copy:
        content: >-
          "{{ ios_l3int | ansible.builtin.to_nice_yaml }}"
        dest: parsed_l3int.cfg
      delegate_to: localhost
```

The `l3int.cfg` file consists of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
interface GigabitEthernet1
 ip address 172.25.250.20 255.255.255.0
 negotiation auto
interface GigabitEthernet2
 ip address 172.25.251.20 255.255.255.0
 ipv6 address 2001:db8:251:20::/64
 negotiation auto
```

After this playbook is run, the `parsed_l3int.cfg` file consists of the following content:

```
"changed: false
failed: false
parsed:
-   ipv4:
    -   address: 172.25.250.20/24
    name: GigabitEthernet1
-   ipv4:
    -   address: 172.25.251.20/24
    ipv6:
    -   address: 2001:db8:251:20::/64
    name: GigabitEthernet2"
```

Note
Consult the module documentation to determine how to generate the correct configuration file to use with the `parsed` state and the `running_config` option.

Parsing Operational State Data

You might want to use the `ansible.utils.cli_parse` module to parse data for the following reasons:

- Because a network resource module does not exist for the particular network platform and resource
- Because fact modules do not return the data that you want
- To convert configuration files into structured data and then send it to an external system, such as an IP address management (IPAM) or network management system
- To parse operational state data, such as the output of `show` commands, and there is no module available to get the data you require

The `ansible.utils.cli_parse` module uses an architecture based on plug-ins and supports several different plug-ins. The `ansible.netcommon.native` plug-in is covered in this section.

Note
For more information on parsing plug-ins that you can use with the `ansible.utils.cli_parse` module, visit `https://docs.ansible.com/ansible/latest/network/user_guide/cli_parsing.html`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

To use the `ansible.netcommon.native` plug-in, you must create a template. The plug-in takes the output of a network device command and applies a YAML template to transform the command output into structured data. The YAML template file uses regular expressions to parse the received data into structured data.

The YAML template is structured as a list of parsers, each containing the following key-value pairs:

Key	Description
`example`	An example line of the text line to be parsed.
`getval`	A regular expression using named capture groups to store the extracted data.
`result`	A data tree, populated as a template, from the parsed data.
`shared`	(optional) The shared key makes the parsed values available to the rest of the parser entries until matched again.

For example, the following play runs the `show ip interface` command on IOS managed nodes, and then parses and displays the output:

```
---
- name: Run a command and parse the output
  hosts: ios
  gather_facts: false
  tasks:
    - name: Parse interface data
      ansible.utils.cli_parse:
        command: "show ip interface"
        parser:
          name: ansible.netcommon.native
          template_path: "./templates/interfaces.yml"
        set_fact: ios_interfaces

    - name: Display structured data
      ansible.builtin.debug:
        var: ansible_facts['ios_interfaces']
```

Important

Set the path to the template file with the `template_path` parameter.

If you do not set the template path, then the default path for a parser template is `templates/{{ short_os }}_{{ command }}.{{ extension }}`.

- `short_os` is derived from the `ansible_network_os` variable and set to lowercase.
- `command` is the command passed to the module with spaces replaced with underscore (_) characters.
- `extension` is specific to the parser used (native=`yaml`, textfsm=`textfsm`, ttp=`ttp`).

The associated YAML template consists of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- example: "GigabitEthernet1 is up, line protocol is up"
  getval: '(?P<name>\S+)\sis\s(?P<admin_state>\S+)\,\sline\sprotocol\sis\s(?
P<oper_state>\S+)'
  result:
    "{{ name }}":
      name: "{{ name }}"
      state:
        admin: "{{ admin_state }}"
        operating: "{{ oper_state }}"
```

When the play is run, the parsed output is displayed as follows:

```
...output omitted...
ok: [ios1.example.com] => {
    "ansible_facts": {
        "ios_interfaces": {
            "GigabitEthernet1": {vendor-independent
}
ok: [ios2.example.com] => {
    "ansible_facts": {
        "ios_interfaces": {
            "GigabitEthernet1": {
                "name": "GigabitEthernet1",
                "state": {
                    "admin": "up",
                    "operating": "up"
                }
            }
        }
    }
}
...output omitted...
```

You can then copy this data to a file and perform further actions with it, such as sending the data to an external system.

Using Ansible Validated Content

Ansible validated content is a set of Ansible Content Collections that are focused on specific use cases.

Validated content stems from and complements the Red Hat Validated Patterns initiative. A selection of Ansible content is curated by experts, such as the Red Hat Automation Community of Practice, to ensure that:

- The use cases are based on successfully deployed customer examples.
- The content creators are trusted and verified subject matter experts.
- The content itself adheres to the latest best practices and guidelines issued by the Red Hat engineering team.
- Ansible validated content is tested against supported versions of Ansible Automation Platform and is ready for use in production environments.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The `network.base` Ansible Content Collection is Ansible validated content focusing on platform-independent network automation, and provides core functionality for other network-oriented Ansible Content Collections that are Ansible validated content, such as the `network.ospf`, `network.interfaces`, and `network.bgp` collections.

- The `persist` action gets the facts for a given resource and stores the facts as inventory host variables.
- The `list` action displays the list of all resource modules supported for a given network OS.
- The `gather` action gathers and displays the facts for specified network resources.
- The `deploy` action deploys changes, based on host variables, to the device.
- The `configure` action provides another way of pushing configuration changes to the device.

For more information on Ansible validated content, visit `https://www.redhat.com/en/blog/automate-expert-ansible-validated-content`.

References

Network Resource Modules - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/network_resource_modules.html

Ansible.Netcommon - Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/index.html

Parsing Semi-structured Text with Ansible - Ansible Documentation
https://docs.ansible.com/ansible/latest/network/user_guide/cli_parsing.html

Automate Like an Expert with Ansible Validated Content
https://www.redhat.com/en/blog/automate-expert-ansible-validated-content

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Simplifying Tasks with Platform-independent Modules

Simplify tasks by using vendor-independent network resource modules and the `ansible.netcommon` Ansible Content Collection.

Outcomes

- Use the platform-independent `ansible.netcommon.cli_command` module to run a command on multiple network platforms and display the results.
- Use the `ansible.utils.cli_parse` module to parse command output and display the results.
- Create and run playbooks with plays that demonstrate various resource module states.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start network-independent
```

Instructions

1. Open the `/home/student/network-independent` directory in VS Code and create the `send_command.yml` playbook to run a command on multiple network platforms and display the results.

   1.1. Open VS Code and then click **File** > **Open Folder**.

   1.2. Navigate to **Home** > **network-independent** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. In VS Code, create the `send_command.yml` playbook file with a play that uses the `ansible.netcommon.cli_command` module to run a command specified by the `show_interfaces` variable.

   The play must target the managed nodes in the `ios`, `junos`, and `eos` inventory groups.

   Add the following lines to the playbook:

```
---
- name: Run a command across multiple network platforms
  hosts: ios,junos,eos
  gather_facts: false
  tasks:
    - name: Run a command
      vars:
        ansible_connection: ansible.netcommon.network_cli
      ansible.netcommon.cli_command:
        command: "{{ show_interfaces }}"
      register: result

    - name: Display the results
      ansible.builtin.debug:
        var: result['stdout_lines']
```

2. Create variable files named `commands.yml` for each of the different network platforms.

The variable files must hold the `show_interfaces` variable, with the value of that variable set to the command to show a summary of interface statuses for the specific network platform.

2.1. In VS Code, create a variable file named `group_vars/ios/commands.yml` that holds the command to send to the IOS network platform.

The completed file must consist of the following content:

```
---
show_interfaces: "show ip interface brief"
```

2.2. In VS Code, create a variable file named `group_vars/eos/commands.yml` that holds the command to send to the EOS network platform.

The completed file must consist of the following content:

```
---
show_interfaces: "show ip interface brief"
```

2.3. In VS Code, create a variable file named `group_vars/junos/commands.yml` that holds the command to send to the Junos network platform.

The completed file must consist of the following content:

```
---
show_interfaces: "show interfaces terse"
```

Note

The `ansible.netcommon.cli_command` module also supports sending multiple commands. You could use a loop and add multiple commands to the variable files to send multiple commands to your managed nodes.

3. Run the `send_command.yml` playbook and inspect the playbook output.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

3.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/network-independent` directory in a GNOME terminal:

```
[student@workstation ~]$ cd network-independent
```

3.2. Use the `ansible-navigator run` command to run the `send_command.yml` playbook.

Inspect the playbook output to ensure that the correct command was run on each network platform, and it produced a summary of interface statuses:

```
[student@workstation network-independent]$ ansible-navigator run \
send_command.yml

PLAY [Run a command across multiple network platforms] ********************

TASK [Run a command] ***********************************************************
ok: [iosxe1.lab.example.com]
ok: [iosxe2.lab.example.com]
ok: [junos2.lab.example.com]
ok: [junos1.lab.example.com]
ok: [arista1.lab.example.com]
ok: [arista2.lab.example.com]

TASK [Display the results] *****************************************************
ok: [iosxe1.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface              IP-Address      OK? Method Status
 Protocol",
        "GigabitEthernet1       172.25.250.20   YES NVRAM  up
 up      ",
        "GigabitEthernet2       unassigned      YES NVRAM  administratively down
 down"
    ]
}
ok: [junos1.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface               Admin Link Proto    Local
 Remote",
        "cbp0                    up    up",
        "demux0                  up    up",
        "dsc                     up    up",
        "em1                     up    up",
        "em1.0                   up    up   inet     10.0.0.4/8      ",
        "                                             128.0.0.1/2     ",
        "                                             128.0.0.4/2     ",
        "                                   inet6    fe80::5254:ff:fe12:bdfe/64",
        "                                             fec0::a:0:0:4/64",
...output omitted...
}
ok: [arista1.lab.example.com] => {
    "result['stdout_lines']": [
        "Address",
```

```
        "Interface         IP Address             Status       Protocol
 MTU    Owner  ",
        "---------------- --------------------- ------------ --------------
 ---------- -------",
        "Management1       172.25.250.24/24       up           up
 1500"
    ]
}
...output omitted...
```

4. Create a playbook named `parse_command.yml` with a play to run the `show ip interface` command on IOS managed nodes and then parse and display the output.

   Use the `ansible.utils.cli_parse` module with the `ansible.netcommon.native` filter to parse the command output.

   The YAML template file `/templates/interfaces.yml` has been provided for you for use with the `ansible.netcommon.native` plug-in. The YAML template file uses regular expressions to parse the command output into structured data.

   4.1. In VS Code, create the `parse_command.yml` playbook file.

   The completed file must consist of the following content:

```
---
- name: Run a command and parse the output
  hosts: ios
  gather_facts: false
  tasks:
    - name: Parse interface data
      ansible.utils.cli_parse:
        command: "show ip interface"
        parser:
          name: ansible.netcommon.native
          template_path: "./templates/interfaces.yml"
        set_fact: ios_interfaces

    - name: Display structured data
      ansible.builtin.debug:
        var: ansible_facts['ios_interfaces']
```

   4.2. Inspect the contents of the `templates/interfaces.yml` file:

```
---
- example: "GigabitEthernet1 is up, line protocol is up"
  getval: '(?P<name>\S+)\sis\s(?P<admin_state>\S+)\,\sline\sprotocol\sis\s(?
P<oper_state>\S+)'
  result:
    "{{ name }}":
      name: "{{ name }}"
      state:
        admin: "{{ admin_state }}"
        operating: "{{ oper_state }}"
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4.3. Return to the terminal and use the `ansible-navigator run` command to run the `parse_command.yml` playbook.

Inspect the playbook output to verify that the values for the `name`, `admin_state`, and `oper_state` variables are present for each interface.

```
[student@workstation network-independent]$ ansible-navigator run \
parse_command.yml

PLAY [Run a command and parse the output] ************************************

TASK [Parse interface data] **************************************************
ok: [iosxe1.lab.example.com]
ok: [iosxe2.lab.example.com]

TASK [Display structured data] ***********************************************
ok: [iosxe2.lab.example.com] => {
    "ansible_facts['ios_interfaces']": {
        "GigabitEthernet1": {
            "name": "GigabitEthernet1",
            "state": {
                "admin": "up",
                "operating": "up"
            }
        }
    }
}
ok: [iosxe1.lab.example.com] => {
    "ansible_facts['ios_interfaces']": {
        "GigabitEthernet1": {
            "name": "GigabitEthernet1",
            "state": {
                "admin": "up",
                "operating": "up"
            }
        }
    }
}
...output omitted...
```

5. Create a playbook named `config_states.yml` that uses the `arista.eos.eos_vlans` resource module with the `merged` state to update the virtual LAN (VLAN) configuration by creating VLANs 100, 200, 300, and 400, and then run the playbook.

5.1. In VS Code, create a playbook named `config_states.yml` with a play that targets the EOS managed nodes and uses the `arista.eos.eos_vlans` resource module with the `state` parameter set to `merged`.

The completed file must consist of the following content:

```
---
- name: Change configuration and demonstrate a resource module state
  hosts: eos
  gather_facts: false
  tasks:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - name: Change configuration using the merged state
      arista.eos.eos_vlans:
        state: merged
        config:
          - name: vlan100
            vlan_id: 100
          - name: vlan200
            vlan_id: 200
          - name: vlan300
            vlan_id: 300
          - name: vlan400
            vlan_id: 400

    - name: Show configuration after configuration change
      arista.eos.eos_command:
        commands:
          - show run | s vlan
```

5.2. Return to the terminal and use the `ansible-navigator run` command to run the `config_states.yml` playbook.

Use the `-vvv` option to increase the output verbosity, so that the resource module state and the return values of `before`, `after`, and `commands` are displayed in the output.

Inspect the output to see that VLANs 100, 200, 300, and 400 are present in the `after` key, which means that they have been added to the configuration:

```
[student@workstation network-independent]$ ansible-navigator run \
config_states.yml -vvv
...output omitted...
TASK [Change configuration using the merged state] ****************************
...output omitted...
changed: [arista1.lab.example.com] => {
    "after": [
        {
            "name": "vlan100",
            "state": "active",
            "vlan_id": 100
        },
        {
            "name": "vlan200",
            "state": "active",
            "vlan_id": 200
        },
        {
            "name": "vlan300",
            "state": "active",
            "vlan_id": 300
        },
        {
            "name": "vlan400",
            "state": "active",
            "vlan_id": 400
        }
```

```
    ],
    "before": [],
    "changed": true,
    "commands": [
        "vlan 100",
        "name vlan100",
        "vlan 200",
        "name vlan200",
        "vlan 300",
        "name vlan300",
        "vlan 400",
        "name vlan400"
    ],
...output omitted...
            "state": "merged"
        }
    }
}
...output omitted...
TASK [Show configuration after configuration change] *************************
ok: [arista1.lab.example.com] => {
    "changed": false,
    "invocation": {
        "module_args": {
            "commands": [
                "show run | s vlan"
            ],
            "interval": 1,
            "match": "all",
            "retries": 10,
            "wait_for": null
        }
    },
    "stdout": [
        "vlan 100\n   name vlan100\nvlan 200\n   name vlan200\nvlan 300\n   name
 vlan300\nvlan 400\n   name vlan400"
    ],
    "stdout_lines": [
        [
            "vlan 100",
            "   name vlan100",
            "vlan 200",
            "   name vlan200",
            "vlan 300",
            "   name vlan300",
            "vlan 400",
            "   name vlan400"
        ]
    ]
}
...output omitted...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

6. Run the `add_vlan_500.yml` playbook to add VLAN 500 to the EOS managed nodes, and then run the `config_states.yml` playbook again.

   Inspect the playbook output to see how the configuration changes when using the `merged` state.

   6.1. Inspect the contents of the `add_vlan_500.yml` file:

```
---
- name: Add a VLAN
  hosts: eos
  gather_facts: false
  tasks:
    - name: Add a VLAN
      arista.eos.eos_vlans:
        state: merged
        config:
          - name: vlan500
            vlan_id: 500
```

   6.2. Use the `ansible-navigator run` command to run the `add_vlan_500.yml` playbook to add VLAN 500 to the EOS managed nodes:

```
[student@workstation network-independent]$ ansible-navigator run add_vlan_500.yml
...output omitted...
```

   6.3. Use the `ansible-navigator run` command to run the `config_states.yml` playbook.

   Use the `-vvv` option to increase the verbosity of the resource module output.

   Inspect the playbook output to see how the configuration changes when using the `merged` state:

```
[student@workstation network-independent]$ ansible-navigator run \
config_states.yml -vvv
...output omitted...
ok: [arista1.lab.example.com] => {
    "changed": false,
    "invocation": {
        "module_args": {
            "commands": [
                "show run | s vlan"
            ],
            "interval": 1,
            "match": "all",
            "retries": 10,
            "wait_for": null
        }
    },
    "stdout": [
        "vlan 100\n   name vlan100\nvlan 200\n   name vlan200\nvlan 300\n   name
 vlan300\nvlan 400\n   name vlan400\nvlan 500\n   name vlan500"
    ],
    "stdout_lines": [
        [
```

```
            "vlan 100",
            "   name vlan100",
            "vlan 200",
            "   name vlan200",
            "vlan 300",
            "   name vlan300",
            "vlan 400",
            "   name vlan400",
            "vlan 500",
            "   name vlan500"
        ]
    ]
}
...output omitted...
```

Important

Notice that the configuration on the EOS managed nodes includes the VLAN you just added; 500. This is because the `merged` state does not remove any existing configuration.

7. Edit the `config_states.yml` playbook and change the value of the `state` parameter to `overridden`.

   Inspect the playbook output to see how the configuration changes when using the `overridden` state.

   7.1. In VS Code, edit the `config_states.yml` playbook and change the value of the `state` parameter to `overridden`.

   The completed file must consist of the following content:

```
---
- name: Change configuration and demonstrate a resource module state
  hosts: eos
  gather_facts: false
  tasks:
    - name: Change configuration using the overridden state
      arista.eos.eos_vlans:
        state: overridden
        config:
          - name: vlan100
            vlan_id: 100
          - name: vlan200
            vlan_id: 200
          - name: vlan300
            vlan_id: 300
          - name: vlan400
            vlan_id: 400

    - name: Show configuration after configuration change
      arista.eos.eos_command:
        commands:
          - show run | s vlan
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

7.2. Return to the terminal and use the `ansible-navigator run` command to run the `config_states.yml` playbook.

Use the `-vvv` option to increase the verbosity of the resource module output:

```
[student@workstation network-independent]$ ansible-navigator run \
config_states.yml -vvv
...output omitted...
ok: [arista1.lab.example.com] => {
    "changed": false,
    "invocation": {
        "module_args": {
            "commands": [
                "show run | s vlan"
            ],
            "interval": 1,
            "match": "all",
            "retries": 10,
            "wait_for": null
        }
    },
    "stdout": [
        "vlan 100\n   name vlan100\nvlan 200\n   name vlan200\nvlan 300\n   name
 vlan300\nvlan 400\n   name vlan400"
    ],
    "stdout_lines": [
        [
            "vlan 100",
            "   name vlan100",
            "vlan 200",
            "   name vlan200",
            "vlan 300",
            "   name vlan300",
            "vlan 400",
            "   name vlan400"
        ]
    ]
}
...output omitted...
```

Important

Notice that the configuration on the EOS managed nodes no longer includes VLAN 500. This is because the `overridden` state enforces all VLAN resources to the data model. This means that it removes any VLANs that are not defined in the data model being sent.

8. Edit the `config_states.yml` playbook and change the value of the `state` parameter to `rendered`.

Inspect the playbook output to see the configuration commands generated by the module.

8.1. In VS Code, edit the `config_states.yml` playbook. Change the value of the `state` parameter to `rendered`, and add VLAN 500 to the file.

The completed file must consist of the following content:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Render a configuration
  hosts: eos
  gather_facts: false
  tasks:
    - name: Render VLAN configuration
      arista.eos.eos_vlans:
        state: rendered
        config:
          - name: vlan100
            vlan_id: 100
          - name: vlan200
            vlan_id: 200
          - name: vlan300
            vlan_id: 300
          - name: vlan400
            vlan_id: 400
          - name: vlan500
            vlan_id: 500

    - name: Show configuration after configuration change
      arista.eos.eos_command:
        commands:
          - show run | s vlan
```

8.2. Return to the terminal and use the `ansible-navigator run` command to run the `config_states.yml` playbook.

Use the `-vvv` option to increase the verbosity of the resource module output.

Inspect the output to see the rendered configuration commands that would be sent to the device:

```
[student@workstation network-independent]$ ansible-navigator run \
config_states.yml -vvv

...output omitted...
    "rendered": [
        "vlan 100",
        "name vlan100",
        "vlan 200",
        "name vlan200",
        "vlan 300",
        "name vlan300",
        "vlan 400",
        "name vlan400",
        "vlan 500",
        "name vlan500"
...output omitted...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

The `rendered` state does not communicate with or change managed nodes. Use the `rendered` state to get the list of commands that would be sent to managed nodes when using the `merged`, `replaced`, or `overridden` state.

9. Close the `/home/student/network-independent` directory in VS Code, and run the `cd` command in a GNOME terminal to return to the `student` home directory:

```
[student@workstation network-independent]$ cd
```

Finish

On the `workstation` machine, 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 network-independent
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Performing Rolling Configuration Updates

Objectives

- Perform rolling network configuration updates, and automatically recover from errors.

Batch Node Configuration Strategy

Ansible supports *rolling updates*, a strategy that staggers deployments to batches of managed nodes. With this strategy, infrastructure deployment can complete with zero downtime.

Ansible can halt the deployment and limit the errors to managed nodes in a particular batch if an unforeseen problem occurs. With tests and monitoring in place, you can configure playbook tasks to perform the following actions:

- Roll back configuration for managed nodes in the affected batch.
- Quarantine affected managed nodes to enable analysis of the failed deployment.
- Send notifications of the deployment to stakeholders.

Configuring Parallelism in Ansible by Using Forks

When Ansible processes a playbook, it runs each play in order. After determining the list of managed nodes for the play, Ansible runs through each task in order. Normally, all managed nodes must complete a task before any managed node starts the next task in the play.

In theory, Ansible could connect to all managed nodes in the play simultaneously for each task. This approach works fine for small lists of managed nodes, but if the play targets hundreds of managed nodes, then these connections 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 the following method:

```
[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...
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

The previous output indicates that you can modify the `forks` parameter by using the `forks` key within the `defaults` section of an Ansible configuration file:

```
[defaults]
forks=5
```

Consider the following scenario. Assume that Ansible is configured with the default value of five forks and that the play has ten managed nodes. Ansible runs the first task in the play on the first five managed nodes, followed by a second round of execution of the first task on the next five managed nodes. After running the first task on all the managed nodes, Ansible runs the next task across all the managed nodes in groups of five nodes at a time. Ansible does this with each task in turn until the play ends.

Most Ansible modules that you use to manage network routers and switches run in the execution environment and not on the network device. Because of the increased load these modules place on the execution environment, you should be careful with increasing the `forks` parameter.

The `ansible-navigator run` command offers the `-f` or `--forks` option, which you can use to override the current value of the `forks` parameter.

Controlling Batch Size

By default, Ansible runs a play by running one task for all managed nodes before running the next task. Imagine that your play has a task that cannot succeed due to some error. If your play runs against all managed nodes in parallel, they all fail and abort the play when Ansible reaches that task, and no node completes the play. As a result, it is likely that none of the managed nodes are working correctly, which could lead to an outage.

To avoid this, you could process *some* managed nodes through the play before starting the next batch of managed nodes. If too many managed nodes fail, then you could have Ansible abort the entire play before all managed nodes attempt to run it.

The following sections describe how to configure your play to abort if too many managed nodes in a batch fail.

Setting a Fixed Batch Size

To process managed nodes through a play in batches, use the `serial` directive in your play. The `serial` directive specifies how many managed nodes should be in each batch. Ansible processes each batch of managed nodes through the play before starting the next batch. If all managed nodes in the current batch fail, the entire play is aborted, and Ansible does not start the next batch.

Consider the beginning part of this play:

```
---
- name: Update Cisco IOS
  hosts: ios
  serial: 2
```

In this example, the `serial` directive instructs Ansible to process managed nodes in the `ios` host group in batches of two managed nodes. The play is repeated with a new batch if the play runs without error.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

This process continues until all the managed nodes in the play are processed. If the total number of managed nodes in the play is not divisible by the batch size, then the last batch might contain fewer managed nodes than the indicated value of the `serial` directive. In the previous example, the last batch contains one host if the total number of managed nodes is an odd value.

Remember, if you use an integer with the `serial` directive, then as the number of managed nodes 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 managed nodes requires 100 batches to complete, but a host group with 20 managed nodes requires ten batches.

Setting Batch Size as a Percentage

You can also specify a percentage for the value of the `serial` directive:

```
---
- name: Update Cisco IOS
  hosts: ios
  serial: 25%
```

The play runs on the percentage of the managed nodes that you specify.

Ansible applies the percentage to the total number of managed nodes in the host group. If the resulting value is not an integer number of managed nodes, then the value is truncated (rounded down) to the nearest integer. The batch size cannot be zero managed nodes. If the truncated value is zero, then Ansible changes the batch size to one managed node.

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 managed nodes	3	13	19
`serial` value	25%	25%	25%
Exact number of managed nodes	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 gradually change the batch size as the play runs 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 managed nodes.

Consider the following example:

```
---
- name: Update Cisco IOS
  hosts: ios
  serial:
    - 1
    - 10%
    - 100%
```

Assume that the `ios` host group consists of 20 managed nodes. The first batch contains a single managed node.

The second batch contains 10% of the total managed nodes in the `ios` host group. Ansible computes the actual value according to the previously discussed rules (20 × 10% = 2).

The third batch contains all the remaining unprocessed managed nodes in the play (20 - 1 - 2 = 17). This setting enables Ansible to process all the remaining managed nodes efficiently.

If unprocessed managed nodes remain after the last batch corresponding to the previous `serial` directive entry, the last batch repeats until all managed nodes are processed. Consider the following play, which runs against the `ios` host group with 100 managed nodes:

```
---
- name: Update Cisco IOS
  hosts: ios
  serial:
    - 1
    - 10%
    - 25%
```

The first batch contains one managed node and the second batch contains ten managed nodes (100 × 10% = 10). The third batch processes 25 managed nodes (100 × 25% = 25), leaving 64 unprocessed managed nodes (100 - 1 - 10 - 25 = 64). Ansible continues executing the play in batch sizes of 25 managed nodes (25% of 100) until fewer than 25 unprocessed managed nodes remain. In this example, the final batch processes the remaining 14 managed nodes (100 - 1 - 10 - 25 - 25 - 25 = 14).

Aborting the Play

By default, Ansible tries to get as many managed nodes to complete a play as possible. If a task fails for a managed node, then the node is dropped from the play, but Ansible continues to run the remaining tasks for other managed nodes. The play only stops if all managed nodes fail.

However, if you use the `serial` directive to organize managed nodes into batches, then if all managed nodes in the current batch fail, Ansible stops the play for *all remaining managed nodes*, not just the remaining managed nodes in the current batch. If all managed nodes in a batch fail, then the play aborts and the next batch does not start.

Ansible keeps a list of the active managed nodes for each batch in the `ansible_play_batch` variable. Ansible removes a managed node that fails a task from the `ansible_play_batch` list, and updates this list after every task.

Consider the following hypothetical playbook, which runs against the `ios` host group with 100 managed nodes:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Update Cisco IOS
  hosts: ios
  tasks:
    - name: Step One
      cisco.ios.ios_command:
        command: SOME COMMAND

    - name: Step Two
      cisco.ios.ios_config:
        src: "{{ inventory_hostname }}.cfg"
```

If 99 managed nodes fail the first task, but one node succeeds, Ansible continues to the second task. When Ansible runs the second task, Ansible only runs that task for the one node that previously succeeded.

If you use the `serial` directive, playbook execution continues, provided that managed nodes remain in the current batch with no failures. Consider this modification to the hypothetical playbook:

```
---
- name: Update Cisco IOS
  hosts: ios
  serial: 2
  tasks:
    - name: Step One
      cisco.ios.ios_command:
        command: SOME COMMAND

    - name: Step Two
      cisco.ios.ios_config:
        src: "{{ inventory_hostname }}.cfg"
```

If the first batch of two contains one managed node that succeeds and one that fails, then the batch completes, and Ansible moves on to the second batch of two. If both managed nodes 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 managed node completes the play.
- Three managed nodes are in an error state.
- The rest of the managed nodes remain unaltered.

Specifying Failure Tolerance

By default, Ansible only halts play execution when all managed nodes in a batch fail. However, you might want a play to abort if more than a certain percentage of managed nodes 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 managed nodes in a batch exceeds this percentage, Ansible halts playbook execution.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Consider the following hypothetical playbook that runs against the `ios` host group, which contains 100 managed nodes:

```
---
- name: Update Cisco IOS
  hosts: ios
  max_fail_percentage: 30%
  serial:
    - 2
    - 10%
    - 100%
  tasks:
    - name: Step One
      cisco.ios.ios_command:
        command: SOME COMMAND

    - name: Step Two
      cisco.ios.ios_config:
        src: "{{ inventory_hostname }}.cfg"
```

The first batch contains two managed nodes. Because 30% of 2 is 0.6, a single node failure causes execution to stop.

If both managed nodes in the first batch succeed, then Ansible continues with the second batch of 10 managed nodes. Because 30% of 10 is 3, more than three node failures must occur for Ansible to stop playbook execution. If three or fewer managed nodes experience errors in the second batch, then 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, then all managed nodes are run through the play in one batch. If all managed nodes fail, then the play fails.
- If the `serial` directive is defined, then managed nodes are run through the play in multiple batches, and the play fails if all managed nodes in any one batch fail.
- If the `max_fail_percentage` directive is defined, then the play fails if more than that percentage of managed nodes in a batch fail.

If a play fails, then Ansible aborts all remaining plays in the playbook.

Running a Task One Time

In certain scenarios, you might only need to run a task one time for an entire batch of managed nodes rather than running the task for each node 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: Pause 30 seconds
  ansible.builtin.pause:
    seconds: 30
  run_once: true
```

Important

Setting the `run_once: true` directive causes a task to run only one time for each batch. If you only need to run a task one time for the entire play, and the play has multiple batches, then you can add the following conditional statement to the task:

```
- name: Pause 30 seconds
  when: inventory_hostname == ansible_play_hosts[0]
  ansible.builtin.pause:
    seconds: 30
  run_once: true
```

This conditional statement runs the task only for the first node in the play.

References

Controlling Playbook Execution: Strategies and More - Ansible Documentation
https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Guided Exercise

Performing Rolling Configuration Updates

Perform rolling configuration changes to ACLs on managed nodes, and automatically recover from any errors.

Outcomes

- Control the process that updates the Access Control Lists (ACLs) on configured network devices by processing the update in multiple batches, rather than updating all devices at once.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise, and to ensure that all required resources are available. This command also creates a project directory with the files needed for the exercise.

```
[student@workstation ~]$ lab start network-rolling
```

Instructions

1. Open the `/home/student/network-rolling` directory in VS Code and review the `junos_config_update.yml` playbook.

   1.1. Open VS Code and navigate to **File** > **Open Folder**.

   1.2. Navigate to **Home** > **network-rolling** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

   1.3. Select the `junos_config_update.yml` playbook. The playbook aims to perform an update of the Access Control Lists (ACLs) on the Juniper Junos managed nodes.

```
---
- name: Update Juniper Junos managed nodes to use new ACLs
  hosts: junos
  gather_facts: false
  tasks:
    - name: Juniper NETCONF configuration
      ansible.builtin.import_tasks: tasks/junos_netconf_enable.yml

    - name: Juniper Junos ACL configuration
      ansible.builtin.import_tasks: tasks/junos_acl_config.yml
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2. The task in the `tasks/junos_acl_config.yml` file updates the ACLs, but that file does not exist yet. Use VS Code to create the `tasks/junos_acl_config.yml` file and then use the `junipernetworks.junos.junos_acls` module to configure ACLs with the following characteristics:

   - The Juniper Junos managed nodes must have a single Access Control List (ACL) named `junos_acl`.
   - The ACL must use the `ipv4` protocol family.
   - The ACL must have the following list of Access Control Entries (ACEs):

First ACE

Description	Value
Filter term name	`permit_igmp`
Action to take	`permit`
Protocol	`igmp`
Source address	`172.25.250.100/32`

Second ACE

Description	Value
Filter term name	`bgp_12`
Source addresses	`12.0.0.0/16` `12.1.0.0/16`
Source port protocol	`bgp`
Protocol	`tcp`

Third ACE

Description	Value
Filter term name	`allow_ssh`
Source port protocol	`ssh`
Protocol	`tcp`

Note
To review the documentation for the `junipernetworks.junos.junos_acls` resource module, run the `ansible-navigator doc junipernetworks.junos.junos_acls -m stdout` command.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.1. In VS Code, right-click the `tasks` folder and select **New File**. Enter `junos_acl_config.yml` as the file name.

2.2. Edit the `tasks/junos_acl_config.yml` file so that it contains a task named `Juniper Junos ACL configuration` that uses the `junipernetworks.junos.junos_acls` module. Start the dictionary of ACLs with one ACL named `junos_acl`, and use the `ace` parameter to start listing the ACEs:

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: merged
    config:
      - acls:
          - name: junos_acl
            aces:
```

2.3. Add the first Access Control Entry (ACE) using the details in the "First ACE" table. Pay attention to the indentation for each level:

- The `name`, `grant`, `protocol`, and `source` parameters have the same level of indentation.
- The address of the source given by the `address` parameter must be indented two spaces more than the `source` parameter.

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: merged
    config:
      - acls:
          - name: junos_acl
            aces:
              - name: permit_igmp
                grant: permit
                protocol: igmp
                source:
                  address: 172.25.250.100/32
```

2.4. Add the second ACE using the details in the "Second ACE" table. Pay attention to the indentation for each level:

- The `name`, `source`, and `protocol` parameters have the same level of indentation.
- The list of addresses of the source given by the `address` parameter must be indented at least as much as the `address` parameter. This file indents the list by an extra two spaces. The `address` parameter must be indented two spaces more than the `source` parameter.
- The source port protocol given by the `eq` parameter must be indented two spaces more than the `port_protocol` parameter. The `port_protocol` parameter must be indented two spaces more than the `source` parameter.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: merged
    config:
      - acls:
          - name: junos_acl
            aces:
              - name: permit_igmp
                grant: permit
                protocol: igmp
                source:
                  address: 172.25.250.100/32

              - name: bgp_12
                source:
                  address:
                    - 12.0.0.0/16
                    - 12.1.0.0/16
                  port_protocol:
                    eq: bgp
                protocol: tcp
```

2.5. Add the third ACE using the details in the "Third ACE" table. Pay attention to the indentation for each level:

- The `name`, `source`, and `protocol` parameters have the same level of indentation.
- The source port protocol given by the `eq` parameter must be indented two spaces more than the `port_protocol` parameter. The `port_protocol` parameter must be indented two spaces more than the `source` parameter.

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: merged
    config:
      - acls:
          - name: junos_acl
            aces:
              - name: permit_igmp
                grant: permit
                protocol: igmp
                source:
                  address: 172.25.250.100/32

              - name: bgp_12
                source:
                  address:
                    - 12.0.0.0/16
                    - 12.1.0.0/16
                  port_protocol:
                    eq: bgp
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
                protocol: tcp

              - name: allow_ssh
                source:
                  port_protocol:
                    eq: ssh
                protocol: tcp
```

2.6. Add the `afi` parameter to the `junos_acl` ACL with a value of `ipv4`. The indentation is the same as the `acls` parameter.

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: merged
    config:
      - acls:
          - name: junos_acl
            aces:
              - name: permit_igmp
                grant: permit
                protocol: igmp
                source:
                  address: 172.25.250.100/32

              - name: bgp_12
                source:
                  address:
                    - 12.0.0.0/16
                    - 12.1.0.0/16
                  port_protocol:
                    eq: bgp
                protocol: tcp

              - name: allow_ssh
                source:
                  port_protocol:
                    eq: ssh
                protocol: tcp
        afi: ipv4
```

3. The `junos_config_update.yml` playbook is ready to configure the ACLs on the managed nodes. Add a task to the playbook to verify that the managed nodes have the expected ACL configuration.

Take the value of the `acl_config` variable, which you can find in the `acls.yml` variables file, as the ACL's configuration that is expected on the managed nodes.

3.1. Edit the `junos_config_update.yml` playbook in VS Code. Add the `acls.yml` variables file to the play, and then add a third task named `Assert that ACLs were correctly configured` that imports the `tasks/junos_acl_verify.yml` tasks file:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Update Juniper Junos managed nodes to use new ACLs
  hosts: junos
  gather_facts: false
  vars_files:
    - acls.yml
  tasks:
    - name: Juniper NETCONF configuration
      ansible.builtin.import_tasks: tasks/junos_netconf_enable.yml

    - name: Juniper Junos ACL configuration
      ansible.builtin.import_tasks: tasks/junos_acl_config.yml

    - name: Assert that ACLs were correctly configured
      ansible.builtin.import_tasks: tasks/junos_acl_verify.yml
```

3.2. Create the `tasks/junos_acl_verify.yml` file with two tasks:

- The first task uses the `junipernetworks.junos.junos_acls` module to gather the ACL configuration from the managed nodes. This collected configuration is the one the managed nodes have after the ACL configuration task finishes.
- The second task uses the `ansible.builtin.assert` module to verify that the collected configuration is as expected. If the collected configuration differs from the expected configuration, then the task fails.

```
---
- name: Gather Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: gathered
  register: result

- name: Assert that ACLs were correctly configured
  ansible.builtin.assert:
    that: acl_config == result['gathered']
```

4. The `junos_config_update.yml` playbook runs against all the managed nodes in the `junos` inventory group. You want to be sure that the change is the expected configuration before applying the change to all the managed nodes. To mitigate the effects of unforeseen deployment errors, configure the play to run in batches.

- The first batch must contain one managed node. If a task in the play fails, Ansible must stop the play for the remaining managed nodes.
- The second batch must contain 25% of the managed nodes.
- The third batch must contain the remaining managed nodes.

Also implement a "fail fast" strategy to ensure that the playbook execution stops if any managed node fails a task.

4.1. Edit the `junos_config_update.yml` playbook in VS Code. Use the `serial` directive to process the managed nodes in three batches:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
- name: Update Juniper Junos managed nodes to use new ACLs
  hosts: junos
  gather_facts: false
  vars_files:
    - acls.yml
  serial:
    - 1
    - 25%
    - 100%
  tasks:
...output omitted...
```

Important

The lab environment in this course only has two Juniper Junos managed nodes in the `junos` inventory group, so this strategy is hypothetical, something that you could implement in your real environment.

With this strategy in the course lab environment, the play runs against one of the managed nodes, and if successful, then it runs against the other.

4.2. Add the `max_fail_percentage` directive to the play with a value of `0` to implement a "fail fast" strategy that stops the execution of the play at any failure:

```
---
- name: Update Juniper Junos managed nodes to use new ACLs
  hosts: junos
  gather_facts: false
  vars_files:
    - acls.yml
  max_fail_percentage: 0
  serial:
   - 1
   - 25%
   - 100%
  tasks:
...output omitted...
```

4.3. The complete `junos_config_update.yml` playbook must consist of the following content:

```
---
- name: Update Juniper Junos managed nodes to use new ACLs
  hosts: junos
  gather_facts: false
  vars_files:
    - acls.yml
  max_fail_percentage: 0
  serial:
   - 1
   - 25%
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - 100%
  tasks:
    - name: Juniper NETCONF configuration
      ansible.builtin.import_tasks: tasks/junos_netconf_enable.yml

    - name: Juniper Junos ACL configuration
      ansible.builtin.import_tasks: tasks/junos_acl_config.yml

    - name: Assert that ACLs were correctly configured
      ansible.builtin.import_tasks: tasks/junos_acl_verify.yml
```

5. Run the `junos_config_update.yml` playbook to update the ACLs. The playbook fails. Increase the output verbosity of the `ansible-navigator` command to provide detailed task execution information of the failure.

   5.1. Switch to the **Terminal** tab in VS Code, or change to the `/home/student/network-rolling` directory in a GNOME terminal:

```
[student@workstation ~]$ cd ~/network-rolling
[student@workstation network-rolling]$
```

   5.2. Use the `ansible-navigator` command to run the playbook:

```
[student@workstation network-rolling]$ ansible-navigator run \
junos_config_update.yml

PLAY [Update Juniper Junos managed nodes to use new ACLs] ************************

TASK [Enable the NETCONF service] ************************************************
changed: [junos1.lab.example.com] (1)

TASK [Juniper Junos ACL configuration] *******************************************

...output omitted...

TASK [Assert that ACLs were correctly configured] ********************************
fatal: [junos1.lab.example.com]: FAILED! => { (2)
    "assertion": "acl_config == result['gathered']"
    "changed": false,
    "evaluated_to": false,
    "msg": "Assertion failed"
}

NO MORE HOSTS LEFT ***************************************************************

...output omitted...

PLAY RECAP ***********************************************************************
junos1.lab.example.com     : ok=3     changed=2     unreachable=0     failed=1 ...
```

(1) The playbook starts running only against one managed node in the `junos` inventory group.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

❷ The `Assert that ACLs were correctly configured` task failed to run against the `junos1.lab.example.com` managed node. This failure is enough to abort the play.

5.3. Use the `ansible-navigator` command to run the playbook again. This time increase the output verbosity by using the `-vvv` option. Notice that the gathered ACL configuration on the managed node contains an ACL called `bgp_13_acl` in addition to the new `junos_acl` ACL.

The `Assert that ACLs were correctly configured` task in the `tasks/junos_acl_verify.yml` file fails because the `acl_config` variable does not contain the additional ACL.

```
[student@workstation network-rolling]$ ansible-navigator run \
junos_config_update.yml -vvv
...output omitted...
    "gathered": [
        {
            "acls": [
                {
                    "aces": [
                        {
                            "name": "bgp_13",
                            "protocol": "tcp",
                            "source": {
                                "address": "13.0.0.0/16",
                                "port_protocol": {
                                    "eq": "bgp"
                                }
                            }
                        }
                    ],
                    "name": "bgp_13_acl"
                },
                {
                    "aces": [
                        {
                            "grant": "permit",
                            "name": "permit_igmp",
                            "protocol": "igmp",
                            "source": {
                                "address": "172.25.250.100/32"
                            }
                        },
                        {
                            "name": "bgp_12",
                            "protocol": "tcp",
                            "source": {
                                "address": [
                                    "12.0.0.0/16",
                                    "12.1.0.0/16"
                                ],
                                "port_protocol": {
                                    "eq": "bgp"
                                }
```

```
                            }
                        },
                        {
                            "name": "allow_ssh",
                            "protocol": "tcp",
                            "source": {
                                "port_protocol": {
                                    "eq": "ssh"
                                }
                            }
                        }
                    ],
                    "name": "junos_acl"
                }
            ],
            "afi": "ipv4"
        }
    ],
...output omitted...
```

6. At this point you can restore any backup that you have made of the managed node before running the `junos_config_update.yml` playbook, send a notification about the failed deployment, and so on. Because you want the expected ACL updated configuration in the complete `junos` group of managed nodes, modify your playbook to fix the issue.

> **Note**
> You can fix the issue in more than one way. For example, you can back up the existing configuration on the managed node, delete the configuration, and then generate a new configuration.
>
> For this guided exercise, you modify the playbook to override the managed node configuration with the configuration provided in the task.

6.1. Modify the `Juniper Junos ACL configuration` task in the `tasks/junos_acl_config.yml` file so that the task overrides the current ACL configuration instead of merging it:

```
---
- name: Juniper Junos ACL configuration
  junipernetworks.junos.junos_acls:
    state: overridden
    config:
      - acls:
          - name: junos_acl
            aces:
...output omitted...
```

6.2. Use the `ansible-navigator` command to run the playbook:

```
[student@workstation network-rolling]$ ansible-navigator run \
junos_config_update.yml
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
PLAY [Update Juniper Junos managed nodes to use new ACLs] ************************

TASK [Enable the NETCONF service] ************************************************
ok: [junos1.lab.example.com] ❶

...output omitted...

TASK [Assert that ACLs were correctly configured] ********************************
ok: [junos1.lab.example.com] => { ❷
    "changed": false,
    "msg": "All assertions passed"
}

PLAY [Update Juniper Junos managed nodes to use new ACLs] ******************** ❸

...output omitted...

TASK [Assert that ACLs were correctly configured] ********************************
ok: [junos2.lab.example.com] => { ❹
    "changed": false,
    "msg": "All assertions passed"
}

PLAY RECAP ***********************************************************************
junos1.lab.example.com    : ok=4     changed=1     unreachable=0     failed=0 ...
junos2.lab.example.com    : ok=4     changed=2     unreachable=0     failed=0 ...
```

❶ They playbook runs against one managed node.

❷ This time the configuration on the managed node is as expected.

❸ The `Update Juniper Junos managed nodes to use new ACLs` play starts running on the next batch of managed nodes.

❹ The classroom environment only has two managed nodes in the `junos` inventory group, so the play now runs against the `junos2.lab.example.com` managed node.

7. Close the `/home/student/network-rolling` directory in VS Code. If you are using the GNOME terminal, return to the `/home/student` directory.

7.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/network-rolling` directory.

7.2. If you are using the GNOME terminal, run the `cd` command to return to the student home directory:

```
[student@workstation network-rolling]$ cd
```

Finish

On the `workstation` machine, 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 network-rolling
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Summary

- You can collect information about your current environment in multiple ways: by using vendor-specific `*_facts` modules, by querying settings saved in backup files, or by using the `network.base.resource_manager` role.
- You can use Jinja2 templates to dynamically customize files by inserting the values of variables into those files when you deploy them.
- You can use Jinja2 templates to customize the configuration of specific managed nodes, and you can create files by using Jinja2 templates that contain information that the play has collected and stored in variables.
- You can use a combination of information collection techniques and Jinja2 templates to create dynamically updated documentation of your current network configuration.
- You can use the platform-independent `ansible.netcommon` and `network.base` Ansible Content Collections to work with multiple network resource modules rather than using vendor-specific resource modules.
- You can limit the impact of playbook failures by adding the `serial` and `max_fail_percentage` keywords to a play.
- You can use the `serial` keyword to configure Ansible to run through a play in batches.
- You can use the `max_fail_percentage` keyword to configure Ansible to abort the play if more than a certain percentage of nodes in the current batch fail.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Chapter 8

Comprehensive Review

Goal

Review tasks from *Network Automation with Red Hat Ansible Automation Platform*.

Sections

- Comprehensive Review

Lab

- Installation and Configuration
- Creating Playbooks
- Scaling Playbooks
- Running Playbooks with Automation Controller

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Comprehensive Review

Objectives

After completing this section, you should have reviewed and refreshed the knowledge and skills that you learned in *Network Automation with Red Hat Ansible Automation Platform*.

Reviewing Network 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, Introducing Red Hat Ansible Automation Platform

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

- Describe the motivation for automating network administration tasks with Ansible, fundamental Ansible concepts, and the basic architecture of Ansible.
- Install automation content navigator and other development tools on a workstation for Ansible automation development and testing.

Chapter 2, Implementing an Ansible Playbook

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

- Describe Ansible inventory concepts and manage a static inventory file.
- Describe where automation content navigator configuration files might be located and edit them to apply changes to default settings.
- Write a basic Ansible Playbook and run it using the automation content navigator.
- Write a playbook that uses multiple plays and per-play privilege escalation, and effectively use automation content navigator to find new modules in available Ansible Content Collections and use them to implement tasks for a play.

Chapter 3, Managing and Running Playbooks

Manage automation code in version control and run Ansible Playbooks from a centrally managed automation controller.

- Create and manage Ansible Playbooks in a Git repository, following recommended practices.
- Navigate and describe the automation controller web UI, and successfully launch a job using a job template, project, credential, and inventory.
- Identify issues in Ansible Playbooks and repair them.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Verify that Ansible can communicate with managed network devices, and troubleshoot any problems with those connections.

Chapter 4, Managing Variables and Facts

Write playbooks that use variables to simplify management of the playbook and facts to reference information about managed nodes.

- Create and reference variables that affect particular managed nodes or groups, the play, or the global environment, and describe how variable precedence works.
- Reference data about managed nodes using Ansible facts.
- Create and use surveys in automation controller to provide information to Ansible Playbook jobs.
- Format, parse, manipulate, and define the values of variables by using filters.

Chapter 5, Implementing Task Control

Manage task control and task errors in Ansible Playbooks.

- Use loops to write efficient tasks and use conditions to control when to run tasks.
- Control what happens when a task fails, and what conditions cause a task to fail.
- Create a workflow job template in automation controller that launches a branching chain of jobs based on the success or failure of each job in the workflow sequence.

Chapter 6, Simplifying Playbooks with Roles and Ansible Content Collections

Use Ansible Roles and Ansible Content Collections to develop playbooks more quickly and to reuse Ansible code.

- Manage large playbooks by importing or including other playbooks or tasks from external files, either unconditionally or based on a conditional test.
- Describe the purpose of an Ansible Role, its structure, and how roles are used in playbooks.
- Create a role in a playbook's project directory and run it as part of one of the plays in the playbook.
- Select and retrieve roles from external sources such as Git repositories or Ansible Galaxy, and use them in your playbooks.
- Obtain a set of related roles, supplementary modules, and other content from an Ansible Content Collection and use them in a playbook.

Chapter 7, Automating Network Administration Tasks

Automate common network administration tasks, discussing recommended practices and approaches to cross-vendor automation.

- Configure connectivity to managed network devices, gather device information, and generate a dynamic report that documents the state of your network infrastructure.
- Generate new device configurations using Jinja2 templates, and use templates to manage existing devices.

- Simplify tasks by using vendor-independent network resource modules and the `ansible.netcommon` Ansible Content Collection.
- Perform rolling network configuration updates, and automatically recover from errors.

For use by Abdul Majeed Khan Yamkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Installation and Configuration

Prepare for playbook development by installing required software, and then creating configuration files, inventory files, and variable files.

Outcomes

- Install software that you use throughout the comprehensive review.
- Use Git to clone a repository, create a branch, commit, and push changes to a branch.
- Generate an automation content navigator configuration file.
- Convert the provided Ansible inventory files to YAML format.
- Create Ansible group variables.

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 compreview-cr1
```

Specifications

This exercise prepares the `workstation` machine to run playbooks by using an automation execution environment with automation content navigator.

- Install the `ansible-navigator` and `podman` RPM packages.
- Download the `hub.lab.example.com/ee-supported-rhel8` automation execution environment from the lab environment's private automation hub. You can log in to the `hub.lab.example.com` container registry by using `student` as the username and `redhat123` as the password.
- Use the information in the following table to configure global Git settings for the `student` user:

Description	Value
User's name	`Student User`
User's email	`student@lab.example.com`

- Clone the `git@git.lab.example.com:student/compreview` repository into the `/home/student/git-repos` directory and then create a branch named `cr1` for this exercise.
- Create an automation content navigator configuration file that meets the following list of settings:
  - Uses the `hub.lab.example.com/ee-supported-rhel8` automation execution environment image

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

  - Specifies `missing` as the image pull policy
  - Defines the `inventory.yaml` file as an Ansible inventory entry
  - Disables the creation of playbook artifacts

- Perform the following actions on the provided `inventory-controller` and `inventory` files:
  - Convert the files from INI format to YAML format. Use `inventory-controller.yaml` and `inventory.yaml` as the respective file names.
  - Delete any variable information from the `inventory-controller.yaml` file.
  - Delete any variable information other than the `ansible_ssh_private_key_file` and `ansible_user` variables from the `inventory.yml` file, and then move those definitions to the `all` group of managed nodes.
  - Delete the `inventory-controller` and `inventory` files.

> **Note**
> The reason for having two inventories is that you can develop and test playbooks on the `workstation` machine using the `inventory.yaml` file, and then you can import the `inventory-controller.yaml` file into automation controller to run the playbooks from that environment.
>
> By defining the `ansible_ssh_private_key_file` variable in the `inventory.yaml` file, rather than in a file in the `group_vars/all` directory, your local playbooks can use the private SSH key without causing problems with automation controller, where you create the credential resources you need.

- Your project already has a subdirectory structure for each inventory group of managed nodes under the `group_vars` directory. Create a file named `connection.yml` for the `eos`, `ios`, and `junos` inventory groups. Populate each `connection.yml` file with the appropriate variables and values from the following tables:

Variable	Value for the `eos` inventory group
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`arista.eos.eos`
`ansible_become`	`true`
`ansible_become_method`	`enable`

Variable	Value for the `ios` inventory group
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`cisco.ios.ios`
`ansible_become`	`true`
`ansible_become_method`	`enable`

Variable	Value for the junos inventory group
ansible_connection	ansible.netcommon.netconf
ansible_network_os	junipernetworks.junos.junos

- Create the host_vars/workstation.lab.example.com directory and then create a file named connection.yml in that directory. Set the ansible_user variable to devops.
- Commit and push all the changes that you made in the /home/student/git-repos/compreview directory to the remote Git repository. Use Prepare for development as the commit message.

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 compreview-cr1
```

Finish

As the student user on the workstation machine, 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 compreview-cr1
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Installation and Configuration

Prepare for playbook development by installing required software, and then creating configuration files, inventory files, and variable files.

Outcomes

- Install software that you use throughout the comprehensive review.
- Use Git to clone a repository, create a branch, commit, and push changes to a branch.
- Generate an automation content navigator configuration file.
- Convert the provided Ansible inventory files to YAML format.
- Create Ansible group variables.

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 compreview-cr1
```

1. Install the `ansible-navigator` and `podman` RPM packages.

   1.1. Use the `dnf` command to install the `ansible-navigator` and `podman` packages:

```
[student@workstation ~]$ sudo dnf install ansible-navigator podman
[sudo] password for student: student
...output omitted...
Is this ok [y/N]: y
...output omitted...
Complete!
```

2. Download the `ee-supported-rhel8` automation execution environment from `hub.lab.example.com`, the lab environment's private automation hub.

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

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

   2.2. Use the `podman pull` command to download the automation execution environment:

```
[student@workstation ~]$ podman pull hub.lab.example.com/ee-supported-rhel8
Trying to pull hub.lab.example.com/ee-supported-rhel8:latest...
Getting image source signatures
Copying blob 9b6252cdda58 done
...output omitted...
```

3. Use the information in the following table to configure global Git settings for the `student` user.

Description	Value
User's name	`Student User`
User's email	`student@lab.example.com`

3.1. Use the `git` command with the `config` option to customize Git globally:

```
[student@workstation ~]$ git config --global user.name 'Student User'
[student@workstation ~]$ git config --global user.email student@lab.example.com
```

3.2. Verify the global configuration settings:

```
[student@workstation ~]$ git config --global -l
user.name=Student User
user.email=student@lab.example.com
```

4. Clone the `git@git.lab.example.com:student/compreview` repository into the `/home/student/git-repos` directory and then create a branch named `cr1` for this exercise.

4.1. Create the `/home/student/git-repos` directory and then change to this directory:

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

4.2. Clone the `git@git.lab.example.com:student/compreview` repository:

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

4.3. Change to the repository directory. Create a branch named `cr1` and check it out:

```
[student@workstation git-repos]$ cd compreview
[student@workstation compreview]$ git checkout -b cr1
Switched to a new branch 'cr1'
```

5. Create an automation content navigator configuration file that meets the following criteria:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Uses the `hub.lab.example.com/ee-supported-rhel8` automation execution environment image
- Specifies `missing` as the image pull policy
- Defines the `inventory.yaml` file as an Ansible inventory entry
- Disables the creation of playbook artifacts

5.1. Use the `ansible-navigator` command with the `settings` subcommand and the `--effective` option to display the effective settings. Add the `--eei` and `--pp` options to specify the automation execution environment image and the pull policy. Redirect the command output into a file named `navigator-sample.yml`.

```
[student@workstation compreview]$ ansible-navigator settings \
--effective --eei hub.lab.example.com/ee-supported-rhel8 \
--pp missing > navigator-sample.yml
```

Important

If you attempt to redirect the output of the `ansible-navigator` command to a file named `ansible-navigator.yml`, then the command fails.

5.2. Rename the `navigator-sample.yml` file to `ansible-navigator.yml`.

```
[student@workstation compreview]$ mv navigator-sample.yml ansible-navigator.yml
```

5.3. Open VS Code and click **File** > **Open Folder**. Navigate to **git-repos** > **compreview** and click **Open**.

Note

If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

5.4. Select the `ansible-navigator.yml` file. Add an entry for the `inventory.yaml` Ansible inventory and disable the creation of playbook artifacts. Optionally, remove any lines not related to what was requested. When finished, save the file.

The final `ansible-navigator.yml` file must consist of at least the following content:

```
---
ansible-navigator:
  ansible:
    inventory:
      entries:
        - ./inventory.yaml
  execution-environment:
    image: hub.lab.example.com/ee-supported-rhel8:latest
    pull:
      policy: missing
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  mode: stdout
  playbook-artifact:
    enable: false
```

6. Perform the following actions on the provided `inventory-controller` and `inventory` files:

- Convert the inventory files to YAML format. Use `inventory-controller.yaml` and `inventory.yaml` as the respective file names.
- Delete any variable information from the `inventory-controller.yaml` file.
- Delete any variable information other than the `ansible_user` and `ansible_ssh_private_key_file` variables from the `inventory.yaml` file.
- Delete the provided `inventory-controller` and `inventory` files to prevent any confusion.

6.1. Use the `ansible-navigator` command with the `inventory` subcommand and the `--list` and `--yaml` options to display the YAML version of the `inventory-controller` file. Redirect the output of the command to create the `inventory-controller.yaml` file:

```
[student@workstation compreview]$ ansible-navigator inventory \
-i inventory-controller --list --yaml > inventory-controller.yaml
```

6.2. Select the `inventory-controller.yaml` file in VS Code. Delete any information related to variables. When finished, save the file.

After editing the `inventory-controller.yaml` file, it contains the following content:

```
all:
  children:
    eos:
      hosts:
        arista1.lab.example.com:
        arista2.lab.example.com:
    ios:
      hosts:
        iosxe1.lab.example.com:
        iosxe2.lab.example.com:
    junos:
      hosts:
        junos1.lab.example.com:
        junos2.lab.example.com:
    ungrouped:
      hosts:
        workstation.lab.example.com:
```

6.3. Use the `ansible-navigator` command with the `inventory` subcommand and the `--list` and `--yaml` options to display the YAML version of the `inventory` file. Redirect the output of the command to create the `inventory.yaml` file:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation compreview]$ ansible-navigator inventory \
-i inventory --list --yaml > inventory.yaml
```

6.4. Select the `inventory.yaml` file in VS Code. Delete any variable information other than the `ansible_user` and `ansible_ssh_private_key_file` variables from the `inventory.yml` file.

6.5. Move the definitions of the `ansible_ssh_private_key_file` and `ansible_user` variables to the `all` group of managed nodes so that you do not have to define those variables in each group of managed nodes. When finished, save the file.

After editing the `inventory.yaml` file, it contains the following content:

```
all:
  vars:
    ansible_ssh_private_key_file: ~/.ssh/lab_rsa
    ansible_user: student
  children:
    eos:
      hosts:
        arista1.lab.example.com:
        arista2.lab.example.com:
    ios:
      hosts:
        iosxe1.lab.example.com:
        iosxe2.lab.example.com:
    junos:
      hosts:
        junos1.lab.example.com:
        junos2.lab.example.com:
    ungrouped:
      hosts:
        workstation.lab.example.com:
```

6.6. Delete the `inventory-controller` and `inventory` files:

```
[student@workstation compreview]$ rm inventory-controller inventory
```

7. Create a file named `connection.yml` under each of the `eos`, `ios`, and `junos` subdirectories of the `group_vars` directory. Populate the `connection.yml` files with the variables and values from the following tables:

Variable	Value for the eos inventory group
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`arista.eos.eos`
`ansible_become`	`true`
`ansible_become_method`	`enable`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Variable	Value for the `ios` inventory group
`ansible_connection`	`ansible.netcommon.network_cli`
`ansible_network_os`	`cisco.ios.ios`
`ansible_become`	`true`
`ansible_become_method`	`enable`

Variable	Value for the `junos` inventory group
`ansible_connection`	`ansible.netcommon.netconf`
`ansible_network_os`	`junipernetworks.junos.junos`

7.1. Create the `group_vars/eos/connection.yml` file with the following content:

```
---
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: arista.eos.eos
ansible_become: true
ansible_become_method: enable
```

7.2. Create the `group_vars/ios/connection.yml` file with the following content:

```
---
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
ansible_become: true
ansible_become_method: enable
```

7.3. Create the `group_vars/junos/connection.yml` file with the following content:

```
---
ansible_connection: ansible.netcommon.netconf
ansible_network_os: junipernetworks.junos.junos
```

8. Create the `host_vars/workstation.lab.example.com` directory and then create a file named `connection.yml` in that directory. Set the `ansible_user` variable to `devops`.

8.1. Create the `host_vars` directory and include a `workstation.lab.example.com` subdirectory:

```
[student@workstation compreview]$ mkdir -p host_vars/workstation.lab.example.com
```

8.2. Create the `host_vars/workstation.lab.example.com/connection.yml` file with the following content:

```
---
ansible_user: devops
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

9. Commit and push all the changes that you made in the `/home/student/git-repos/compreview` directory to the remote Git repository. Use `Prepare for development` as the commit message.

9.1. Use VS Code or the `git add` command to add the new and updated files and directories:

```
[student@workstation compreview]$ git add .
```

9.2. Use VS Code or the `git status` command to display the list of files that you staged for the next commit:

```
[student@workstation compreview]$ git status
On branch cr1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ansible-navigator.yml
        new file:   group_vars/eos/connection.yml
        new file:   group_vars/ios/connection.yml
        new file:   group_vars/junos/connection.yml
        new file:   host_vars/workstation.lab.example.com/connection.yml
        deleted:    inventory
        deleted:    inventory-controller
        new file:   inventory-controller.yaml
        new file:   inventory.yaml
```

9.3. Use VS Code or the `git commit` command to commit the files. Use `Prepare for development` as the commit message:

```
[student@workstation compreview]$ git commit -m "Prepare for development"
[cr1 3e21bbd] Prepare for development
 9 files changed, 66 insertions(+), 30 deletions(-)
 create mode 100644 ansible-navigator.yml
 create mode 100644 group_vars/eos/connection.yml
 create mode 100644 group_vars/ios/connection.yml
 create mode 100644 group_vars/junos/connection.yml
 create mode 100644 host_vars/workstation.lab.example.com/connection.yml
 delete mode 100644 inventory
 delete mode 100644 inventory-controller
 create mode 100644 inventory-controller.yaml
 create mode 100644 inventory.yaml
```

9.4. Use VS Code or the `git push` command to update the remote Git repository:

```
[student@workstation compreview]$ git push -u origin cr1
Enumerating objects: 20, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 6 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (15/15), 1.50 KiB | 1.50 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for cr1, visit:
```

```
remote:   https://git.lab.example.com/student/compreview/-/merge_requests/new?
merge_request%5Bsource_branch%5D=cr1
remote:
To git.lab.example.com:student/compreview.git
 * [new branch]      cr1 -> cr1
Branch 'cr1' set up to track remote branch 'cr1' from 'origin'.
```

10. Close the `/home/student/git-repos/compreview` directory in VS Code. Return to the `/home/student` directory in the GNOME terminal.

10.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/compreview` directory.

10.2. Run the `cd` command in the GNOME terminal to return to the student home directory:

```
[student@workstation compreview]$ cd
```

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 compreview-cr1
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr1
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Creating Playbooks

Create a playbook that includes task files and uses loops, conditional statements, and filters to configure managed nodes.

Outcomes

- Install collections from a `requirements.yml` file.
- Gather and display network resource information using the `resource_manager` role from the `network.base` collection.
- Use loops and variable files to configure interfaces for each network platform (IOS and Junos), based on conditional statements.
- Run a platform-independent task to verify managed node configurations.

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 compreview-cr2
```

Specifications

Important

All files in this exercise must be stored on the `workstation` machine in the `/home/student/git-repos/compreview` directory, or a subdirectory thereof.

The `/home/student/git-repos/compreview` directory contains an Ansible project managed by Git. The Git project is cloned to that directory when you run the `lab start` command.

Create a branch named `cr2`, commit all your finished work to that branch, and then push it to the remote repository at the end of this activity.

- Configure the Ansible project to search for Ansible Content Collections in the project `collections` directory and the `/usr/share/ansible/collections` directory.
- Configure the Ansible project so that the `ansible-galaxy` command retrieves collections from the classroom's private automation hub, enabling access to the `rh-certified` and `validated` repositories.

  You can access the classroom's private automation hub at `https://hub.lab.example.com`. Log in as `student` using `redhat123` as the password.
- Create a `requirements.yml` file in the project `collections` directory to install the following collections:

Collection name	Version	Repository
`network.base`	`2.0.0`	Validated
`ansible.netcommon`	`4.1.0`	Red Hat Certified
`ansible.utils`	`2.7.0`	Red Hat Certified

For each collection requirement, specify the source URL for the repository that contains the collection.

- The source URL for Red Hat Ansible Certified Content is `https://hub.lab.example.com/api/galaxy/content/rh-certified/`.
- The source URL for Ansible validated content is `https://hub.lab.example.com/api/galaxy/content/validated/`. When done, install the required collections.
- Create the following task files in the `/home/student/git-repos/compreview/tasks` directory to gather the `ansible_facts['network_resources']['interfaces']` fact for IOS and Junos managed nodes respectively:

  `- cisco.ios.ios.yml`

  `- junipernetworks.junos.junos.yml`

  Include a task in the `junipernetworks.junos.junos.yml` file to enable the NETCONF service on the Junos managed nodes.

  Use the `to_nice_yaml` filter to format the output file contents, and then store the facts in a file with the name of the managed node and the `.txt` extension.

- Create the following task files in the `/home/student/git-repos/compreview/tasks` directory to configure interface descriptions for IOS and Junos managed nodes respectively:

  `- cisco.ios.ios_interface.yml`

  `- junipernetworks.junos.junos_interface.yml`

  Use loops to configure interface descriptions for each network platform. The `group_vars/ios/interfaces.yml` and `group_vars/junos/interfaces.yml` files contain the interface description configurations to be deployed on the managed nodes.

- Create a playbook named `config_interfaces.yml`, with a single play, in the `/home/student/git-repos/compreview` project directory that targets the `ios` and `junos` inventory groups of managed nodes, and disables fact gathering for them.

  The playbook must include the following task files, which you created in the `tasks` directory, that gather facts and configure interface descriptions for the IOS and Junos network platforms:

  `- cisco.ios.ios.yml`

  `- junipernetworks.junos.junos.yml`

  `- cisco.ios.ios_interface.yml`

  `- junipernetworks.junos.junos_interface.yml`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

- Add a task to the `config_interfaces.yml` playbook that uses the `resource_manager` role from the `network.base` collection to gather `interfaces` network resources from managed nodes in the `ios` and `junos` inventory groups.

  This task must be the first task in the play.

> **Important**
> The correct variable for the `resource_manager` role in the `network.base` collection in this course is the `action` variable. You might see documentation that shows a different variable. Use the `action` variable.

- Add a task to the `config_interfaces.yml` playbook that uses the `ansible.netcommon.cli_command` module to verify that the changes were correctly applied.

  This task must be the last task in the play.

  Create the `group_vars/ios/verify_config.yml` and `group_vars/junos/verify_config.yml` variable files to hold the verification commands for each network platform in the `show_interfaces` variable. The verification commands are listed in the following table:

Managed node	Verification command
`iosxe1.lab.example.com`	`show interface description`
`iosxe2.lab.example.com`	`show interface description`
`junos1.lab.example.com`	`show interfaces descriptions`
`junos2.lab.example.com`	`show interfaces descriptions`

- Run the `config_interfaces.yml` playbook and inspect the output to verify that the interfaces have been correctly configured.

- Commit and push all the changes that you made in the `/home/student/git-repos/compreview` directory to the remote Git repository. Use `Gather facts and configure interface descriptions` as the commit message.

> **Important**
> Make sure that you are in the `/home/student` directory before grading your work with the `lab grade` command.

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 compreview-cr2
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr2
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Creating Playbooks

Create a playbook that includes task files and uses loops, conditional statements, and filters to configure managed nodes.

Outcomes

- Install collections from a `requirements.yml` file.
- Gather and display network resource information using the `resource_manager` role from the `network.base` collection.
- Use loops and variable files to configure interfaces for each network platform (IOS and Junos), based on conditional statements.
- Run a platform-independent task to verify managed node configurations.

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 compreview-cr2
```

1. Change to the `/home/student/git-repos/compreview` Git repository directory and then create a branch named `cr2` for this exercise.

   1.1. Change to the `/home/student/git-repos/compreview` Git repository directory:

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

   1.2. Create a branch named `cr2` and check it out:

```
[student@workstation compreview]$ git checkout -b cr2
Switched to a new branch 'cr2'
```

2. Configure the Ansible project to search for collections in the project `collections` directory and the `/usr/share/ansible/collections` directory.

   2.1. Open VS Code and click **File** > **Open Folder**.

   2.2. Navigate to **git-repos** > **compreview** and click **Open**.

> **Note**
> If prompted, select **Trust the authors of all files in the parent folder 'student'**, and then click **Yes, I trust the authors**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

2.3. Create the `/home/student/git-repos/compreview/ansible.cfg` file and set the `collections_paths` variable:

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

3. Configure the Ansible project so that the `ansible-galaxy` command retrieves collections from the classroom's private automation hub, enabling access to the `rh-certified` and `validated` repositories.

You can access the classroom's private automation hub at `https://hub.lab.example.com`. Log in as `student` using `redhat123` as the password.

3.1. 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.

3.2. 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.

3.3. Update the `ansible.cfg` file and add the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file contains the following content:

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

[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>
```

3.4. 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 **validated** row. Do not close your web browser window.

3.5. Update the `ansible.cfg` file to append the lines that you copied to the clipboard in the preceding step. The updated `ansible.cfg` file now contains the following content:

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

[galaxy]
server_list = rh-certified_repo
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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

[galaxy]
server_list = validated_repo

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

3.6. Update the `ansible.cfg` file so that it only contains one `[galaxy]` section. Update the first `[galaxy]` section so that it lists both the `rh-certified_repo` and the `validated_repo` repositories. Remove the second `[galaxy]` section. The updated `ansible.cfg` file now contains the following content:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

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

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

3.7. Return to the private automation hub web UI. Navigate to **Collections** > **API token management** and then click **Load token**. Copy the API token.

> **Important**
> Loading a new token invalidates all your previous tokens.

3.8. Using the copied token, update both token lines in the `ansible.cfg` file. Your token is different from the token displayed in this example. Save and close the file when done:

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

[galaxy]
server_list = rh-certified_repo, validated_repo

[galaxy_server.rh-certified_repo]
url=https://hub.lab.example.com/api/galaxy/content/rh-certified/
token=6ab90d6835a930fa6cb298aa0d03d11a60735577

[galaxy_server.validated_repo]
url=https://hub.lab.example.com/api/galaxy/content/validated/
token=6ab90d6835a930fa6cb298aa0d03d11a60735577
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4. Create a `requirements.yml` file in the project `collections` directory to install the following collections:

Collection name	Version	Repository
`network.base`	`2.0.0`	Validated
`ansible.netcommon`	`4.1.0`	Red Hat Certified
`ansible.utils`	`2.7.0`	Red Hat Certified

For each collection requirement, specify the source URL for the repository that contains the collection, such as the `https://hub.lab.example.com/api/galaxy/content/rh-certified/` URL. When done, install the required collections.

4.1. Switch to the **Terminal** tab in VS Code, or return to the GNOME terminal. Use the `mkdir` command to create the `collections` directory:

```
[student@workstation compreview]$ mkdir collections
```

4.2. Create the `/home/student/git-repos/compreview/collections/requirements.yml` file. Add the following content to specify the required collections:

```
---
collections:
  - name: network.base
    version: 2.0.0
    source: https://hub.lab.example.com/api/galaxy/content/validated/
  - name: ansible.netcommon
    version: 4.1.0
    source: https://hub.lab.example.com/api/galaxy/content/rh-certified/
  - name: ansible.utils
    version: 2.7.0
    source: https://hub.lab.example.com/api/galaxy/content/rh-certified/
```

4.3. In the `/home/student/git-repos/compreview` directory, use the `ansible-galaxy` command to install the required collections. Add the `-r` option to specify the location of the `collections/requirements.yml` file:

```
[student@workstation compreview]$ ansible-galaxy collection install \
-r collections/requirements.yml
...output omitted...
network.base:2.0.0 was installed successfully
...output omitted...
ansible.netcommon:4.1.0 was installed successfully
...output omitted...
ansible.utils:2.7.0 was installed successfully
```

4.4. Use the `ansible-galaxy` command to list the installed collections:

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

# /home/student/git-repos/compreview/collections/ansible_collections
```

```
Collection        Version
----------------- -------
ansible.netcommon 4.1.0
ansible.utils     2.7.0
network.base      2.0.0

# /usr/share/ansible/collections/ansible_collections
Collection               Version
------------------------ -------
redhat.rhel_system_roles 1.20.1
```

Important

The output in the `/usr/share/ansible/collections/ansible_collections` section displays collections installed on the local system, which is different from collections installed in an automation execution environment.

To see collections available to automation content navigator from the project directory, run the following command:

```
[student@workstation compreview]$ ansible-navigator collections \
-m interactive
```

5. Create the following task files in the `/home/student/git-repos/compreview/tasks` directory to gather the `ansible_facts['network_resources']['interfaces']` fact for IOS and Junos managed nodes, respectively:

   - `cisco.ios.ios.yml`
   - `junipernetworks.junos.junos.yml`

   Include a task in the `junipernetworks.junos.junos.yml` file to enable the NETCONF service on the Junos managed nodes.

   Use the `to_nice_yaml` filter to format the output file contents, and then store the facts in a file with the name of the managed node and the `.txt` extension.

   5.1. Switch to the **Terminal** tab in VS Code, or return to the GNOME terminal. Use the `mkdir` command to create the `tasks` directory:

```
[student@workstation compreview]$ mkdir tasks
```

   5.2. Create the `tasks/cisco.ios.ios.yml` task file with the following content:

```
---
- name: Gather IOS facts
  cisco.ios.ios_facts:
    gather_network_resources:
      - interfaces

- name: Save facts to file
  ansible.builtin.copy:
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    content: "{{ ansible_facts | to_nice_yaml }}"
    dest: "{{ inventory_hostname }}.txt"
    mode: "0644"
```

> **Note**
> The `mode` parameter sets the OS-level permissions of the destination file. This parameter is optional, but omitting the parameter results in a lint warning.

5.3. Create the `tasks/junipernetworks.junos.junos.yml` task file with the following content:

```
---
- name: Enable the NETCONF service
  vars:
    ansible_connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
    netconf_port: 830
    state: present

- name: Gather Junos facts
  junipernetworks.junos.junos_facts:
    gather_network_resources:
      - interfaces

- name: Save facts to file
  ansible.builtin.copy:
    content: "{{ ansible_facts | to_nice_yaml }}"
    dest: "{{ inventory_hostname }}.txt"
    mode: "0644"
```

6. Create the following task files in the `/home/student/git-repos/compreview/tasks` directory to configure interface descriptions for IOS and Junos managed nodes, respectively:

- `cisco.ios.ios_interface.yml`
- `junipernetworks.junos.junos_interface.yml`

Use loops to configure interface descriptions for each network platform. The `group_vars/ios/interfaces.yml` and `group_vars/junos/interfaces.yml` files contain the interface description configurations to be deployed on the managed nodes.

6.1. In VS Code, examine the `group_vars/ios/interfaces.yml` file.

This file contains the interface description configurations to be deployed on IOS managed nodes:

```
---
interfaces:
  iosxe1.lab.example.com:
    - name: Loopback0
      description: Managed by Ansible

    - name: GigabitEthernet2
      description: Managed by Ansible
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  iosxe2.lab.example.com:
    - name: Loopback0
      description: Managed by Ansible

    - name: GigabitEthernet2
      description: Managed by Ansible
```

6.2. In VS Code, examine the `group_vars/junos/interfaces.yml` file.

This file contains the interface configurations to be deployed on Junos managed nodes.

```
---
interfaces:
  junos1.lab.example.com:
    - name: em1
      unit_name: 0
      unit_description: Managed by Ansible

    - name: fxp0
      unit_name: 0
      unit_description: Managed by Ansible

  junos2.lab.example.com:
    - name: em1
      unit_name: 0
      unit_description: Managed by Ansible

    - name: fxp0
      unit_name: 0
      unit_description: Managed by Ansible
```

6.3. Create the `tasks/cisco.ios.ios_interface.yml` file to configure interface descriptions for IOS managed nodes.

The task file must consist of the following content:

```
---
- name: Configure interface descriptions on IOS
  cisco.ios.ios_interfaces:
    config:
      - name: "{{ item['name'] }}"
        description: "{{ item['description'] }}"
    state: merged
  loop: "{{ interfaces[inventory_hostname] }}"
```

6.4. Create the `tasks/junipernetworks.junos.junos_interface.yml` file to configure interface descriptions for Junos managed nodes.

The task file must consist of the following content:

```
---
- name: Configure interface descriptions on Junos
  junipernetworks.junos.junos_interfaces:
```


For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    config:
      - name: "{{ item['name'] }}"
        units:
          - name: "{{ item['unit_name'] }}"
            description: "{{ item['unit_description'] }}"
    state: merged
  loop: "{{ interfaces[inventory_hostname] }}"
```

7. Create a playbook named `config_interfaces.yml` in the `/home/student/git-repos/compreview` project directory that targets the `ios` and `junos` inventory groups, and disables fact gathering for these managed nodes. The playbook must include the tasks under the `tasks` directory that gather facts and configure interface descriptions for IOS and Junos managed nodes.

   7.1. In VS Code, create the `config_interfaces.yml` playbook. Target the `ios` and `junos` inventory groups. Disable fact gathering.

```
---
- name: Gather network resource information
  hosts:
    - ios
    - junos
  gather_facts: false
```

   7.2. Add a first task to the `config_interfaces.yml` playbook that includes the `tasks/{{ ansible_network_os }}.yml` file:

```
---
- name: Gather network resource information
  hosts:
    - ios
    - junos
  gather_facts: false
  tasks:
    - name: Facts task
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}.yml
```

Note

Notice that by using the `{{ ansible_network_os }}` variable as part of the task file name, on Cisco IOS managed nodes the task included is `cisco.ios.ios.yml` and on Juniper Junos managed nodes the task included is `junipernetworks.junos.junos.yml`.

Another way to include the tasks is by using conditionals.

   7.3. Add a second task to the `config_interfaces.yml` playbook that includes the `tasks/{{ ansible_network_os }}_interface.yml` file:

```
---
- name: Gather network resource information
  hosts:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - ios
    - junos
  gather_facts: false
  tasks:
    - name: Facts task
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}.yml

    - name: Configure interface descriptions
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}_interface.yml
```

8. Add a task to the `config_interfaces.yml` playbook that uses the `resource_manager` role from the `network.base` collection to gather `interfaces` network resources from managed nodes in the `ios` and `junos` inventory groups.

> **Important**
> The `resource_manager` role in the `network.base` collection uses a variable to determine the type of actions that the role should perform. The name of this variable changes depending on which version of the collection that you use. The correct variable for the `resource_manager` role in this course is the `action` variable.

8.1. From the private automation hub web UI, navigate to **Collections** > **Collections**.

8.2. In the **Filter by repository** list, choose **Validated**.

8.3. Click the **base** collection and go to its **Documentation** tab to review the documentation.

8.4. Edit the `config_interfaces.yml` playbook in VS Code. Add a third task to the playbook that uses the `resource_manager` role from the `network.base` collection to gather `interfaces` network resources from managed nodes in the `ios` and `junos` inventory groups:

```
---
- name: Gather network resource information
  hosts:
    - ios
    - junos
  gather_facts: false
  tasks:
    - name: Facts task
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}.yml

    - name: Configure interface descriptions
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}_interface.yml

    - name: Network Resource Manager
      ansible.builtin.include_role:
        name: network.base.resource_manager
      vars:
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        action: gather
        resources:
          - 'interfaces'
```

9. Add a task to the `config_interfaces.yml` playbook that uses the `ansible.netcommon.cli_command` module to verify that the changes were correctly applied.

   Create the `group_vars/ios/verify_config.yml` and `group_vars/junos/verify_config.yml` variable files to hold the verification commands for each network platform in the `show_interfaces` variable. The verification commands are listed in the following table:

Managed node	Verification command
`iosxe1.lab.example.com`	`show interface description`
`iosxe2.lab.example.com`	`show interface description`
`junos1.lab.example.com`	`show interfaces descriptions`
`junos2.lab.example.com`	`show interfaces descriptions`

   9.1. Edit the playbook to use the `ansible.netcommon.cli_command` module to verify that the changes were correctly applied:

```
---
- name: Gather network resource information
  hosts:
    - ios
    - junos
  gather_facts: false
  tasks:
    - name: Facts task
      ansible.builtin.include_tasks:
        file: "tasks/{{ ansible_network_os }}.yml"

    - name: Configure interface descriptions
      ansible.builtin.include_tasks:
        file: tasks/{{ ansible_network_os }}_interface.yml

    - name: Network Resource Manager
      ansible.builtin.include_role:
        name: network.base.resource_manager
      vars:
        action: gather
        resources:
          - 'interfaces'

    - name: Run a command to verify interface configurations
      vars:
        ansible_connection: ansible.netcommon.network_cli
      ansible.netcommon.cli_command:
        command: "{{ show_interfaces }}"
      register: result
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
    - name: Display the results
      ansible.builtin.debug:
        var: result['stdout_lines']
```

9.2. Create a variable file named `group_vars/ios/verify_config.yml` that holds the verification commands for IOS managed nodes.

The variable file must consist of the following content:

```
---
show_interfaces: "show interface description"
```

9.3. Create a variable file named `group_vars/junos/verify_config.yml` that holds the verification commands for IOS managed nodes.

The variable file must consist of the following content:

```
---
show_interfaces: "show interfaces descriptions"
```

10. Run the `config_interfaces.yml` playbook again and inspect the output to verify that the interfaces have been correctly configured:

```
[student@workstation compreview]$ ansible-navigator run config_interfaces.yml
...output omitted...

TASK [Display the results] *****************************************************
ok: [junos1.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface       Admin Link Description",
        "em1.0           up    up   Managed by Ansible",
        "fxp0.0          up    up   Managed by Ansible"
    ]
}
ok: [junos2.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface       Admin Link Description",
        "em1.0           up    up   Managed by Ansible",
        "fxp0.0          up    up   Managed by Ansible"
    ]
}
ok: [iosxe2.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface                     Status       Protocol Description",
        "Gi1                           up           up       ",
        "Gi2                           up           up       Managed by Ansible",
        "Lo0                           up           up       Managed by Ansible"
    ]
}
ok: [iosxe1.lab.example.com] => {
    "result['stdout_lines']": [
        "Interface                     Status       Protocol Description",
        "Gi1                           up           up       ",
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
        "Gi2                          up           up       Managed by Ansible",
        "Lo0                          up           up       Managed by Ansible"
    ]
}
...output omitted...
```

11. Commit and push all the changes that you made in the `/home/student/git-repos/compreview` directory to the remote Git repository. Use `Gather facts and configure interface descriptions` as the commit message.

11.1. Use VS Code or the `git add` command to add the new and updated files and directories:

```
[student@workstation compreview]$ git add .
```

11.2. Use VS Code or the `git status` command to display the list of files that you staged for the next commit:

```
[student@workstation compreview]$ git status
On branch cr2
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
...output omitted...
```

11.3. Use VS Code or the `git commit` command to commit the files. Use `Gather facts and configure interface descriptions` as the commit message:

```
[student@workstation compreview]$ git commit -m \
"Gather facts and configure interface descriptions"
[cr2 46a0ba4] Gather facts and configure interface descriptions
...output omitted...
```

11.4. Use VS Code or the `git push` command to update the remote Git repository:

```
[student@workstation compreview]$ git push -u origin cr2
...output omitted...
To git.lab.example.com:student/compreview
 * [new branch]      cr2 -> cr2
Branch 'cr2' set up to track remote branch 'cr2' from 'origin'.
```

12. Close the `/home/student/git-repos/compreview` directory in VS Code.

Return to the `/home/student` directory in the GNOME terminal.

12.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/compreview` directory.

12.2. Run the `cd` command in the GNOME terminal to return to the student home directory:

```
[student@workstation compreview]$ cd
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

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 compreview-cr2
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr2
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Scaling Playbooks

Create a role to back up managed network nodes, and then use the backups to help create Jinja2 templates to generate configuration files.

Outcomes

- Use Git to create a branch, commit changes, and push changes to a branch.
- Create and use a role that backs up managed nodes.
- Create a playbook that generates configuration files from Jinja2 templates.
- Configure NTP, DNS, and Syslog on managed nodes by using generated configuration files.

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 compreview-cr3
```

Specifications

This exercise creates a role to back up managed nodes. The backups are then converted to Jinja2 templates, which are used to generate new configuration files to configure NTP, DNS, and Syslog on the managed nodes.

Important

All files in this exercise must be stored on the `workstation` machine in the `/home/student/git-repos/compreview` directory, or a subdirectory thereof.

The `/home/student/git-repos/compreview` directory contains an Ansible project managed by Git. The `lab start` command cloned a Git project to that directory. Create a branch named `cr3`, commit all your finished work to that branch, and at the end of this activity, push that branch to the remote repository.

- Create a role named `net_backup` in the `/home/student/git-repos/compreview/roles` directory.
- Define the following variables in the `defaults/main.yml` file for the `net_backup` role:

Variable name	Variable value
`backup_tmp`	`/tmp`
`backup_store`	`/backups`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Variable name	Variable value
`delegate_host`	`workstation.lab.example.com`

- Create the `tasks/junos.yml` file for the `net_backup` role. The file must accomplish the following objectives:
  - Enable the `NETCONF` service.
  - Create backups of the configurations of the Junos-based managed nodes. Use the `backup_tmp` variable for the `dir_path` parameter and use `"{{ inventory_hostname }}.cfg"` as the value for the `filename` parameter.
- Create the `tasks/ios.yml` file for the `net_backup` role. The file must accomplish the following objective:
  - Create backups of the configurations of the IOS-based managed nodes. Use the `backup_tmp` variable for the `dir_path` parameter and use `"{{ inventory_hostname }}.cfg"` as the value for the `filename` parameter.
- Create the `tasks/store_backups.yml` file for the `net_backup` role. The file must accomplish the following objectives:
  - Delegate all tasks in the file to the host defined by the `delegate_host` variable.
  - Use the `ansible.builtin.file` module to create the directory defined by the `backup_store` variable if it does not already exist on the delegated host.
  - Copy the configuration backup files from the directory defined by the `backup_tmp` variable to the directory defined by the `backup_store` variable on the delegated host.
  - Run all tasks with privilege escalation enabled.
- Update the `tasks/main.yml` file for the `net_backup` role. The file must accomplish the following objectives:
  - Include the `junos.yml` task file for managed nodes that define the `junipernetworks.junos.junos` value for the `ansible_network_os` variable.
  - Include the `ios.yml` task file for managed nodes that define the `cisco.ios.ios` value for the `ansible_network_os` variable.
  - Include the `store_backups.yml` task file.
- Create and run a playbook named `net_backup.yml` that targets the managed nodes in the `junos` and `ios` inventory groups, does not gather facts, and includes the `net_backup` role.
- Modify Jinja2 templates to add NTP, DNS, and Syslog servers.
  - The `junos.j2` and `ios.j2` templates are stored in the `/home/student/git-repos/compreview/templates` directory.
  - Update the templates to add the additional NTP, DNS, and Syslog information using the variables in the `/home/student/git-repos/compreview/group_vars/all` directory.
- Create and run the `generate_config.yml` playbook.
  - The play should target the `workstation.lab.example.com` host, be run using privilege escalation, and should become the `student` user.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

  - Use the `ansible.builtin.template` module to generate new configuration files.
  - Add a task that uses the `templates/junos.j2` template to create a configuration file for each host in the `junos` group.
  - Add a task that uses the `templates/ios.j2` template to create a configuration file for each host in the `ios` group.
  - Save the filenames for the configuration files in the format of *`hostname`*`.cfg`, such as `iosxe1.lab.example.com.cfg`, and save the configuration files in the project `configs` directory.
  - Use the `playbook_dir` magic variable to help specify a path to the project `configs` directory.

- Inspect the newly generated configuration files to verify that the correct values were used for the NTP, DNS, and Syslog information from the variables in the `/home/student/git-repos/compreview/group_vars` directory.
- Create and run the `deploy.yml` playbook to deploy the newly generated configuration files to the appropriate managed node. The playbook should contain one play that targets the `ios` and `junos` inventory groups. Run the `show_run.yml` playbook to verify that the configuration changes were made.
- Commit and push all the changes that you made in the `/home/student/git-repos/compreview` directory to the remote Git repository. Use `Generate and apply configurations` as the commit message.

> **Important**
> Make sure that you are in the `/home/student` directory before grading your work with the `lab grade` command.

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 compreview-cr3
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr3
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Scaling Playbooks

Create a role to back up managed network nodes, and then use the backups to help create Jinja2 templates to generate configuration files.

Outcomes

- Use Git to create a branch, commit changes, and push changes to a branch.
- Create and use a role that backs up managed nodes.
- Create a playbook that generates configuration files from Jinja2 templates.
- Configure NTP, DNS, and Syslog on managed nodes by using generated configuration files.

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 compreview-cr3
```

1. Change to the `/home/student/git-repos/compreview` directory and then create a branch named `cr3` for this exercise.

   1.1. Change to the `/home/student/git-repos/compreview` directory:

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

   1.2. Create a branch named `cr3` and check it out:

```
[student@workstation compreview]$ git checkout -b cr3
Switched to a new branch 'cr3'
```

2. Create the `net_backup` role in the `/home/student/git-repos/compreview/roles` directory.

   2.1. Create the `roles` directory and create the initial role structure and files:

```
[student@workstation compreview]$ mkdir roles; cd roles
[student@workstation roles]$ ansible-galaxy init net_backup
- Role net_backup was created successfully
```

   2.2. Set the `backup_tmp` variable to `/tmp` in the `defaults/main.yml` file. Add the following line to the `net_backup/defaults/main.yml` file:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
---
# defaults file for net_backup
backup_tmp: /tmp
```

2.3. Set the `backup_store` variable to `/backups` in the `defaults/main.yml` file. Add the following line to the `net_backup/defaults/main.yml` file:

```
---
# defaults file for net_backup
backup_tmp: /tmp
backup_store: /backups
```

2.4. Set the `delegate_host` variable to `workstation.lab.example.com` in the `defaults/main.yml` file. Add the following line to the `net_backup/defaults/main.yml` file:

```
---
# defaults file for net_backup
backup_tmp: /tmp
backup_store: /backups
delegate_host: workstation.lab.example.com
```

2.5. Create the `net_backup/tasks/junos.yml` task file to enable the `NETCONF` service and create backups of the configurations from the Junos managed nodes. Use the `backup_tmp` variable for the `dir_path` parameter and use `"{{ inventory_hostname }}.cfg"` as the value for the `filename` parameter. Create the `net_backup/tasks/junos.yml` file with the following content:

```
---
- name: Enable the netconf service on port 830
  vars:
    ansible_connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
    netconf_port: 830
    state: present

- name: Back up Junos configurations
  junipernetworks.junos.junos_config:
    backup: true
    backup_options:
      dir_path: "{{ backup_tmp }}"
      filename: "{{ inventory_hostname }}.cfg"
```

2.6. Create the `net_backup/tasks/ios.yml` task file to create backups of the configurations from the IOS managed nodes. Use the `backup_tmp` variable for the `dir_path` parameter and use `"{{ inventory_hostname }}.cfg"` as the value for the `filename` parameter. Create the `net_backup/tasks/ios.yml` file with the following content:

```
---
- name: Back up IOS XE configurations
  cisco.ios.ios_config:
    backup: true
    backup_options:
      dir_path: "{{ backup_tmp }}"
      filename: "{{ inventory_hostname }}.cfg"
```

2.7. Create the `net_backup/tasks/store_backups.yml` task file to copy the configuration backup files from the directory defined by the `backup_tmp` variable to the directory defined by the `backup_store` directory on the delegated host.

Delegate all tasks in the file to the host defined by the `delegate_host` variable and enable privilege escalation for all tasks.

Use the `ansible.builtin.file` module to create the directory defined by the `backup_store` variable if it does not already exist on the delegated host, and then copy the configuration backup files.

Create the `net_backup/tasks/store_backups.yml` file with the following content:

```
---
- name: Create the backup store directory
  ansible.builtin.file:
    path: "{{ backup_store }}"
    state: directory
    recurse: true
  become: true
  delegate_to: "{{ delegate_host }}"

- name: Store the latest backup files
  ansible.builtin.copy:
    src: "{{ backup_tmp }}/{{ inventory_hostname }}.cfg"
    dest: "{{ backup_store }}"
    mode: "0664"
  become: true
  delegate_to: "{{ delegate_host }}"
```

Note

The `mode` parameter sets the operating system level permissions of the destination file. This parameter is optional, but omitting the parameter results in a lint warning.

2.8. Update the `net_backup/tasks/main.yml` task file to include the tasks in the `junos.yml`, `ios.yml`, and `store_backups.yml` task files. The tasks in the `junos.yml` task file should only run on managed nodes running the `junipernetworks.junos.junos` network OS. The tasks in the `ios.yml` task file should only run on managed nodes running the `cisco.ios.ios` network OS. Update the `net_backup/tasks/main.yml` file with the following content:

```
---
# tasks file for net_backup
- name: Back up Junos network device configs
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  when: ansible_network_os == 'junipernetworks.junos.junos'
  ansible.builtin.include_tasks:
    file: junos.yml

- name: Back up IOS network device configs
  when: ansible_network_os == 'cisco.ios.ios'
  ansible.builtin.include_tasks:
    file: ios.yml

- name: Persistently store backup files
  ansible.builtin.include_tasks:
    file: store_backups.yml
```

2.9. *(Optional)* Delete the files and directories that the role does not use:

```
[student@workstation roles]$ cd net_backup
[student@workstation net_backup]$ rm -rf README.md files handlers meta \
templates tests vars
[student@workstation net_backup]$ tree
.
├── defaults
│   └── main.yml
└── tasks
    ├── ios.yml
    ├── junos.yml
    ├── main.yml
    └── store_backups.yml

2 directories, 5 files
```

2.10. Return to the `/home/student/git-repos/compreview` directory:

```
[student@workstation net_backup]$ cd ~/git-repos/compreview
```

3. Create and run the `net_backup.yml` playbook in the `/home/student/git-repos/compreview` directory. The playbook targets the managed nodes in the `junos` and `ios` inventory groups, and includes the `net_backup` role.

3.1. Create the `net_backup.yml` playbook with the following content:

```
---
- name: Back up managed network node configurations
  hosts:
    - junos
    - ios
  gather_facts: false
  tasks:
    - name: Back up network managed node configurations
      ansible.builtin.include_role:
        name: net_backup
```

3.2. Run the `net_backup.yml` playbook:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation compreview]$ ansible-navigator run net_backup.yml
...output omitted...
```

4. Modify the Jinja2 templates to add NTP, DNS, and Syslog servers. The `junos.j2` and `ios.j2` templates are stored in the `/home/student/git-repos/compreview/templates` directory. Update the templates to add the additional NTP, DNS, and Syslog information with the variables in the `/home/student/git-repos/compreview/group_vars/all` directory.

4.1. Inspect the `ntp_servers` variable in the `group_vars/all/ntp_servers.yml` file:

```
---
ntp_servers:
  - 172.25.23.240
  - 172.25.23.230
```

4.2. Inspect the `nameservers` variable in the `group_vars/all/nameservers.yml` file:

```
---
nameservers:
  - 172.25.250.220
  - 172.25.240.220
```

4.3. Inspect the `syslog_servers` variable in the `group_vars/all/syslog_servers.yml` file:

```
---
syslog_servers:
  - name: syslog01
    address: 192.168.10.10
  - name: syslog02
    address: 192.168.11.10
```

4.4. Inspect the `templates/junos.j2` file. Note the format used by the Juniper Junos configuration file to configure the NTP, DNS, and Syslog server IP addresses:

```
{# Edit this template to perform the actions stated in the comments #}
{# loop through each name server #}
set system name-server 1.2.3.4
set system name-server 5.6.7.8
{# loop through each syslog server #}
set system syslog host host1 source-address 1.2.3.4
set system syslog host host2 source-address 5.6.7.8
{# loop through each ntp server #}
set system ntp server 1.2.3.4
set system ntp server 5.6.7.8
```

4.5. Inspect the `templates/ios.j2` file. Note the format used by the Cisco IOS configuration file to configure the NTP, DNS, and Syslog server IP addresses:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
{# Edit this template to perform the actions stated in the comments #}
{# Add both nameservers from variable #}
ip name-server {{ nameservers[0] }} {{ nameservers[1] }}
{# loop through each syslog server #}
logging host 1.2.3.4
logging host 5.6.7.8
{# loop through each ntp server #}
ntp server 1.2.3.4 version 2
ntp server 5.6.7.8 version 2
```

4.6. Modify the `templates/junos.j2` template file. Replace the NTP, DNS, and Syslog server sections under the comments using `for` statements to loop through the variables in the `group_vars/all` directory. The final Jinja2 template must consist of the following content:

```
{# Edit this template to perform the actions stated in the comments #}
{# loop through each name server #}
{% for NAMESERVER in nameservers %}
set system name-server {{ NAMESERVER }}
{% endfor %}
{# loop through each syslog server #}
{% for SYSLOG in syslog_servers %}
set system syslog host {{ SYSLOG['name'] }} source-address {{ SYSLOG['address'] }}
{% endfor %}
{# loop through each ntp server #}
{% for NTP in ntp_servers %}
set system ntp server {{ NTP }}
{% endfor %}
```

4.7. Modify the `templates/ios.j2` template file. Replace the Syslog and NTP server sections under the comments using `for` statements to loop through the variables in the `group_vars/all` directory. The final Jinja2 template must consist of the following content:

```
{# Edit this template to perform the actions stated in the comments #}
{# Add both nameservers from variable #}
ip name-server {{ nameservers[0] }} {{ nameservers[1] }}
{# loop through each syslog server #}
{% for SYSLOG in syslog_servers %}
logging host {{ SYSLOG['address'] }}
{% endfor %}
{# loop through each ntp server #}
{% for NTP in ntp_servers %}
ntp server {{ NTP }} version 2
{% endfor %}
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Note

The DNS servers have already been configured to use the correct variables in this file by using one method to place both name servers on the same line. Another option is to use a `for` statement inline as shown in the following example:

```
{# Add both nameservers from variable #}
ip name-server{% for SERVER in nameservers %} {{ SERVER }}{% endfor %}

{# loop through each syslog server #}
```

Note that there is an extra blank line in this example. Without this extra blank line, the first logging host is placed on the same line as the `ip name-server` line.

5. Create and run the `generate_config.yml` playbook, which uses the `ansible.builtin.template` module to generate new configuration files. Save the filenames for the configuration files in the format of *`hostname`*`.cfg`, such as `iosxe1.lab.example.com.cfg`, and save the configuration files in the project `configs` directory.

   5.1. Begin the `generate_config.yml` playbook by targeting the `workstation.lab.example.com` host and setting the `config_dir` and `template_dir` variables:

```
---
- name: Generate configuration files from templates
  hosts: workstation.lab.example.com
  become: true
  become_user: student
  gather_facts: false
  vars:
    config_dir: "{{ playbook_dir }}/configs"
  tasks:
```

   5.2. Create a task that generates the configuration files for the managed Juniper nodes:

```
    - name: Generate Juniper configuration files
      ansible.builtin.template:
        src: templates/junos.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
        mode: "0664"
      loop: "{{ groups['junos'] }}"
```

   5.3. Create a second task that generates the configuration files for the managed Cisco nodes. The completed `generate_config.yml` playbook must consist of the following content:

```
---
- name: Generate configuration files from templates
  hosts: workstation.lab.example.com
  become: true
  become_user: student
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
  gather_facts: false
  vars:
    config_dir: "{{ playbook_dir }}/configs"
  tasks:
    - name: Generate Juniper configuration files
      ansible.builtin.template:
        src: templates/junos.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
        mode: "0664"
      loop: "{{ groups['junos'] }}"

    - name: Generate Cisco configuration files
      ansible.builtin.template:
        src: templates/ios.j2
        dest: "{{ config_dir }}/{{ item }}.cfg"
        mode: "0664"
      loop: "{{ groups['ios'] }}"
```

5.4. Run the `generate_config.yml` playbook:

```
[student@workstation compreview]$ ansible-navigator run generate_config.yml
...output omitted...
```

6. Inspect the newly generated configuration files to verify that the correct values were used for the NTP, DNS, and Syslog information from the variables in the `/home/student/git-repos/compreview/group_vars` directory.

6.1. Inspect the `configs/junos1.lab.example.com.cfg` file. Note that the IP address values match the values in the `/home/student/git-repos/compreview/group_vars` directory:

```
set system name-server 172.25.250.220
set system name-server 172.25.240.220
set system syslog host syslog01 source-address 192.168.10.10
set system syslog host syslog02 source-address 192.168.11.10
set system ntp server 172.25.23.240
set system ntp server 172.25.23.230
```

6.2. Inspect the `configs/iosxe1.lab.example.com.cfg` file. Note that the IP address values match the values in the `/home/student/git-repos/compreview/group_vars` directory:

```
ip name-server 172.25.250.220 172.25.240.220
logging host 192.168.10.10
logging host 192.168.11.10
ntp server 172.25.23.240 version 2
ntp server 172.25.23.230 version 2
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important

The `ip name-server` line should have a space before the first IP address, a space between the IP addresses, and this line should not include the first `logging host` line.

7. Create and run the `deploy.yml` playbook to deploy the newly generated configuration files to the appropriate managed node.

7.1. Begin the `deploy.yml` playbook by targeting the `junos` and `ios` inventory groups:

```
---
- name: Configure managed nodes
  hosts:
    - junos
    - ios
  gather_facts: false
  tasks:
```

7.2. Add a task to deploy the Junos configurations:

```
    - name: Deploy Junos configuration
      when: ansible_network_os == 'junipernetworks.junos.junos'
      junipernetworks.junos.junos_config:
        src: configs/{{ inventory_hostname }}.cfg
```

7.3. Add a task to deploy the IOS configurations. The completed `deploy.yml` playbook must consist of the following content:

```
---
- name: Configure managed nodes
  hosts:
    - junos
    - ios
  gather_facts: false
  tasks:
    - name: Deploy Junos configuration
      when: ansible_network_os == 'junipernetworks.junos.junos'
      junipernetworks.junos.junos_config:
        src: configs/{{ inventory_hostname }}.cfg

    - name: Deploy IOS configuration
      when: ansible_network_os == 'cisco.ios.ios'
      cisco.ios.ios_config:
        src: configs/{{ inventory_hostname }}.cfg
```

7.4. Run the `deploy.yml` playbook:

```
[student@workstation compreview]$ ansible-navigator run deploy.yml
...output omitted...
```

7.5. Run the `show_run.yml` playbook to verify that the NTP, DNS, and Syslog server configurations have been added to the managed nodes:

```
[student@workstation compreview]$ ansible-navigator run show_run.yml
...output omitted...

TASK [Display configured NTP servers] ****************************************
ok: [junos1.lab.example.com] => {
    "actual_ntp_servers": [
        "1.2.3.4",
        "5.6.7.8",
        "172.25.23.240",
        "172.25.23.230"
    ]
}
...output omitted...
ok: [iosxe1.lab.example.com] => {
    "actual_ntp_servers": [
        "5.6.7.8",
        "172.25.23.240",
        "1.2.3.4",
        "172.25.23.230"
    ]
}
...output omitted...

TASK [Display configured DNS name servers] ***********************************
ok: [junos1.lab.example.com] => {
    "actual_dns_servers": [
        "1.2.3.4",
        "5.6.7.8",
        "172.25.250.220",
        "172.25.240.220"
    ]
}
...output omitted...
ok: [iosxe1.lab.example.com] => {
    "actual_dns_servers": [
        "1.2.3.4",
        "5.6.7.8",
        "172.25.250.220",
        "172.25.240.220"
    ]
}
...output omitted...

TASK [Display configured Syslog servers] *************************************
ok: [junos1.lab.example.com] => {
    "actual_syslog_servers": [
        {
            "name": "host1",
            "source-address": "1.2.3.4"
        },
        {
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
            "name": "host2",
            "source-address": "5.6.7.8"
        },
        {
            "name": "syslog01",
            "source-address": "192.168.10.10"
        },
        {
            "name": "syslog02",
            "source-address": "192.168.11.10"
        }
    ]
}
...output omitted...
ok: [iosxe1.lab.example.com] => {
    "actual_syslog_servers": [
        "1.2.3.4",
        "5.6.7.8",
        "192.168.10.10",
        "192.168.11.10"
    ]
}
...output omitted...
```

8. Commit and push all the changes that you made in the `/home/student/git-repos/compreview` directory to the remote Git repository. Use `Generate and apply configurations` as the commit message.

8.1. Use VS Code or the `git add` command to add the new and updated files and directories:

```
[student@workstation compreview]$ git add .
```

8.2. Use VS Code or the `git status` command to display the list of files that you staged for the next commit:

```
[student@workstation compreview]$ git status
On branch cr3
Your branch is up to date with 'origin/cr3'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   configs/iosxe1.lab.example.com.cfg
        new file:   configs/iosxe2.lab.example.com.cfg
        new file:   configs/junos1.lab.example.com.cfg
        new file:   configs/junos2.lab.example.com.cfg
        new file:   deploy.yml
        new file:   generate_config.yml
        new file:   net_backup.yml
...output omitted...
```

8.3. Use VS Code or the `git commit` command to commit the files. Use `Generate and apply configurations` as the commit message:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

```
[student@workstation compreview]$ git commit -m \
"Generate and apply configurations"
[cr3 5574r79] Generate and apply configurations
...output omitted...
```

8.4. Use VS Code or the `git push` command to update the remote Git repository:

```
[student@workstation compreview]$ git push -u origin cr3
...output omitted...
To git.lab.example.com:student/compreview.git
 * [new branch]      cr3 -> cr3
Branch 'cr3' set up to track remote branch 'cr3' from 'origin'.
```

9. Close the `/home/student/git-repos/compreview` directory in VS Code. Return to the `/home/student` directory in the GNOME terminal.

9.1. Click **File** > **Close Folder** in VS Code to close the `/home/student/git-repos/compreview` directory.

9.2. Run the `cd` command in the GNOME terminal to return to the student home directory:

```
[student@workstation compreview]$ cd
```

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 compreview-cr3
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr3
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Lab

Running Playbooks with Automation Controller

Create resources within the automation controller web UI so that you can launch job templates and workflow job templates.

Outcomes

- Create automation controller source control and machine credentials.
- Create an automation controller project and then create an inventory that uses files stored in the project.
- Create automation controller job templates and workflow templates.
- Launch jobs from the automation controller web UI.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your environment for this exercise, and to ensure that all required resources are available:

```
[student@workstation ~]$ lab start compreview-cr4
```

Specifications

You can log in to your automation controller at `https://controller.lab.example.com` as the `admin` user with `redhat` as the password.

- Create a source control credential resource that automation controller can use to download files from the `git.lab.example.com` Git server. Create the credential with the following settings:

Field	Value
Name	`GitLab Projects`
Organization	`Default`
Credential Type	`Source Control`
Username	`student`
SCM Private Key	Content of the `/home/student/.ssh/gitlab_rsa` file.

- Create a machine credential resource that automation controller can use to access the managed nodes in your environment. Create the credential with the following settings:

Field	Value
Name	`Managed Nodes`

Field	Value
Organization	`Default`
Credential Type	`Machine`
Username	`student`
SSH Private Key	Content of the `/home/student/.ssh/lab_rsa` file.

- Create a project resource so that automation controller can download files from the `git@git.lab.example.com:student/compreview` Git repository. Create the project with the following settings:

Field	Value
Name	`Comprehensive Review`
Organization	`Default`
Source Control Type	`Git`
Source Control URL	`git@git.lab.example.com:student/compreview`
Source Control Branch/Tag/Commit	`main`
Source Control Credential	`GitLab Projects`
Options	`Delete`
	`Allow Branch Override`

- Create an inventory resource that contains the same hosts and host groups that you used in the previous comprehensive review exercises. Create the inventory with the following settings:

Field	Value
Name	`Common Nodes`
Organization	`Default`

Add an inventory source to the `Common Nodes` inventory with the following settings:

Field	Value
Name	`Project Inventory`
Source	`Sourced from a Project`
Project	`Comprehensive Review`
Inventory file	`inventory-controller.yaml`

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Important
Although the `inventory-controller.yaml` file does not appear in the **Inventory file** list, you can type the name of the file. After typing the file name, press the `Enter` key rather than the `Tab` key.

After you synchronize the inventory source, you should see that the following hosts and host groups belong to the `Common Nodes` inventory.

Host	Host group
`arista1.lab.example.com`	`eos`
`arista2.lab.example.com`	`eos`
`iosxe1.lab.example.com`	`ios`
`iosxe2.lab.example.com`	`ios`
`junos1.lab.example.com`	`junos`
`junos2.lab.example.com`	`junos`
`workstation.lab.example.com`	Ungrouped

Important
The `Common Nodes` inventory does not define the `ansible_user` and `ansible_ssh_private_key_file` variables.

- Create a job template named `Configure Interfaces` that uses the `config_interfaces.yml` playbook to add interface descriptions. Create the job template with the following settings:

Field	Value
Name	`Configure Interfaces`
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`config_interfaces.yml`
Credentials	`Managed Nodes`

- Create a job template named `Deploy DNS, NTP, & Syslog` that uses the `deploy.yml` playbook to add DNS, NTP, and Syslog servers. Create the job template with the following settings:

Field	Value
Name	`Deploy DNS, NTP, & Syslog`

Field	Value
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`deploy.yml`
Credentials	`Managed Nodes`

- Create a job template named `Create Backups` that uses the `net_backup.yml` playbook to generate new backup files. Create the job template with the following settings:

Field	Value
Name	`Create Backups`
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`net_backup.yml`
Credentials	`Managed Nodes`

- Create a workflow template named `Configure & Back up` that belongs to the `Default` organization and uses the `Common Nodes` inventory. Configure the workflow to satisfy the following criteria:
  - Start the workflow by synchronizing the `Comprehensive Review` project. Use `Project Sync` as the node alias.
  - If the `Project Sync` node succeeds, then synchronize the `Project Inventory` source. Use `Inventory Sync` as the node alias.
  - If the `Inventory Sync` node succeeds, then run both the `Configure Interfaces` and `Deploy DNS, NTP, & Syslog` job templates.
  - If **both** the `Configure Interfaces` and `Deploy DNS, NTP, & Syslog` nodes succeed, then run the `Create Backups` job template.

  The completed workflow template should appear as follows:

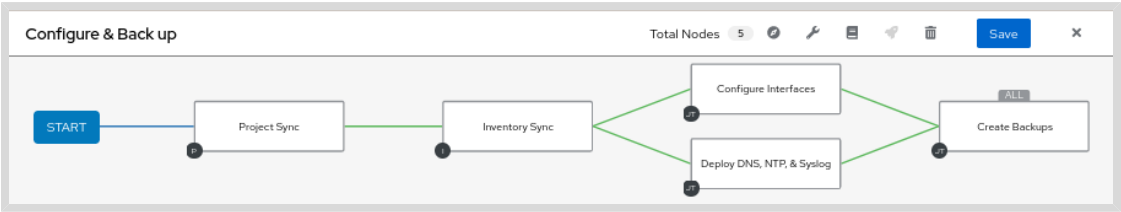


Figure 8.1: Completed workflow template

- Add a survey to the `Configure & Back up` workflow template that prompts for a location to store the backup files. Create the survey question with the following settings:

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Field	Value
Question	`Enter an absolute path to the backup directory:`
Answer variable name	`backup_store`
Answer type	`Text`
Required	*(selected)*
Minimum length	`2`
Maximum length	`100`
Default Answer	`/backups`

Note
The `net_backup.yml` playbook (used by the `Create Backups` node) includes the `net_backup` role. This role creates the directory defined by the value of the `backup_store` variable.

- Launch the `Configure & Back up` workflow template. Enter `/compreview` as the response to the survey question.
- After the jobs launched by the `Configure & Back up` workflow template complete, verify that the `/compreview` directory exists and that the directory contains a configuration file for each of the Junos and IOS managed nodes.

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 compreview-cr4
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr4
```

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

Solution

Running Playbooks with Automation Controller

Create resources within the automation controller web UI so that you can launch job templates and workflow job templates.

Outcomes

- Create automation controller source control and machine credentials.
- Create an automation controller project and then create an inventory that uses files stored in the project.
- Create automation controller job templates and workflow templates.
- Launch jobs from the automation controller web UI.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your environment for this exercise, and to ensure that all required resources are available:

```
[student@workstation ~]$ lab start compreview-cr4
```

1. From the automation controller web UI, create the `GitLab Projects` source control credential resource. Configure the 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**. If you do not see the **Resources** menu, then click the main menu icon at the upper left of the page. Create the credential with the following settings:

Field	Value
Name	`GitLab Projects`
Organization	`Default`
Credential Type	`Source Control`
Username	`student`
SCM Private Key	Content of the `/home/student/.ssh/gitlab_rsa` file.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

> **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.

1.3. Click **Save**.

2. Create the `Managed Nodes` machine credential such that it contains authentication information for the devices in the lab environment.

2.1. Navigate to **Resources** > **Credentials** and click **Add**. Create the credential with the following settings and then click **Save**:

Field	Value
Name	`Managed Nodes`
Organization	`Default`
Credential Type	`Machine`
Username	`student`
SSH Private Key	Content of the `/home/student/.ssh/lab_rsa` file.

3. Create the `Comprehensive Review` project resource and configure the project to pull files from the `git@git.lab.example.com:student/compreview` Git repository.

3.1. Navigate to **Resources** > **Projects** and click **Add**. Create the project with the following settings and then click **Save**:

Field	Value
Name	`Comprehensive Review`
Organization	`Default`
Source Control Type	`Git`
Source Control URL	`git@git.lab.example.com:student/compreview`
Source Control Branch/Tag/Commit	`main`
Source Control Credential	`GitLab Projects`
Options	`Delete`
	`Allow Branch Override`

3.2. After you click **Save**, automation controller immediately attempts to synchronize the project. Verify that the **Last Job Status** field updates to indicate that the project synchronization succeeded.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

4. Create the `Common Nodes` inventory resource and add the `Project Inventory` source to the inventory. Synchronize the inventory source and verify that the inventory contains the expected hosts and host groups.

4.1. Navigate to **Resources** > **Inventories** and then click **Add** > **Add inventory**. Create the inventory with the following settings:

Field	Value
Name	`Common Nodes`
Organization	`Default`

4.2. Click **Save**.

4.3. From the **Sources** tab, click **Add**. Create the inventory source with the following settings:

Field	Value
Name	`Project Inventory`
Source	`Sourced from a Project`
Project	`Comprehensive Review`
Inventory file	`inventory-controller.yaml`

Important
Although the `inventory-controller.yaml` file does not appear in the **Inventory file** list, you can type the name of the file. After typing the file name, press the `Enter` key rather than the `Tab` key.

4.4. Click **Save**.

4.5. Click **Sync** to synchronize the inventory source. The **Last Job Status** field updates to indicate that the synchronization succeeded.

4.6. Click the **Back to Sources** tab and then click the **Details** tab. The **Details** page for the **Common Nodes** inventory does not define the `ansible_user` and `ansible_ssh_private_key_file` variables.

Important
If the `Common Nodes` inventory defines the `ansible_user` and `ansible_ssh_private_key_file` variables, then you entered the `inventory.yaml` file as the source. Update the `Project Inventory` source so that it uses the `inventory-controller.yaml` file. You might also need to update the variables defined for the `Common Nodes` inventory to remove the undesired variables.

4.7. From the **Details** page for the **Common Nodes** inventory, click the **Hosts** tab to display the imported hosts. Click the **Groups** tab to display the imported groups. The inventory source imported the following hosts and host groups:

Host	Host group
`arista1.lab.example.com`	`eos`
`arista2.lab.example.com`	`eos`
`iosxe1.lab.example.com`	`ios`
`iosxe2.lab.example.com`	`ios`
`junos1.lab.example.com`	`junos`
`junos2.lab.example.com`	`junos`
`workstation.lab.example.com`	Ungrouped

5. Create a job template named `Configure Interfaces` that uses the `config_interfaces.yml` playbook to configure interface descriptions.

5.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. Create the job template with the following settings and then click **Save**:

Field	Value
Name	`Configure Interfaces`
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`config_interfaces.yml`
Credentials	`Managed Nodes`

6. Create a job template named `Deploy DNS, NTP, & Syslog` that uses the `deploy.yml` playbook to add DNS, NTP, and Syslog servers.

6.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. Create the job template with the following settings and then click **Save**:

Field	Value
Name	`Deploy DNS, NTP, & Syslog`
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`deploy.yml`
Credentials	`Managed Nodes`

7. Create a job template named `Create Backups` that uses the `net_backup.yml` playbook to generate new backup files.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

7.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add job template**. Create the job template with the following settings and then click **Save**:

Field	Value
Name	`Create Backups`
Inventory	`Common Nodes`
Project	`Comprehensive Review`
Playbook	`net_backup.yml`
Credentials	`Managed Nodes`

8. Create a workflow template named `Configure & Back up` that belongs to the `Default` organization and uses the `Common Nodes` inventory.

8.1. Navigate to **Resources** > **Templates** and then click **Add** > **Add workflow template**. Create the workflow template with the following settings and then click **Save**:

Field	Value
Name	`Configure & Back up`
Organization	`Default`
Inventory	`Common Nodes`

9. Add a `Project Sync` node named `Project Sync` to synchronize the `Comprehensive Review` project.

9.1. In the visualizer, click **Start**.

9.2. On the **Add Node** page, select **Project Sync** in the **Node Type** list.

9.3. Select the **Comprehensive Review** project.

9.4. Enter `Project Sync` in the **Node Alias** field and then click **Save**.

10. Add an `Inventory Source Sync` node named `Inventory Sync` that synchronizes the `Project Inventory` source if the `Project Sync` node succeeds.

10.1. In the visualizer, hover over the **Project Sync** node and click the **+** icon to add a new node.

> **Note**
> You might need to increase the display resolution on the `workstation` machine, decrease the web browser zoom, or both, to see the entire visualizer display screen.
>
> If your mouse has a scroll wheel, then you can use the scroll wheel to zoom in and out of the visualizer.

10.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

10.3. On the **Node type** page, select **Inventory Source Sync** in the **Node Type** list.

10.4. Select the **Project Inventory** source.

10.5. Enter `Inventory Sync` in the **Node Alias** field and then click **Save**.

11. Add a `Job Template` node that launches the `Configure Interfaces` job template to configure interface descriptions if the `Inventory Sync` node succeeds.

11.1. In the visualizer, hover over the **Inventory Sync** node and click the + icon to add a new node.

11.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

11.3. On the **Node type** page, select the **Configure Interfaces** job template.

11.4. Click **Save**.

12. Add a `Job Template` node that launches the `Deploy DNS, NTP, & Syslog` job template to add DNS, NTP, and Syslog servers if the `Inventory Sync` node succeeds.

12.1. In the visualizer, hover over the **Inventory Sync** node and then click the + icon to add a new node.

12.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

12.3. On the **Node type** page, select the **Deploy DNS, NTP, & Syslog** job template.

12.4. Click **Save**.

13. Add a job template node that launches the `Create Backups` job template to generate new backup files if **both** the `Configure Interfaces` and `Deploy DNS, NTP, & Syslog` nodes succeed.

13.1. In the visualizer, hover over the **Configure Interaces** node and then click the + icon to add a new node.

13.2. On the **Run type** page, select the **On Success** condition and then click **Next**.

13.3. On the **Node type** page, select the **Create Backups** job template.

13.4. Select **All** from the **Convergence** menu and then click **Save**. This convergence setting indicates that all parent nodes that point to the **Create Backups** node must match their defined conditions.

Notice that the icon for the **Create Backups** node contains the word `ALL` to indicate the convergence setting.

13.5. Hover over the **Deploy DNS, NTP, & Syslog** node and then click the **Link to an available node** icon.

13.6. Hover over the **Create Backups** node. The visualizer displays a dashed line with an arrow pointing to the node. Click the **Create Backups** node.

13.7. Click **Save** to accept the default **On Success** condition.

13.8. Click **Save** again in the upper-right corner to save this workflow template.

After this step, the completed workflow template should appear as follows:

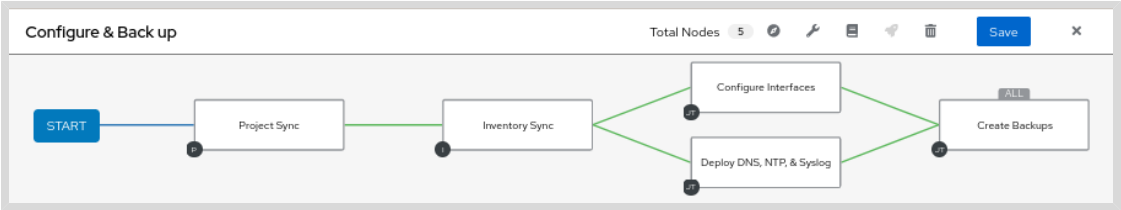


Figure 8.2: Completed workflow template

14. Add a survey to the `Config & Back up` workflow template, which prompts for a location to store the backup files.

14.1. Navigate to **Resources** > **Templates** and click the **Configure & Back up** link.

14.2. Click the **Survey** tab and then click **Add** to begin. Create the survey question with the following settings:

Field	Value
Question	`Enter an absolute path to the backup directory:`
Answer variable name	`backup_store`
Answer type	`Text`
Required	*(selected)*
Minimum length	`2`
Maximum length	`100`
Default Answer	`/backups`

> **Note**
> The `net_backup.yml` playbook (used by the `Create Backups` node) includes the `net_backup` role. This role creates the directory defined by the value of the `backup_store` variable.

14.3. Click **Save**.

14.4. Enable the survey for the `Config & Back up` job template. Click **Survey Disabled** and verify that the label changes to **Survey Enabled**.

15. Launch the `Configure & Back up` workflow template and observe the result.

15.1. Navigate to **Resources** > **Templates**.

15.2. Click the **Launch Template** icon for the **Configure & Back up** workflow template.

15.3. Enter `/compreview` as the response to the survey question and then click **Next**.

15.4. Click **Launch** to launch the workflow template with `/compreview` as the value for the `backup_store` variable.

15.5. Monitor the workflow. All nodes in the workflow succeed.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

For use by Abdul Majeed Khan Y amkhan@vectratech.in amkhan@vectratech.in Copyright © 2023 Red Hat, Inc.

16. Verify that the `/compreview` directory exists on the `workstation` machine and that the directory contains a backup file for each managed node in the `ios` and `junos` inventory host groups.

16.1. List the contents of the `/compreview` directory:

```
[student@workstation ~]$ ls -1 /compreview
iosxe1.lab.example.com.cfg
iosxe2.lab.example.com.cfg
junos1.lab.example.com.cfg
junos2.lab.example.com.cfg
```

16.2. *(Optional)* Inspect one or more of the backup files for changes made by the `Configure Interfaces` and the `Deploy DNS, NTP, & Syslog` job templates.

The `Configure Interfaces` job template added the `Managed by Ansible` description to some interfaces on the managed nodes.

The `Deploy DNS, NTP, & Syslog` job template added the following servers:

Server type	Added servers
DNS	`172.25.250.220` and `172.25.240.220`
NTP	`172.25.23.240` and `172.25.23.230`
Syslog	`192.168.10.10` and `192.168.11.10`

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 compreview-cr4
```

Finish

As the `student` user on the `workstation` machine, 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 compreview-cr4
```