Red Hat Training and Certification

DO374 - Instructor Demo Guide

Travis Michette

Version 1.0

Table of Contents

1. Developing Playbooks with Ansible Automation Platform 2

1.1. Introducing Red Hat Ansible Automation Platform 2

Describing the architecture of Red Hat Ansible Automation Platform 2 (AAP2) and new features for Ansible development.

1.1.1. Orientation to Red Hat Ansible Automation Platform 2

New evolution of Ansible Platform providing customization with Ansible Execution Environments (EEs), Ansible Navigator, and a redesign of Ansible Tower which has now become Ansible Controller. Ansible Automation Platform now also provides Ansible Automation Hub which is a private Ansible Galaxy as well as a container registry service for Ansible EEs.

1.1.2. Red Hat Ansible Automation Platform 2 Components

1.1.2.1. Ansible Core

The Ansible Core package is provided by **ansible-core** and is version Ansible Core 2.11 in AAP2.0. This package provides the **ansible** command as well as the built-in modules allowing administrators to run playbooks with the **ansible-playbook** command. The **ansible-core** package only contains a minimal set of modules (***ansible.builtin***) collection and all other modules have been moved to Ansible collections.

The **ansible** *Package*

It is still possible to install the package called **ansible**. This will install Ansible 2.9 which is AAP1.2. This version of Ansible will support collections, but is not the full AAP2.0 version of Ansible.

1.1.2.2. Ansible Content Collections

Ansible content and modules have now been re-organized into what is referred to as Ansible Content Collections (***Content Collections***) in order to support the growth and rapid development of modules and packages. This separation allows modules, roles, plug-in to be separated from the **Ansible Core** for a simpler management style.

This separation provides the following

- Developers can easily upgrade and deploy new version of their modules without depending on Ansible
- Only needed modules can be present on the Ansible system or in the execution environment
- New modules and content doesn’t need to wait for a new version of Ansible to be deployed

ansible.builtin

The **ansible.builtin** collection is a special collection that will always be part of Ansible Core. However, this has a limited number of modules. Things like the **Firealld** module have now been moved as part of the **POSIX** Ansible Collection.

Collection Mapping

Ansible mapping of content collections: https://github.com/ansible/ansible/blob/devel/lib/ansible/config/ansible_builtin_runtime.yml

Red Hat Official Collections are available from: https://console.redhat.com/ansible/ansible-dashboard

1.1.2.3. Ansible Content Navigator

AAP provides **ansible-navigator** which is the new *preffered* tool to run and interact with Ansible on the CLI. It extends and includes the functionality of the **ansible-playbook**, **ansible-inventory**, and **ansible-config** commands.

While Ansible Navigator still leverages **ansible.cfg**, it has its own configuration file that must point to both the **ansible.cfg** being used as well as using its own **ansible-navigator.yml** configuration file which has even more options to extend and control the behavior of Ansible Navigator.

Why **ansible-navigator***?*

The purpose of **ansible-navigator** is to separate the control node from the execution environment. This makes it easier for playbooks to be run in a production environment from Ansible Controller Nodes (formerly known as Ansible Tower).

1.1.2.4. Ansible Execution Environments

Ansible Execution Environments (EEs) as container images which contain the following items:

Ansible EEs

- Ansible Core
- Ansible Content Collections
    - Ansible Modules
    - Ansible Roles
- Python Libraries
- Other dependencies

The default AAP2 environment provides Ansible Core 2.11 and Red Hat Certified Content Collections to give a similar experience to AAP1.2 which is what provides Ansible 2.9.

AAP1.2 and Ansible 2.9

Ansible 2.9 is part of AAP1.2, but it supports things like Ansible Collections. You must have AAP2 to support things link Ansible Navigator, and other components of the AAP2 platform.

The **ansible-builder** package can be used to create and develop your own custom execution environments.

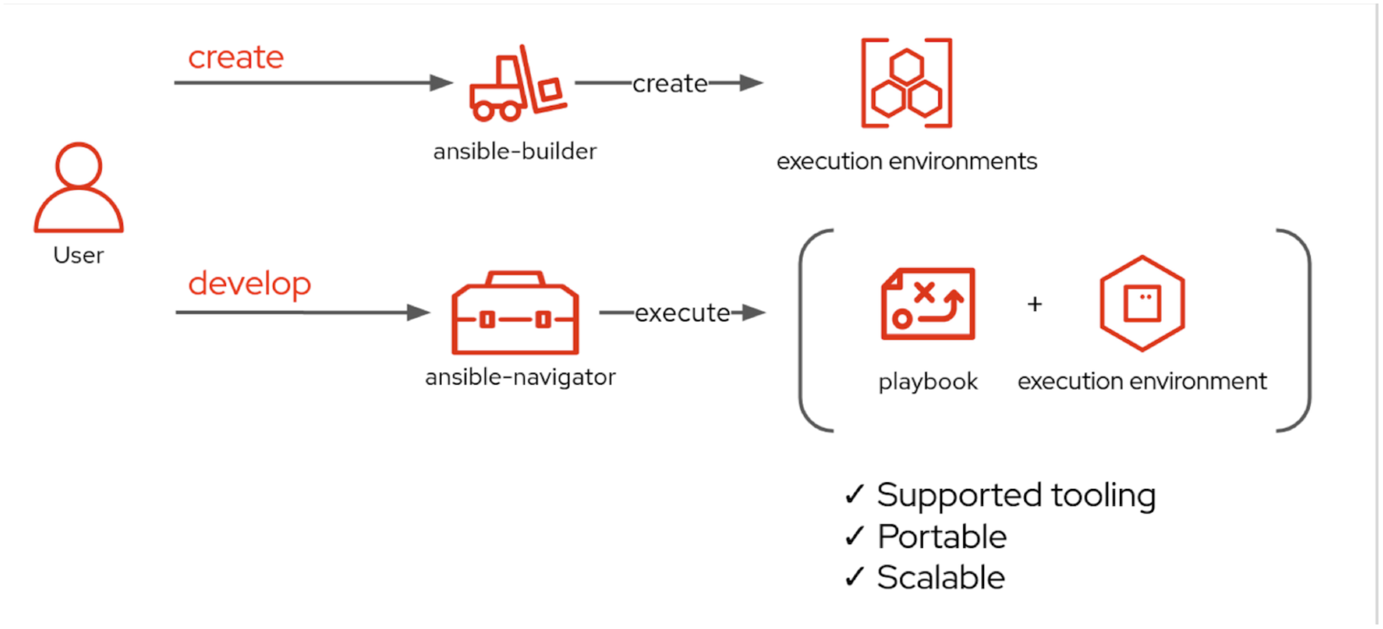


Figure 1. AAP2 Utilities

1.1.2.5. Automation Controller

Automation Controller provides a central web-based UI and REST API which can be used to automate Ansible jobs. Previous iterations of Ansible leveraged Ansible Tower which was the control node and execution environment. With the deployment of AAP2, Ansible Tower was re-named to Ansible Automation Controller and serves as the control node ***only***, as with Ansible Automation Controller, the execution environment can be separated from the controller node as it now runs in a container.

Figure 2. AAP2 Automation Controller Components, align=

By separating the control node functionality and execution environments, it is much easier to leverage the system when playbooks could require different python environments or other requirements to run.

Automation Controller

AAP2 Automation Controllers has the ability to use multiple execution environments on playbook and project levels as the execution plan is 100% separate from the control plane.

1.1.2.6. Ansible Automation Hub

Ansible Automation Hub allows easy management and distribution of Ansible automation content. Red Hat maintains supported and certified content collections and Ansible Galaxy maintains the community-based content. The addition of Automation Hub also provides the ability to host a private automation hub which is basically a self-hosted version of Ansible Galaxy or Red Hat's **console.redhat.com** version of Automation Hub.

The private automation hub provides a container registry for distribution of custom execution environments as well as a repository for Ansible Collections and namespaces.

1.1.2.7. Hosted Services

Red Hat provides three (3) hosted Ansible Automation services

- Ansible Automation Hub
- Ansible Automation Services Catalog
- Ansible Insights for Red Hat AAP

1.1.3. Red Hat Ansible Automation Platform 2 Architecture

1.1.3.1. Developing Playbooks with Ansible Automation Platform 2

Ansible Execution Engines (EEs) can be built and customized to contain everything needed to execute playbooks developed by your organization. These playbooks can be leveraged seamlessly between content navigator and automation controller providing access is available to the EEs being used (which is where automation hub comes into play). :pygments-style: tango :source-highlighter: pygments :toc: :toclevels: 7 :sectnums: :sectnumlevels: 6 :numbered: :chapter-label: :icons: font :icons: font :imagesdir: ./images/

1.2. Running Playbooks with Automation Content Navigator

Section Info Here

1.2.1. Introducing Automation Content Navigator

Ansible Content Navigator (**ansible-navigator**) is a new tool created for AAP2 designed to make it easier to write and test playbooks and more importantly leverage Ansible Controller with the playbooks. **ansible-navigator** uses and combines the features from the previous ansible commands into a single top-level command tool and interface.

Ansible Commands Combined in Navigator

- ansible-playbook
- ansible-inventory
- ansible-config
- ansible-doc

Ansible Ad-Hoc Commands

Ansible ad-hoc commands are not supported with Ansible Navigator and not reccomended as a best practice. However, ad-hoc commands can still be run by installing the Ansible package and leveraging the **ansible** command.

In order to run a playbook using Ansible Navigator, you must use the **ansible-navigator run** command. It is possible to use Ansible Navigator to provide the same output as the **ansible-playbook** command by providing the argument with the run command and using **-m stdout**.

Listing 1. **ansible-playbook** *Command*

```
[student@workstation navigator (main)]$ ansible-playbook playbook.yml

... OUTPUT OMITTED ...

PLAY RECAP ***************************************************************************
servera.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

Listing 2. **ansible-navigator** *Equivalent to* **ansible-playbook**

```
[student@workstation navigator (main)]$ ansible-navigator run playbook.yml -m stdout

... OUTPUT OMITTED ...

PLAY RECAP ***************************************************************************
servera.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

ansible-navigator *Use*

If the **-m stdout** is not provided, **ansible-navigator** runs the playbook in interactive mode. This mode allows analyzing plays, tasks, and the runtime in a more detailed fashion. Typically, you use number for what should be displayed, but if the number is >9 it is necessary to use **:** followed by the number. The interactive mode interface can be exited by hitting the escape key (multiple times, depending on the level being analyzed).

1.2.1.1. Improving Portability with Automation Execution Environments

Execution environments were introduced as part of AAP2. The introduction of EEs meant that Ansible could be run from a container image that included Ansible Engine runtimes, content collections, software dependencies, and python components needed to run playbooks and interact with Ansible. EEs allow **ansible-navigator** and **Ansible Automation Controller** to leverage automation execution environments simplifying development, testing, and deployment of Ansible playbooks in a consistent and predictable fashion. Red Hat provides several supported EEs from Red Hat's Ansible Automation Hub.

EEs allow **ansible-navigator** and **Ansible Controller** to easily leverage custom execution environments by specifying an **Execution Environment Image** (**--eei**) to be used for running playbooks. By specifying EEIs, it is no longer necessary to have multiple configurations on control nodes to run Ansible playbooks.

1.2.2. Installing Automation Content Navigator

Ansible Navigator is part of the **Ansible Automation Platform 2.0** repository. It can be installed with a **yum** command.

Listing 3. Installing **ansible-navigator**

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

1.2.3. Configuring Authentication to Managed Hosts

Even though Ansible Navigator leverages EEs, it must also be able to log in to managed nodes as well as gain privileged access on managed nodes. Therefore, it is best to implement **SSH keys** and **sudo** without a password.

1.2.3.1. Preparing SSH Key-Based Authentication

SSH access can be prepared by creating users on the systems and setting up SSH key-pairs between the systems. The SSH key pair is created with **ssh-keygen** and usually resides in **~/.ssh/** directory. The public key is installed on the remove system in the **~/.ssh/authorized_keys** file usually with the **ssh-copy-id** command.

SUDO access is generally granted without password access by creating a sudoers file for the user in the **/etc/sudoers.d** directory.

*Listing 4. Example Sudoers File (**/etc/sudoers.d/devops**)*

```
# User rules for devops
devops ALL=(ALL) NOPASSWD:ALL ①
```

① Allows the **devops** user SUDO access for all commands without requiring a password.

1.2.3.2. Providing Private Keys to the Automation Execution Environment

There are some tricks to running and leveraging **ansible-navigator** as the SSH private key must somehow become available to the EE. When running in a GUI environment, **ssh-agent** is already running and will add private keys to the agent. This same behavior doesn't happen when logged into the systems via SSH.

Using SSH on the Control Node

A major difference with AAP2 is the use of EEs. When **ansible-navigator** uses an EE, it is running from a container and doesn't have access to the user's SSH keys or settings. In order to use **ansible-navigator** on a system where the login is through SSH vs. a graphical login, it is necessary to use SSH-Agent to manage and store SSH private keys so the container has them available for use.

Listing 5. Storing SSH Keys and Leveraging SSH-Agent

```
[student@workstation ~]$ eval $(ssh-agent) ①
Agent pid 240212

[student@workstation ~]$ ssh-add ~student/.ssh/lab_rsa ②
Identity added: /home/student/.ssh/lab_rsa (/home/student/.ssh/lab_rsa)
```

① Starting **ssh-agent**

② Adding Identities to SSH-Agent Keyring

1.2.4. Running Automation Content Navigator

1.2.4.1. Running Playbooks

1.2.4.2. Reviewing Previous Playbook Runs

1.2.4.3. Reading Documentation

1.2.4.4. Getting Help

1.3. Demo - Ansible Content Navigator

Ansible Content Navigator can be used to run playbooks in place of the Ansible command. At this point, the **ansible-navigator.yml** file doesn't exist, so additional command line options will need to exist. Later chapters introduce how to fully configure navigator for execution environments.

Example 1. **Navigator Demo**

1. Switch to Demo Directory

```
[student@workstation ~]$ cd /home/student/github/do374/Demos/CH1/navigator
```

2. Install Navigator

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

... OUTPUT OMITTED ...

Installed:
  ansible-navigator-1.0.0-2.el8ap.noarch

Complete!
```

3. Login to **hub.lab.example.com** to allow downloading of the EE **ee-supported-rhel8:2.0** for navigator

```
[student@workstation navigator]$ podman login -u admin -p redhat
hub.lab.example.com
Login Succeeded!
```

4. Set an execution environment variable and verify

```
[student@workstation navigator]$ export EE=ee-supported-rhel8:2.0 ; echo $EE
ee-supported-rhel8:2.0
```

5. Run the playbook with the **ansible-navigator run** command

```
[student@workstation navigator]$ ansible-navigator run playbook.yml -m stdout
--eei $EE ①

... OUTPUT OMITTED ...

servere.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
serverf.lab.example.com    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

① The **$EE** environment variable provides the EE for the **ansible-navigator** command

SSH Key Errors from Execution Environment

If you receive this as a message … it is possible you are running ansible using SSH and the SSH keys haven't been added. it is necessary to use an **eval $(ssh-agent)** followed by adding the key to your keyring.

Listing 6. Error

```
fatal: [servera.lab.example.com]: UNREACHABLE! => {"changed":
false, "msg": "Failed to connect to the host via ssh: Warning:
Permanently added 'servera.lab.example.com,172.25.250.10'
(ECDSA) to the list of known
hosts.\r\ndevops@servera.lab.example.com: Permission denied
(publickey,gssapi-keyex,gssapi-with-mic,password,keyboard-
interactive).", "unreachable": true}
```

Listing 7. Adding SSH Keys for Ansible Execution Environment

```
[student@workstation navigator]$ eval $(ssh-agent) ①
Agent pid 234883

[student@workstation navigator]$ ssh-add ~/.ssh/lab_rsa ②
Identity added: /home/student/.ssh/lab_rsa
(/home/student/.ssh/lab_rsa)
```

① Starting **ssh-agent**

② Adding key to keyring for SSH-Agent

ansible-playbook *Equivalence*

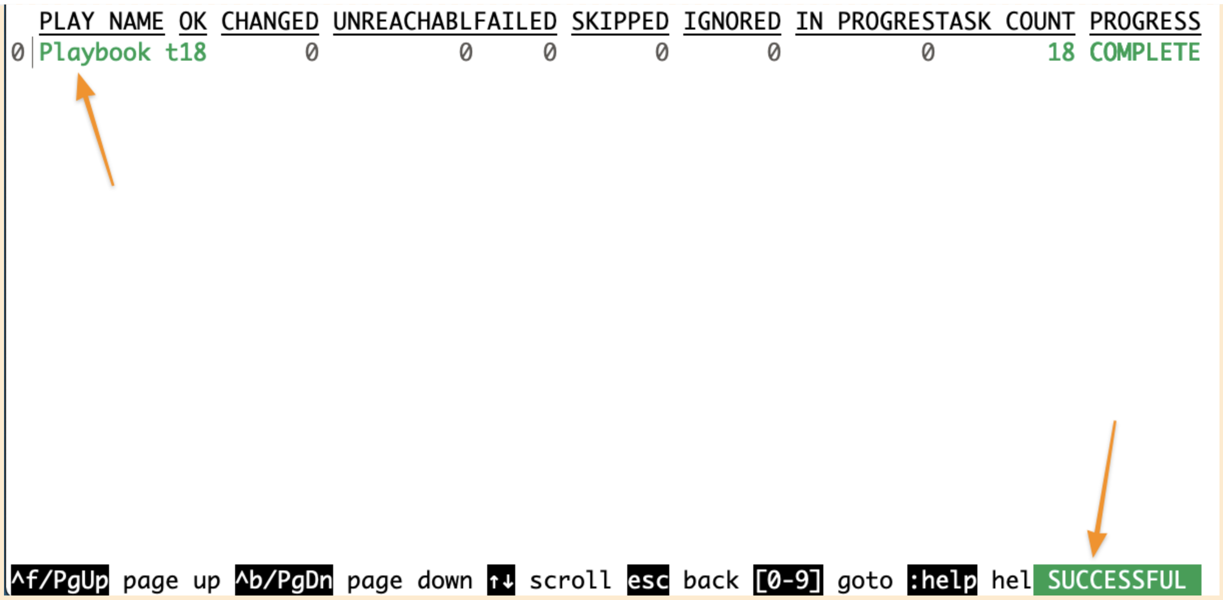
The **ansible-navigator run playbook.yml -m stdout** will provide the same STDOUT as the **ansible-playbook** command. There are some other features about **ansible-navigator** but those will be covered in a later chapter and section.

6. Run the **ansible-navigator run** command interactively (***Leave out the -m stdout***)

   a. Get output of first playbook/play (Hit **0** and Enter to navigate)

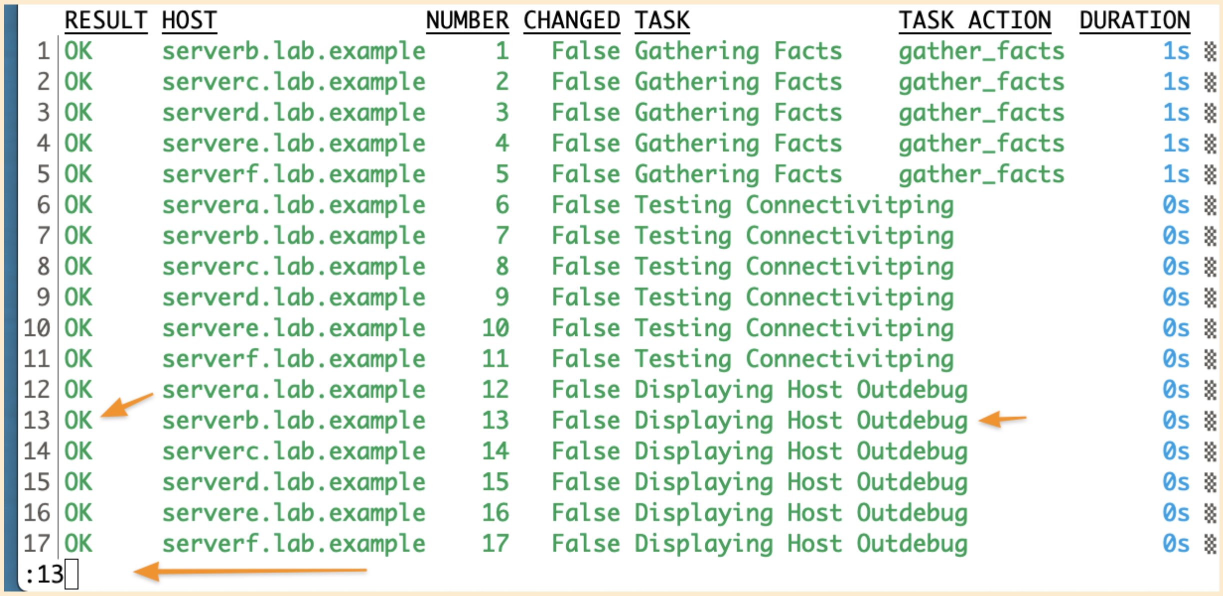
   b. Get detailed output of **Task 13** (Hit **:** and then hit **13** and enter to navigate)

```
[student@workstation navigator]$ ansible-navigator run playbook.yml --eei
$EE
```



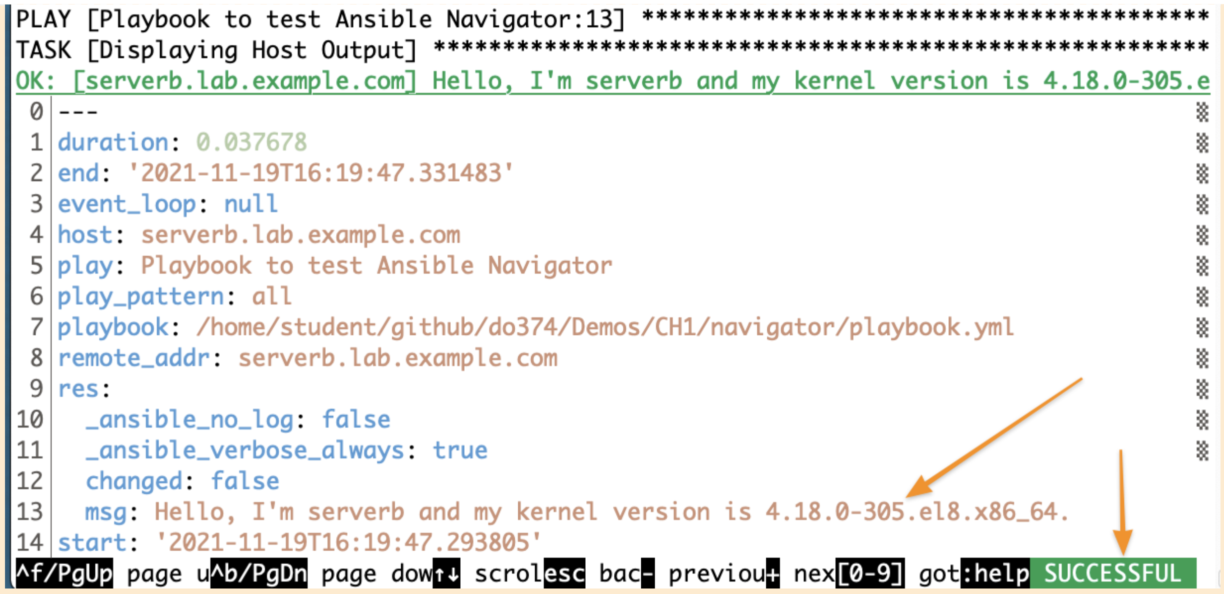
```
  PLAY NAME OK CHANGED UNREACHABLFAILED SKIPPED IGNORED IN PROGRESTASK COUNT PROGRESS
0│Playbook t18        0           0      0       0       0           0      18 COMPLETE










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

Figure 3. Navigator Interactive Window



```
   RESULT HOST                 NUMBER CHANGED TASK              TASK ACTION   DURATION
 1│OK     serverb.lab.example       1   False Gathering Facts   gather_facts        1s ▒
 2│OK     serverc.lab.example       2   False Gathering Facts   gather_facts        1s ▒
 3│OK     serverd.lab.example       3   False Gathering Facts   gather_facts        1s ▒
 4│OK     servere.lab.example       4   False Gathering Facts   gather_facts        1s ▒
 5│OK     serverf.lab.example       5   False Gathering Facts   gather_facts        1s ▒
 6│OK     servera.lab.example       6   False Testing Connectivitping               0s ▒
 7│OK     serverb.lab.example       7   False Testing Connectivitping               0s ▒
 8│OK     serverc.lab.example       8   False Testing Connectivitping               0s ▒
 9│OK     serverd.lab.example       9   False Testing Connectivitping               0s ▒
10│OK     servere.lab.example      10   False Testing Connectivitping               0s ▒
11│OK     serverf.lab.example      11   False Testing Connectivitping               0s ▒
12│OK     servera.lab.example      12   False Displaying Host Outdebug              0s ▒
13│OK     serverb.lab.example      13   False Displaying Host Outdebug              0s ▒
14│OK     serverc.lab.example      14   False Displaying Host Outdebug              0s ▒
15│OK     serverd.lab.example      15   False Displaying Host Outdebug              0s ▒
16│OK     servere.lab.example      16   False Displaying Host Outdebug              0s ▒
17│OK     serverf.lab.example      17   False Displaying Host Outdebug              0s ▒
:13
```

Figure 4. Attempting to get Task 13 Information



```
PLAY [Playbook to test Ansible Navigator:13] ****************************************
TASK [Displaying Host Output] *******************************************************
OK: [serverb.lab.example.com] Hello, I'm serverb and my kernel version is 4.18.0-305.e
 0 ---
 1 duration: 0.037678
 2 end: '2021-11-19T16:19:47.331483'
 3 event_loop: null
 4 host: serverb.lab.example.com
 5 play: Playbook to test Ansible Navigator
 6 play_pattern: all
 7 playbook: /home/student/github/do374/Demos/CH1/navigator/playbook.yml
 8 remote_addr: serverb.lab.example.com
 9 res:
10   _ansible_no_log: false
11   _ansible_verbose_always: true
12   changed: false
13   msg: Hello, I'm serverb and my kernel version is 4.18.0-305.el8.x86_64.
14 start: '2021-11-19T16:19:47.293805'
^f/PgUp page u^b/PgDn page dow↑↓ scrolesc bac- previou+ nex[0-9] got:help SUCCESSFUL
```

Figure 5. Task 13 Information

7. Exit Ansible Navigator by hitting the **ESC** key multiple times to exit each layer.

```
[student@workstation navigator]$
```

1.4. Managing Ansible Project Materials Using Git

Section Info Here

1.4.1. Defining Infrastructure as Code

1.4.2. Introducing Git

1.4.3. Describing Initial Git Configuration

1.4.4. Starting the Git Workflow

1.4.4.1. Examining the Git Log

1.4.5. Working with Branches and References

1.4.5.1. Creating Branches

1.4.5.2. Merging Branches

1.4.5.3. Creating Branches from Old Commits

1.4.5.4. Pushing Branches to Remote Repositories

1.4.6. Structuring Ansible Projects in Git

1.4.6.1. Roles and Ansible Content Collections

1.4.6.2. Configuring Git to Ignore Files

1.5. Demo - Using Git

Ansible playbooks can be leveraged for Infrastructure-as-Code (IaC). In order to do this, playbooks and other assets should exist in version control. One way to accomplish this is by using Github or Gitlab. The course has been setup to use Gitlab, but this demo, we will see how to use Github and personal access tokens.

Example 2. **Git Demo**

1. Update the BASHRC file to use the **git-prompt.sh** Assets

   Listing 8. **.bashrc** *File*

```
[student@workstation ~]$ vim .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

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


# User specific environment
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging
feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

2. Apply changes for BASHRC

```
[student@workstation ~]$ source .bashrc
```

3. Configure system for PAT (Personal Access Tokens)

```
[student@workstation ~]$ git config --global credential.helper cache
```

4. Verify credential helper and other configurations

```
[student@workstation ~]$ git config --global -l
user.name=Git Lab
user.email=git@lab.example.com
push.default=simple
```

5. Create Github Directory and Switch to it

```
[student@workstation ~]$ mkdir Github ; cd Github
```

6. Clone **DO374** Repository

```
[student@workstation Github]$ git clone https://github.com/tmichett/do374.git
Cloning into 'do374'...
remote: Enumerating objects: 56, done.
remote: Counting objects: 100% (56/56), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 56 (delta 11), reused 51 (delta 9), pack-reused 0
Unpacking objects: 100% (56/56), 556.15 KiB | 2.93 MiB/s, done
```

7. Change to **do374** Directory

```
[student@workstation Github]$ cd do374/
[student@workstation do374 (main)]$ ①
```

① Notice it shows main branch

8. Create a dummy file and observe prompt change

```
[student@workstation do374 (main)]$ echo "I'm a dummy file" > test.txt
[student@workstation do374 (main %)]$ ①
```

① Prompt changed to **%** indicating new "untracked" files

9. Add and Commit File

Listing 9. Adding File for Tracking

```
[student@workstation do374 (main %)]$ git add .
[student@workstation do374 (main +)]$ ①
```

① Prompt changed to **+** indicating new files being tracked, but not committed

Listing 10. Commiting File Locally

```
[student@workstation do374 (main +)]$ git commit -m "Testing"
[main 9697a39] Testing
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
[student@workstation do374 (main)]$ ①
```

① Normal Prompt

10. Get status of repository

```
[student@workstation do374 (main)]$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

11. Push to remote repository

```
[student@workstation do374 (main)]$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 285 bytes | 285.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/tmichett/do374.git
   2b7cf28..9697a39  main -> main
```

First time pushing saves credentials

Listing 11. SSH/CLI Version - Warning doesn't appear if using X11/Wayland and Gnome in Graphical Environment

```
[student@workstation CH1]$ git push

(gnome-ssh-askpass:236143): Gtk-WARNING **: 11:50:21.480: cannot
open display:
error: unable to read askpass response from
'/usr/libexec/openssh/gnome-ssh-askpass'
Username for 'https://github.com': tmichett

(gnome-ssh-askpass:236144): Gtk-WARNING **: 11:50:23.638: cannot
open display:
error: unable to read askpass response from
'/usr/libexec/openssh/gnome-ssh-askpass'
Password for 'https://tmichett@github.com':
```

1.6. Implementing Recommended Ansible Practices

Section Info Here

1.6.1. The Effectiveness of Ansible

1.6.2. Keeping Things Simple

1.6.2.1. Keeping Your Playbooks Readable

1.6.2.2. Use Existing Modules

1.6.2.3. Adhering to a Standard Style

1.6.3. Staying Organized

1.6.3.1. Following Conventions for Naming Variables

1.6.3.2. Standardizing the Project Structure

1.6.3.3. Using Dynamic Inventories

1.6.3.4. Taking Advantage of Groups

1.6.3.5. Using Roles and Ansible Content Collections for Reusable Content

1.6.3.6. Running Playbooks Centrally

1.6.3.7. Building Automation Execution Environments

1.6.4. Testing Often

1.6.4.1. Testing the Results of Tasks

1.6.4.2. Using Block/Rescue to Recover or Rollback

1.6.4.3. Developing Playbooks with the Latest Ansible Version

1.6.4.4. Using Test Tools

2. Managing Content Collections and Execution Environments

2.1. Reusing Content from Ansible Content Collections

Section Info Here

2.1.1. Defining Ansible Content Collections

2.1.1.1. Organizing Ansible Content Collections in Namespaces

2.1.2. Using Ansible Content Collections

2.1.2.1. Accessing Ansible Content Collection Documentation

2.1.2.2. Using Ansible Content Collections in Playbooks

2.1.2.3. Finding Ansible Content Collections

2.1.2.4. Using the Built-in Ansible Content Collection

2.2. Finding and Installing Ansible Content Collections

Section Info Here

2.2.1. Sources for Ansible Content Collections

2.2.1.1. Finding Collections on Ansible Automation Hub

2.2.2. Installing Ansible Content Collections

2.2.2.1. Installing Collections from the Command Line

2.2.2.2. Installing Collections with a Requirements File

2.2.2.3. Listing Installed Collections

2.2.3. Configuring Collection Sources

2.2.3.1. Installing Collections from Ansible Automation Hub

2.2.3.2. Installing Collections from Private Automation Hub

2.3. Selecting an Execution Environment

Section Info Here

2.3.1. Describing Automation Execution Environments

2.3.2. Selecting a Supported Automation Execution Environment

2.3.3. Inspecting Automation Execution Environments

2.3.4. Using Automation Execution Environments with Ansible Content Navigator

3. Running Playbooks with Automation Controller

3.1. Explaining the Automation Controller Architecture

Section Info Here

3.1.1. Introduction to Automation Controller

3.1.2. Describing the Architecture of Automation Controller

3.1.3. Automation Controller Features

3.2. Running Playbooks in Automation Controller

Section Info Here

3.2.1. Exploring Resources in Automation Controller

3.2.2. Creating Credential Resources

3.2.2.1. Listing Credentials

3.2.2.2. Creating a Machine Credential

3.2.2.3. Creating a Source Control Credential

3.2.3. Creating Project Resources

3.2.4. Creating Inventory Resources

3.2.4.1. Manually Creating Groups and Hosts

3.2.4.2. Populating Groups and Hosts Using a Project Inventory File

3.2.5. Creating Job Template Resources

3.2.6. Launching and Reviewing Jobs

4. Working with Ansible Configuration Settings

4.1. Examining Ansible Configuration with Automation Content Navigator

Section Info Here

4.1.1. Inspecting Configuration in Interactive Mode

4.1.1.1. Searching for Specific Configuration Parameters

4.1.1.2. Accessing Parameter Details

4.1.1.3. Inspecting Local Configuration

4.1.2. Inspecting Ansible Configuration in Standard Output Mode

4.2. Configuring Automation Content Navigator

Section Info Here

4.2.1. Format of the Settings File

4.2.2. Locating the Settings File

4.2.2.1. Selecting a Settings File to Use

4.2.3. Editing the Settings File

4.2.3.1. Setting a Default Automation Execution Environment

4.2.3.2. Default to Running in Standard Output Mode

4.2.3.3. Disabling Playbook Artifacts

4.2.3.4. Overview of an Example Settings File

5. Managing Inventories

5.1. Managing Dynamic Inventories

Section Info Here

5.1.1. Generating Inventories Dynamically

5.1.2. Discussing Inventory Plug-ins

5.1.2.1. Using Inventory Plug-ins

5.1.3. Developing Inventory Scripts

5.1.3.1. Using Inventory Scripts

5.1.4. Managing Multiple Inventories

5.2. Writing YAML Inventory Files

Section Info Here

5.2.1. Discussing Inventory Plug-ins

5.2.2. Writing YAML Static Inventory Files

5.2.2.1. Setting Inventory Variables

5.2.3. Converting a Static Inventory File in INI Format to YAML

5.2.4. Troubleshooting YAML Files

5.2.4.1. Protecting a Colon Followed by a Space

5.2.4.2. Protecting a Variable that Starts a Value

5.2.4.3. Knowing the Difference Between a String and a Boolean or Float

5.3. Managing Inventory Variables

Section Info Here

5.3.1. Describing the Basic Principles of Variables

5.3.2. Variable Merging and Precedence

5.3.2.1. Determining Command-line Option Precedence

5.3.2.2. Determining Role Default Precedence

5.3.2.3. Determining Host and Group Variable Precedence

5.3.2.4. Determining Play Variable Precedence

5.3.2.5. Determining the Precedence of Extra Variables

5.3.3. Separating Variables from Inventory

5.3.4. Using Special Inventory Variables

5.3.4.1. Configuring Human Readable Inventory Host Names

5.3.5. Identifying the Current Host Using Variables

6. Managing Task Execution

6.1. Controlling Privilege Escalation

Section Info Here

6.1.1. Privilege Escalation Strategies

6.1.1.1. Privilege Escalation by Configuration

6.1.1.2. Defining Privilege Escalation in Plays

6.1.1.3. Privilege Escalation in Tasks

6.1.1.4. Grouping Privilege Escalation Tasks with Blocks

6.1.1.5. Applying Privilege Escalation in Roles

6.1.1.6. Listing Privilege Escalation with Connection Variables

6.2. Choosing Privilege Escalation Approaches

6.3. Controlling Task Execution

Section Info Here

6.3.1. Controlling the Order of Execution

6.3.1.1. Importing or Including Roles as a Task

6.3.1.2. Defining Pre- and Post-tasks

6.3.1.3. Reviewing the Order of Execution

6.3.2. Listening to Handlers

6.3.2.1. Notifying Handlers

6.3.3. Controlling the Order of Host Execution

6.4. Running Selected Tasks

Section Info Here

6.4.1. Tagging Ansible Resources

6.4.2. Managing Tagged Resources

6.4.2.1. Running Tasks with Specific Tags

6.4.2.2. Combining Tags to Run Multiple Tasks

6.4.2.3. Skipping Tasks with Specific Tags

6.4.2.4. Listing Tags in a Playbook

6.4.3. Assigning Special Tags

6.5. Optimizing Execution for Speed

Section Info Here

6.5.1. Optimizing Playbook Execution

6.5.1.1. Optimizing the Infrastructure

6.5.1.2. Disabling Fact Gathering

6.5.1.3. Reusing Gathered Facts with Fact Caching

6.5.1.4. Limiting Fact Gathering

6.5.1.5. Increasing Parallelism

6.5.1.6. Avoiding Loops with the Package Manager Modules

6.5.1.7. Efficiently Copying Files to Managed Hosts

6.5.1.8. Using Templates

6.5.1.9. Enabling Pipelining

6.5.2. Profiling Playbook Execution with Callback Plug-ins

6.5.2.1. Timing Tasks and Roles

7. Transforming Data with Filters and Plug-ins

7.1. Processing Variables Using Filters

Section Info Here

7.1.1. Ansible Filters

7.1.2. Variable Types

7.1.3. Manipulating Lists

7.1.3.1. Extracting list elements

7.1.3.2. Modifying the Order of List Elements

7.1.3.3. Merging Lists

7.1.3.4. Operating on Lists as Sets

7.1.4. Manipulating Dictionaries

7.1.4.1. Joining dictionaries

7.1.4.2. Converting Dictionaries

7.1.5. Hashing, Encoding, and Manipulating Strings

7.1.5.1. Hashing strings and passwords

7.1.5.2. Encoding strings

7.1.5.3. Formatting Text

7.1.5.4. Replacing Text

7.1.6. Manipulating JSON Data

7.1.6.1. JSON Queries

7.1.6.2. Parsing and Encoding Data Structures

7.2. Templating External Data using Lookups

Section Info Here

7.2.1. Lookup Plug-ins

7.2.2. Calling Lookup Plug-ins

7.2.3. Selecting Lookup Plug-ins

7.2.3.1. Reading the Contents of Files

7.2.3.2. Applying Data with a Template

7.2.3.3. Reading Command Output in the Execution Environment

7.2.3.4. Getting Content from a URL

7.2.3.5. Getting Information from the Kubernetes API

7.2.3.6. Using Custom Lookup Plug-ins

7.2.4. Handling Lookup Errors

7.3. Implementing Advanced Loops

Section Info Here

7.3.1. Comparing Loops and Lookup Plug-ins

7.3.2. Example Iteration Scenarios

7.3.2.1. Iterating over a List of Lists

7.3.2.2. Iterating Over Nested Lists

7.3.2.3. Iterating Over a Dictionary

7.3.2.4. Iterating Over a File Globbing Pattern

7.3.2.5. Retrying a Task

7.4. Using Filters to Work with Network Addresses

Section Info Here

7.4.1. Gathering and Processing Networking Information

7.4.2. Network Information Filters

7.4.2.1. Testing IP Addresses

7.4.2.2. Filtering Data

7.4.2.3. Manipulating IP Addresses

7.4.2.4. Reformatting or Calculating Network Information

8. Coordinating Rolling Updates

8.1. Delegating Tasks and Facts

Section Info Here

8.1.1. Delegating Tasks

8.1.1.1. Delegating to localhost

8.1.2. Delegating Facts

8.2. Configuring Parallelism

Section Info Here

8.2.1. Configure Parallelism in Ansible Using Forks

8.2.2. Running Batches of Hosts Through the Entire Play

8.3. Managing Rolling Updates

Section Info Here

8.3.1. Overview

8.3.2. Controlling Batch Size

8.3.2.1. Setting a Fixed Batch Size

8.3.2.2. Setting Batch Size as a Percentage

8.3.2.3. Setting Batch Sizes to Change During the Play

8.3.3. Aborting the Play

8.3.3.1. Specifying Failure Tolerance

8.3.4. Running a Task Once

9. Creating Content Collections and Execution Environments

9.1. Writing Ansible Content Collections

Section Info Here

9.1.1. Developing Ansible Content Collections

9.1.1.1. Selecting a Namespace for Collections

9.1.1.2. Creating Collection Skeletons

9.1.1.3. Adding Content to Collections

9.1.1.4. Updating Collection Metadata

9.1.1.5. Declaring Collection Dependencies

9.1.1.6. Building Collections

9.1.1.7. Validating and Testing Collections

9.1.2. Publishing Collections

9.2. Building a Custom Execution Environment

Section Info Here

9.2.1. Deciding When to Create a Custom Automation Execution Environment

9.2.2. Preparing for a New Automation Execution Environment

9.2.2.1. Declaring the Ansible Content Collections to Install

9.2.2.2. Declaring Python Packages

9.2.2.3. Declaring RPM Packages

9.2.3. Building a New Automation Execution Environment

9.2.3.1. Interacting with the Build Process

9.3. Validating a Custom Execution Environment

Section Info Here

9.3.1. Testing Automation Execution Environments Locally

9.3.1.1. Running a Test Playbook

9.3.1.2. Providing Authentication Credentials

9.3.2. Sharing an Automation Execution Environment from Private Automation Hub

9.4. Using Custom Content Collections and Execution Environments in Automation Controller

Section Info Here

9.4.1. Using Custom Collections with Existing Execution Environments

9.4.1.1. Preparing Ansible Projects for Automation Controller

9.4.1.2. Storing Authentication Credentials for Collections

9.4.2. Using Custom Automation Execution Environments with Automation Controller

9.4.2.1. Storing Container Registry Credentials

9.4.2.2. Configuring Automation Execution Environments

9.4.2.3. Configuring the Default Automation Execution Environment for a Project

9.4.2.4. Specifying an Automation Execution Environment in a Template

Appendix A: Exam Objectives

A.1. Understand and use Git

- Clone a Git repository
- Create, modify and push files in a Git repository

A.2. Manage inventory variables

- Structure host and group variables using multiple files per host or group
- Use special variables to override the host, port, or remote user for a specific host
- Set up directories containing multiple host variable files for managed hosts
- Override names used in inventory files with a different name or IP address

A.3. Manage task execution

- Control privilege execution
- Run selected tasks from a playbook

A.4. Transform data with filters and plugins

- Populate variables with data from external sources using lookup plugins
- Use lookup and query functions to incorporate data from external sources into playbooks and deployed template files
- Implement loops using structures other than simple lists using lookup plugins and filters
- Inspect, validate, and manipulate variables containing networking information with filters

A.5. Delegate tasks

- Run a task for a managed host on a different host
- Control whether facts gathered by a task are delegated to the managed host or the controlling host

A.6. Manage content collections

- Create a content collection
- Install a content collection
- Publish a content collection

A.7. Manage execution environments

- Build an execution environment

- Run playbooks in a execution environment
- Upload execution environments into automation hub
- Using execution environments in automation controller

A.8. Manage inventories and credentials

- Manage advanced inventories
- Create a dynamic inventory from an identity management server or a database server
- Create machine credentials to access inventory hosts
- Create a source control credential

A.9. Manage automation controller

- Run playbooks in automation controller
- Pull content into automation controller from either git or automation hub
- Pull an execution environment from automation hub and run a playbook in it.