Red Hat
Training and
Certification

DO467

Travis Michette

Version 3.0 | 2024-03-27

Table of Contents

Introduction

Classroom Machines

Machine name	IP addresses	ROLE
workstation.lab.example.com	172.25.250.9	Graphical workstation used for system administration
servera.lab.example.com	172.25.250.10	Managed server "A"
serverb.lab.example.com	172.25.250.11	Managed server "B"
serverc.lab.example.com	172.25.250.12	Managed server "C"
serverd.lab.example.com	172.25.250.13	Managed server "D"
servere.lab.example.com	172.25.250.14	Managed server "E"
serverf.lab.example.com	172.25.250.15	Managed server "F"
git.lab.example.com	172.25.250.5	GitLab server
hub.lab.example.com	172.25.250.6	Private automation hub server
controller.lab.example.com	172.25.250.7	Automation controller server
control2.lab.example.com	172.25.250.16	Second automation controller
utility.lab.example.com	172.25.250.8	System with utility services required for classroom
db.lab.example.com	172.25.250.20	Database server
exec1.lab.example.com	172.25.250.21	Execution node
exec2.lab.example.com	172.25.250.22	Execution node
exec3.lab.example.com	172.25.250.17	Execution node
hop1.lab.example.com	172.25.250.24	Hop node
bastion.lab.example.com	172.25.250.254	Bridges classroom and student networks

Repositories for this Course

Main Repository

The DO467 Demo repository contains the PDF of the instructor notes, and various pre-built demos for the chapters. Demos in this repository are meant to setup and configure the environment automatically to provide consistent demos from course-to-course. Each of the various chapter directories contains one or more playbooks for demonstration or for setting up and configuring the environment for running demonstrations from the various **Demo Repositories**,

- **DO467 Demo**: https://github.com/tmichett/DO467_Demo
- **DO467 Notes (Private)**: https://github.com/tmichett/DO467_Notes

The DO467 Notes repository contains the Asciidoc code for the PDF as well as where the examples get developed. Both of these repositories are part of Jenkins workflows. The primary workflow task monitors the **NOTES** repository for changes. Upon changes, a new PDF is built. The PDF, along with the **Demos** directory are then promoted to the **DEMO** repository and published to the **main** branch publicly for everyone to access (including students).

Demo Repositories

These repositories contain demo playbooks that are used as projects in Ansible Controller. They also contain inventory files as well as configuration files needed to run the playbooks locally from workstation for testing.

- **AAP2 Controller Demo**: https://github.com/tmichett/AAP2_Controller_Demo
- **AAP2 Demos**: https://github.com/tmichett/AAP2_Demos

Demo Setup/Preparing to Teach

The primary demo uses all playbooks to setup an Organizations, Users, Teams, Projects, Credentials, Roles, Job Templates, and finally a Job Workflow Template for approvals.

1. Create Github directory

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

2. Clone Repository

```
[student@workstation Github]$ git clone https://github.com/tmichett/DO467_Demo.git
Cloning into 'DO467_Demo'...
remote: Enumerating objects: 339, done.
remote: Counting objects: 100% (339/339), done.
remote: Compressing objects: 100% (251/251), done.
remote: Total 339 (delta 167), reused 198 (delta 31), pack-reused 0
Receiving objects: 100% (339/339), 5.11 MiB | 10.36 MiB/s, done.
Resolving deltas: 100% (167/167), done.
```

3. Change to **DO467_Demo** Directory and the Demo Setup directory

```
[student@workstation Github]$ cd DO467_Demo/Demos/Demo_Setup/
```

4. Update the **vars.yml** file with the Github Personal Access Token (PAT)

```
[student@workstation Demo_Setup]$ vim vars.yml
---
Github_PAT: <Insert_Token_Here> ①
```

① Replace "<Insert_Token_Here>" with your access token

5. Run the **Demo_Prep.sh** script

```
[student@workstation Demo_Setup]$ ./Demo_Prep.sh
```

Setup and Prepare Advanced Demo Environment

1. Change to the Gitlab Directory

```
[student@workstation Demo_Setup]$ cd Gitlab/
```

2. Run the **Setup_Gitlab.sh** script

```
[student@workstation Gitlab]$ ./Setup_Gitlab.sh
*********************************************
***** Clone Repo from Github *****************
mkdir: cannot create directory □/tmp/Github□: File exists
fatal: destination path 'AAP2_Demos' already exists and is not an empty directory.
*********************************************
***** Push Repository to Gitlab ***************
*********************************************

Creating Gitlab Project
[sudo] password for student:
```

SUDO and Other Passwords

Depending on the environment and what has been setup, the script is made to allow prompting for passwords. If the **student** user has been configured for sudo without a password, the prompt will not appear.

It may also prompt for the classroom environment Gitlab credentials which are:

- **Username**: student
- **Password**: Student@123

The playbook utilizes some Ansible modules and collections to clone from the public Github repositories mentioned above and create a new project and repository on the Internal classroom environment for testing Gitlab WebHooks.

Preparing Automation Controller Timeout

It is important to note that the default timeout for Automation Controller often requires multiple logins throughout the day and week. It is suggested that the timeout be changed in the classroom environment so that you avoid logging in and out of the Automation Controller constantly.

Changing Timeouts

I suggest changing timeouts to last the entire week. This works very well on the ILT version of the course. However, if you are using ROL and VT labs, the systems sometimes have the automated shutdown at the end of the day. In this instance, the timeouts may only last for the day depending on the browser and how things were shutdown.

Adujusting the Timeouts

Settings ⇒ Miscellaneous Authentication Settings ⇒ Idle Time Force Logout

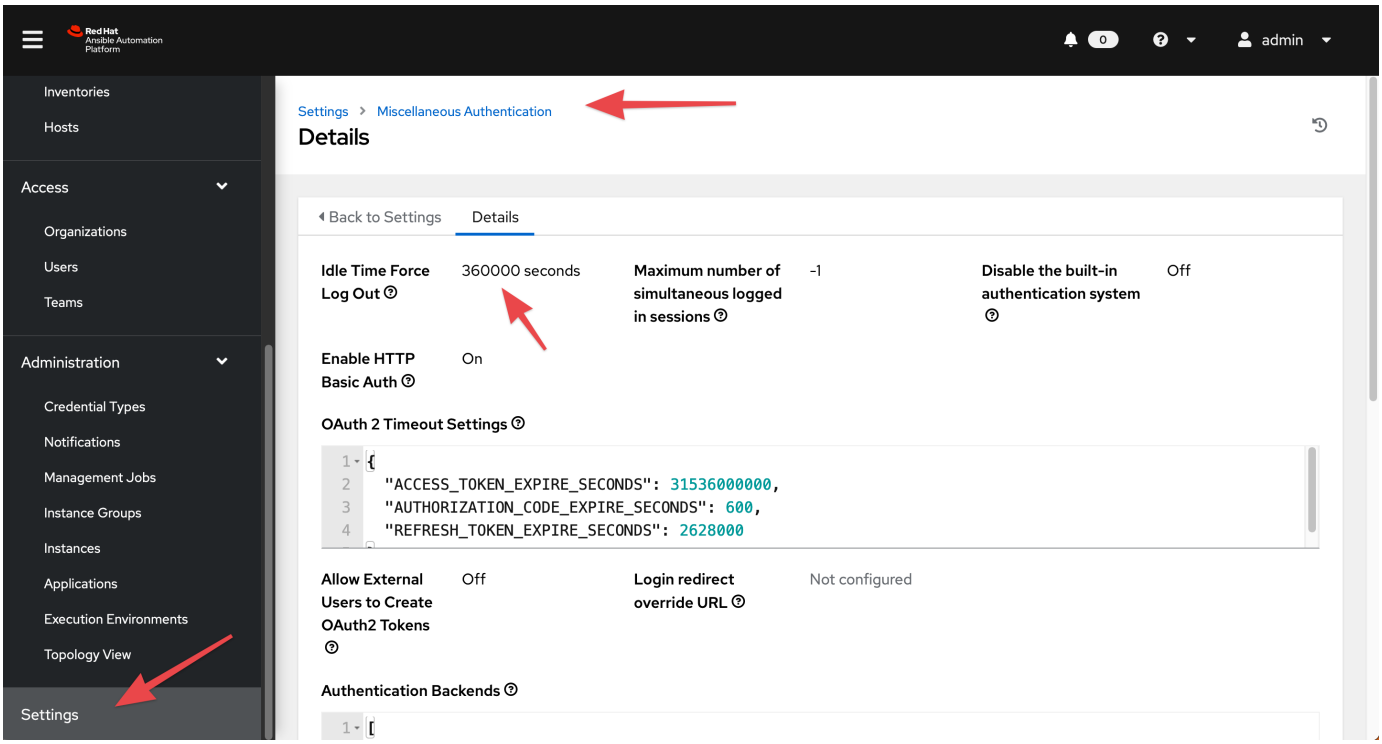

Figure 1. Adjusting Forced Logout Time

Changing Fact Cache Timeout

There are some small issues with the timeouts on cached facts. This has been pointed out as tips in the Instructor Guide. It will be necessary to point out for demos here too in order to ensure facts are being cached properly.

Message and Information from Instructor Guide

- Manually created static inventories (such as those created in chapter 3) seem to have issues with fact caching even though the equivalent inventory (created by a playbook) does not. Because of this problem, students must run lab finish host-inventory (if they create the Prod inventory) and they must run lab finish host-review (if they create the Dev inventory).
- Interestingly, a manually created static inventory will cache facts if you set a small enough cache timeout, such as 60 seconds.

Changing Fact Cache Timeouts

For my purposes, I've made it five (5) minutes so that I can properly show demos with the students and have questions. Keep in mind, these demos are shown using the automatically generated projects and components from this Github repository (https://github.com/tmichett/DO467_Demo).

Settings ⇒ Jobs ⇒ Per-Host Ansible Fact Cache Timeout

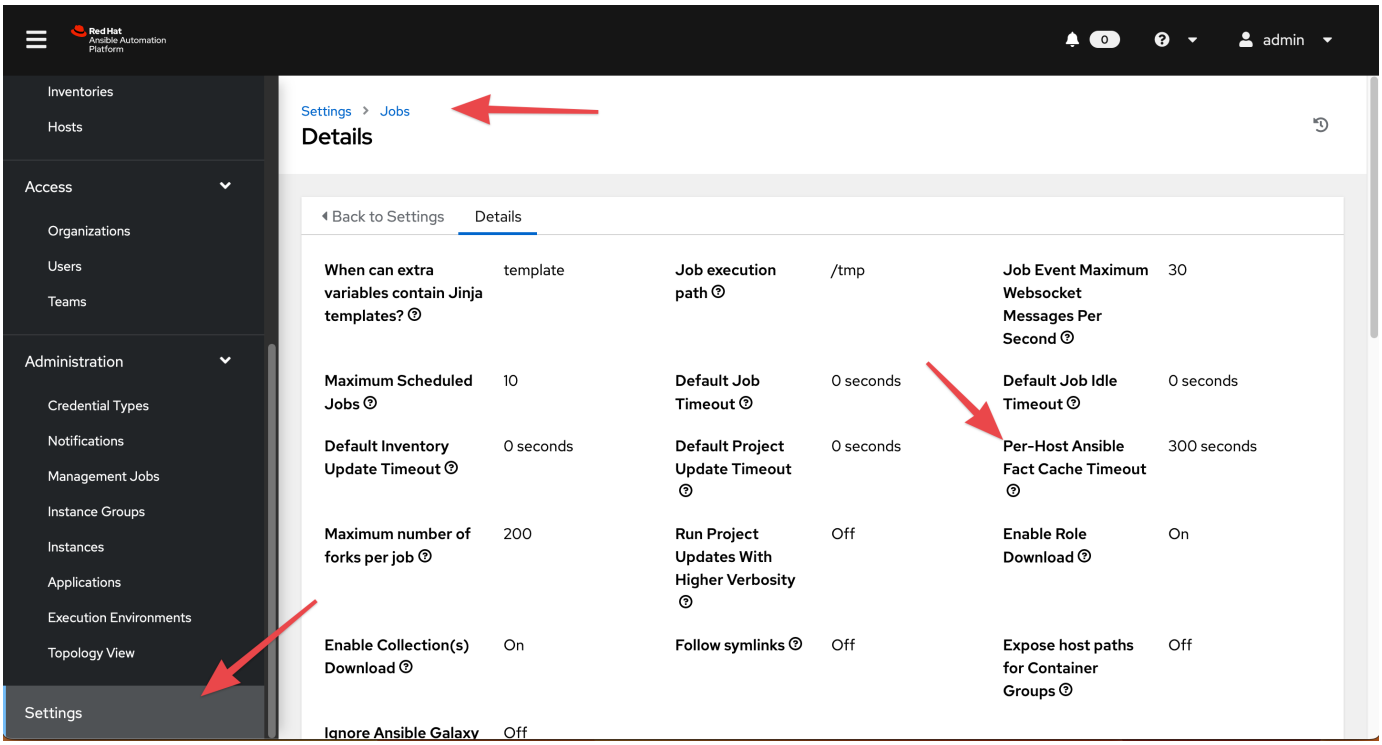

Figure 2. Adjusting Cached Fact Timeout

1. Installing Red Hat Ansible Automation Platform

1.1. Explaining the Red Hat Ansible Automation Platform Architecture

1.1.1. Red Hat Ansible Automation Platform

New version of Ansible Automation.

AAP2.x provides

- Separation of the **Ansible Core** project development and Ansible Modules by introducing content collections
- Execution Environments (leverages containerized execution of playbooks)
  - Ansible Navigator
  - Ansible Builder
- Replaces Ansible Tower with Ansible Automation Controller
- Introduces Ansible Private Automation Hub (container registry and private collection source)

1.1.2. Red Hat Ansible Automation Platform Components

Ansible Core

Minimal implementation of Ansible providing functionality to run Ansible playbooks. Includes the **ansible.builtin** modules.

- **CLI** - Includes the **ansible-playbook**, **ansible-doc**, and the **ansible** ad-hoc commands and utilities
- **Language** - Uses YAML to construct playbooks
- **Framework** - provides platform to extend Ansible by leveraging Content Collections
- **Functions** - Allows use of various logic components such as conditionals, blocks, includes, loops, and other Ansible items.

Ansible Content Collections

Provides rapid growth and expansion for Ansible modules, functions, and filters. Allows only the **ansible.builtin** collection to be part of Ansible Core and separates development of all other collections and their corresponding modules, filters, and other components.

Supportability

Allows a means for vendors to provide certified and supported collections and modules and leverages Ansible Automation Hub as a means of acquiring these collections.

Ansible Content Navigator

New tool being leveraged to develop and test Ansible playbooks. The **ansible-navigator** command leverages Ansible Execution Environments (EEs) to execute playbooks. The EE allows separation of the control node from the environment executing the playbook. In other words, the EE through **ansible-navigator** provides a self-contained execution environment eliminating the need for any of the Ansible Core utilities (ansible-playbook, ansible-inventory, ansible-config) to be locally installed and further increases portability of the tested playbook because EEs can also be used from Ansible Automation Controller.

Automation Execution Environments

An Ansible Execution Environment Image (EEI) contains at a minimum the Ansible Core Package, Ansible Content Collections, any Python libraries, executables, and other dependencies needed to run a given playbook.

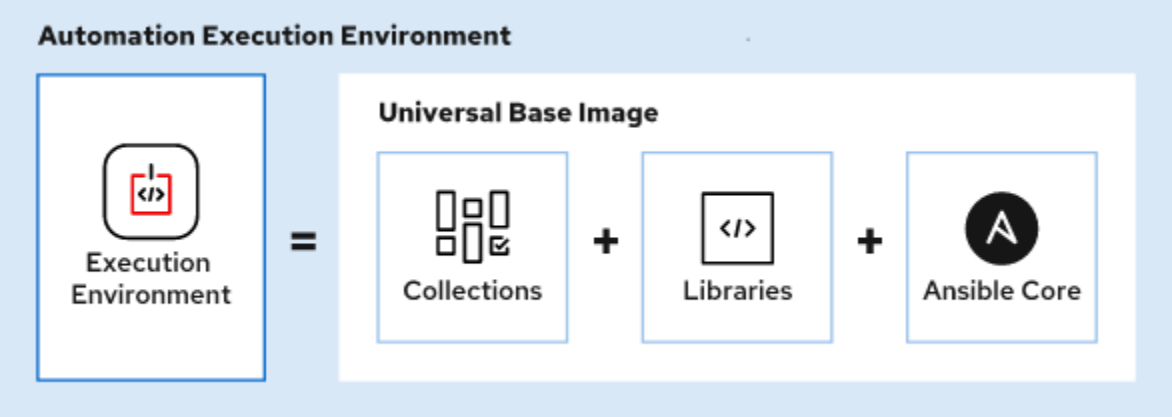

Figure 3. Ansible Execution Environment

Selecting an EEI

The **ansible-navigator** command allows selecting and EEI for executing and running a playbook. When developing playbooks for others to run, or more importantly Automation Controller, once tested, the EEI name will be provided so that future runs for the given playbook will be executed in the same environment ensuring the same outcome.

Listing 1. Selecting EEI on Command-Line

```
[student@workstation General_Demos]$ ansible-navigator run Hello_Demo.yml
--pp missing --eei hub.lab.example.com/ee-supported-rhel8:latest -m stdout
①
```

① The **-eei** line specified the image and container registry.

Listing 2. Controlling EEI on with ansible-navigator.yml Config File

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

The EEI can be customized and extended with the Ansible Builder package.

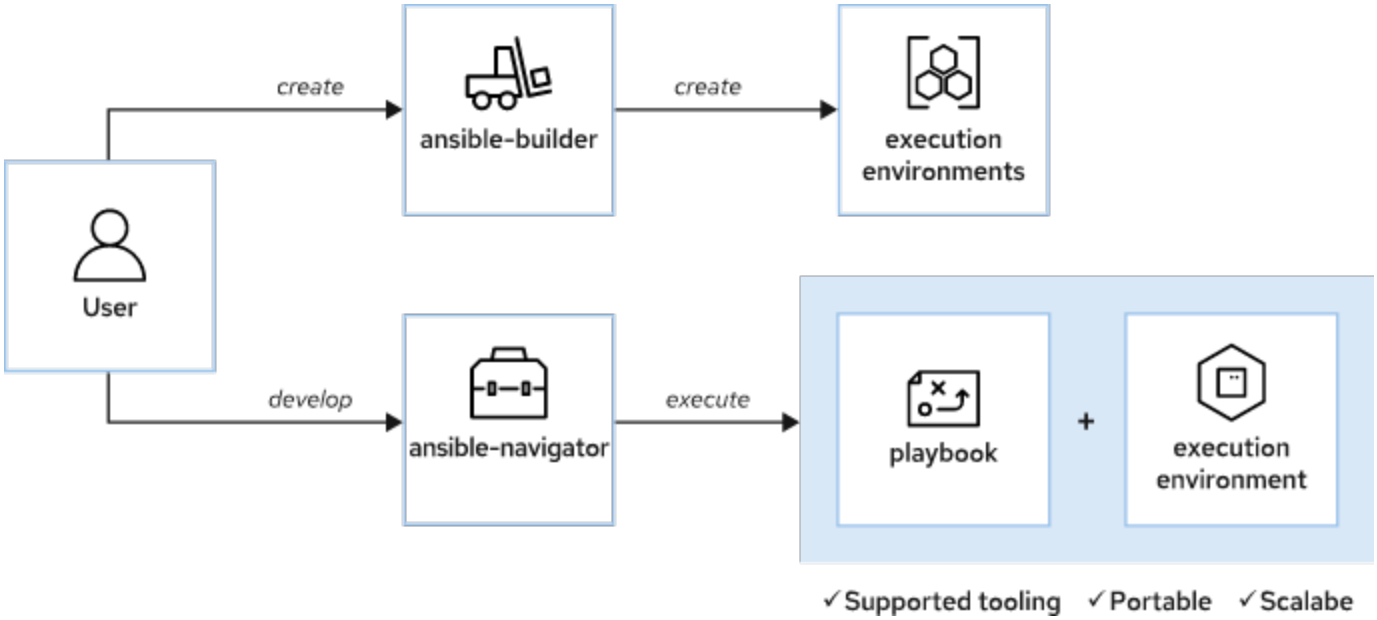

Figure 4. Creating an Ansible Execution Environment

Automation Controller

Replacement for Ansible Tower. Provides a multi-function environment with a RestfulAPI, WebUI, and both control plane and execution planes for enterprise environments. The Automation Controller provides the ability to integrate directly with CI/CD tools and provides a centralized place to manage all Ansible automation tasks.

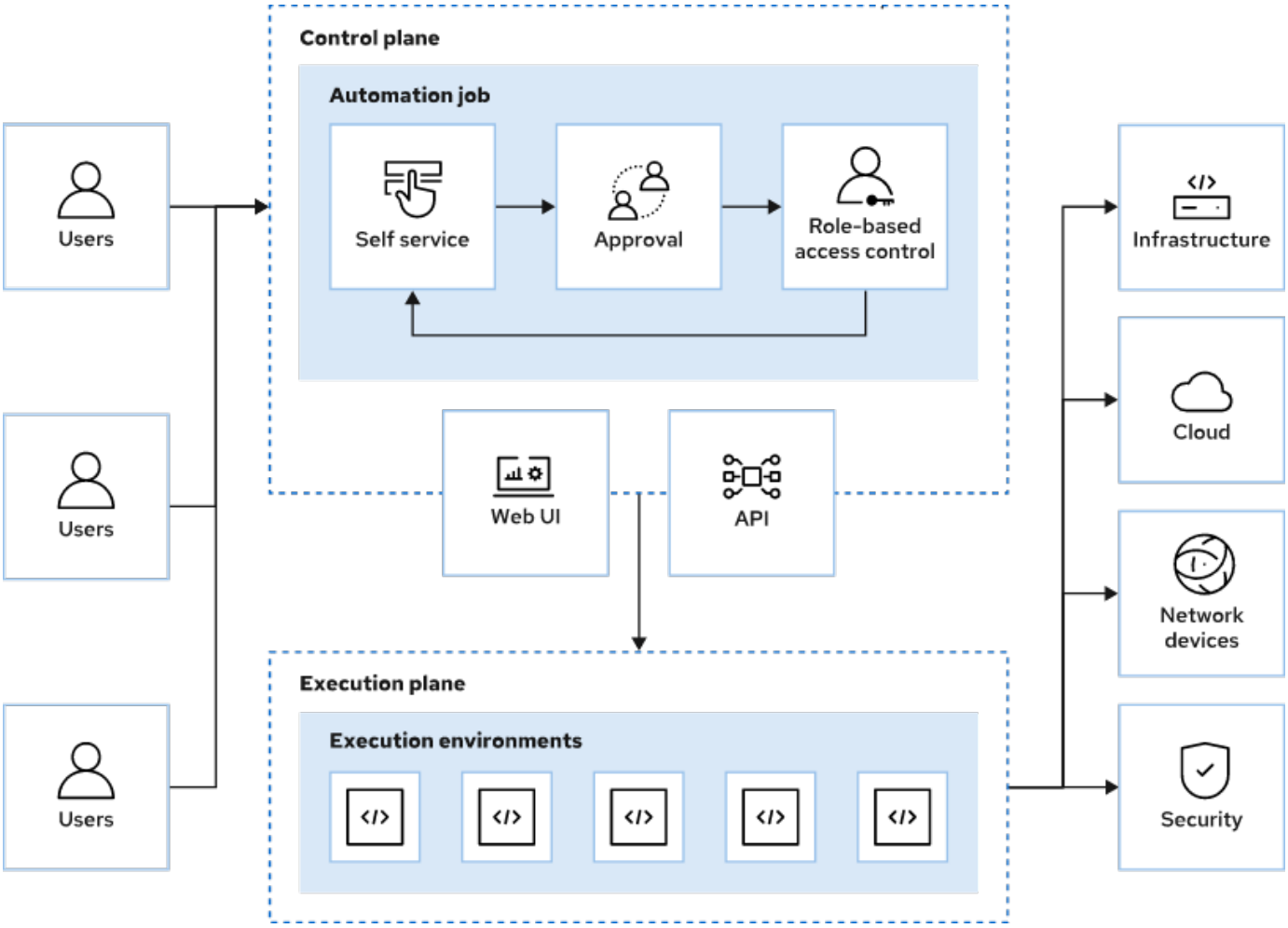

Figure 5. Ansible Automation Controller Components

Automation Hub and Private Automation Hub

Ansible Automation Hub provides a central source to manage Ansible content collections and receive certified and supported collections and modules. The use of Private Automation Hub allows organizations to setup and manage an internal container registry for managing Ansible EEIs as well as storing and managing both certified and home-grown Ansible content collections.

References

Easy way to find Collections on Ansible Automation Platform.

Automation Hub ⇒ **Partners** ⇒ **Ansible** then you can select the collection you want.

- **Ansible Posix Collection**: https://console.redhat.com/ansible/automation-hub/repo/published/ansible/posix

1.2. Installing Automation Controller and Private Automation Hub

Installing Exercise Time

It can take up to 15 minutes for the GE **lab start install-installation** so it is recommended to run this script at the start of the lecture.

1.2.1. Planning the Installation

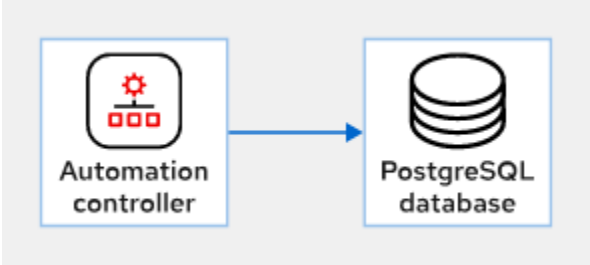

Figure 6. Standalone Automation Controller

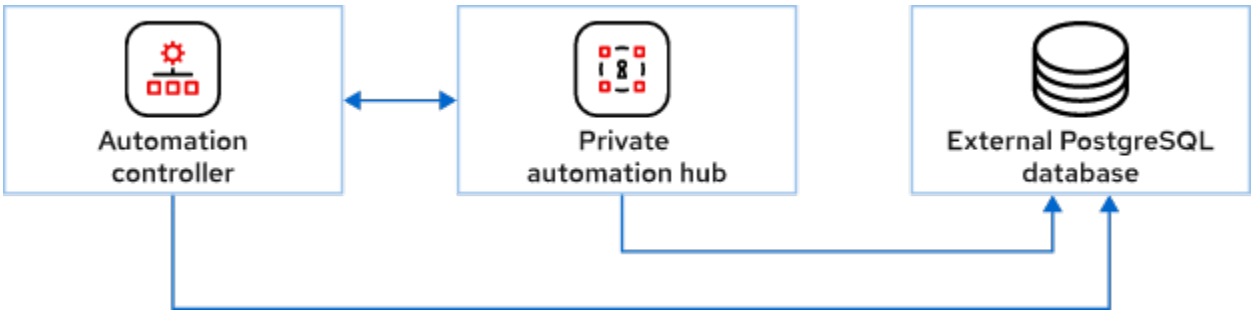

Figure 7. Automation Controller with Private Automation Hub

Automation Mesh Considerations

The Database server must be a separate machine is a requirement for using Ansible Automation Mesh.

Ease of Installation

When installing both Ansible Automation Controller and Private Automation Hub, it is recommended to install both of these items using the same setup command and inventory source. This will allow deployments of both services, but also provides additional benefits with automatic configurations such as creating the "link-style" resources between Controller and Hub like execution environments, and other items such as credentials.

Installation of Automation Controller and Private Automation Hub

In environments having both Automation Controller and Private Automation Hub, it is required that these be installed on separate nodes as they cannot be installed on the same node.

1.2.1.1. Standalone Automation Controller with a Database on the Same Node

1.2.1.2. Standalone Private Automation Hub with a Database on the Same Node

1.2.1.3. Automation Controller and Private Automation Hub with External Database Servers

Classroom and Lab Environment

The deployment for this course will be using both Controller and Hub connected to a shared **External** PostgreSQL database.

1.2.1.4. Advanced Deployment Scenarios

1.2.2. Installation Requirements

Table 1. Hardware Requirements

Machine name	RAM	CPU
Controller	16GB	4 CPUs
Hub	8GB	2 CPUs

Minimum vs. Practical Requirements

The memory and CPU requirements depend really on the size and implementation in the environment. Essentially a good rule of thumb is to have 1GB RAM (memory) for every ten (10) forks and keep at least 2GB for automation controller services. It is also important that with additional forks CPU capacity will also be increased.

Classroom Environment Doesn't Meet Specifications

The classroom environment being utilized doesn't meet the minimum specifications, so during the exercise, the **./setup.sh -e ignore_preflight_errors=true** is run. Specifically, the **-e ignore_preflight_errors=true** instructs the installer to ignore the checks for system requirements. This should not be done in a production environment.

Additionally, for the installation, we are running the setup as **root** since the root user exists on all systems being modified and SSH keys have been pre-distributed.

The classroom environment also uses an internal CA and custom certificates that have been created by the Red Hat Training team. These certificates must be obtained for your environment prior to installation if you want to leverage custom internal CAs.

1.2.2.1. Database Storage

Red Hat recommends at least 150GB storage. If using in the cloud, **m5.large** at a minimum and if managing more than 100 hosts, need **m4.xlarge**.

1.2.3. Subscription and Support

Listing 3. Repository and Channel for RHEL 8

```
subscription-manager repos --enable ansible-automation-platform-2.2-for-rhel-8-x86_64-
rpms
```

1.2.4. Installing Red Hat Ansible Automation Platform

As mentioned above, it is recommended to install both Controller and Hub from the same inventory file using a single **setup.sh** command. In order to successfully install Controller and Hub, the inventory file must be updated and modified providing credentials and FQDMs in the various section headers.

Extra Resources Added when Installed Together

If installed together, the setup script will create additional controller resources such as hub credentials and perform the configuration automatically as part of the installation/setup process. If done separately, it will be necessary to create the resources in controller manually so that controller can communicate with hub. The other benefit of using a combined installation from the bundled installer is that there are three (3) execution environment images (EEIs) included with the bundled installer and these are automatically loaded into hub.

1.2.5. Installing AAP2 Components

When installing, it is suggested to install both items together by making modifications in the single inventory file and then running the **setup.sh** script once.

1.2.5.1. Installing Automation Controller

1.2.5.2. Installing Private Automation Hub

Combined Installation

Installing the components requires modifying the **inventory** file

Listing 4. Ansible Controller and Hub Inventory File

```
[student@workstation Install_AAP2]$ grep -o '^[^#]*' inventory
[automationcontroller] ①
controller.lab.example.com

[automationcontroller:vars] ②
peers=execution_nodes

[execution_nodes] ③

[automationhub] ④
hub.lab.example.com

[automationcatalog] ⑤

[database] ⑥
db.lab.example.com

[sso] ⑦

[all:vars] ⑧
admin_password='redhat'
pg_host='db.lab.example.com'
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='redhat'
pg_sslmode='prefer'
registry_url='hub.lab.example.com'
registry_username='admin'
registry_password='redhat'
receptor_listener_port=27199
automationhub_admin_password='redhat'
automationhub_pg_host='db.lab.example.com'
```

```
automationhub_pg_port=5432
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='redhat'
automationhub_pg_sslmode='prefer'
automationcatalog_pg_host=''
automationcatalog_pg_port=5432
automationcatalog_pg_database='automationservicescatalog'
automationcatalog_pg_username='automationservicescatalog'
automationcatalog_pg_password=''
sso_keystore_password=''
sso_console_admin_password=''

## Certificate Section ⑨
custom_ca_cert=/home/student/certs/classroom-ca.pem
web_server_ssl_cert=/home/student/certs/controller.lab.example.com.crt
web_server_ssl_key=/home/student/certs/controller.lab.example.com.key
automationhub_ssl_cert=/home/student/certs/hub.lab.example.com.crt
automationhub_ssl_key=/home/student/certs/hub.lab.example.com.key
postgres_use_ssl=True
postgres_ssl_cert=/home/student/certs/db.lab.example.com.crt
postgres_ssl_key=/home/student/certs/db.lab.example.com.key
```

① Automation controller definition FQDN or IP

② Automation controller specific variables

③ Execution Nodes (discussed in a later chapter, but defines special execution nodes within the AAP environment)

④ Private automation hub definition FQDN or IP

⑤ Defines automation catalog settings (Beyond scope of the course)

⑥ Section defining FQDN or IP of database server (PostgreSQL server for Controller/Hub)

⑦ Section for SSO (Single-Sign-On which is beyond the scope of this course)

⑧ This is the variables section. At a minimum, FQDNs should be added for various components and also **username**/**password** combinations should be added.

⑨ Typically the end of the file allows specifying things like SSL certificates and configuring the connection security for web apps.

1.2.6. Replacing the CA Certificate

1.2.6.1. Gathering Certificates and Private Keys

1.2.6.2. Preparing the Systems

1.2.6.3. Trusting Custom CA Certificates

Configuring Additional Systems for Trusting Certificates

Copy the CA certificate to the **/etc/pki/ca-trust/source/anchors/** directory on the system and then run **update-ca-trust** to add the trusted CA certificates.

Installing Exercise Time

It can take up to 15 minutes for the GE **lab start install-installation** so it is recommended to run this script at the start of the lecture.

1.2.7. DEMO: Installing Automation Controller and Private Automation Hub

Automation Controller and Private Automation Hub can both be installed from the **same** machine provided that they are both specified in the inventory file and that the installation user and installation machine has access to all systems specified in the **inventory** file and that the user has the ability to SSH/SUDO without passwords.

Automation Hub and Controller Placement

Ansible Controller and Ansible Private Automation Hub must be installed on separate systems and cannot be installed on the same system.

Example 1. DEMO: Installing Automation Hub and Controller

1. Obtain the bundled installer and untar the file

```
[student@workstation ~]$ tar xvf ansible-automation-platform-setup-bundle-2.2.0-6.1.tar.gz

[student@workstation ~]$ mv ansible-automation-platform-setup-bundle-2.2.0-6.1 AAP2

[student@workstation ~]$ cd AAP2/
```

2. Update the inventory file with the system FQDNs or IP Addresses

Listing 5. Update the Inventory File

```
[student@workstation AAP2]$ vim inventory
```

```
[automationcontroller] ①
controller.lab.example.com


[execution_nodes]

[automationhub] ②
hub.lab.example.com

[automationcatalog]

[database] ③
db.lab.example.com


[all:vars]
admin_password='redhat' ④

pg_host='db.lab.example.com' ⑤
pg_port=5432 ⑥

pg_database='awx'
pg_username='awx'
pg_password='redhat' ⑦


registry_url='hub.lab.example.com' ⑧
```

```
registry_username='admin' ⑨
registry_password='redhat' ⑩



# Automation Hub Configuration ⑪
#

automationhub_admin_password='redhat'

automationhub_pg_host='db.lab.example.com'
automationhub_pg_port=5432

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='redhat'
automationhub_pg_sslmode='prefer'

# SSL Settings ⑫

custom_ca_cert=/home/student/certs/classroom-ca.pem
web_server_ssl_cert=/home/student/certs/controller.lab.example.com.crt
web_server_ssl_key=/home/student/certs/controller.lab.example.com.key
automationhub_ssl_cert=/home/student/certs/hub.lab.example.com.crt
automationhub_ssl_key=/home/student/certs/hub.lab.example.com.key
postgres_use_ssl=True
postgres_ssl_cert=/home/student/certs/db.lab.example.com.crt
postgres_ssl_key=/home/student/certs/db.lab.example.com.key
```

① Specify the Controller Node

② Specify the Private Automation Hub Node

③ Specify the Database Node

④ Specify the **admin** password for Controller

⑤ Specify the Database FQDN

⑥ Specify the Database Port

⑦ Specify the Database Password

⑧ URL and Registry for Container Images/Execution Environments

⑨ Username for Registry

⑩ Password for Registry

⑪ Ansible Automation Hub Configuration Settings

⑫ SSL Settings

Database

If you are running the database locally and not as a separate installation, you can leave the database section blank and the **pg_host** and **pg_port** blank. This will cause the installer to setup the database locally with the deployed AAP application.

Registry

Setting the registry for **hub.example.com** will allow the installer to link and configure Ansible Automation Hub to Ansible Controller. It will also ensure that the execution environments container in the bundled installer will be loaded properly into Ansible Automation Hub.

SSL

The classroom and lab environment has been configured to run with SSL enabled. In order for the certificates to work properly, the SSL certificates have been supplied in the **/home/student/certs** directory. These certificates must be specified in the **inventory** file. In the default inventory file, the certificates and SSL settings are generally commented out, so it is possible to just place the certificate information at the bottom of the inventory file to prevent searching for each line.

Listing 6. Default SSL Certificate

```
# SSL-related variables

# If set, this will install a custom CA certificate to the system
trust store.
# custom_ca_cert=/home/student/certs/classroom-ca.pem

# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
```

3. View final inventory file

```
[student@workstation AAP2]$ grep -Ev "^#|^$" inventory
[automationcontroller]
controller.lab.example.com
[automationcontroller:vars]
peers=execution_nodes
[execution_nodes]
[automationhub]
hub.lab.example.com
[automationcatalog]
[database]
db.lab.example.com
[sso]
[all:vars]
admin_password='redhat'
pg_host='db.lab.example.com'
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='redhat'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL
registry_url='hub.lab.example.com'
registry_username='admin'
registry_password='redhat'
receptor_listener_port=27199
automationhub_admin_password='redhat'
automationhub_pg_host='db.lab.example.com'
automationhub_pg_port=5432
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='redhat'
automationhub_pg_sslmode='prefer'
automationcatalog_pg_host=''
automationcatalog_pg_port=5432
automationcatalog_pg_database='automationservicescatalog'
automationcatalog_pg_username='automationservicescatalog'
automationcatalog_pg_password=''
sso_keystore_password=''
sso_console_admin_password=''
custom_ca_cert=/home/student/certs/classroom-ca.pem
web_server_ssl_cert=/home/student/certs/controller.lab.example.com.crt
web_server_ssl_key=/home/student/certs/controller.lab.example.com.key
automationhub_ssl_cert=/home/student/certs/hub.lab.example.com.crt
automationhub_ssl_key=/home/student/certs/hub.lab.example.com.key
postgres_use_ssl=True
postgres_ssl_cert=/home/student/certs/db.lab.example.com.crt
postgres_ssl_key=/home/student/certs/db.lab.example.com.key
```

Using grep to remove comments and blank lines

Listing 7. Source Description

```
grep -Ev "^#|^$" <FILENAME>
```

4. Run the installation **setup.sh** script as the root user with **ignore_preflight_errors=true** as the systems in this course don't meet the minimum hardware requirements.

```
[student@workstation AAP2]$ sudo -i
[sudo] password for student:

[root@workstation ~]# cd ~student/AAP2/

[root@workstation AAP2]# ./setup.sh -e ignore_preflight_errors=true
```

Bundled Software Installer

It is important to at least save the bundled software installer archive **TGZ** file or to save the entire bundled installation directory. In addition, you will also want to save the **Inventory** file that was created so that adding additional components later, performing system backups/restores, and other administrative and maintenance tasks can be performed easily.

5. Install the licenses for Controller by providing the **manifest.zip** file to controller in the WebUI.

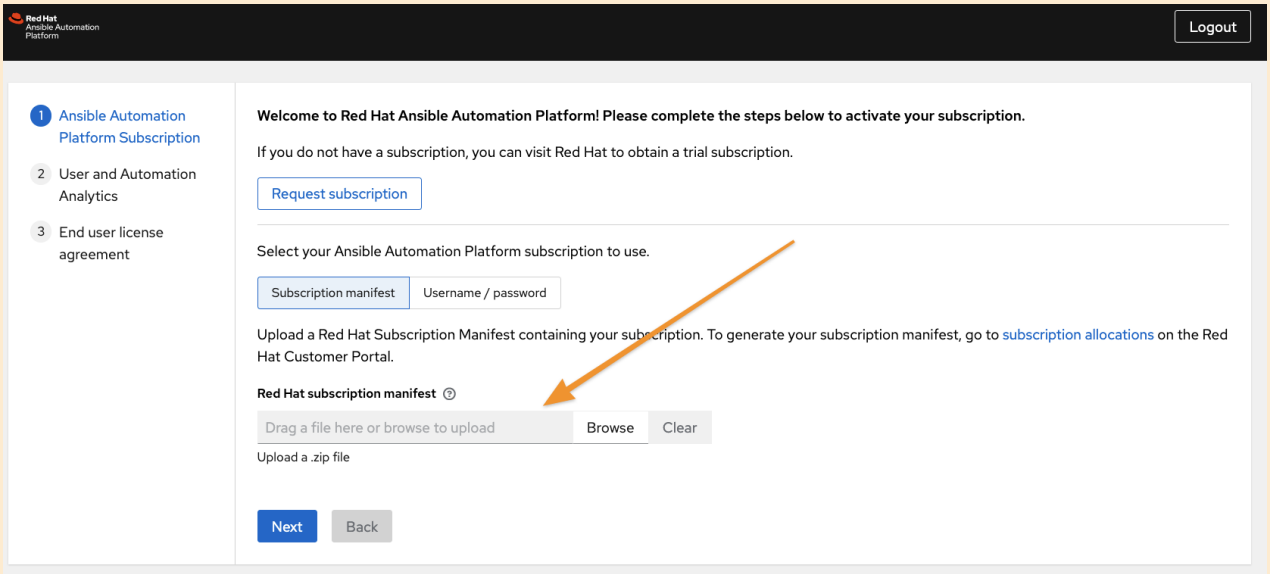

Figure 8. Ansible Controller License

6. Verify **Automation Hub** is installed

1.3. Initial Configuration of Automation Controller and Private Automation Hub

1.3.1. Configuration Overview

Main benefit of AAP2 is the Controller uses execution environments just like developers have tested with **ansible-navigator** so playbooks can run directly on controller without modification. Initial installation will import the base container images for execution environments, but synchronization and some initial configuration is often necessary for custom environments.

1.3.2. Making Automation Execution Environments Available from Private Automation Hub

Private Automation hub provides a container registry where needed execution environment images (EEIs) can be synchronized and stored. The EEIs can also be uploaded manually to private automation hub as well as collections which we will find out about later.

1.3.2.1. Synchronizing Automation Execution Environments

In most instances, you will want to sync the supported EEIs from **https://registry.redhat.io** and get the latest supported versions for the AAP2 platform.

Synchronizing all EEIs

It is possible to synchronize all remote registries getting all versions from a remote catalog by selecting **Index execution environments** from the vertical dots icon.

1.3.2.2. Manually Adding Container Images

There are multiple ways to copy and inspect container images. The **skopeo** command is probably the best, however it can also be done with **Podman**. The benefit of **Skopeo** is that once you are logged into both the remote container registry and private automation hub, it is possible to use **skopeo copy** directly without needing to first download the container image and set the tags.

1.3.2.3. Managing Container Repositories, Images, and Tags

Management of container images within private automation hub can be done through the WebUI or using the **skopeo** command on the CLI. Images can be easily tagged and displayed here.

1.3.3. Synchronizing Ansible Content Collections

Another key piece of AAP2 is the need for content collections. Many modules that were built-in for Ansible have been moved to content collections (firewalld, podman, and networking components and filters). In order to leverage these modules, collections must be installed and available.

Collection Locations

- **Red Hat Certified (Supported) Content Collections** - https://console.redhat.com/ansible/automation-hub
- **Ansible Community (Unsupported) Content Collections provided by Ansible Galaxy** - https://galaxy.ansible.com
- **Homegrown/Manual Collections**: Manually uploaded to private automation hub

1.3.3.1. Synchronizing Red Hat Certified Ansible Content Collections

Login and Credentials required for RH Certified Collections

Red Hat Certified Ansible Collections require a multi-step process.

1. Login to Ansible Automation Platform
   a. Select Collections
   b. Click "Sync"
2. Create an Authentication Token with the **Connect to Hub**
3. Login to Private Automation Hub
   a. Collections ⇒ Repository Management and remote tab
   b. Select **rh-certified** and **Edit** to provide your token.
   c. Click **Save** and then from Repository Management page, select Sync and it will sync all collections marked as Sync.

1.3.3.2. Synchronizing Ansible Content Collections from Ansible Galaxy

Galaxy Doesn't Require Authentication

Since Ansible Galaxy doesn't require authentication, if is possible to configure a **Community** colelction or set of collections by providing a single **requirements.yml** file. This file provides a list of all content collections to synchronize.

1.3.3.3. Manually Adding Ansible Content Collections

In order to manually upload connections, you must first create a **Namespace** in private automation hub.

Collection Security and Signing

The concept of **collection signing** which is signing collection content similar to signing RPMs is currently in Tech Preview for AAP 2.2. This feature is expected to provide an additional level of security with respect to the download content and where it originated from and that it is in its intact and intended format.

1.3.4. Testing Basic Automation Controller Functionality

DEMO Project Benefits

The **Demo** project is essentially there to provide some "smoke" tests allowing a quick way to see if Controller is performing as expected.

1.3.4.1. The Demo Project

Verification of EE and Project Synchronization

The new thing that Controller needs is the ability to use EEIs. The **Demo** project in addition to testing project synchronization components is now able to verify that the EEI can be downloaded and leveraged with Controller.

1.3.4.2. Default Execution Environment Registry Credential

Registry Credential

This is a valid credential and is created based on information at install found in the **inventory** file. This credential cannot be changed from the WebUI and must be modified in the inventory file and have the **setup.sh** script executed again.

1.3.4.3. The Demo Credential

Machine Credential Doesn’t Work

This is the only non-working component in the project. The **Machine** credential will need to be updated with a valid username and password or SSH key so that connections can be made to the remote hosts.

1.3.4.4. The Demo Inventory

Modify Inventory to Add Systems

Initially the **Demo** inventory only contains localhost. It should be modified to include one or more hosts from the environment. Ideally, all hosts would be added so that it is possible to verify Controller connectivity to your entire environment.

1.3.4.5. The Demo Job Template

1.3.5. DEMO: Initial Configuration of Automation Controller and Private Automation Hub

Ensuring EEs Available Locally

It is possible to run the demo without the steps with Skopeo ...

1. Run the **lab start** command

Listing 8. Ensuring EE Tar Files Exist

```
lab start install-configuration
```

2. Execute the Load EE Playbook

```
[student@workstation ~]$ cd ~/Github/DO467_Demo/Demos/Add_EEs/
```

```
[student@workstation Add_EEs]$ ansible-galaxy collection install -r
collections/requirements.yml -p collections/
```

Listing 9. Run the Ansible Playbook

```
[student@workstation Add_EEs]$ ansible-playbook Load_EE_Demo.yml
```

Example 2. DEMO: Initial Configuration of Automation Controller and Private Automation Hub

Working with Execution Environments

Manually uploading and adding container images (EEs) to Ansible Private Automation Hub.

1. Login to Registries to both Push/Pull and Copy container images

```
[student@workstation Add_EEs]$ skopeo login hub.lab.example.com
```

2. Inspect available containers and tags

```
[student@workstation Add_EEs]$ skopeo inspect docker://hub.lab.example.com/ee-29-
rhel8
```

Grabbing Tags and Release Information from the CLI

Listing 10. skopeo inspect to get release and skopeo tags to get tags

```
[student@workstation Add_EEs]$ skopeo inspect
docker://hub.lab.example.com/ee-29-rhel8 --format "{{
.Labels.version }}-{{ .Labels.release }}"
1.0.0-119

[student@workstation Add_EEs]$ skopeo list-tags
docker://hub.lab.example.com/ee-29-rhel8
```

It is also possible to use **podman** to search and list tags, but that is generally considered less reliable. It should also be noted that only **skopeo** has the ability to inspect and act with images remotely. As such, this course will leverage **skopeo** over Podman for many of the exercises.

Listing 11. podman Tag Listing

```
[student@workstation Add_EEs]$ podman search --list-tags
docker://hub.lab.example.com/ee-29-rhel8
```

The skopeo Command

Skopeo is another command that can be used with containers and was introduced as part of the **container-tools** suite with RHEL8. The **container-tools** suite installs the RHEL 8 toolchain to work with containers which includes: **podman**, **buildah**, and **skopeo**.

Testing Automation Controller

1. Create Job Template (**DO467 Demo Job - Testing Inventory**) which launches the **hello_world_demo.yml** Playbook.

```
[student@workstation CH1]$ ansible-navigator run Create_Job_Template.yml
```

2. Login to Automation Controller and navigate to **Resources** ⇒ **Templates**

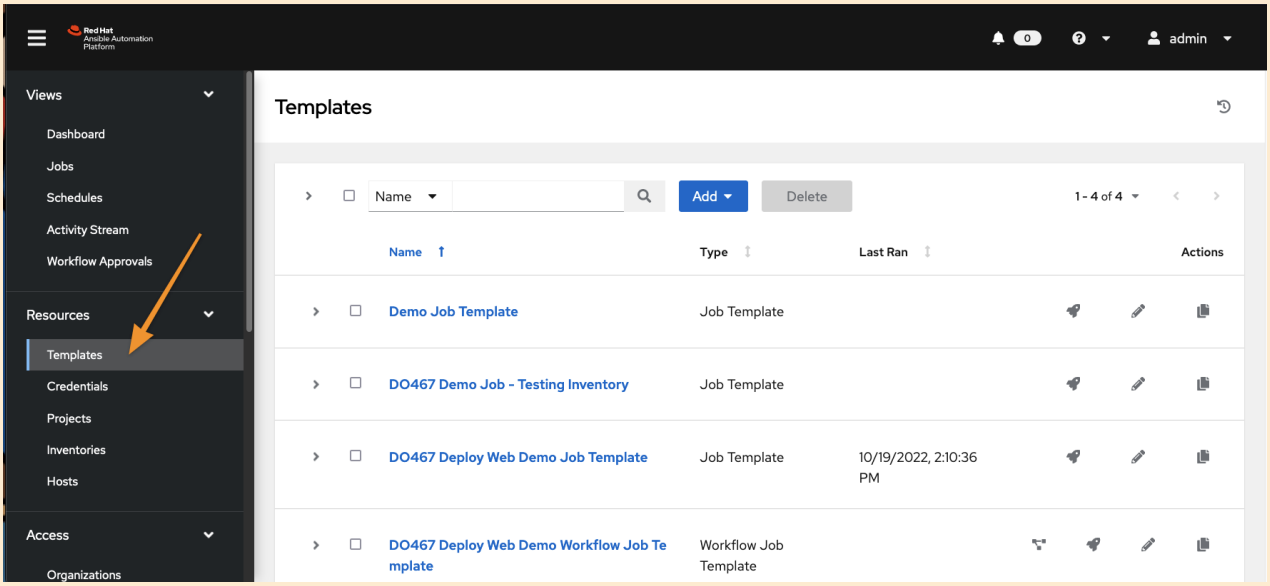

Figure 9. DO467 Demo Job - Testing Inventory

3. Click the **Rocket** to launch the job template and click **Next** for the other prompts and **Launch** on the job preview

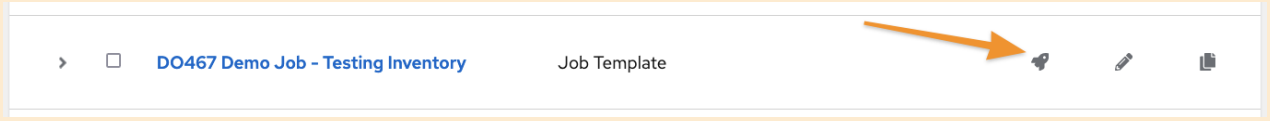

Figure 10. Launching the **DO467 Demo Job - Testing Inventory** *Job*

A Look at the hello_world_demo.yml Playbook.

The **hello_world_demo.yml** playbook is mean to check inventory and show that Automation Controller can execute a playbook on inventory hosts and return data.

Listing 12. hello_world_demo.yml Playbook

```
- name: Hello World Sample Message
  hosts: all
  tasks:
    - name: Hello Message - Demo
      debug:
        msg: "Hello World! I'm a message from {{
inventory_hostname }}!!!"
```

4. Review that all systems are able to be contacted and responded properly based on the job output.

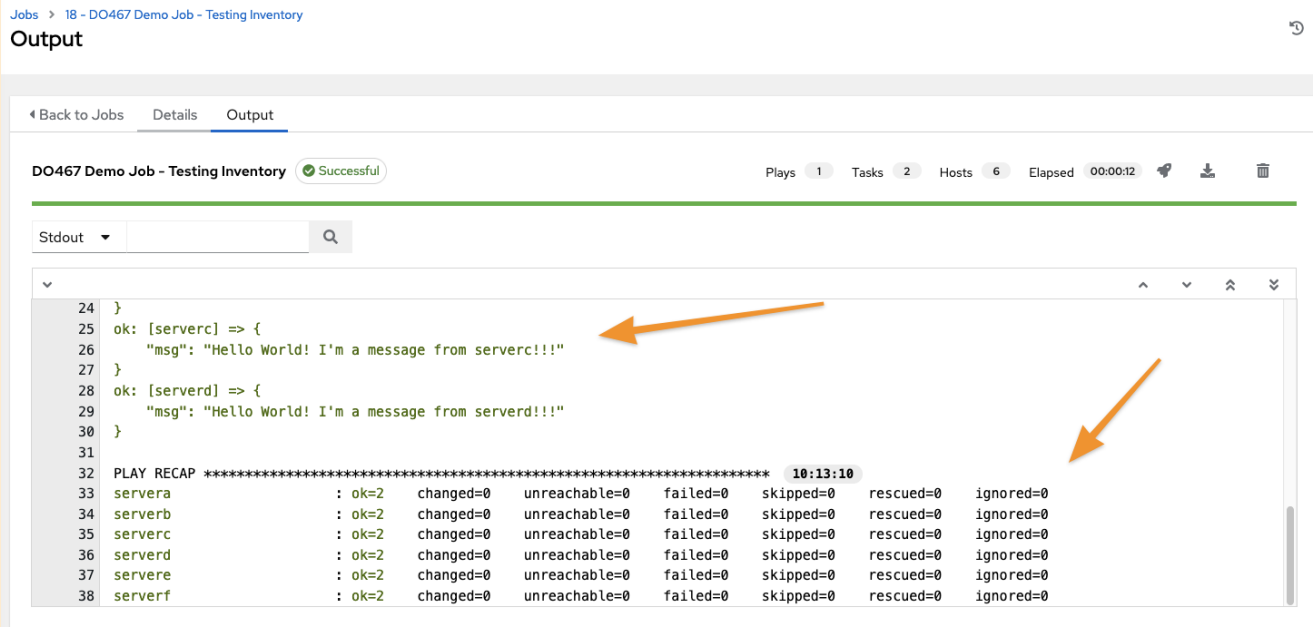

Figure 11. **DO467 Demo Job - Testing Inventory** *Job Output*

Configuring Private Automation Hub

For configuring Private Automation hub, we will want to ensure that custom collections and custom execution environments can be loaded. We will use Quay.io to get the EEI and the built-in synchronization that is provided as part of Private Automation Hub.

Execution Environment Configuration

1. Login to Private Automation Hub and navigate to **Execution Environments** ⇒ **Remote Registries**

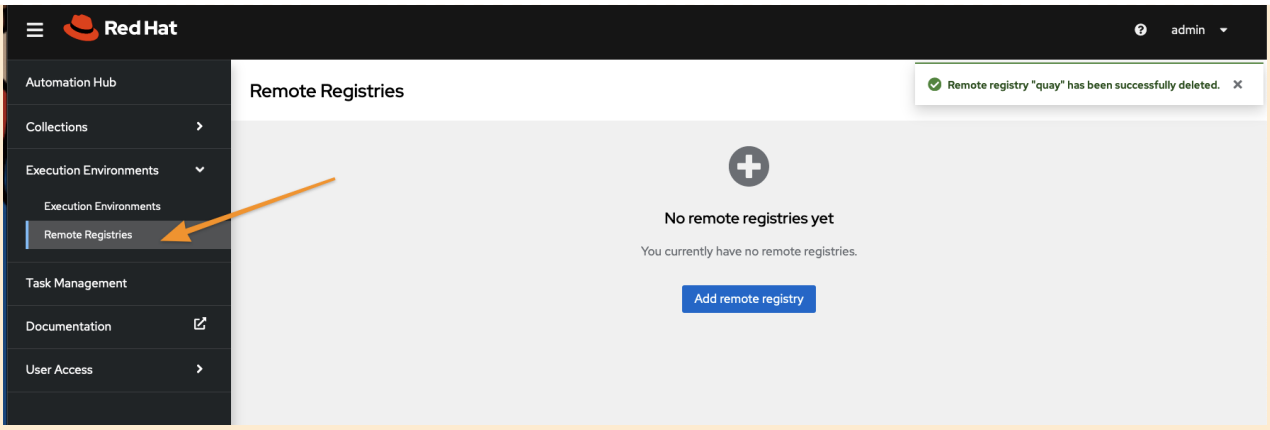

Figure 12. EE Remote Registries

2. Click **Add remote Registry**
3. Complete the fields
   a. Name: **quay**
   b. URL: **https://quay.io/user/tmichett**
   c. Username: **Quay_IO_Name**
   d. Password: **Quay_IO_Password**

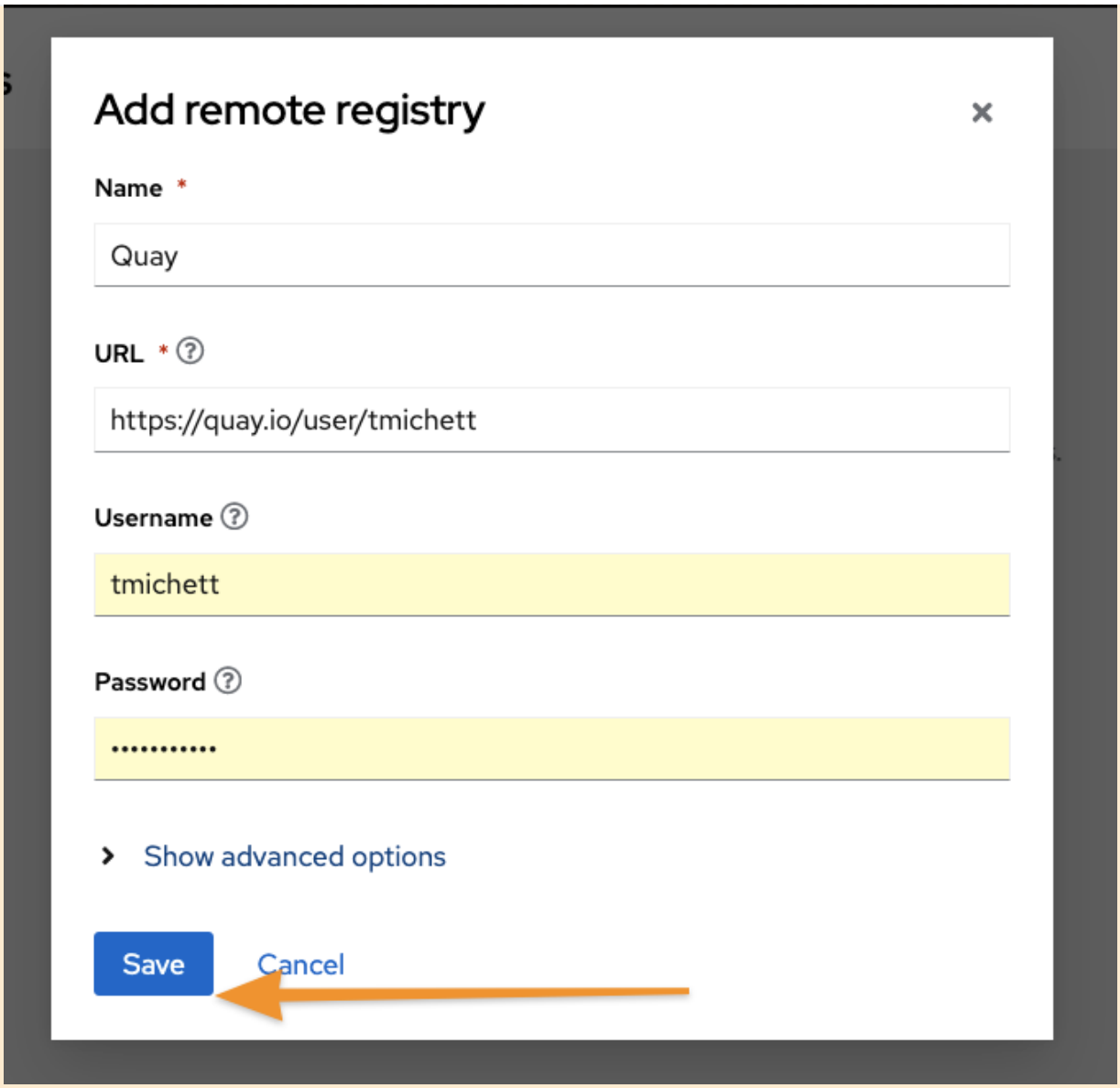

Figure 13. EE Remote Registry Configuration

4. Click **Sync from registry**

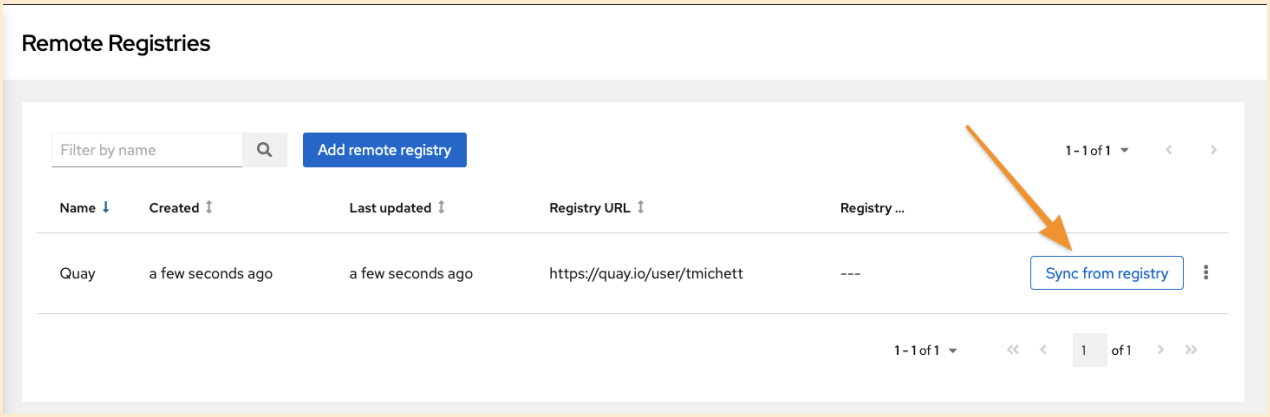

Figure 14. EE Remote Registry Sync

Testing

This ensures that you can synchronize from the specified registry.

5. Navigate to **Execution Environments** ⇒ **Execution Environments**

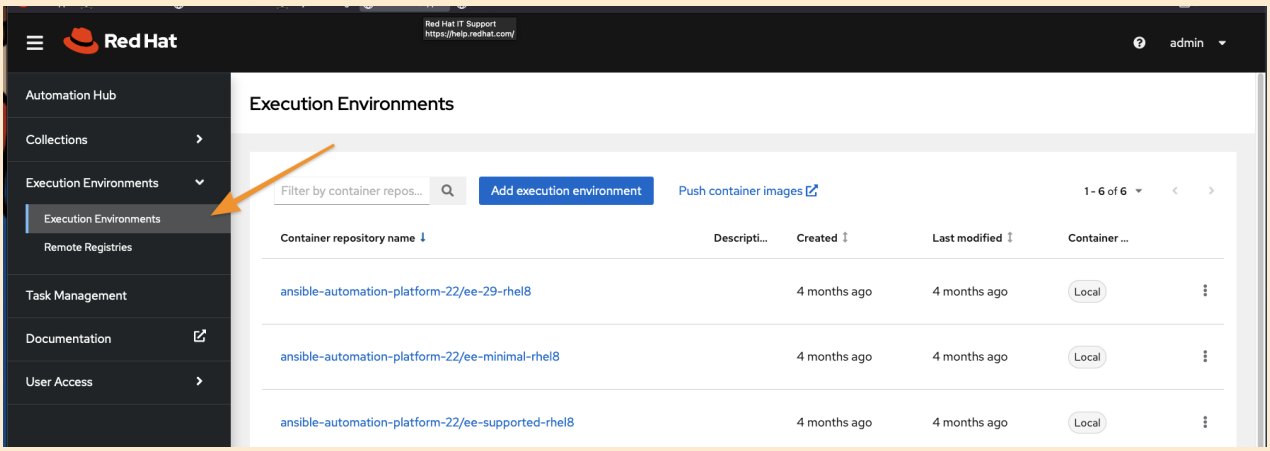

Figure 15. Execution Environment List

6. Click **Add execution environment**
7. Complete the **Add execution environment** Form and then click **Save**
   a. Name: **travis-aap2-demo**
   b. Upstream Name: tmichett/aap2-ee-demo
   c. Registry: **Quay**
   d. Add tag(s) to include: **2.2 latest**

Note Header

- **Name**: Private automation hub name for the execution environment image locally
- **Upstream Name**: The **name** and **repository** of the container. In this instance, we are using **https://quay.io/repository/tmichett/aap2-ee-demo**

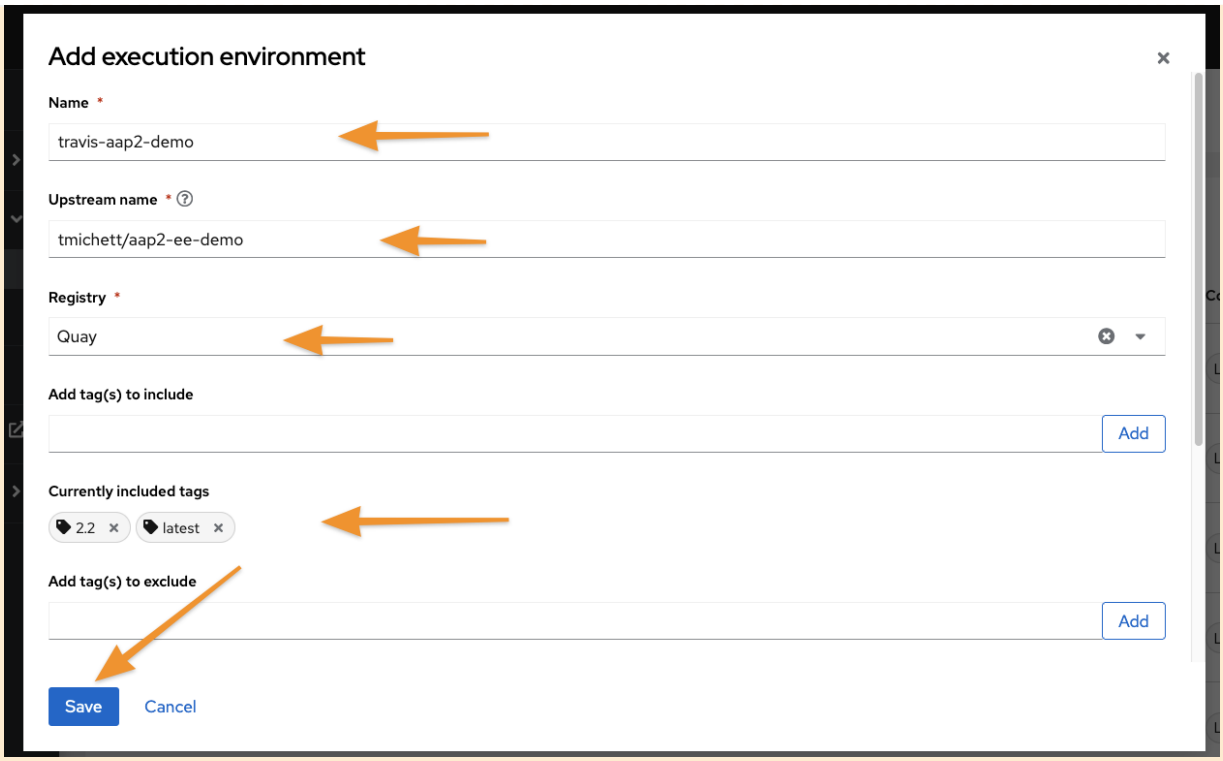

Figure 16. Execution Environment Details

8. Navigate to the new EE, click the stacked **.** and select **Sync from Registry**

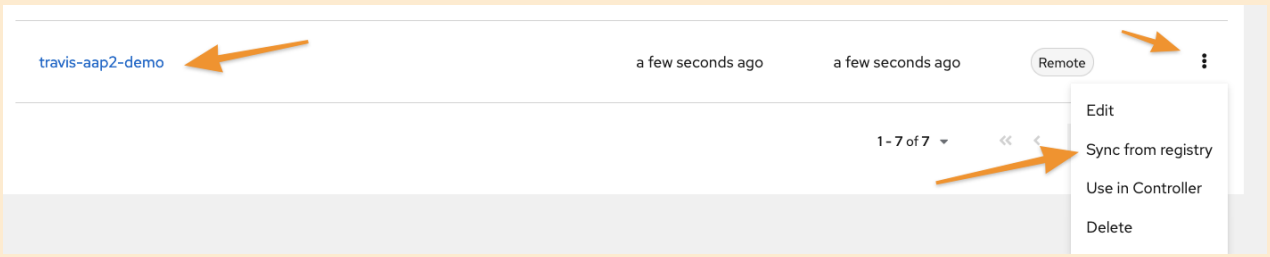

Figure 17. Sync Container Image

9. Verify Image exists by clicking on the **Images** tab

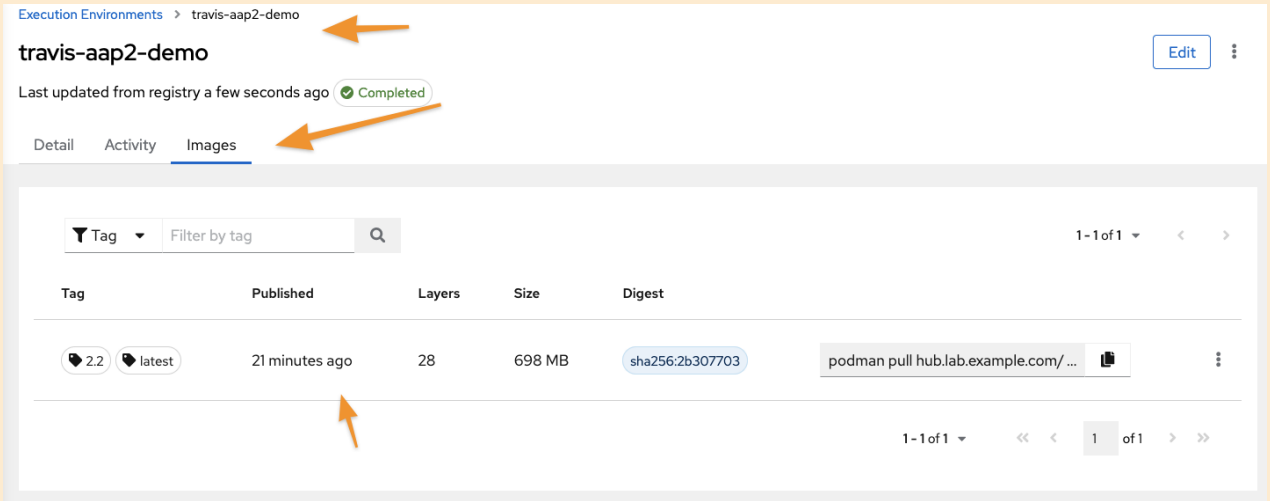

Figure 18. Execution Environment Verification

10. Verify image can be downloaded from Private Automation Hub using **podman pull**

```
[student@workstation ~]$ podman pull hub.lab.example.com/travis-aap2-demo:2.2

... OUTPUT OMITTED ...

Copying config 9e3638983c done
Writing manifest to image destination
Storing signatures
9e3638983c756bc94b6e00fa2f0ef29dd577f274c44072005869a3758d713fab
```

Namespace and Collection Configuration

1. Login to Private Automation Hub and navigate to **Collections** ⇒ **Namespaces**

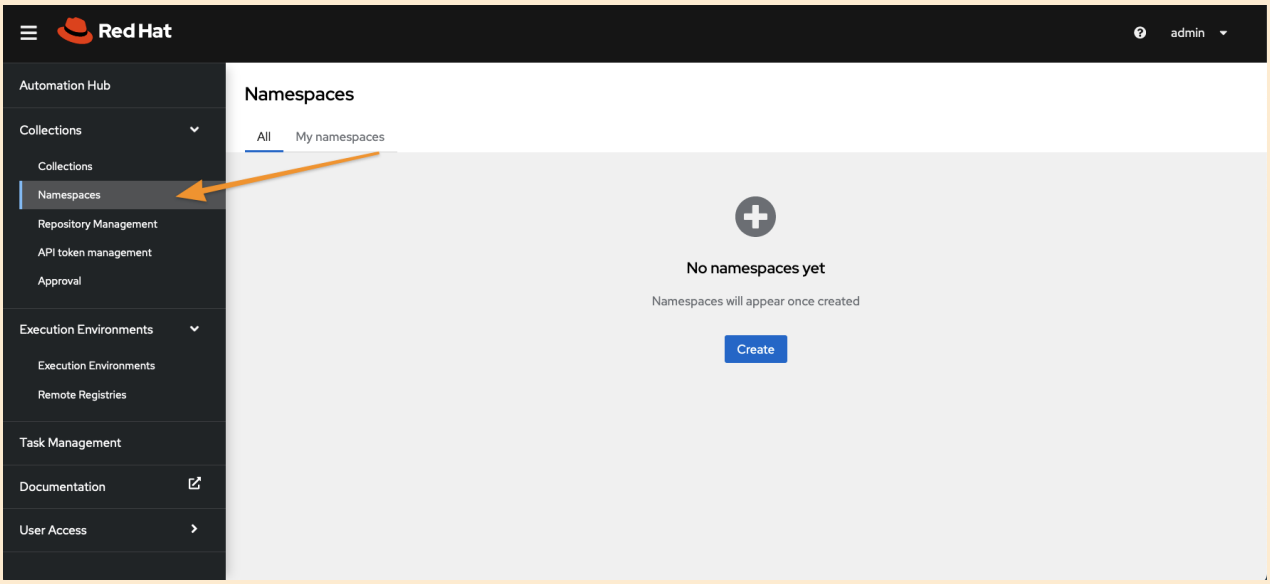

Figure 19. Collection Namespaces

2. Click **Create** to create a new Namespace

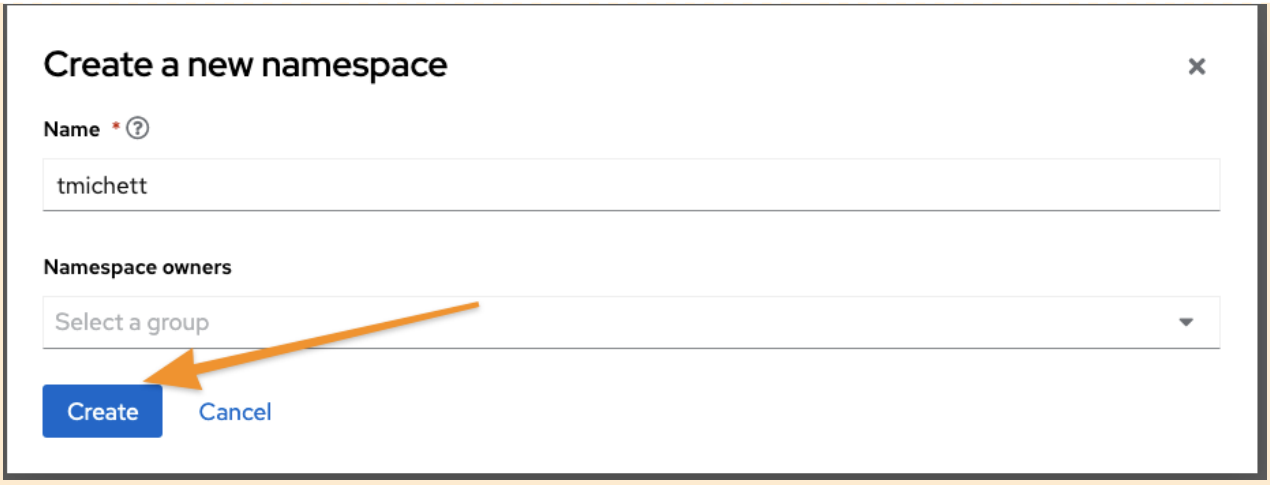

Figure 20. New Namespace

3. Provide a **Name** for the namespace and click **Create**

Namespace Names Must be Correct

If using/importing collections that have been published to Ansible Galaxy, the namespace is defined as part of the collection. The same **Namespace** must be used for Private Automation Hub.

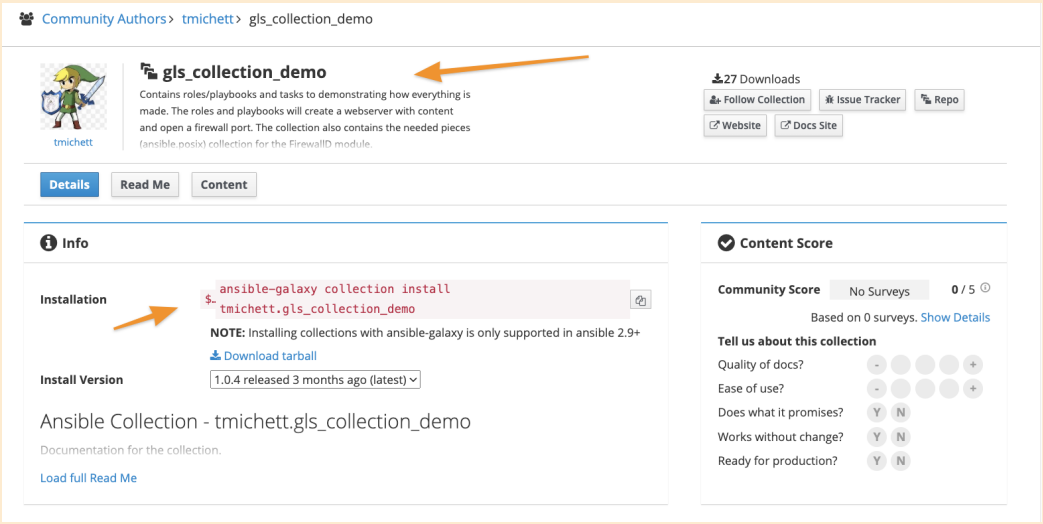

Figure 21. Collection on Ansible Galaxy

4. Once namespace has been created, you can click **Upload collection** to upload the collection

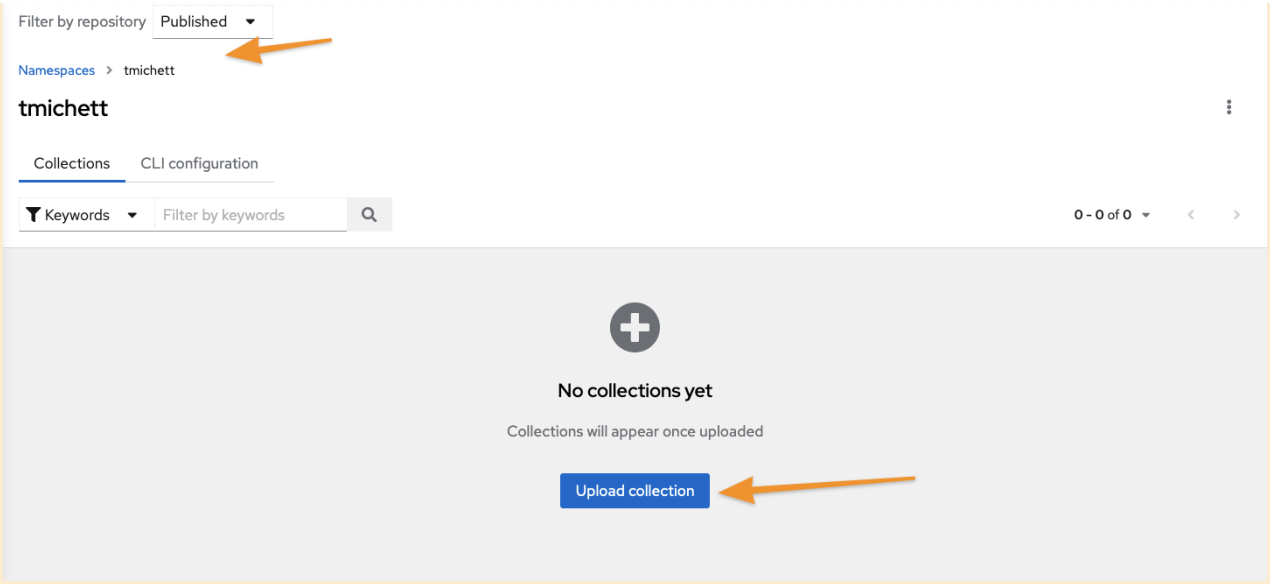

Figure 22. Collection Namespace

5. Select the collection file you want to upload and click **Upload**

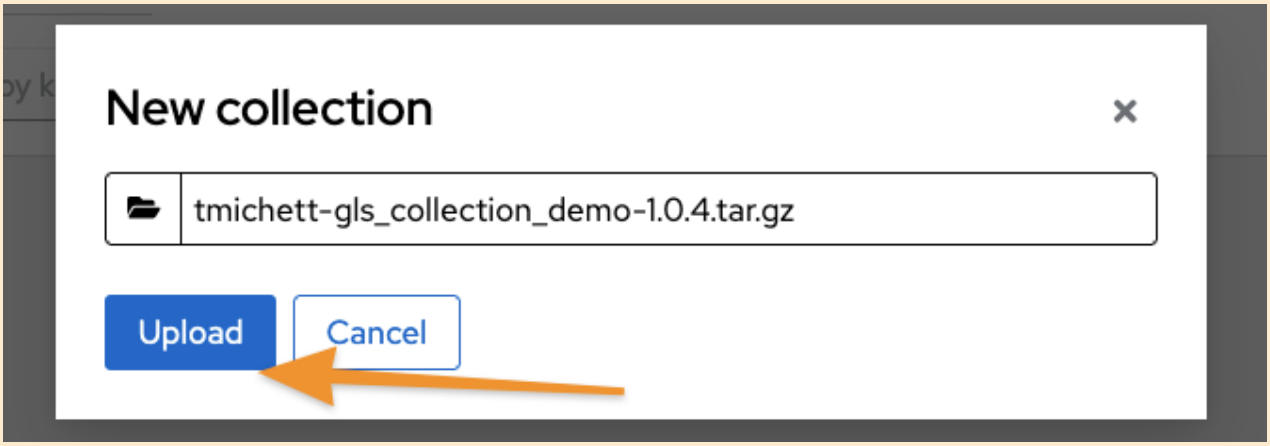

Figure 23. Uploading Collections

6. Verify collection uploaded successfully.

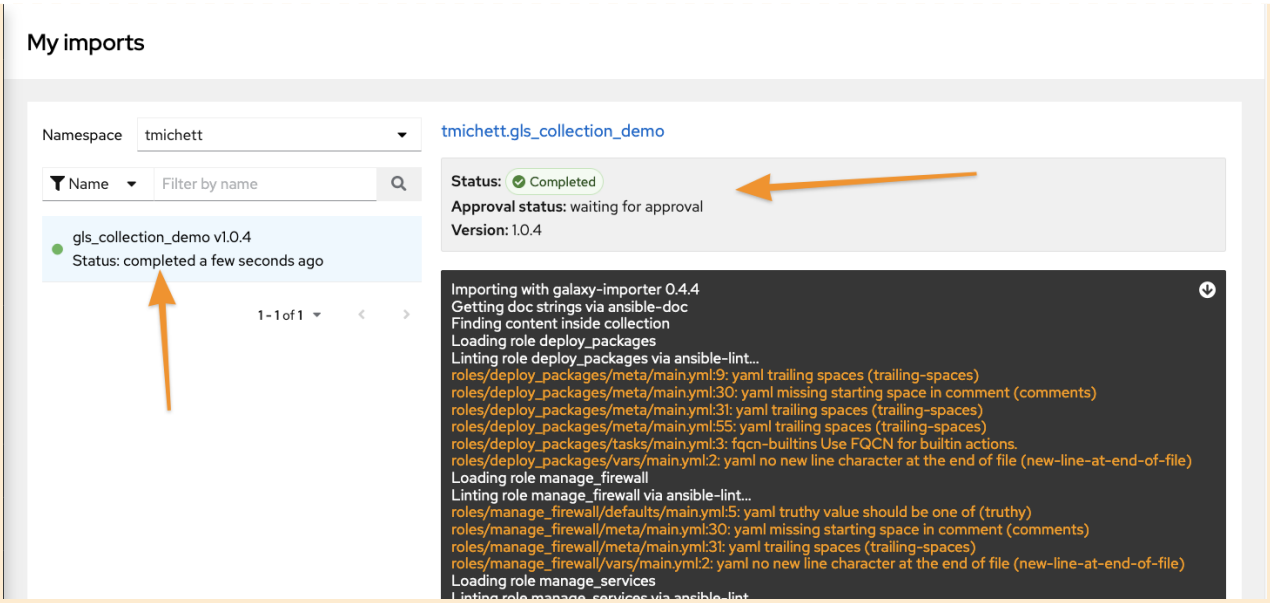

Figure 24. Collection Upload Verification

7. Approve the collection by navigating to **Collections** ⇒ **Approval** and then clicking **Approve**

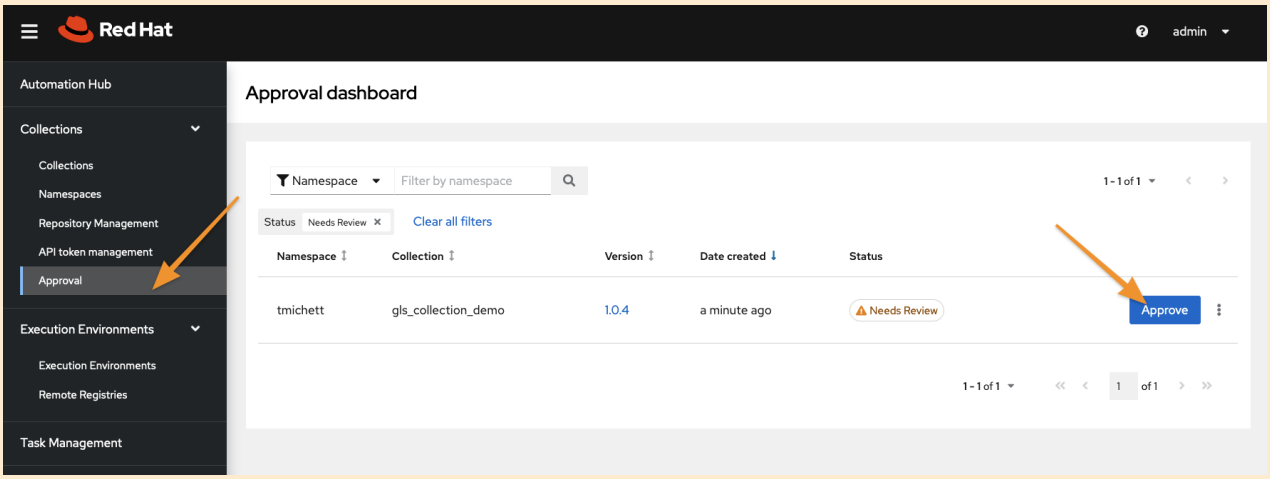

Figure 25. Collection Approval

2. Managing User Access

2.1. Creating and Managing Automation Controller Users

2.1.1. Role-based Access Controls

Users assigned roles which grants one or more permissions (RBAC). Roles can be applied to both teams and users and all users in a team will inherit the team's roles.

2.1.2. Automation Controller Organizations

Top-level component in Controller. Organization can have large numbers of users and teams and this is one way to segregate resources such as teams/users/projects into a logical structure to control access.

Enabling Automated Role and Collection Installation

Roles and Collections can be installed automatically by Ansible Automation Controller. This is an Organization-Level setting and must be configured. It is possible that roles are not automatically installing and this would be because **AWX_ROLES** aren't enabled. To configure automatic installation of roles/collections, you must navigate to **Access** = **Organizations** and "Edit" the Organization.

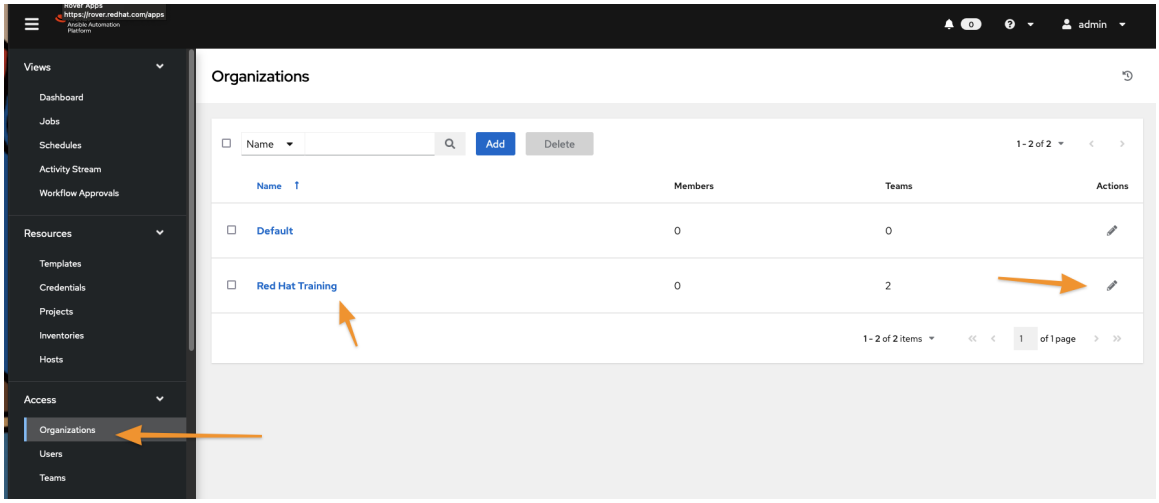

Figure 26. Automation Controller Organizations

You must ensure that **Galaxy Credentials** have been supplied which will then by default enable the automated role/collection installation. It is possible to have multiple credentials. For example, you will most likely have **Ansible Galaxy** and **Private Automation Hub**.

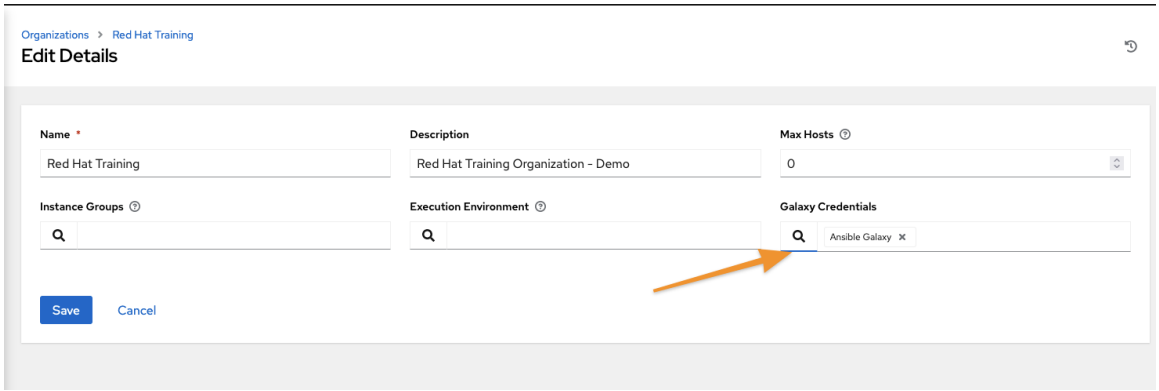

Figure 27. Galaxy Credentials

It should also be noted, in order to function correctly, the **Roles** and **Collections** directory must be at the project's top-level directory.

2.1.3. Types of Users

- **System Adminstrator**: The built in **superuser** role providing unrestricted access to the entire controller installation. This has **read/write** permissions on all objects in controller regardless of

organizations and structure present.

- **System Auditor**: Special **read-only** role with access to everything on the automation controller
- **Normal User**: Standard user with minimal access and no special roles assigned.

2.1.4. Creating Users

Basically done interactively from the WebUI.

Access ⇒ **Users** ⇒ **Add** and then fill in the form.

2.1.5. Editing Users

Access ⇒ **Users** ⇒ **Edit** and then modify values the form.

2.1.6. Organization Roles

Roles within an organization can provide users/teams with additional privileges. Users and teams can be assigned multiple roles to accomplish various tasks and functions.

- **Admin Role**: Provides ability to manage all aspects at the organization level and below. It should be noted that a **System Administrator** automatically inherits an **Admin** role at the organization level.
- **Auditor Role**: Provides read-only access to all aspects at the organization level and below. It should be noted that a **System Auditor** automatically inherits an **Auditor** role at the organization level.
- **Member Role**: Users with this role have **read** access to the organization. There are no special permissions or authorizations given with a member role, so permissions must be assigned to users or teams in order for users to have permissions on organization objects.

2.1.7. Managing User Organization Roles

Organizations can be created and managed from **Access** ⇒ **Organizations** ⇒ **Access** ⇒ **Add**

2.1.8. DEMO: Creating and Managing Automation Controller Users

Automation Controller relies on users and teams to have RBAC assigned to various controller objects.

Example 3. DEMO: Creating and Managing Automation Controller Users

1. Run the **Create_Users.yml** Playbook

```
[student@workstation Add_Users]$ ansible-navigator run Create_Users.yml
```

2. Login to Automation Controller as the Admin User and navigate to **Access** ⇒ **Teams**

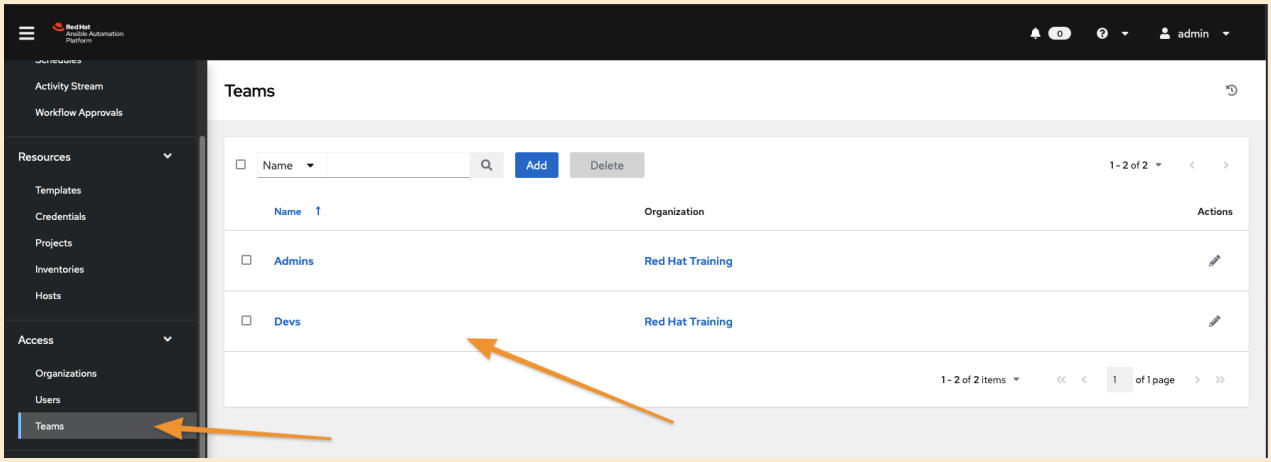

Figure 28. Controller Team List

3. Login to Automation Controller as the Admin User and navigate to **Access** ⇒ **Users**

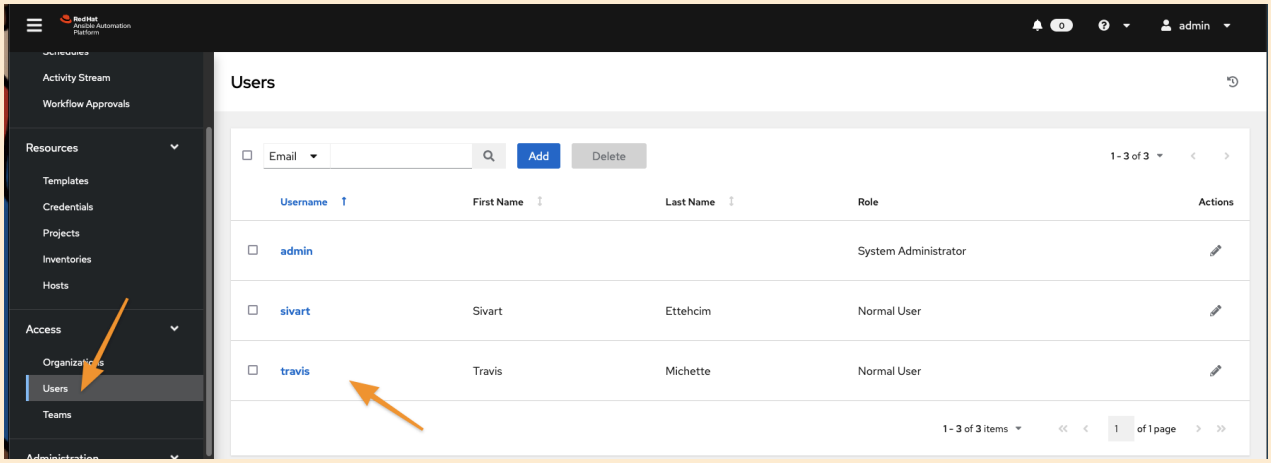

Figure 29. Controller User List

4. Show user role details by clicking on the "travis" user, then going to ROLES

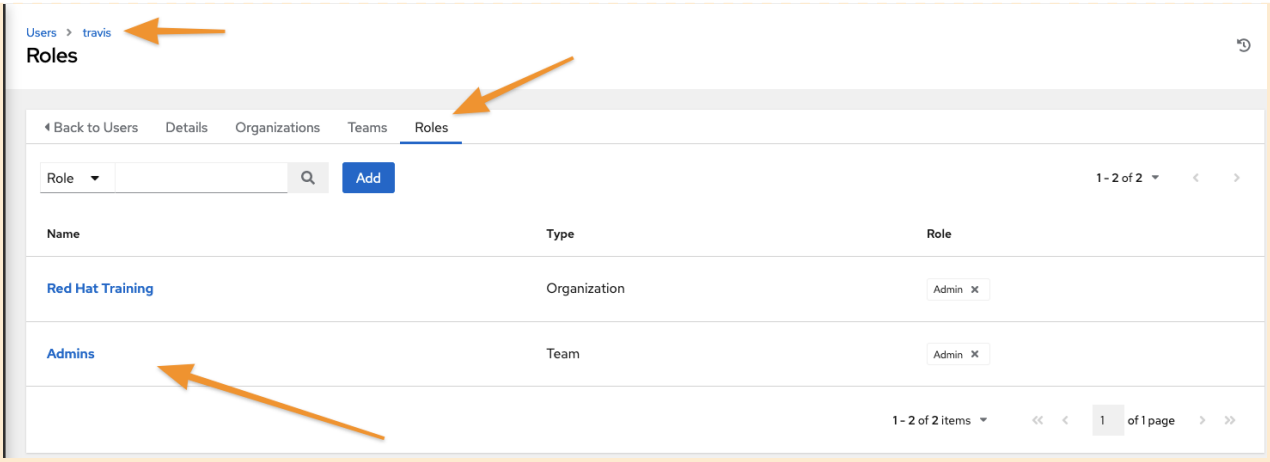

Figure 30. User Roles/Permissions

Continuing the Demos

From here, you can show how to make other users by hand, login/logout as various users and extend demos based on questions.

2.2. Managing Automation Controller Access with Teams

2.2.1. Teams in Automation Controller

Teams are groups of users and provide the ability to assign permissions to team members by assigning one or more roles to a team.

Users vs. Teams

It is important to understand that while a user can belong to one or more organization having a variety of different permissions, a **Team** can belong to exectly one organization.

2.2.2. Creating Teams

Access ⇒ **Teams** ⇒ **Add** to create teams.

2.2.3. Team Roles

- **Member Role**: Inherits roles with resources granted to a team
- **Admin Role**: Full control of a team. This role specifically applies to the team, in order to manage other resources as an admin, the **Admin** role must be assigned to the team for a given resources.
- **Read Role**: Provides ability for team members to view resources and team roles.

2.2.4. Adding Users to a Team and Assigning Team Roles

Access ⇒ **Teams** = **Access** ⇒ **Add** ⇒ **Users** ⇒ **Next** and then complete the form.

2.2.5. Organization Roles

- **Execute**
- **Project Admin**
- **Credential Admin**
- **Workflow Admin**
- **Notification Admin**
- **Job Template Admin**
- **Execution Environment Admin**
- **Auditor**
- **Read**
- **Approve**

2.2.6. Managing Organization Roles

Access ⇒ Organizations ⇒ Access ⇒ Add

2.3. Creating and Managing Users and Groups for Private Automation Hub

2.3.1. User Access

Private automation hub allows administrators to setup and manage granular access to content for end-users. This access is based on managing permissions to system objects.

See permission table in book!!

Superusers

Superusers are assigned all permissions regardless of groups or other permissions assigned.

2.3.1.1. Creating Groups

User Access ⇒ **Groups** ⇒ **Create**

Edit, select permissions and then save.

2.3.1.2. Creating Users

User Access ⇒ **Users** ⇒ **Create** then fill in the form and hit "Save"

2.3.1.3. Creating Groups to Manage Content

3. Managing Inventories and Machine Credentials

3.1. Creating a Static Inventory

3.1.1. Red Hat Ansible Inventory

When leveraging Ansible at the CLI, the inventory is generally specified by one or more inventory files/scripts and defined within the **ansible.cfg** file. These are known as managed hosts. When using Ansible Automation Controller, inventory can be specified within the WebUI or inventories can be provided as part of projects for static files (for example Git repository) or generated dynamically from an external source.

3.1.2. Creating an Inventory Using the Automation Controller Web UI

Licensing Concerns

It is extremely important to remember that Ansible Automation Controller is a licensed and supported project. Each unique entry in an inventory file consumes a single license. So naming conventions for hosts (especially when using Dynamic Inventory) must be considered when developing and constructing inventories.

Listing 13. Inventory

```
servera
servera.lab.example.com
172.25.250.10
```

The above inventory all refer to the same system but referenced in different ways: Hostname, FQDN, and IP address. Since each of these is a unique entry, this inventory file would consume three (3) entitlements instead of just a single entitlement. Therefore, it is important to note how inventories are created and license entitlements are managed.

3.1.2.1. Creating a New Inventory

Resources ⇒ Inventories ⇒ Add ⇒ Add Inventory

3.1.2.2. Creating a Host Group in an Inventory

3.1.2.3. Creating Hosts in an Inventory

3.1.3. Inventory Roles

- **Admin**
- **Update**
- **Ad Hoc**
- **Use**
- **Read**

3.1.3.1. Assigning Roles

Resources ⇒ Inventories ⇒ Access ⇒ Add

Assign users/teams to an inventory and assign one ore more roles, then click "Save"

3.1.4. Inventory Variables

It is possible from the inventory screen to provide **inventory** variables which would apply to all systems within the inventory. It is possible to provide group-based or host-based variables to inventory as well by selecting the hosts or host groups.

Automation Mesh and Instance Groups

The **Instance Groups** is primarily used for Automation Mesh.

3.1.5. DEMO: Creating and Managing Inventories

Credentials are needed prior to creating and running Job Templates. They are also used against inventories which supply the host list for the Job Templates. Inventories are required so that Job Templates and Job Workflows have a list of hosts to run the playbooks on.

Example 4. DEMO: Creating Inventories

1. Run the playbooks provided
   - **Setup_Inventory.yml**

```
[student@workstation CH3]$ ansible-navigator run Setup_Inventory.yml
```

2. Open WebUI and show the inventory that was created with no hosts.

Inventory Module

Point out that the inventory module is generally setup to pull in inventories from source control or other locations. Show how to add inventory manually and mention that "Advanced Inventories" will be covered in an upcoming chapter.

3.2. Creating Machine Credentials for Access to Inventory Hosts

3.2.1. Storing Secrets in Credentials

Ansible automation controller objects can store secrets such as passwords, SSH keys and other credentials. It is also possible to store vault credentials which can then decrypt files from projects.

Encryption and Decryption

Once secrets or sensitive information has been entered into the WebUI and encrypted, it can no longer be retrieved in decrypted form through the WebUI.

3.2.2. Credential Types

- **Ansible Galaxy/Automation Hub API Token**
- **Container Registry**
- **Github Personal Access Token**
- **Machine**
- **Network**
- **Source Control**
- **Vault**

Note on Source Control and PAT

In order to use standard source control Github/Gitlab without SSH keys, it is possible to still use the Username/Password. However, you must create a Personal Access Token (PAT) to be used and this will be done with the **Source Control** credential type and not the **Github Personal Access Token**.

3.2.3. Creating Machine Credentials

Resources ⇒ **Credentials** ⇒ **Add** and complete the form.

3.2.4. Editing Machine Credentials

3.2.5. Credential Roles

- **Admin**
- **Use**
- **Read**

3.2.6. Managing Credential Access

Resources ⇒ **Credentials** ⇒ **Access** and then add users/teams with one or more roles.

3.2.7. Common Credential Scenarios

Credentials Protected by Controller (Not known to Users)

- SSH and Machine Keys
- Sudo keys

Credential Prompts for Sensitive Password, Not Stored in Automation Controller

- Prompts user for passwords at job launch because don't want credentials stored on the controller (Based on compliance or other regulations)

3.2.8. DEMO: Creating and Managing Credentials

Credentials are needed prior to creating and running Job Templates. They are also used against inventories which supply the host list for the Job Templates.

Credentials from Local Files

When using Ansible Execution Environments, sometimes files aren't directly attainable or accessible from the Execution Environment. The **ansible.builtin.slurp** module can read a file and leverage it within the execution environments. The file contents are B64 encoded, so they must be decoded.

Listing 14. Playbook to Create Machine Credentials with SSH Key

```
---
- name: Create Credentials Username and SSH Key
  hosts: workstation

  tasks:
    - name: "Read in SSH Key"
      ansible.builtin.slurp:
        src: /home/student/.ssh/lab_rsa
      register: SSH_KEY

    - name: "DO467 Machine Credentials - CH3 Demo - Key"
      ansible.controller.credential:
        name: "DO467 Machine Credentials - CH3 Demo - Key"
        description: "SSH Username and SSH Key"
        organization: Red Hat Training
        credential_type: Machine
        inputs:
          username: devops
          ssh_key_data: "{{ SSH_KEY['content'] | b64decode }}"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        state: present
```

Example 5. DEMO: Creating Machine Credentials

1. Run the playbooks provided
   - **Create_Machine_Creds_SSH_Key_EE.yml**
   - **Create_Machine_Creds_Password.yml**

```
[student@workstation CH3]$ ansible-navigator run
Create_Machine_Creds_Password.yml

[student@workstation CH3]$ ansible-navigator run
Create_Machine_Creds_SSH_Key_EE.yml
```

2. Open WebUI and show the credentials that were created and explain a few things and items.

4. Managing Projects and Launching Ansible Jobs

4.1. Creating a Project for Ansible Playbooks

4.1.1. Automation Controller Projects

Generally backed by a source control repository (Git) and contains at least one playbook. It is possible for Ansible Controller to automatically download resources and project materials specified by the **requirements.yml** when the job template using the project template is launched.

Automatically Installing Collections and Roles

The automated installation of roles/collections depends on the **roles** or **collections** directory containing a **requirements.yml** file. It further requires that these sub-directories be at the top level of the source control project. Meaning they cannot be nested down within other sub-directories in the project.

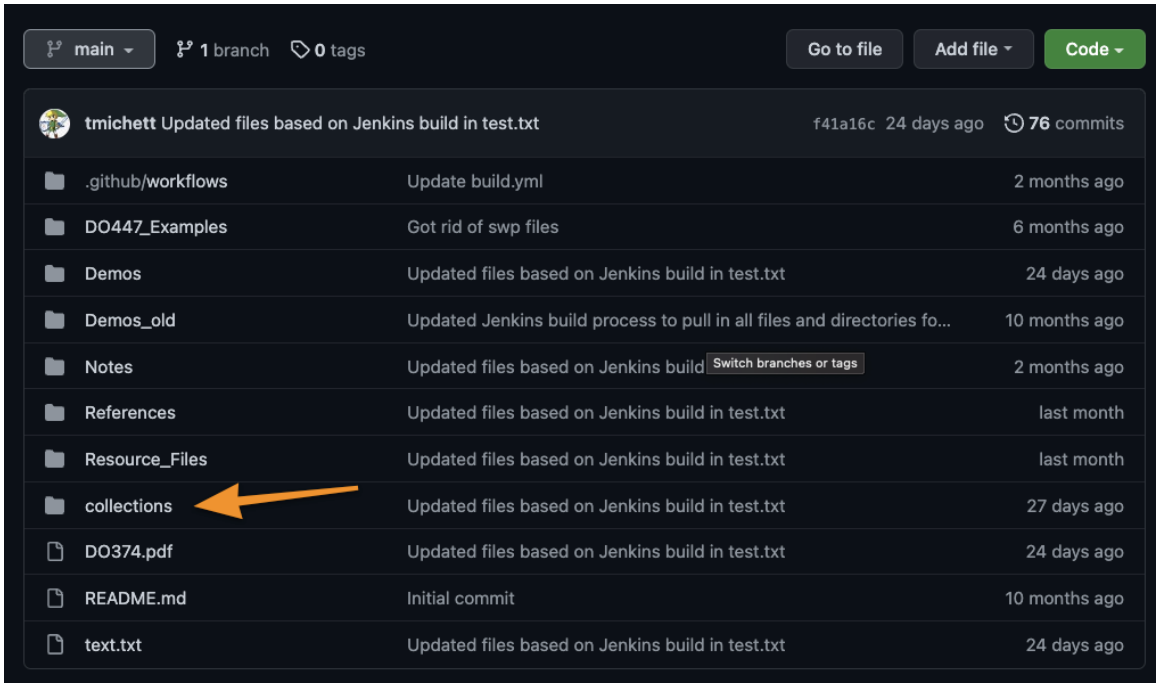

Figure 31. Source Control Project with Collections directory

4.1.2. Creating a Project

Resources ⇒ Projects ⇒ Add ⇒ Create New Project

Complete all the entries and select source control location.

Credentials must exist

The source control credentials must be created in advance of creating a project.

4.1.3. Project Roles

- **Admin**
- **Use**
- **Update**
- **Read**

4.1.4. Managing Project Access

Resources ⇒ Projects ⇒ (name of project) ⇒ Access ⇒ Add

Add the users/teams to the project and select one or more roles.

4.1.5. Creating SCM Credentials

Resources ⇒ Credentials ⇒ Add and complete the form on the new credentials page.

4.1.6. SCM Credential Roles

- **Admin**
- **Use**
- **Read**

4.1.7. Managing Access to SCM Credentials

Resources ⇒ Credentials ⇒ (credential name) ⇒ Access ⇒ Add

Add the users/teams to the credential and select one or more roles.

4.1.8. Updating Projects

SCM projects require copies of playbooks and other version control objects to be replicated locally and stored in Controller. Based on this functionality, it must be determined how project assets get updated.

Clean

Removes any local modifications before pulling latest SCM revision.

Delete

Deletes local copy of repository and clones down a the repository from the remote system.

Allow Branch Override

Allows using items from other branches in the source control.

Update Revision on Launch

Ensures that each time project is used source control is updated before any other actions take place.

Manual Updates

Project source versions can be updated manually or custom workflow nodes can be created when Advanced Job Workflows are made.

4.1.9. Support for Ansible Content Collections and Roles

Ansible Content Collections can be installed automatically at runtime by Controller providing that ...

- There is a **collections/requirements.yml** file present in the project
- Ansible controller has access to the collection source

Ansible Automation Controller Projects

When projects are created they are located in the **/var/lib/awx/projects** directory. Controller also downloads collections and roles based on the **requirements.yml** file and these can be viewed by SSHing as the **awx** user to Controller and looking in the **.__awx_cache** location.

Listing 15. Connection to Controller

```
[student@workstation ~]$ ssh awx@controller
```

Listing 16. AWX Directory

```
[awx@controller ~]$ pwd
/var/lib/awx

[awx@controller ~]$ ls -alF
total 8
drwxr-xr-x. 11 awx  awx   179 Aug 29 15:21 ./
drwxr-xr-x. 53 root root 4096 Aug 29 15:20 ../
drwx------.  3 awx  awx    17 Jun 14 11:42 .ansible/
drwx------.  4 awx  awx    35 Jun 14 11:47 .config/
drwxr-x---.  2 awx  awx     6 May 18 17:21 job_status/
drwx------.  3 awx  awx    19 Jun 14 11:47 .local/
drwxr-x---.  4 awx  awx    91 Aug 29 16:07 projects/
drwxr-xr-x.  3 root awx    20 Jun 14 11:41 public/
drwxr-xr-x.  3 awx  awx    40 Aug 29 15:21 rsyslog/
drwx------.  2 awx  awx     6 Jun 14 11:41 .ssh/
-rw-r--r--.  1 root root    5 Jun 14 11:44 .tower_version
srw-rw----.  1 awx  awx     0 Aug 29 15:21 uwsgi.stats=
drwxr-xr-x.  3 root root   17 Jun 14 11:40 venv/
```

Listing 17. AWX Projects

```
[awx@controller ~]$ cd projects/

[awx@controller projects]$ ls -alF
total 0
drwxr-x---.  4 awx awx  91 Aug 29 16:07 ./
drwxr-xr-x. 11 awx awx 179 Aug 29 15:21 ../
drwxr-xr-x.  5 awx awx 166 Aug 29 16:07 _8__do467_demo_project/
-rwxr-xr-x.  1 awx awx   0 Aug 29 16:07 _8__do467_demo_project.lock*
drwxr-xr-x.  3 awx awx  36 Aug 29 16:07 .__awx_cache/

[awx@controller projects]$ cd .__awx_cache/
```

Listing 18. AWX Projects and Collections

```
[awx@controller 14]$ cd
/var/lib/awx/projects/.__awx_cache/_13__ch4_do467_demo_project/14

[awx@controller 14]$ ls requirements_collections/ansible_collections/
ansible  ansible.posix-1.4.0.info  tmichett  tmichett.gls_collection_demo-
1.0.4.info
```

The Organization MUST have the Galaxy Credentials Configured

In order to "enable" the automatic synchronization of collections and roles, the organization must be configured to allow this. At the Organization-level, it needs to have one or more credentials selected for **Galaxy Credentials**. The **Ansible Galaxy** credential is created by default at install. Depening on how Ansible Private Automation Hub was installed, there may also be some credentials created for using collections stored there.

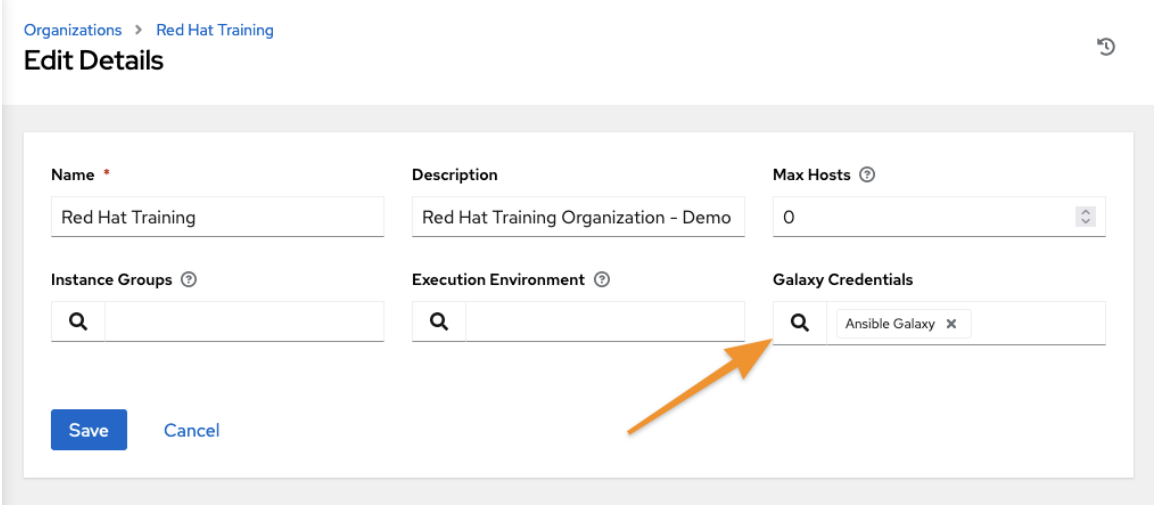

Figure 32. Organization Galaxy Credentials

If projects have **requirements.yml** files and **Galaxy Credentials** have not been defined, the following error will occur on Project sync.

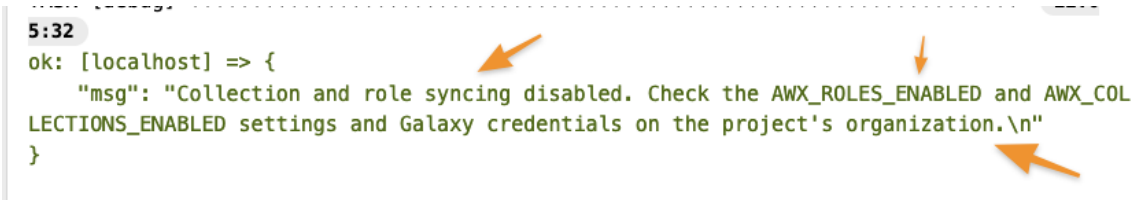

Figure 33. Galaxy Credentials - Sync Error

4.1.10. DEMO: Ansible Automation Controller and Automatic Installation of Collections and Roles

Automation Controller can install roles and collections automatically from a **requirements.yml** file.

Automatic Installation of Collections/Roles

The **collections** and **roles** directory must exist at the top-layer of the project. The contents of this directory will contain a **requirements.yml** file with the collections or roles that need to be installed for the project to work. These directories cannot be nested anywhere else within the project.

Location of Collections and Automated Installation

Collections and roles are installed automatically when creating a project from a version control system providing the directories are at the top level. When installed, they will be placed on the automation contoller **/var/lib/awx/projects** location with the project they support.

In this demo, `/var/lib/awx/projects/.awx_cache/_27ch4_do467_demo_project/` will have the collections installed there. The # will be generated based on the job number for the cache so will be unique for each run.

The `_awx_cache has two (2) underscores ()`.

Example 6. DEMO: Creating an Automation Controller Project

CH4 Project Source

The source project is located at https://github.com/tmichett/AAP2_Controller_Demo. This project will also demonstrate the automatic installation of Ansible Collections using automation controller. It will further provide playbooks that can be leveraged to demonstrate the use of variables and "extra" variables when executing the job template.

Quick Setup

The Project and Source Credentials can be setup quickly from the https://github.com/tmichett/DO467_Demo project. In the **/home/student/Github/DO467_Demo/Demos/CH4** there is a **Demo_Prep.sh** script that will setup the entire CH4 demo. There is also the individual **Create_CH4_Project.yml** that will create and setup a Github project.

Listing 19. Setting up Project Only

```
[student@workstation CH4]$ ansible-navigator run
Create_CH4_Project.yml
```

1. Show the creation of the project in WebUI
2. Show **sync** of project and in the job details, specifically point out where it doesn't have roles to sync, by it finds a **collections/requirements.yml** file and installs the collections automatically.

Examining Structure on Automation Controller

```
[root@controller ~]# cd
/var/lib/awx/projects/.__awx_cache/_27__ch4_do467_demo_project/ ①


[root@controller 70]# ls
requirements_collections/ansible_collections/
ansible  ansible.posix-1.4.0.info  tmichett
tmichett.gls_collection_demo-1.0.4.info ②
```

① Navigate to the project and **awx_cache** directory

② View installed collections

4.2. Creating Job Templates and Launching Jobs

4.2.1. Job Templates

Job templates are predefined settings that are used to launch jobs for running a playbook. A job template allows customization of how a playbook will be run from a given project. Job templates provide all parameters for the execution of a playbook including:

- inventory
- credentials
- execution environment
- variables
- ansible paramters (fork, etc)

This allows jobs to be easily run, scheduled, and maintained.

4.2.2. Creating Job Templates

Resources ⇒ Termplates ⇒ Add ⇒ Add job template

Things to Remember

Be aware of template naming and how job templates are named. Also, it is extremely important to select an **Execution Environment** so that you know which environment will be executing your playbook.

4.2.3. Modifying Job Execution

- **Execution Environment**
- **Forks**: Specifies the **forks** which controls parallel processes. Remember this is a memory dependent item
- **Limit**: Used to restrict specified hosts from inventory defined in job template
- **Fact Storage**: Used to save gathered facts at host level (will be seen in a later chapter)

Will want to mention all of the options, but the above are the most relevant.

4.2.4. Prompting for Job Parameters

When any of the prompt on launch parameters are set, this will create interactive prompts for the user executing the **Job Template** or if set at here will also propagate to the **Job Template Workflow**.

4.2.5. Job Template Roles

- **Admin**
- **Execute**
- **Read**

4.2.6. Managing Job Template Access

Resources ⇒ Templates ⇒ (Job Template Name) ⇒ Access ⇒ Add

Select users/teams and then select one or more roles to add.

4.2.7. Launching Jobs

Resources ⇒ Templates ⇒ (Job Template Name - Launch Template)

4.2.8. Evaluating the Results of a Job

4.2.9. DEMO: Project and Job Template with Prompting for input.

Automation Controller Job Templates can prompt for input such as special variables. This example will show a generic playbook running that installs a webserver, but doesn't create any content. It then uses the **site2.yml** file with a **variable_host** variable that can control where and how the playbook runs.

Example 7. DEMO: Creating a Job Template

CH4 Project Playbooks

The source project is located at https://github.com/tmichett/AAP2_Controller_Demo.

Collections Used:

- tmichett.gls_collection_demo

Playbooks Used:

- site2.yml
- Deploy_WS_Content2.yml
- Deploy_WS_Services_Collection.yml

Quick Setup

The Project and Source Credentials can be setup quickly from the https://github.com/tmichett/DO467_Demo project. In the **/home/student/Github/DO467_Demo/Demos/CH4** there is a **Demo_Prep.sh** script that will setup the entire CH4 demo. There is also the individual **Create_CH4_Project.yml** that will create and setup a Github project.

Listing 20. Setting up Job Template Only

```
[student@workstation CH4]$ ansible-navigator run
Create_CH4_Job_Template.yml
```

1. Create Job template based on the **site2.yml** from the project created earlier.
2. Ensure that **Prompt for Input** on the extra variables has been selected.

Explaining the playbooks

Show the actual playbooks and variables being used. Mention specifically the **variable_host** variable as this will be a variable being prompted for as part of the job launch.

Variable in Playbook

In the **site2.yml** playbook, there is a variable called "variable_host". The playbook is setup to default to **serverb**, however, it is possible to override this variable with the variable prompt.

Listing 21. Variable Host

```
---
- name: Playbook to Fully Setup and Configure a Webserver
  hosts: "{{ variable_host | default('serverb') }}"
  gather_facts: "{{ variable_facts | default('true') }}"
```

5. Advanced Job Configuration

5.1. Improving Performance with Fact Caching

5.1.1. Fact Caching

Unless specifically disabled in the playbook or the Job Template, controller will automatically run the **setup** module to gather facts as the first task just like the **ansible-playbook** command. When optimizing playbooks, it is common to disable gather facts at the playbook level for the play. However, facts are often very useful. It is possible to use **Fact Caching** to obtain facts from the setup module and cache within Ansible Controller.

Cached Facts and Timeouts

There are a few things to keep in mind regarding fact caching

1. There are per-host Ansible Fact Cache Timeouts
2. Cached facts may become stale or outdated so should be updated frequently

5.1.1.1. Enabling Fact Caching in Automation Controller

Fact Cache settings are global at the Automation Controller level and are set from **Settings** ⇒ **Jobs** and then editing the **Per-Host Ansible Fact Cache Timeout**.

Gathering and Caching Facts

If all playbooks and jobs have been optimized and have Fact Gathering disabled, it is possible to create a simple job from a playbook setup to just gather facts. The playbook doesn't even require tasks. The special fact-gathering playbook can be run on regular intervals to ensure that all hosts have relevant up-to-date facts available on the system.

Listing 22. Fact Gathering

```
---
- name: Collect or Refresh Fact Cache
  hosts: all
  gather_facts: true
...
```

Enabling Fact Caching

It is important to note that fact caching must be enabled. To enable fact caching **Resources** ⇒ **Templates** ⇒ **(Job Template Name)** ⇒ **Edit Template** then select **Enable Fact Storage**.

Lab Environment Changes

It may be necessary to change the timeouts in the lab environment. Currently there are issues with the **0** being no timeout and Automation Controller caching facts properly. It may be necessary to set to something like 60 or 360 to get the fact caching to work.

Settings ⇒ Job Settings ⇒ Per-Host Ansible Fact Cache Timeout

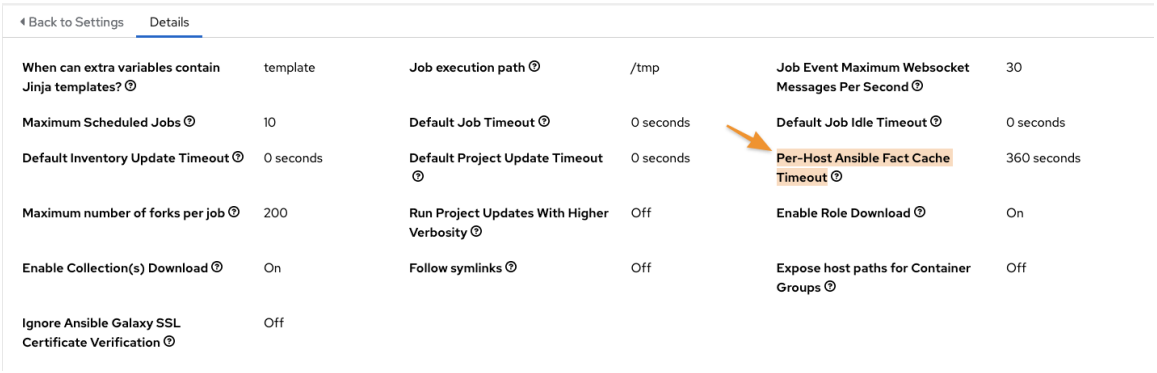

Figure 34. Changing Fact Cache Timeout

5.1.2. DEMO: Ansible Automation Controller Fact Caching

Example 8. DEMO: Using Fact Caching

1. Use the **Gather_Facts_Hello_Demo.yml** to create a job template that will allow demonstrating turning on facts and disabling facts.

Can run a playbook to setup the job for this demo

```
[student@workstation ~]$ cd ~/Github/DO467_Demo/Demos/CH5/

[student@workstation CH5]$ ansible-navigator run
Create_CH5_Job_Template.yml
```

5.2. Creating Job Template Surveys to Set Variables for Jobs

5.2.1. Managing Variables

Job templates can prompt for defining extra variables. However, one thing about this is that the **variable_name** must be known so that you can correctly specify the variable and its value.

vars_promt in Playbooks

The **vars_prompt** playbook directive for interactively specifying variables doesn't work in controller as there is no way to interactively set variables via this means. Instead, you must use the **prompt on launch** or Job template surveys.

5.2.2. Defining Extra Variables

Relaunching Job Templates

When defining the variables with the **Prompt on launch** and then re-running the same job, the same variables and values are used. In order to change the variables or values, a new job must be launched from the Job Template.

5.2.3. Job Template Surveys

Job Template Surveys are available to ask users to specify values for given variables. This eliminates the need for the person to know anything about the playbook, or the variables used in the playbook and also can provide some additional assistance and input validation for variables.

5.2.3.1. Managing Answers to Survey Questions

5.2.3.2. Creating a Job Template Survey

Resources ⇒ **Templates** ⇒ **(Job Template Name)** ⇒ **Survey** ⇒ **Add** and then fill in the form.

Surveys Created via Ansible Playbooks or the API

Currently there is a bug preventing editing of surveys within the WebUI if those have been created using the API/controller module. This bug has been identified as well as a fix, but is not implemented in the current version of AAP 2.2 used for this course. The fix will be released as part of AAP 2.3 and will also be back-ported to AAP 2.1.x and AAP 2.2.x releases.

Github Issue: https://github.com/ansible/awx/issues/12696

Internal Red Hat JIRA for Tracking: https://issues.redhat.com/browse/AAP-5546

Looks like backports are being tracked in the following:

- 2.2 backport being tracked here: https://github.com/ansible/tower/issues/6099
- 2.1 backport being tracked here: https://github.com/ansible/tower/issues/6100

Working Sometimes

There are two separate demo playbooks as part of the repository that setup jobs with surveys. The job setting up a single survey question appears to work.

- Working: **Create_CH5_Survey_Demo_Hello.yml**
- Not-Working: **Create_CH5_Survey_Demo.yml**

Both playbooks will setup working job templates and surveys in Automation Controller, however, if the survey is viewed or edited from the webUI, the job created with **Create_CH5_Survey_Demo.yml** will result in errors in the webUI.

5.2.4. DEMO: Job Template Surveys

Example 9. DEMO: Job Template Surveys

1. Use the **Gather_Facts_Hello_Demo.yml** to create a job template survey that will allow demonstrating turning on facts and disabling facts.

Can run a playbook to setup the job for this demo

This playbook only has a single survey item so it allows you to run the playbook to setup the demo as well as navigate the webUI to show and modify the various items.

```
[student@workstation ~]$ cd ~/Github/DO467_Demo/Demos/CH5/

[student@workstation CH5]$ ansible-navigator run
Create_CH5_Survey_Demo_Hello.yml
```

5.3. Scheduling Jobs and Configuring Notifications

DEMO or GE Notes

Before doing a demo with notifications, you must first run the **lab start job-notification** script as this sets up and configures the e-mail relays and SMTP server in the lab environment. This process takes about five (5) minutes.

```
[student@workstation ~]$ lab start job-notification
```

5.3.1. Scheduling Job Execution

Resources ⇒ **Templates** ⇒ **(Job Template Name)** ⇒ **Schedules** ⇒ **Add** to add the job template schedule.

Scheduling Reminder

Allows launching of Job Templates based on a schedule. Keep in mind, if a template or workflow requires input from a user, then it cannot be scheduled.

5.3.1.1. Temporarily Disabling a Schedule

Schedules can be enabled and disabled from the **Schedules** page.

5.3.1.2. Scheduled Management Jobs

5.3.2. Reporting Job Execution Results

In addition to scheduling, Controller has the ability to provide notifications based on various notification templates.

5.3.2.1. Notification Templates

Notifications can be created for the following:

- Job Start
- Job Success/Failure
- Approvals

5.3.2.2. Creating Notification Templates

Administration ⇒ **Notifications** ⇒ **Add**

5.3.2.3. Enabling Job Result Notification

Resources ⇒ Templates ⇒ (Job Template Name) ⇒ Notifications

5.3.3. DEMO: Notification Templates

Need to Run the Lab Start Script to Setup E-mail

The lab start script will setup the e-mail server for the system. The e-mail server will be the **utility** server, and you can watch the e-mails come into the system by using the **tail -f /var/mail/student** command on the Utility server as the **root** user.

Listing 23. Run the Start Script

```
[student@workstation ~]$ lab start job-notification
```

Example 10. DEMO: Notification Templates

Table 2. E-mail Settings

Field	Value
Host*	localhost
Recipient list	aap-admins@lab.example.com
Sender e-mail	system@controller.lab.example.com
Port	25
Timeout	60

6. Constructing Job Workflows

6.1. Creating Workflow Job Templates and Launching Workflow Jobs

6.1.1. Workflow Job Templates

Job templates run single Ansible playbooks as jobs. In order to run multiple playbooks, you must use a **Workflow Job Template** which allows running multiple playbooks (job templates) in a single sequential workflow. This prevents a user from manually launching jobs in a specified order and makes the process less error-prone and more repeatable.

Workflow Execution

While it is possible to have workflows execute jobs serially (sequentially) it is often the case where there are execution paths based on either success or failures as well as paths requiring approval for a job to run.

6.1.2. Creating Workflow Job Templates

Resources ⇒ **Templates** ⇒ **Add** ⇒ **Add workflow template** and fill in the form.

At a minimum, the Job Template Workflow must have a name.

Using the Workflow Visualizer

The **Workflow Visualizer** allows adding nodes to a workflow. Nodes can be of multiple types which include ..

- Approval
- Inventory Source Sync
- Job Template (actually running a playbook)
- Project Sync
- Management Job
- Workflow Job Template (kicking off another Job Workflow)

Table 3. Workflow Node Relationships

Run Type	Node Relationship
On Success	Node executed when previous run completed successfully

Run Type	Node Relationship
On Failure	Node executed when previous run results in a failure
Always	Node always executes regardless of previous run success/failure.

Adding Multiple Nodes with the Same Relationship

It is possible to have multiple nodes attached with the same relationship.

Creating Convergent Nodes

These nodes rely on one or more previous nodes succeeding.

Node Convergence Requirements

- **Any** - Default setting to where any of the previous nodes success can trigger running job for the convergence node
- **All** - Requires all of the previous nodes to have success success in order to trigger running job for the convergence node

Workflow Job Template Surveys

Method for providing questions to a user running the workflow to interactively set extra variables.

6.1.3. Launching Workflow Jobs

6.1.3.1. Evaluating Workflow Job Execution

6.2. Requiring Approvals in Workflow Jobs

6.2.1. Approval Nodes

Special nodes to obtain approval before proceeding to additional node and job runs. Approval nodes will pause/stop the job workflow and only continue once an approval is given or if the approval node times out the request can be automatically denied.

6.2.2. Adding Approval Nodes to Workflows

Approval nodes are generally added to workflows with the **Workflow Visualizer**

6.2.3. Approving and Denying Workflow Approval Requests

Workflow requests can be approved/denied/monitored. These can be accessed from the **Pending Workflow Approvals** in the WebUI.

Viewing Workflow Approvals (Required Permissions on Workflow Job Template)

- Admin
- Execute
- Read
- Approve
- System Administrator/Auditor

Approving Workflows (Required Permissions on Workflow Job Template)

- Admin
- Approve
- System Administrator/Auditor

6.2.4. Approval Time-outs

Ability to set default action for approval nodes and a timeout if something wasn't approved/denied.

6.2.5. Approval Notifications

Ability to create notifications for e-mail or other type based on approval workflow being required.

6.2.6. DEMO: Constructing Job Workflows Using Ansible

Job workflow templates are relatively straightforward to create within the WebUI. However, when using Ansible, careful planning must be leveraged as once the workflow template has been created, nodes in the workflow must be constructed and the ordering and linkage of these nodes must be considered.

Using Ansible for Job Workflows

When constructing an Ansible playbook, the first task should be to create the **Job Workflow Template**. After the workflow template has been created, then nodes can be created and added to the system.

Listing 24. ansible.controller.workflow_job_template Module

```
- name: Create Job Workflow Template
  ansible.controller.workflow_job_template:
    name: "DO467 Deploy Web Demo Workflow Job Template"
    organization: Red Hat Training
    controller_username: admin
    controller_password: redhat
    controller_host: https://controller.lab.example.com
    validate_certs: false
    state: present
```

Example 11. Analysis: Constructing Job Workflows Using Ansible

The following playbook demonstrates the creation of the **DO467 Deploy Web Demo Workflow Job Template**. Once that is created, it should be noted that we are starting with the last node in the sequence and moving to the first node. This ordering was purposeful to ensure that resources existed within the Automation Controller environment.

Linking Nodes and Building Workflows from a playbook.

Ansible playbook tasks are sequential and require resources to exist in order to manipulate those resources. In this instance, we must start at the last **node** in oder to link based on success throughout the workflow, otherwise, Ansible will fail when trying to link nodes to one that didn't exist.

Listing 25. Job Workflow Template - Playbook

```
---
- name: Create Job WorkflowTemplate
  hosts: localhost

  tasks:

### Create Job Workflow Template

    - name: Create Job Workflow Template ①
      ansible.controller.workflow_job_template:
        name: "DO467 Deploy Web Demo Workflow Job Template"
        organization: Red Hat Training
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        state: present

### Run Job Template for Prod Environment (after approval)
    - name: Create Run Job Template PROD Environment Step ⑧
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        identifier: Execute PROD
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        unified_job_template: "DO467 Deploy Web Demo Job Template"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        extra_data:
          inv_host_var: PROD
```

```
### Create Approval to Prod
    - name: Create Approval to PROD Step ⑦
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        identifier: Approve to Prod
        approval_node:
          description: "Approval to Prod"
          name: Execute PROD
          timeout: 86400
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false

### Run Job Template for QA Environment
    - name: Create Run Job Template QA Environment Step ⑥
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        identifier: Execute QA
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        unified_job_template: "DO467 Deploy Web Demo Job Template"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        extra_data:
          inv_host_var: QA
        success_nodes:
          - Approve to Prod

### Link Approval to Prod
    - name: Link Approval Node to Production  ⑤
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        identifier: Approve to Prod
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        success_nodes:
          - Execute PROD

### Run Job Template for Dev Environment
    - name: Create Run Job Template Dev Environment Step ④
      ansible.controller.workflow_job_template_node:
```

```
        organization: Red Hat Training
        identifier: Execute DEV
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        unified_job_template: "DO467 Deploy Web Demo Job Template"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        extra_data:
          inv_host_var: DEV
        success_nodes:
          - Execute QA

### Synchronize Inventory
    - name: Create Synchronize Inventory Source Step ③
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        identifier: Synchronize Inventory
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        unified_job_template: "DO467 Demo Project Inventory Source"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        success_nodes:
          - Execute DEV

### Synchronize Project
    - name: Create Synchronize Project Step ②
      ansible.controller.workflow_job_template_node:
        organization: Red Hat Training
        identifier: Synchronize Project
        workflow_job_template: "DO467 Deploy Web Demo Workflow Job Template"
        unified_job_template: "DO467 Demo Project"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        success_nodes:
          - Synchronize Inventory
```

① The **ansible.controller.workflow_job_template** is used to create the Job Workflow Template. This creates the initial workflow allowing one or more job workflow nodes to be added.

② The first job workflow node needs to be a **Project Synchronization**. This is the last task in the playbook as it will ultimately link to every node throughout the workflow, so all other nodes must exist prior to creation of this node.

③ On success of project synchronization, the inventory should be synchronized, this is done next to last as the first job template run must exist so that it can be linked as a success node.

④ The deployment to **QA** occurs after test and before the workflow approval. Therefore, this node is between the **DEV** and the **Approval** nodes.

⑤ Approval nodes link to two different nodes as they must come before those nodes. Therefore, the linking of approval nodes to other nodes requires that both the previous node and the success node must exist prior to this task being executed.

⑥ This node is the node responsible for running the push to **QA** and must exist before the approval node as the approval node is the path for "success."

⑦ The approval node is created here. This node waits for approval before moving forward and must be created so that a link can be made through the various **Job Template** nodes, in this case approval from **QA** to **PROD**.

⑧ The last task in the workflow, but the second task in the playbook sets up the **Production** deployment. This workflow must be completed after the **QA** workflow node and the **Approval** node and must exist prior to linking either of those nodes.

Ansible Workflow Nodes

The **ansible.controller.workflow_job_template_node** allows each node to be linked based on a **success**, **failure**, or **always** path. In order to properly link nodes in a single step using Ansible, nodes must first exist. For this reason, tasks can be implemented in reverse order of the workflow to ensure objects exist prior to the task attempting to link them.

7. Managing Advanced Inventories

7.1. Importing External Static Inventories

7.1.1. Importing Existing Static Inventories

Inventories can be added manually in the webUI or retrieved from a source control project.

7.1.2. Storing an Inventory in a Project

When inventory is stored in a project, an inventory source must be created for inventory.

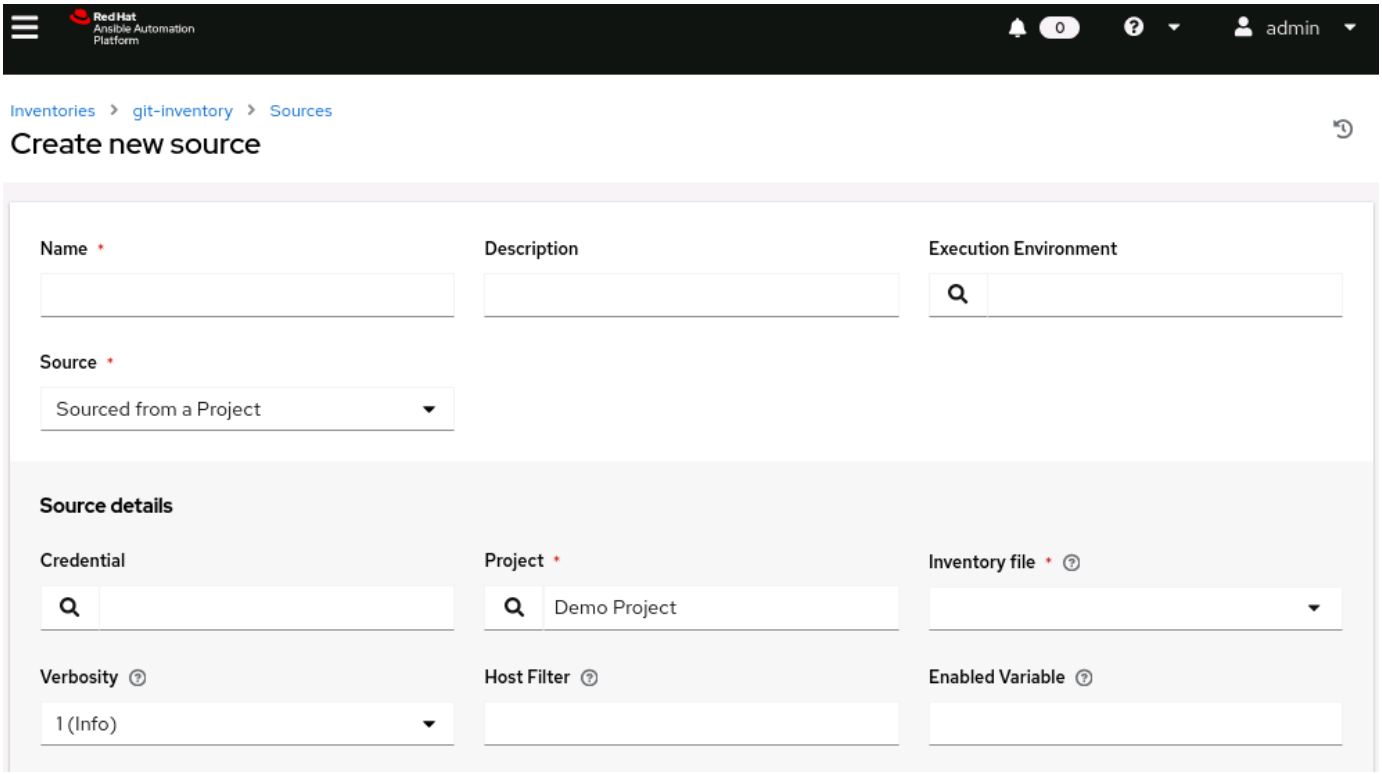

Figure 35. Project Inventory Source

7.1.3. DEMO: Using Project-Based Inventory

Important Header

Need to setup the demo inventory so it imports variables from the project into Ansible Automation Controller. The **CH7 Demo Inventory** will be created.

Listing 26. Creating Demo Inventory

```
[student@workstation Inventory_Project]$ ansible-navigator run
Demo_Setup.yml
```

Example 12. DEMO -Project Based Inventory

1. Change to the Chapter 7 Inventory Project Demo

```
[student@workstation ~]$ cd Github/DO467_Demo/Demos/CH7/Inventory_Project
```

2. View and List Inventory

```
ansible-inventory --list
```

3. View Inventory and Variables for Specific Host

```
ansible-inventory --vars --host servera
```

4. Run Playbook for Inventory Demo

```
[student@workstation Inventory_Project]$ ansible-navigator run Inventory_Demo.yml

PLAY [Create Project]
*********************************************************************************
***********************************

TASK [Gathering Facts]
*********************************************************************************
***********************************
ok: [serverd]
ok: [servera]
ok: [serverb]
ok: [serverc]
ok: [servere]
ok: [serverf]

TASK [Display variables from inventory]
*********************************************************************************
******************
ok: [servera] => {
    "msg": "The variable value for servera for the value of Course SKU is do374"
①
}
ok: [serverb] => {
    "msg": "The variable value for serverb for the value of Course SKU is do467"
}
ok: [serverc] => {
```

```
    "msg": "The variable value for serverc for the value of Course SKU is do467"
}
ok: [serverd] => {
    "msg": "The variable value for serverd for the value of Course SKU is do467"
}
ok: [servere] => {
    "msg": "The variable value for servere for the value of Course SKU is do467"
}
ok: [serverf] => {
    "msg": "The variable value for serverf for the value of Course SKU is do467"
}

TASK [Display variables from hist vars for servera]
*********************************************************************************
******
ok: [servera] => {
    "msg": "The values for servera additional variables are course_user travis and
password redhat" ②
}
skipping: [serverb]
skipping: [serverc]
skipping: [serverd]
skipping: [servere]
skipping: [serverf]

PLAY RECAP
*********************************************************************************
*********************************************
servera                    : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
serverb                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
serverc                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
serverd                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
servere                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
serverf                    : ok=2    changed=0    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

① Notice that the **Course SKU** changed because the host-based variable for Server A had a higher precedence

② Notice that the other variables were displayed as only Server A had those available

After running the playbook and showing everything on the CLI, open Automation Controller's webUI

and show the resulting inventory for the **CH7 Demo Inventory**.

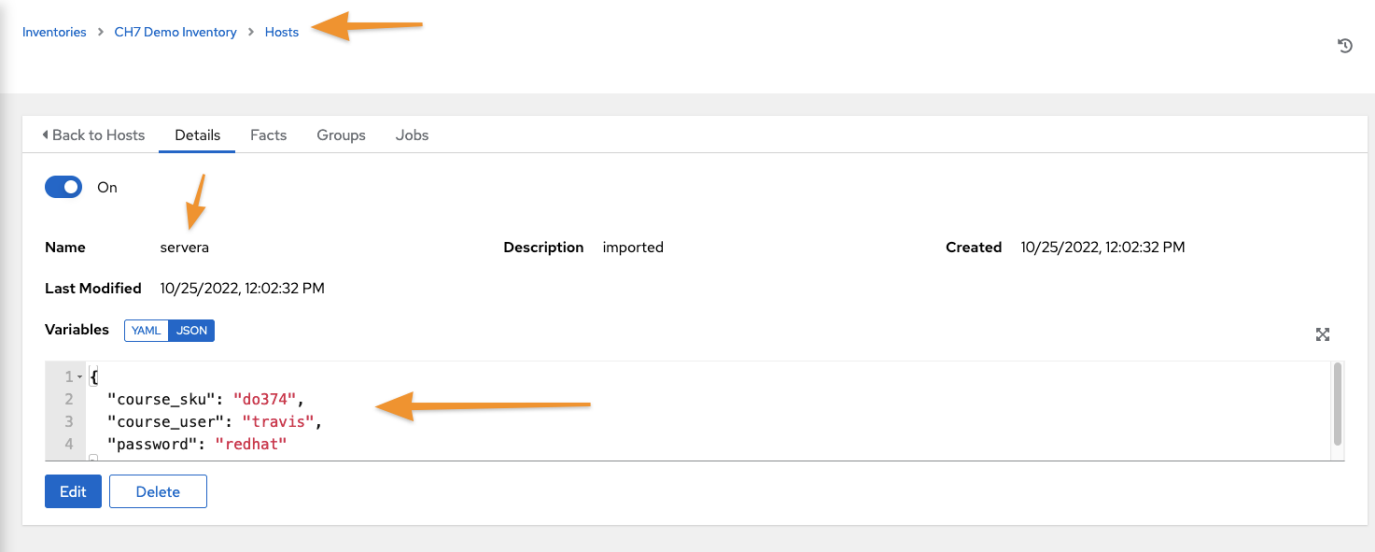

Figure 36. Inventory Host Variables

7.2. Configuring Dynamic Inventory Plug-ins

7.2.1. Dynamic Inventories

Inventories in where hosts and groups can be dynamically imported into automation controller from an external source.

Built-in Inventory Sources

- AWS EC2
- GCP
- Azure
- vCenter
- Satellite
- OpenStack
- RHV
- AAP
- Insights

In addition to built-in inventories, there are multiple inventory plugins available and provided by Ansible Content Collections.

Listing 27. Obtaining Inventory Plugin Documentation

```
[student@workstation Demo_Setup]$ ansible-navigator doc --list -t inventory --eei
hub.lab.example.com/ee-supported-rhel8:latest --mode stdout

advanced_host_list            Parses a 'host list' with ranges
amazon.aws.aws_ec2            EC2 inventory source
amazon.aws.aws_rds            rds instance source
ansible.controller.controller Ansible dynamic inventory plugin for the Automation...

... OUTPUT OMITTED ...
```

7.2.2. OpenStack Dynamic Inventories

7.2.3. Red Hat Satellite 6 Dynamic Inventories

Dynamic Inventory Scripts

AAP2 is attempting to move away from dynamic inventory scripts as AAP2 leverages execution environments and python packages and other libraries might not exist and therefore cannot be run from within the execution environment.

7.3. Filtering Hosts with Smart Inventories

7.3.1. Defining Smart Inventories

Smart inventories are a way of creating a new dynamic inventory based on already existing inventories. Generally smart inventories key off of facts about systems, so **Enable Fact Storage** option must be selected to save facts.

7.3.2. Using Ansible Facts in Smart Inventory Filters

Smart inventories use filters selecting hosts based on specified criteria, many of those criteria are based on Ansible facts.

Creating a Smart Inventory Based on Ansible Facts

Resources ⇒ Inventories ⇒ Add ⇒ Add smart inventory

7.3.3. Other Smart Inventory Filters

It is possible to create simple filters based on names of systems and host groups which do not require Ansible facts.

8. Automating Configuration of Ansible Automation Platform

8.1. Configuring Red Hat Ansible Automation Platform with Collections

8.1.1. Automating Red Hat Ansible Automation Platform Configuration

Benefits

- Rebuilding from scratch
- Version control and historical changes
- Configuration-as-Code to validate and test infrastructure before deployment to production

The **ansible.controller** collection is a certified and supported collection provided by Red Hat which allows the automation, management, and configuration of Ansible Automation Controller. There are also community collections such as **awx**, but those are not officially supported by Red Hat.

Subscriptions

In order to obtain the **ansible.controller** collection you must have a valid subscription entitlement and be able to download the collection from Ansible Automation Hub.

8.1.2. Getting the Supported Ansible Content Collection

- Part of the **ee-supported-rhel8** EE
- Available from the **ansible.controller** collection from private automation hub or Ansible Automation Hub at: https://console.redhat.com/ansible/automation-hub

8.1.3. Exploring the Supported Ansible Content Collection

The documentation for the modules and plugins for the **ansible.controller** collection can be viewed in two ways:

- Using **ansible-navigator collections** to view the documentation interactively
- Using the web browser and searching for the documentation on Ansible Automation Hub

8.1.3.1. Reading Documentation with Ansible Content Navigator

Listing 28. Reading Documentation

```
ansible-navigator collections --eei hub.lab.example.com/ee-supported-rhel8
```

8.1.3.2. Reading Documentation on Automation Hub

Login to Ansible Automation hub and search for the **ansible.controller** collection.

ansible.controller Collection

It is possible to use a username/password combination or OAuth2 tokens to authenticate to Ansible Controller. The **OAuth2** token is the preferred method.

Listing 29. Ansible Controller (tower_cli.cfg) Syntax

```
[general]
host = https://localhost:8043
verify_ssl = true
oauth_token = LEdCpKVKc4znzffcpQL5vLG8oyeku6
```

https://console.redhat.com/ansible/automation-hub/repo/published/ansible/controller/docs

8.1.4. Examples of Automation with ansible.controller

All **ansible.controller** collection modules require authentication. The table below shows the options that can be in a configuration file or in the playbook itself.

Table 4. **ansible.controller** *Collection - Authentication Settings*

Option	Description
controller_config_file	File that can be provided from localhost to authenticate to Ansible Controller
controller_host	The FQDN or IP address of the controller host
controller_oauthtoken	Allows authentication with **Oauth** token. If this is present, there is no need for **controller_username** and **controller_password** to be provided.
controller_password	Password for the controller user

Option	Description
controller_username	Username with permissions on controller to perform the given task
validate_certs	**true/false** for validation of SSL certs

ansible.controller Modules

By this point in the course, students should have seen several example playbooks from the demo setup and become very familiar with the various modules in the **ansible.controller** collection. However, if you chose to ignore some of the Ansible playbooks to setup demos and explanations for Chapters 1-7, you will need to focus in more detail on explaining the modules as part of this chapter.

8.1.4.1. Creating Automation Controller Users

ansible.controller.user

8.1.4.2. Creating Automation Controller Teams

ansible.controller.team

8.1.4.3. Adding Users to Organizations and Teams

Once users and teams are added to the system, roles must be assigned to give authorization to perform given tasks.

ansible.controller.role

8.1.5. Community-supported Ansible Content Collections

Community collections available on Ansible Galaxy. The following are community collections (not supported) by Red Hat that are available to perform various tasks. The collections can provide additional capabilities for management of Ansible Automation platform.

- **redhat_cop.controller_configuration** - Red Hat Community of Practice collection for configuring Ansible Automation Controller (https://github.com/redhat-cop/controller_configuration) (https://galaxy.ansible.com/redhat_cop/controller_configuration)
- **redhat_cop.ah_configuration** - Red Hat Community of Practice collection for configuring Ansible Automation Controller (https://github.com/redhat-cop/ah_configuration) (https://galaxy.ansible.com/redhat_cop/ah_configuration)

All Red Hat CoP Collection Resources on Ansible Galaxy

https://galaxy.ansible.com/redhat_cop

8.1.6. DEMO: Configuring Red Hat Ansible Automation Platform with Collections

The demo from this section will show how to create users for an Organization using the Ansible Controller collection.

Ansible Content Collections

The **ansible.controller** is a supported Ansible Content collection and is available as part of the **ee-supported-rhel8**. If access isn't available to this content collection, the **awx** collection can be obtained from Ansible Galaxy.

Example 13. DEMO: Configuring Red Hat Ansible Automation Platform with Collections

1. Login to **hub** and run **ansible-navigator collections** for the RHEL8 supported EE.

Listing 30. Reading Documentation

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

[student@workstation ~]$ ansible-navigator collections --eei
hub.lab.example.com/ee-supported-rhel8
```

2. Locate the collection you want to review documentation and type in that number (if >9, then **:** followed by a number [ex. **:27**])

Listing 31. Collection List

```
   Name                Version Shadowed Type  Path
 0 | amazon.aws          3.2.0   False    contained
/usr/share/ansible/collections/ansib
 1 | ansible.builtin     2.13.0  False    contained /usr/lib/python3.9/site-
packages/ans
 2 | ansible.controller  4.2.0   False    contained
/usr/share/ansible/collections/ansib
 3 | ansible.netcommon   3.0.0   False    contained
/usr/share/ansible/collections/ansib
 4 | ansible.network     1.2.0   False    contained
/usr/share/ansible/collections/ansib
 5 | ansible.posix       1.3.0   False    contained
/usr/share/ansible/collections/ansi
```

Listing 32. ansible.controller Collection

```
  Ansible.controller      Type      Added Deprecated Description
 0 | ad_hoc_command          module    4.0.0 False      create, update, or destroy
Auto
 1 | ad_hoc_command_cancel   module    None  False      Cancel an Ad Hoc Command.

 2 | ad_hoc_command_wait     module    None  False      Wait for Automation
Platform Co

... OUTPUT OMITTED ...

27 | organization           module None  False      create, update, or destroy
Automat
28 | project                module None  False      create, update, or destroy
Automat
29 | project_update         module None  False      Update a Project in Automation
Pla

... OUTPUT OMITTED ...
```

Listing 33. ansible.controller Organization Module

```
Image: ansible.controller.organization
Description: create, update, or destroy Automation Platform Controller
organizations
  0 | ---

  1 | additional_information: {}

  2 | collection_info:
  3 |   authors:
  4 |   - AWX Project Contributors <awx-project@googlegroups.com>

... OUTPUT OMITTED ...

163 | examples: |-
164 |   - name: Create organization
165 |     organization:
166 |       name: "Foo"
167 |       description: "Foo bar organization"
168 |       state: present
169 |       controller_config_file: "~/tower_cli.cfg"

... OUTPUT OMITTED ...
```

3. Users can be created with the **ansible.controller.tower_user** Module

Listing 34. ansible.controller.tower_user Module

```
    - name: Create Users
      ansible.controller.tower_user:
        username: "{{ item.username }}"
        password: "{{ item.password }}"
        email: "{{ item.email }}"
        first_name: "{{ item.first_name }}"
        last_name: "{{ item.last_name }}"
        controller_username: admin
        controller_password: redhat
        controller_host: https://controller.lab.example.com
        validate_certs: false
        state: present
      loop: "{{ users }}"
```

Listing 35. tower_users.yml Variables File

```
---
users:
  - first_name: Sheldon
    last_name: Cooper
    username: scooper
    email:  scooper@redhat.com
    password: redhat123

  - first_name: Frodo
    last_name: Baggins
    username: frodo
    email:  frodo@redhat.com
    password: redhat123
```

8.2. Automating Configuration Updates with Git Webhooks

8.2.1. Introducing Red Hat Ansible Automation Platform Webhooks

Webhooks are used-defined HTTP callbacks. Ansible automation controller can use webhooks to be notified when something in a Git repository has changed. New commits added to a specific branch or other things might trigger a webhook.

8.2.1.1. What Are the Benefits of Webhooks

A variety of webhooks can be created. A common webhook would be for a version control system. Changes to a version control system would occur, be pushed to the main repository, and then automation controller could be notified via the webhook.

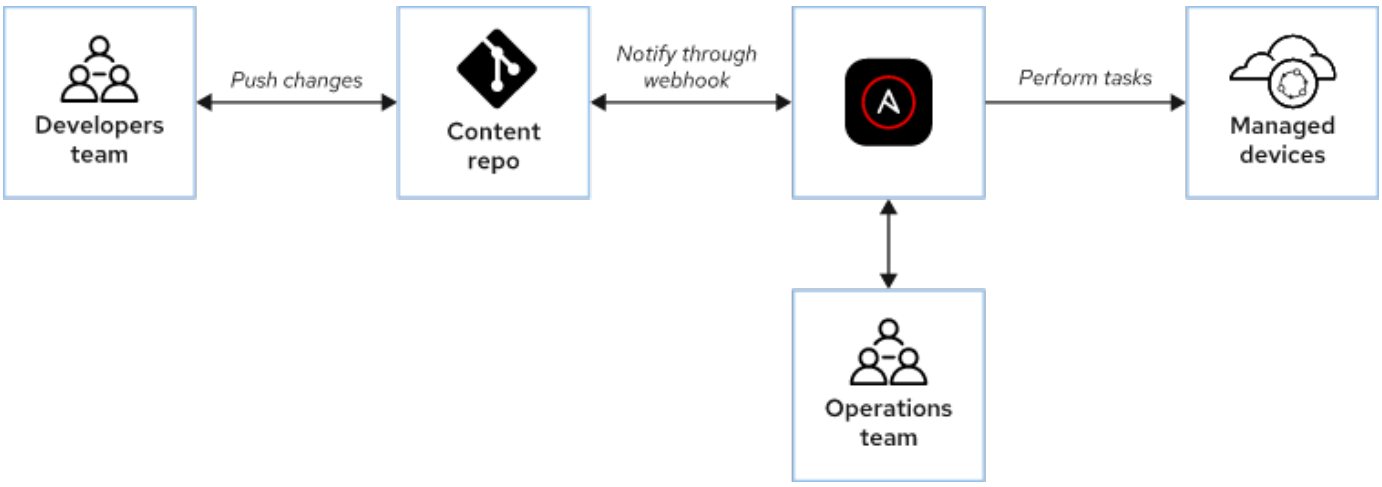

Figure 37. Version Control Webhooks

8.2.2. Configuring Webhooks

In order to leverage webhooks, both automation controller and the remote system must be configured for webhooks. In this instance, Gitlab must create a webhook to trigger and automation controller must be configured to listen.

Configuring a Webhook for a Job Template

In order to use webhooks, projects and jobs must be configured to meet certain requirements.

Requirements * Project must have **Update Revision on Launch** to ensure controller pulls latest revision. * All "Prompt on Launch" settings for a job template must be disabled

Templates ⇒ **<Job Template Name>** ⇒ **Enable Webhook** then choose Gitlab from the Webhook service.

Creating the Webhook for the Repository in GitLab

Before creating webhooks in Gitlab, the administrator must allow webhooks to be used.

Requirements * Admin Area ⇒ **Settings** ⇒ **Network** ⇒ **Outbound requests** and configure local IP addresses and domain names that can be accessed.

Projects ⇒ **Your Projects** ⇒ **<Project Name>** ⇒ **Settings** ⇒ **Webhooks** and then complete the form information paying close attention to the **Trigger** section.

Testing Webhook

After the hook has been created, you should **Test** ⇒ **Push Events** to ensure that it is correctly executed in automation controller.

8.2.3. Use Cases for Using Webhooks

- Triggering Different Job Templates Using Branches

Branches

The **Source Control Branch** is only available if the project is configured to **Allow Branch Override**.

- Configuration as Code for Automation Controller

8.2.4. DEMO: Automating Workflows with Webhooks

Gitlab Configuration

In order for Gitlab to be configured properly for allowing Gitlab/Ansible Automation Controller Webhooks, the configuration on Gitlab for settings as administrator must allow communication on the network. The **lab start code-webhooks** will complete the configuration automatically of Gitlab. Run this script prior to the demo.

You will also need to run the **Site.yml** playbook out of the Chapter 8 Demo directory.

Listing 36. Ensure Gitlab Setup for Webhooks (*takes about 5 minutes*)

```
[student@workstation ~]$ lab start code-webhooks
```

Listing 37. Login to Private Automation Hub

```
[student@workstation ~]$ podman login hub.lab.example.com -u admin -p
redhat
```

Listing 38. Run the site.yml Playbook with Ansible Navigator

```
[student@workstation ~]$ cd
/home/student/Github/DO467_Notes/Demos/CH8/Webhooks

[student@workstation Webhooks]$ ansible-navigator run Site.yml
```

More details available in the Github Readme for Chapter 8 Webhook Demos: https://github.com/tmichett/DO467_Demo/tree/main/Demos/CH8/Webhooks

Example 14. DEMO: Automating Workflows with Webhooks

1. Configure Webhook in Auomation Controller
   - **Resources** ⇒ **Templates** ⇒ ***CH8 Deploy Web Demo Workflow Job Template*** ⇒ **Enable WebHook (Gitlab)** and hit save.

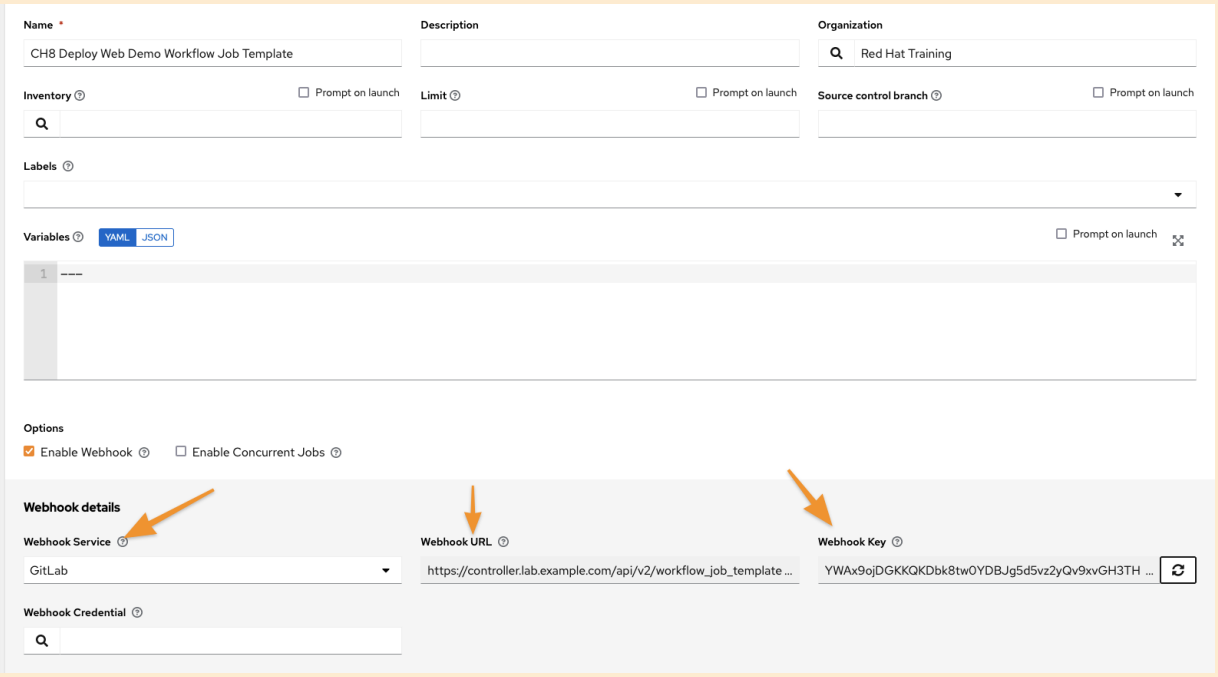

Figure 38. Creating Webhook

2. Copy Credentials and information to Gitlab and create the Webhook for the **aap2_demo** project.
   - Select **AAP2_Demos** Project
   - Click **Settings**
   - Click **Webhooks**
   - Enter in the **URL** and **Secret Token**
   - Ensure **Push events** is checked for Trigger.
   - Ensure **Enable SSL verification** is unchecked.
   - Click **Add Webhook**

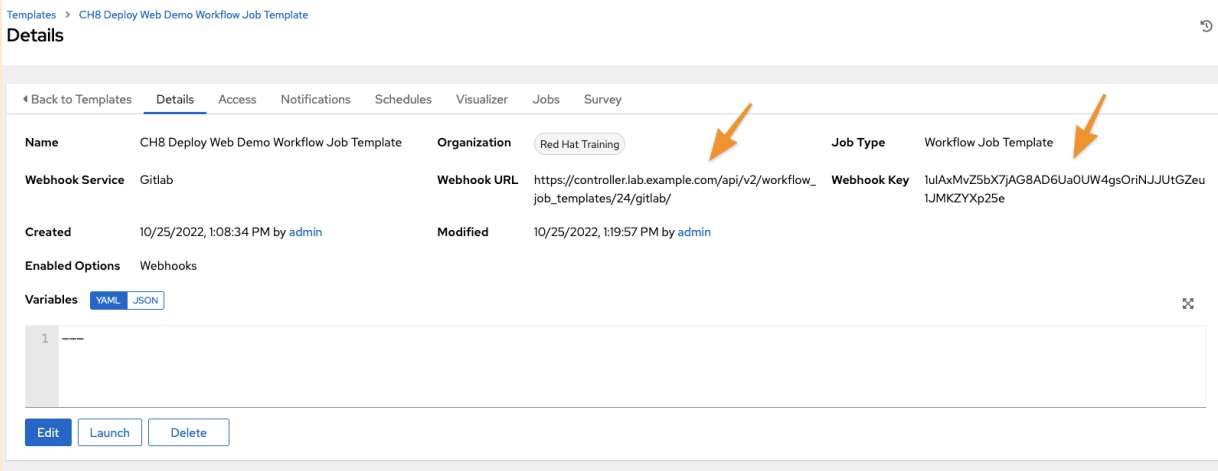

Figure 39. Getting Webhook Information from Controller

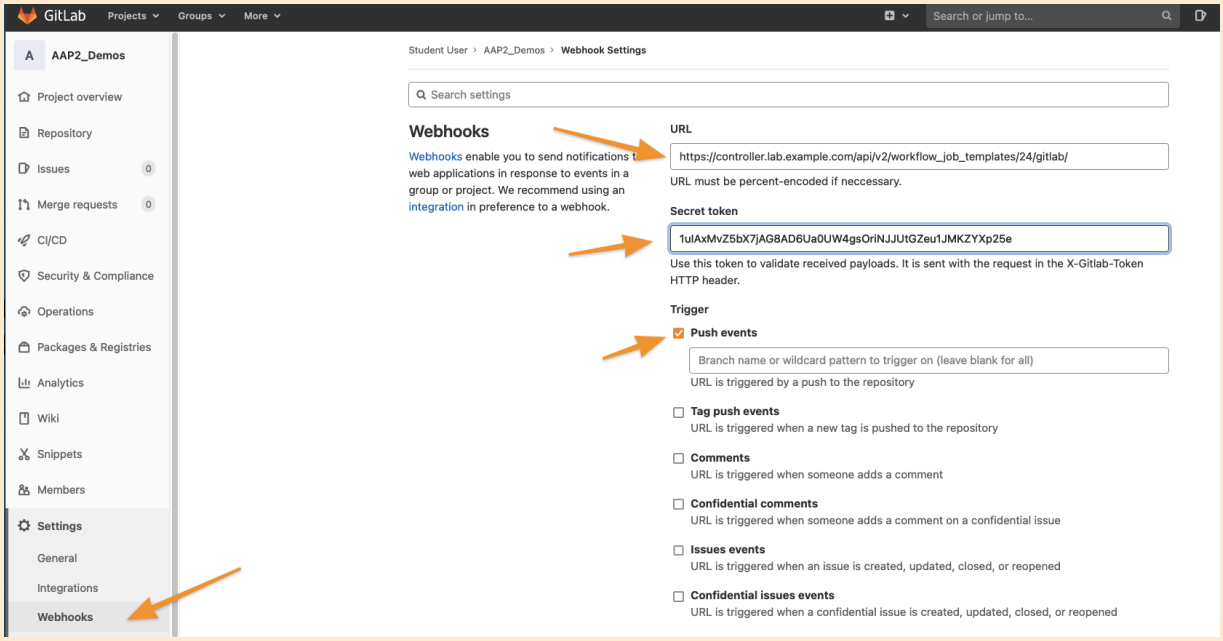

Figure 40. Gitlab Webhook Configuration

Use the FQDN of Controller

It is important to note that the **lab start code-webhooks** pre-configures items in Gitlab. This configuration allows Gitlab and Controller to communicate. The lab script adds the **FQDN** of automation controller allowing Gitlab to communicate with Automation Controller. You MUST use the FQDN when generating the Webhook in controller, otherwise, the webhook will fail as Gitlab will be unable to contact Automation Controller.

3. Test the webhook by clicking **Test** and then selecting **Push events**
   - You should see a **succeeded** in Gitlab

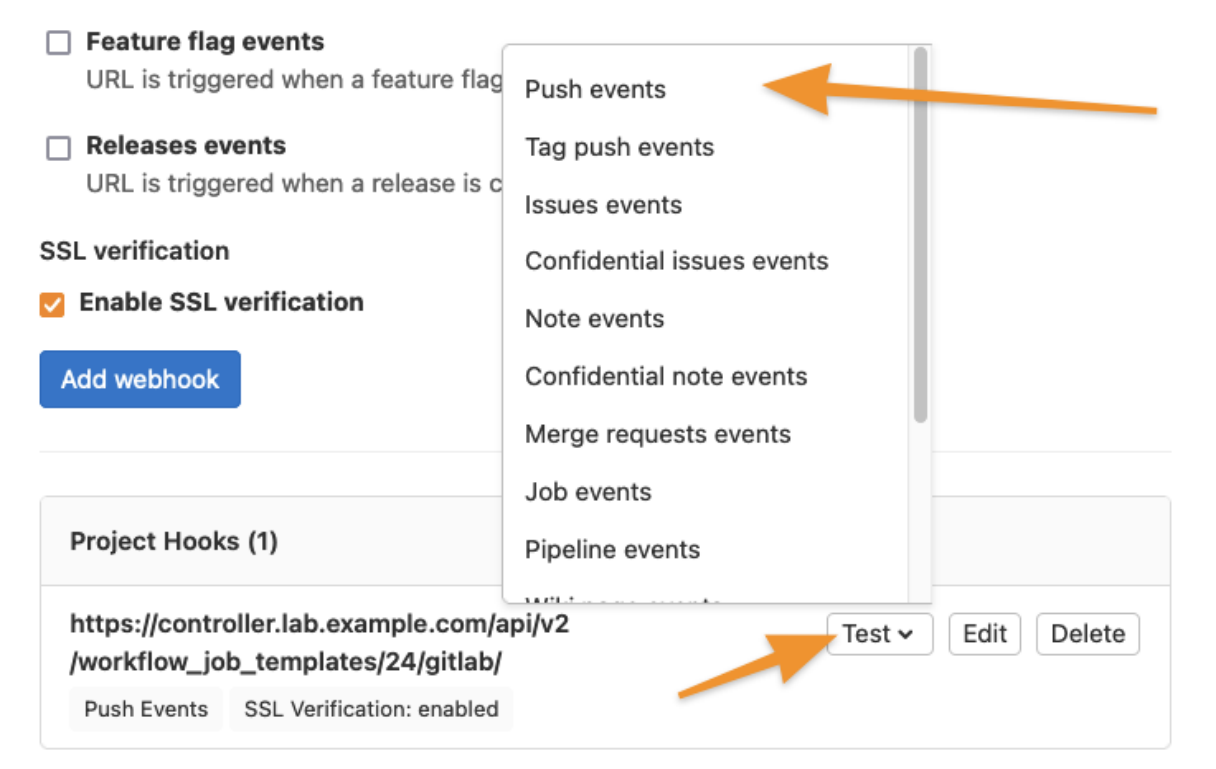

Figure 41. Testing Webhook Configuration

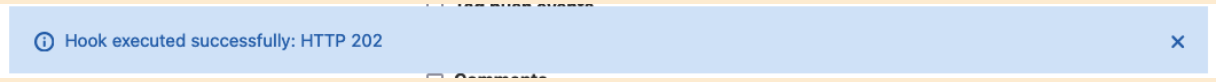

Figure 42. Testing Webhook Validating Successful

4. Show **servera**, **serverc**, and **servere** before changing the **index.html.j2** file.

Verifying Servers and Jobs

At this point, you can demonstrate in Automation Controller the job. Since it has an "approval" node, you will need to approve before showing anything on **servere**.

Essentially bring up the websites in a graphical browser.

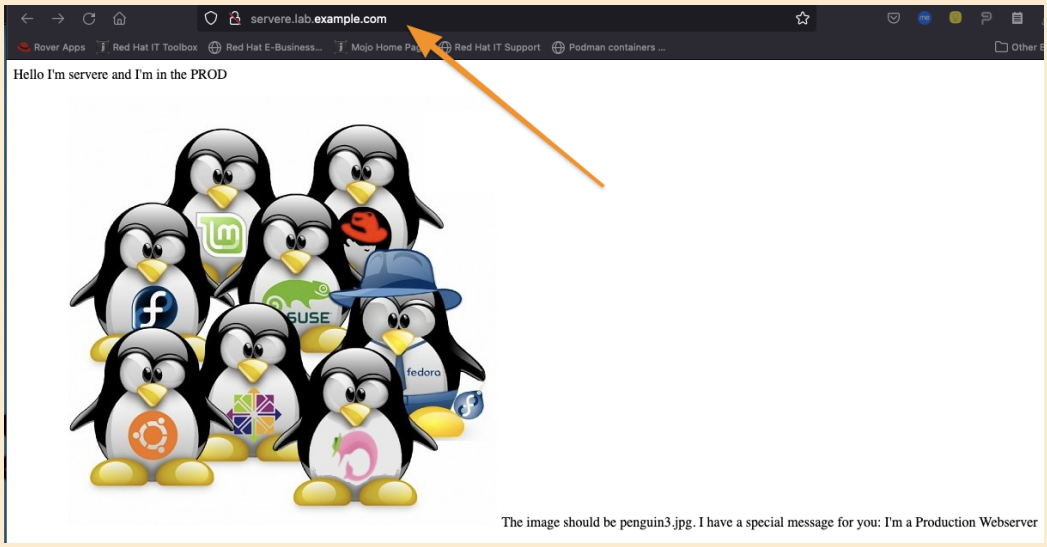

Figure 43. Verifying Workflow Results - SERVERE

5. Copy the **index.html.j2** file from CH8 demo to the **~/Gitlab/aap2_demos** location and commit/push to repo then show again.

Listing 39. Copying the Files

```
[student@workstation Webhooks]$ cp index.html.j2
/home/student/Gitlab/aap2_demos/templates/
```

Listing 40. Committing Changes to Git

```
[student@workstation Webhooks]$ cd /home/student/Gitlab/aap2_demos/templates/

[student@workstation templates]$ git status

[student@workstation templates]$ git add .

[student@workstation templates]$ git commit -m "Improved Website"
[main 2e6731e] Improved Website
 1 file changed, 13 insertions(+), 13 deletions(-)

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

The webhook will have triggered a new workflow job. It is possible to watch the job like we did above and you will also still need to "approve" the rollout to the Production Environment.

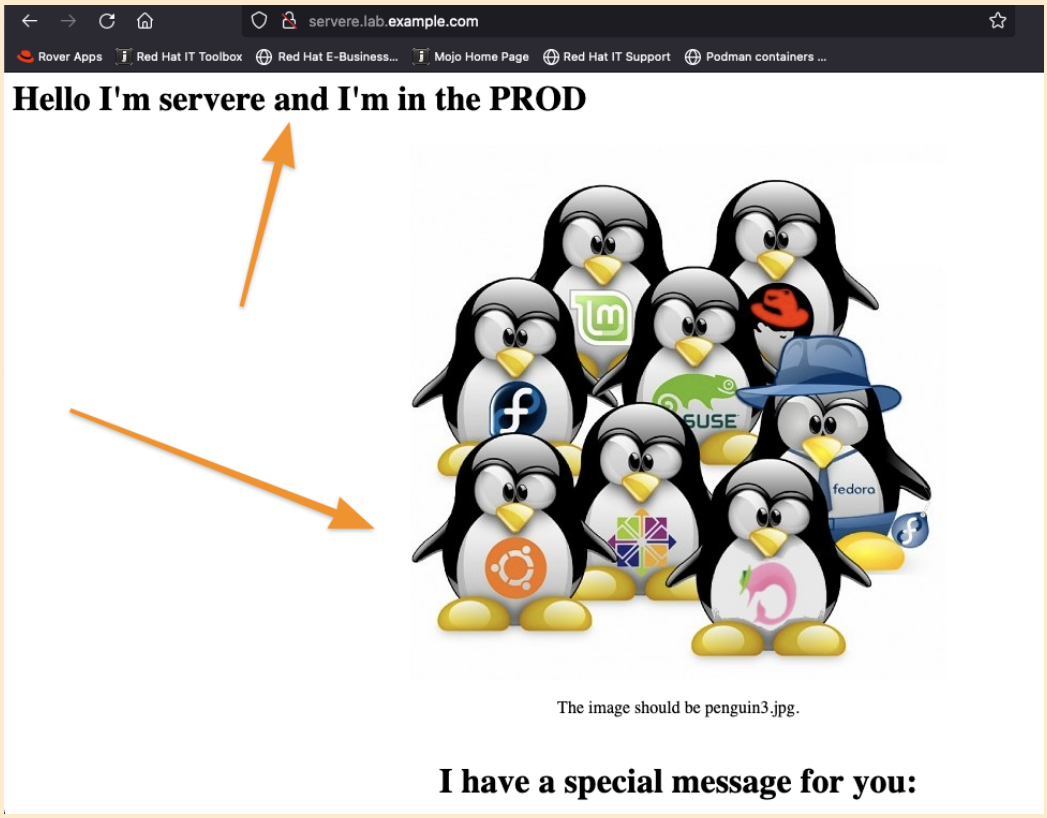

Figure 44. Final Workflow Results - SERVERE

8.3. Launching Jobs with the Automation Controller API

8.3.1. The Automation Controller REST API

Automation controller provides a REST API. This API allows for any programming language or framework to be used to automate tasks for automation controller.

API Development and Availability

The REST API version 2 is the only API currently available.

8.3.1.1. Using the REST API

Listing 41. REST API Demo

```
[student@workstation Demo_Setup]$ curl -X GET https://controller/api/ -k
{"description":"AWX REST API","current_version":"/api/v2/","available_versions":{"v2"
:"/api/v2/"},"oauth2":"/api/o/","custom_logo":"","custom_login_info":"","login_redirect_o
verride":""}
```

JSON Format

The REST API output is JSON which can be sometimes hard to read. In order to have a more "readable" format, it is possible to use JSON formatting tools like **jq** to format the output.

Listing 42. Better Formatted view with jq

```
[student@workstation Demo_Setup]$ curl -X GET https://controller/api/ -k -s | jq
{
  "description": "AWX REST API",
  "current_version": "/api/v2/",
  "available_versions": {
    "v2": "/api/v2/"
  },
  "oauth2": "/api/o/",
  "custom_logo": "",
  "custom_login_info": "",
  "login_redirect_override": ""
}
```

REST API Authentication

The REST API requires authentication. When using **curl** it is possible to specify the user credentials with the curl GET commands by using **--user admin:redhat** which specifies the admin username/password combination for the automation controller.

Listing 43. Retrieving Information from API

```
[student@workstation Demo_Setup]$ curl -X GET --user admin:redhat
https://controller/api/v2/activity_stream/ -k -s | jq
{
  "count": 132,
  "next": "/api/v2/activity_stream/?page=2",
  "previous": null,
  "results": [
    {
      "id": 1,
      "type": "activity_stream",
      "url": "/api/v2/activity_stream/1/",
      "related": {
        "user": [
          "/api/v2/users/1/"
        ]
      },
      "summary_fields": {
        "user": [
          {
            "id": 1,
            "username": "admin",
            "first_name": "",
            "last_name": ""
          }
        ]
      },
      "timestamp": "2022-06-14T15:44:41.419854Z",
      "operation": "create",
      "changes": {
        "username": "admin",
        "first_name": "",
        "last_name": "",
        "email": "admin@example.com",
        "is_superuser": true,
        "password": "hidden",
        "id": 1
      },
      "object1": "user",
      "object2": "",
      "object_association": "",
      "action_node": "controller.lab.example.com",
      "object_type": ""
    },

... OUTPUT OMITTED ...
```

8.3.1.2. JSON Pagination

8.3.1.3. Accessing the REST API From a Graphical Web Browser

The API can also be accessed from a web browser.

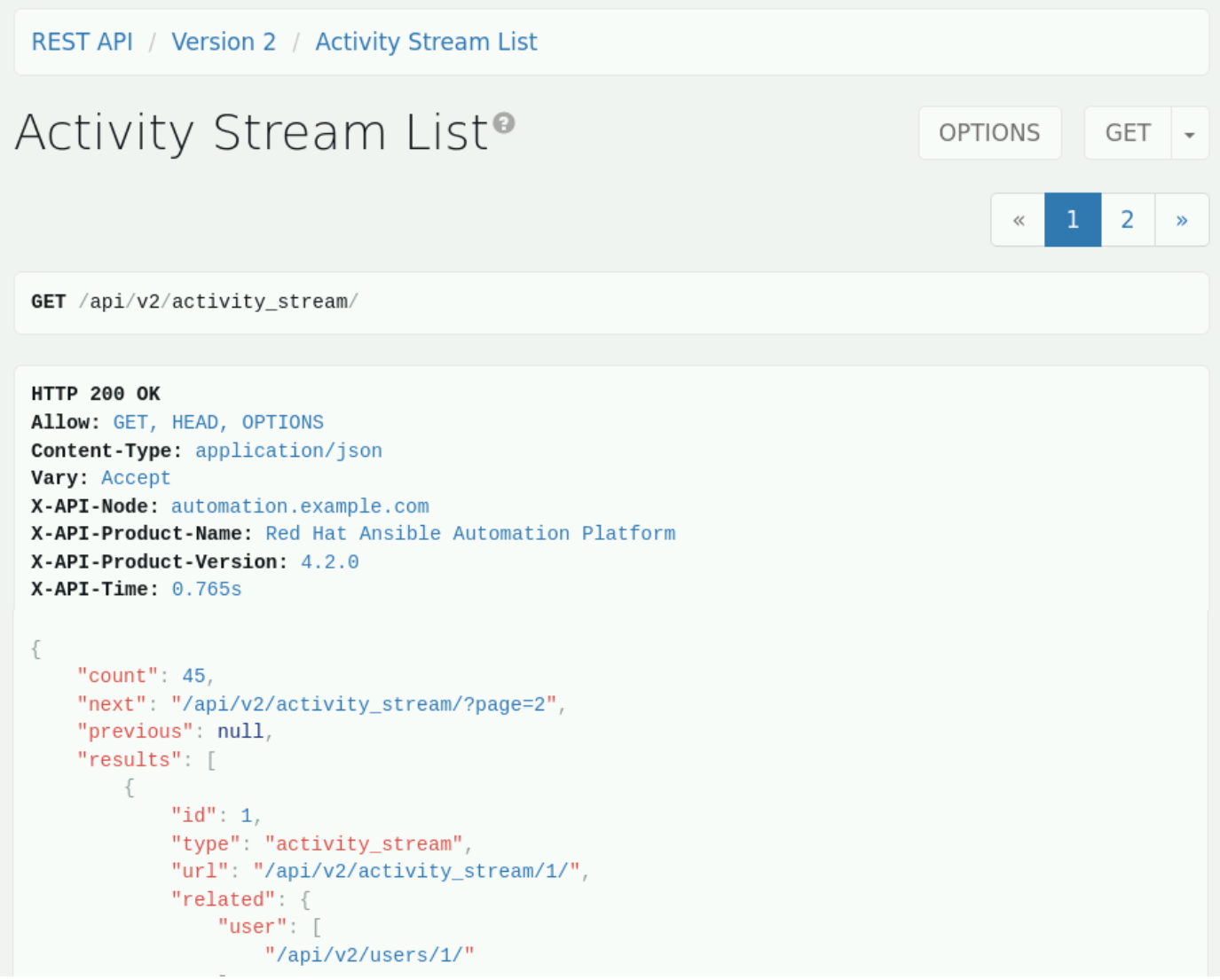

Figure 45. Web Browser Activity Stream

API Documentation

When accessing the API via the browser, there is documentation available by clicking on the **?** icon.

8.3.2. Launching a Job Template Using the API

When launching a Job Template using the API from CLI with **curl** you must first **GET** the information from the API to form the **POST** request which will launch the job template.

8.3.3. Launching a Job Using the API from an Ansible Playbook

When using Ansible playbooks, the **URI** module can be used in a similar way to **curl** to launch the jobs. When launching jobs this way, username/password are provided and **force_basic_auth** can be used.

8.3.3.1. Vault Credentials

Automation controller can store various types of credentials (including **vault** credentials). Vault credentials must be created and available for automation tasks, especially for leveraging the API as there isn't a mechanism to prompt for user's input.

Multiple Vault Files

Newer versions of Ansible allow encrypting files with different vault passwords. When using multiple vaults, these credentials must be supplied to the job template to ensure automation controller can decrypt files and project resources.

8.3.4. Token-based Authentication

Ansible Automation Controller can use token-based authentication built on OAuth 2.

Two kinds of tokens

- **Application Token** - Requested for an application and accessed by multiple users and jobs
- **Personal Access Tokens (PAT)** - Token for access to the API for a single user

8.3.5. DEMO: Using the Ansible Automation Controller REST API

Dealing with JSON Output

JSON isn't the most human readable format. There are JSON formatters and parsers available to make JSON more friendly to read. By leveraging the **jq** command, it is possible to format JSON to be more consumable.

There are some JSON/JQ Cheatsheets available

- https://lzone.de/cheat-sheet/jq
- https://stedolan.github.io/jq/manual/
- https://programminghistorian.org/en/lessons/json-and-jq

Example 15. DEMO: Automation Controller API

Listing 44. Retrieving API Information (formatted with jq)

```
[student@workstation API]$ curl -X GET https://controller.lab.example.com/api/v2/ -k
-s | jq .
{
  "ping": "/api/v2/ping/",
  "instances": "/api/v2/instances/",
  "instance_groups": "/api/v2/instance_groups/",

... OUTPUT OMITTED ...
```

Listing 45. Retrieving Job Template Information (formatted with jq)

```
[student@workstation API]$ curl -X GET --user admin:redhat
https://controller.lab.example.com/api/v2/job_templates/ -k -s | jq .
```

Listing 46. Retrieving Job Template Information (formatted with jq and looking for results.name)

```
[student@workstation API]$ curl -X GET --user admin:redhat
https://controller.lab.example.com/api/v2/job_templates/ -k -s | jq '.results[] |
{name}'
{
  "name": "Add Teams"
}
{
  "name": "Add Users"
}
... OUTPUT OMITTED ...
{
  "name": "DO467 CH5 Job Survey and Facts"
}
{
  "name": "DO467 Demo Job - Testing Inventory" ①
... OUTPUT OMITTED ...
```

① Job name we want to launch

Listing 47. Launching the DO467 Demo Job - Testing Inventory Job Template

```
[student@workstation API]$ curl -X POST --user admin:redhat
https://controller.lab.example.com/api/v2/job_templates/"DO467 Demo Job - Testing
Inventory"/launch/ -k -s | jq .
{
  "job": 122,
  "ignored_fields": {},
  "id": 122,
  "type": "job",
  "url": "/api/v2/jobs/122/",
  "related": {
    "created_by": "/api/v2/users/1/",
    "modified_by": "/api/v2/users/1/",
    "labels": "/api/v2/jobs/122/labels/",
    "inventory": "/api/v2/inventories/2/",
    "project": "/api/v2/projects/8/",
    "organization": "/api/v2/organizations/2/",
    "credentials": "/api/v2/jobs/122/credentials/",
    "unified_job_template": "/api/v2/job_templates/26/",
    "stdout": "/api/v2/jobs/122/stdout/",
    "job_events": "/api/v2/jobs/122/job_events/",
    "job_host_summaries": "/api/v2/jobs/122/job_host_summaries/",
    "activity_stream": "/api/v2/jobs/122/activity_stream/",
    "notifications": "/api/v2/jobs/122/notifications/",
    "create_schedule": "/api/v2/jobs/122/create_schedule/",
    "job_template": "/api/v2/job_templates/26/",
    "cancel": "/api/v2/jobs/122/cancel/",
    "relaunch": "/api/v2/jobs/122/relaunch/"
  },
  "summary_fields": {
    "organization": {
      "id": 2,
      "name": "Red Hat Training",
      "description": "Red Hat Training Organization - Demo"
    },

... OUTPUT OMITTED ...
```

Bash Script and Ansible Playbook

There is a bash script and Ansible playbook also available for a demo.

Listing 48. Accessing Additional Demo Files

```
[student@workstation ~]$ cd
/home/student/Github/DO467_Notes/Demos/CH8/API
```

Bash Scripts

Launch_Job_Template.sh - Launches a Job Template **Search_Job_Template_Names.sh** - Returns names of various Job Templates (_more advanced as it also uses JQ/sed for some filtering)

Ansible Playbook

Launch_Job.yml - Launches the **DO467 Demo Job - Testing Inventory** job.

In the **Launch_Job.yml** we are also utilizing JINJA2 filters to allow the URL to be encoded as many of our Ansible Job Templates have spaces which must be encoded properly when using as URLs or in API calls.

9. Maintaining Red Hat Ansible Automation Platform

9.1. Performing Basic Troubleshooting of Automation Controller

9.1.1. Automation Controller Components

Automation controller is made up of multiple processes and services.

Automation Controller Components

- **Nginx** - Provides webserver for hosting applications and supports the WebUI and API
- **PostgreSQL** - Database storing controller data, configuration, and history
- **Supervisord** - Process control system managing various controller operations including job scheduling.
- **Receptor** - Provides overlay network for work distribution across dispersed workers (used for Automation Mesh network communication)
- **memcached** - Object caching daemon for local object caching

Most services run on ports **80/tcp** or **443/tcp**. The PostgreSQL listens on **5432/tcp** and **receptor** uses certain ports for automation mesh communications.

9.1.1.1. Starting, Stopping, and Restarting Automation Controller

Because of the many components that create automation controller, there is an administrative utility script **automation-controller-service** that ships with automation controller which helps centralize the management and controlling of the independent services.

Listing 49. automation-controller-service status

```
[root@controller ~]# automation-controller-service status
● automation-controller.service - Automation Controller service
   Loaded: loaded (/etc/systemd/system/automation-controller.service; enabled; vendor>
   Active: active (exited) since Wed 2022-09-07 16:26:35 EDT; 23h ago
  Process: 1269 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 1269 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 36152)
   Memory: 0B
   CGroup: /system.slice/automation-controller.service

Sep 07 16:26:35 controller.lab.example.com systemd[1]: Starting Automation Controller>
Sep 07 16:26:35 controller.lab.example.com systemd[1]: Started Automation Controller >

● redis.service - Redis persistent key-value database
   Loaded: loaded (/usr/lib/systemd/system/redis.service; enabled; vendor preset: dis>
  Drop-In: /etc/systemd/system/redis.service.d
           └──limit.conf, override.conf
   Active: active (running) since Wed 2022-09-07 16:25:32 EDT; 23h ago
 Main PID: 913 (redis-server)
```

Supported automation-controller-service Options

- start
- stop
- restart
- status

9.1.1.2. Supervisord Components

Like SystemD, another set of components **Supervisord** is a collection of processes that controls the various Django-based applications. In order to manage **supervisord** components, the **supervisorctl** command is available.

```
[root@controller ~]# supervisorctl status
master-event-listener                   RUNNING   pid 1273, uptime 23:23:10
tower-processes:awx-callback-receiver   RUNNING   pid 1275, uptime 23:23:10
tower-processes:awx-daphne              RUNNING   pid 1277, uptime 23:23:10
tower-processes:awx-dispatcher          RUNNING   pid 1274, uptime 23:23:10
tower-processes:awx-rsyslogd            RUNNING   pid 1309, uptime 23:23:05
tower-processes:awx-uwsgi               RUNNING   pid 1276, uptime 23:23:10
tower-processes:awx-wsbroadcast         RUNNING   pid 1278, uptime 23:23:10
```

9.1.2. Automation Controller Configuration and Log Files

Configuration Files

/etc/tower

Most Important Configuration File

One of the most important config files is **/etc/tower/settings.py**.

There may also be service specific configuration files like **/etc/nginx** to consider as well.

Log Files

- /var/log/tower/
- /var/log/supervisor/

Log files of interest

It should be noted that even though the product is now Automation Controller, the directory name still reflects **tower**.

- **/var/log/tower/tower.log** - Main log file for automation controller.
- **var/log/tower/task_system.log** - Log file capturing the log of tasks being run.

Other Automation Controller Files

- **/var/lib/awx/projects** - Main directory for storing projects. Source control projects will be cloned to this directory.
- **/var/lib/awx/job_status** - Contains output of playbook job runs

9.1.3. Common Troubleshooting Scenarios

Problems Running Playbooks

- Is the correct user being leveraged?
- Are the YAML files correctly formatted? (yamllint and **ansible-lint** can be used to help)

Currently, **ansible-lint** is in tech preview and is not fully supported by Red Hat.

- Are lists formatted properly
- Are dictionaries formatted properly
- Is automation controller properly entitled?

Problems Connecting to Your Host

- Verify machine credentials for SSH/WinRM

- Verify inventory hostname and IP addresses

Playbooks Do Not Appear in the List of Job Templates

- Review YAML syntax, if incorrect, won't appear in Ansible and can't be processed

Playbook Stays in Pending State

- Check status of **supervisord** with **supervisorctl status**
- Check controller storage, specifically **/var** has sufficient free space
- Restart automation controller components

Error: Provided Hosts List Is Empty

- Check inventory host patterns in playbook
- Check group names for spaces
- Check the **limit** on **Job Template**

9.1.4. Performing Command-Line Management

The **awx-manage** command can change passwords, create new superusers, and other tasks for automation controller.

Listing 50. Changing Built-in Admin Password

```
[root@control ~]# awx-manage changepassword admin
```

Listing 51. Creating New Superusers

```
[root@control ~]# awx-manage createsuperuser
```

9.2. Backing Up and Restoring Red Hat Ansible Automation Platform

9.2.1. Backing Up Red Hat Ansible Automation Platform

The ability to perform a backup/restore is built into Ansible Tower. The procedure users the same **setup.sh** script and **inventory** file that was used to install Ansible Tower.

If the original installation directory has been deleted, you can still setup backups by unpacking the **tar** archive for the installer of the same version of Ansible Tower you are using.

After that has been completed, the **inventory** file will need to be edited with the current passwords for the Ansible Tower services (admin_password, pg_password, and rabbitmq_password). Any other changes/edits to the original inventory file must also be made to this file.

The backup process is started by running **./setup.sh -b** in the installation directory on the Ansible Tower server as the root user.

The backup archive consists of:

- **tower.db**: PostgreSQL database dump file
- **./conf**: The configuration directory, containing files from the **/etc/tower** directory
- **./job_status**: The directory for job output files
- **./projects**: The directory for manual projects
- **./static**: The directory for webUI customization such as custom logos

Additional Archives for AAP2

- Creates archive for each controller in the environment
- Creates a **common.tar.gz** archive for SECRET_KEY, **tower.db** and version files.
- If automation hub is part of the environment, it will create an **automationhub.tar.gz** file.

Backup Procedure

1. As root, change to the Ansible Tower installation directory
2. As root, run the **setup.sh** script with the **-b** option to initiate the backup process

Listing 52. Backup Process

```
[root@tower ansible-tower-setup-bundle-3.4.2-1.el7]# ./setup.sh -b
Using /etc/ansible/ansible.cfg as config file
/root/ansible-tower-setup-bundle-3.4.2-1.el7/inventory did not meet host_list
requirements, check plugin documentation if this is unexpected
/root/ansible-tower-setup-bundle-3.4.2-1.el7/inventory did not meet script
requirements, check plugin documentation if this is unexpected
 [WARNING]: Could not match supplied host pattern, ignoring: instance_group_*

PLAY [tower:instance_group_*] **************************************************

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

... output omitted ...

RUNNING HANDLER [backup : Remove backup dest stage directory.] *****************
changed: [localhost] => {"changed": true, "path": "/root/ansible-tower-setup-bundle-
3.4.2-1.el7/2019-12-06-17:41:48", "state": "absent"}

PLAY RECAP *********************************************************************
localhost                  : ok=37   changed=29   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2019-12-06-17:41:41.log
```

3. List the current directory to ensure the archive was created.

Storing the Backup

The manual backup procedure creates the backup archive file locally. The administrator is responsible for ensuring that the backup gets transferred and stored in a safe place.

9.2.2. Restoring Ansible Automation Platform From Backup

The **setup.sh** script can be used with the **-r** option to restore Ansible Tower from a backup archive. You need the backup, the installer of the same version of Ansible Tower and the **inventory** file for the installer.

Restore Procedure

1. As the root user, change to the Ansible Tower installation directory
2. Ensure the backup archive is present in the directory
3. Run **setup.sh -r** script to start restoring the Ansible Tower configuration and database

4. Login to the webUI and verify the server has been restored from backup correctly

10. Getting Insights into Automation Performance

10.1. Gathering Data for Cloud-based Analysis

Section Info Here

10.1.1. Introducing Red Hat Hybrid Cloud Console Services

10.1.2. Collecting Data for Cloud Services

10.1.3. Registering Managed Hosts with Insights for Ansible Automation Platform

10.1.4. Accessing the Red Hat Hybrid Cloud Console

10.2. Getting Insights into Automation Performance

Section Info Here

10.2.1. Insights for Ansible Automation Platform

10.2.2. Generating Remediation Playbooks with Advisor

10.2.2.1. Automating Remediation of an Issue for Multiple Systems

10.2.2.2. Automating Remediation of Multiple Issues for One System

10.2.3. Comparing Systems with Drift

10.2.3.1. Finding Differences Between Systems

10.2.3.2. Comparing the State of One System at Different Times

10.2.3.3. Comparing Systems to a Standard Baseline

10.2.4. Sending Alerts Based on Ansible Facts with Policies

10.3. Evaluating Performance with Automation Analytics

Section Info Here

10.3.1. Automation Analytics

10.3.2. Reporting Playbook Execution Status

10.3.3. Examining Job History

10.3.4. Monitoring Notifications

10.4. Producing Reports from Automation Analytics

Section Info Here

10.4.1. Producing Reports from Automation Analytics

10.4.1.1. Choosing an Appropriate Report

10.4.1.2. Using Automation Calculator to Compute Savings

10.4.1.3. Exporting a Report

10.4.2. Predicting the Cost Savings of Automation

10.4.2.1. Creating a Savings Plan

10.4.2.2. Reviewing the Cost Savings Calculations

11. Building a Large Scale Red Hat Ansible Automation Platform Deployment

11.1. Designing a Clustered Ansible Automation Platform Implementation

11.1.1. Running Red Hat Ansible Automation Platform at Scale

The new architecture and design of AAP2 allows running Ansible automation on a larger scale. Automation controller provides the ability to configure two separate modes by separating the API/Control nodes from the Execution nodes. This separation allows execution nodes to be places closer to managed hosts.

Automation Mesh

Automation mesh and new automation architecture replaces less powerful isolated nodes.

11.1.2. Automation Mesh

Provides overlay networ to distribute work from machines running the controller and instead directs it to dedicated and dispersed execution nodes.

11.1.2.1. Benefits of Automation Mesh

- Provides ability to scale both control and execution capacity.
- Uses end-to-end encrypted and authenticated network protocols.
- Provides multiple direction and multiple hop network allowing communication across various networks with security constraints.
- Allows reconfiguring how traffic is routed across the mesh if one or more nodes fail.
- Reduces overhead providing a single distributed platform

11.1.2.2. Types of Nodes on Automation Mesh

- **Control Plane** Runs controller services, webUI, task dispatcher, project updates and management jobs.
  - **Hybrid nodes** - Performs both control plane and execution plane tasks
  - **Control Nodes** - Performs only control plane tasks and does not perform any execution plan tasks
- **Execution Plane** Executes functions on behalf of the control planes.

- **Execution nodes**: Run jobs using container-based execution environments (uses **ansible-runner**)
- **Hop nodes**: Node to route traffic to other execution nodes.

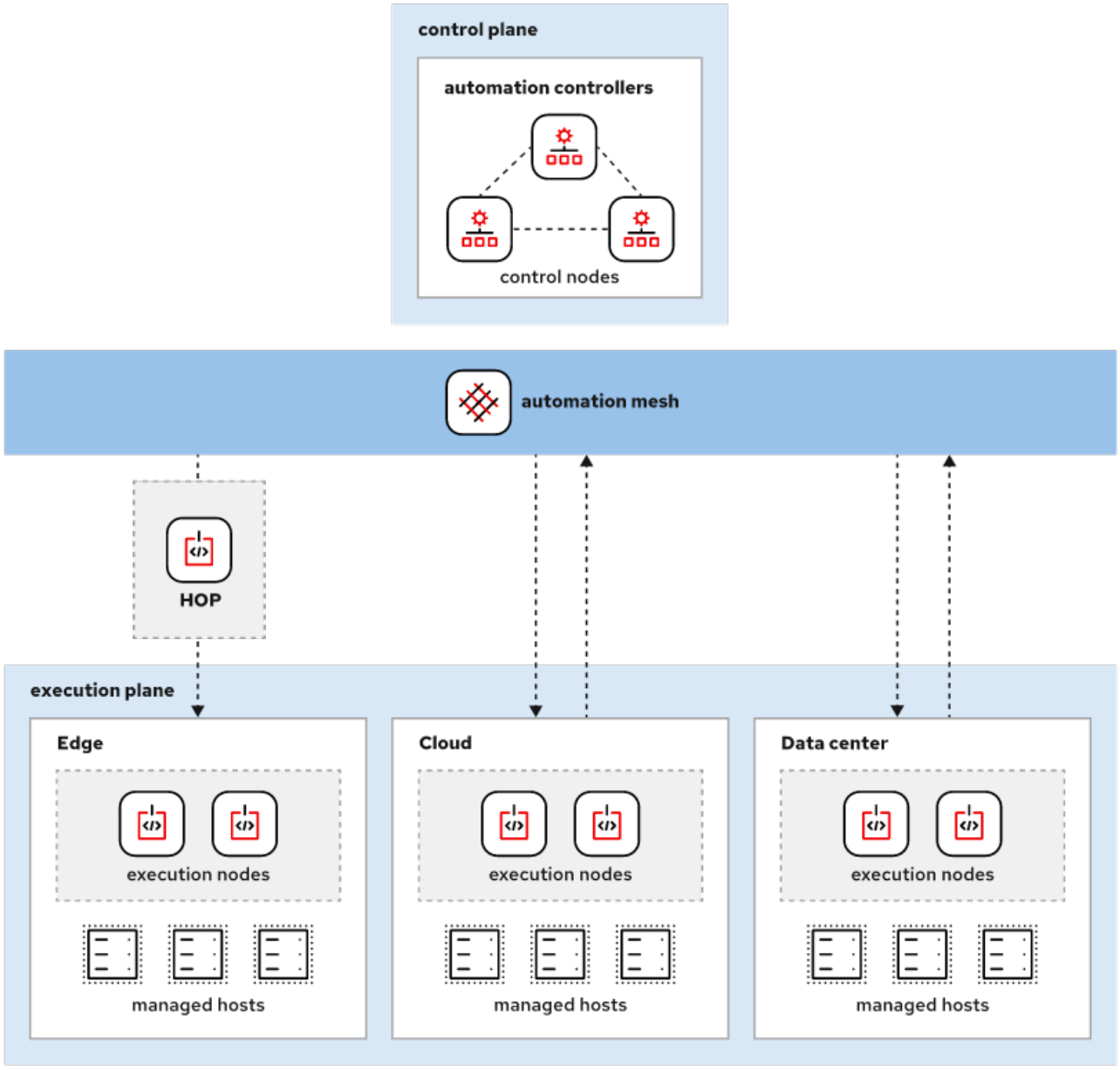

Figure 46. Ansible Automation Mesh Nodes

11.1.2.3. What Are Instance Groups?

Groups of execution or control nodes. Allows using instance groups to run automation jobs on specific execution nodes. Ansible Automation mesh creates the **default** instance group for all execution and hybrid nodes and the **controlplane** instance group for all control nodes and hybrid nodes.

Default Instance Group

The **default** instance group cannot be removed.

Instance Group Precedence

Instance groups have precedence (highest to lowest)

- Job Template
- Inventory
- Organization default

11.1.3. Planning Network Communication and Firewalls

Automation mesh uses its own network protocol for communication between nodes. It is a TLS-encrypted protocol connecting to port 27199/TCP. The port and TLS certificates may be changed at installation time. The **receptor** service listens on the port for automation mesh communications.

Execution nodes continue to communicate directly with managed hosts using SSH protocol. Hop nodes may be used to relay communications from control nodes to execution nodes using port 27199/TCP.

Table 5. Requirements for Control Nodes and Hybrid Nodes

Network ports	Service	Purpose
27199/TCP	Receptor	Used for communication between the control plane and hop nodes.
80/TCP & 443/TCP	HTTP/HTTPS	Used for accessing the automation controller web UI and API.
22/TCP	SSH	Used for secure access and administration, including configuration performed by Ansible using the `setup.sh` script.

Table 6. Requirements for Hop Nodes

Network ports	Service	Purpose
27199/TCP	Receptor	Used for communication between the control plane and hop nodes.
22/TCP	SSH	Used for secure access and administration, including configuration performed by Ansible using the `setup.sh` script.

Table 7. Requirements for Execution Nodes

Network ports	Service	Purpose
27199/TCP	Receptor	Used for communication between the control plane and hop nodes.
80/TCP & 443/TCP	HTTP & HTTPS	Connection to container registry to pull down execution environments.
22/TCP	SSH	Used for secure access and administration, including configuration performed by Ansible using the `setup.sh` script.

Execution Node Communication

The execution node only needs to be allowed to communicate on port 27199/TCP with control nodes/hop nodes that the execution node directly peers with.

11.1.4. Planning for Automation Mesh

The Ansible Automation Mesh can be both planned and deployed in stages. As the deployment evolves, it is possible to introduce **Instance Groups** controlling execution of job templates ensuring that execution happens closer to the managed host.

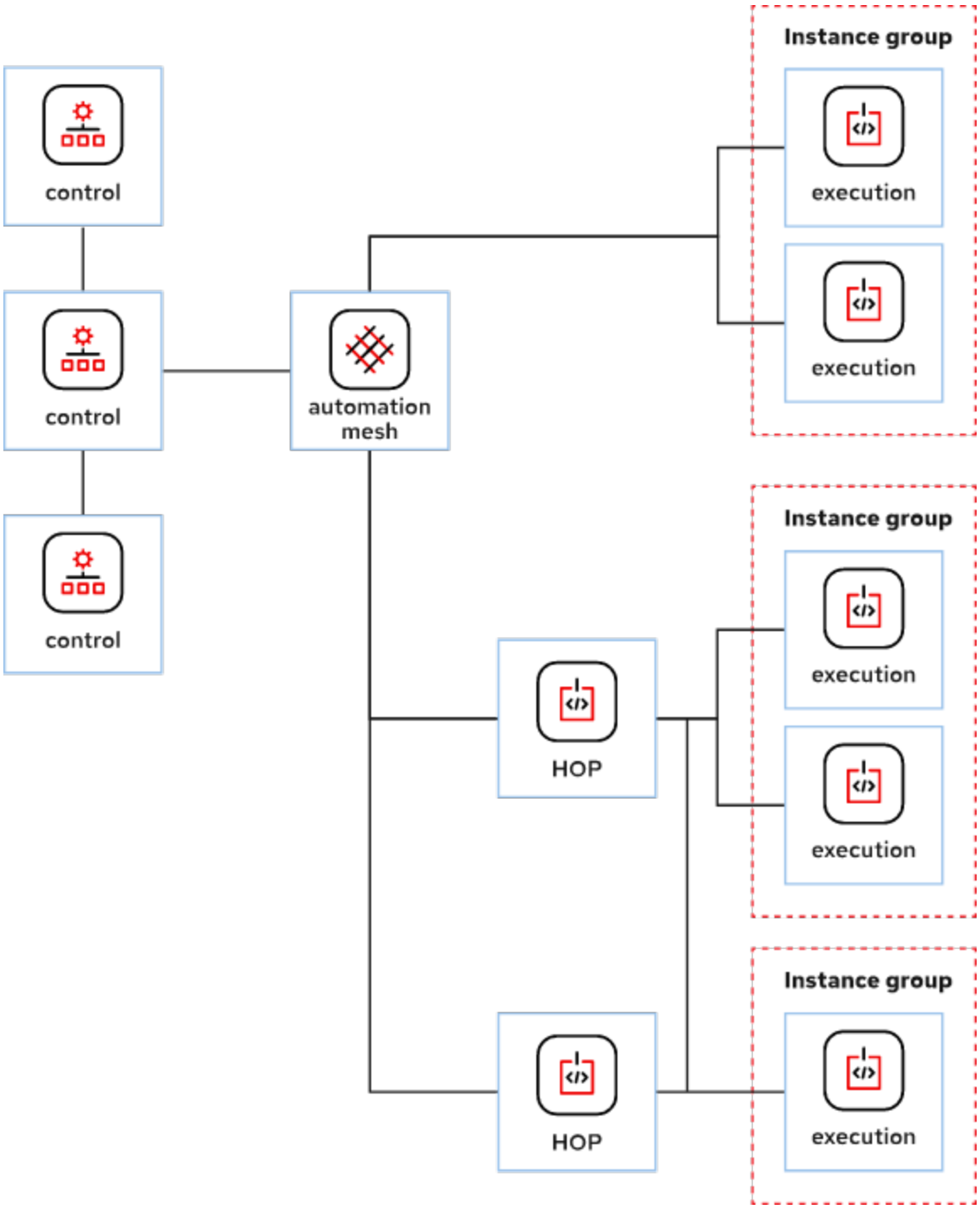

Figure 47. Ansible Automation Mesh with Instance Groups

11.1.4.1. Providing Resilient Services

When designing with multiple controllers and private automation hubs, a network load balancer can provide high-availability by ensuring traffic gets routed to systems that are operational and avoid downtime by preventing access to systems that have gone offline.

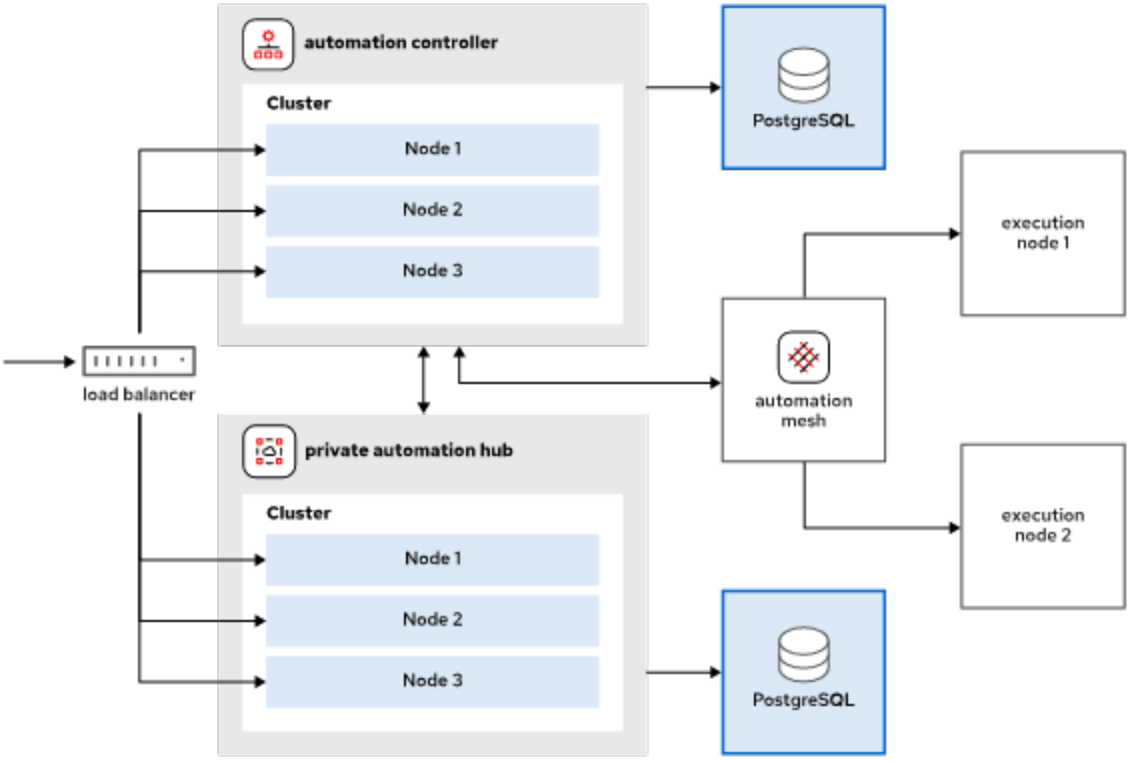

Figure 48. Enterprise Automation Mesh with High-Availability

Multiple Private Automation Hubs

In addition to multiple control and execution nodes, multiple private automation hubs should also be considered in the event of failures.

11.1.5. DEMO: Installing Automation Mesh

Examining Configuration

Listing 53. Configuration File without Comments

```
[student@workstation Mesh]$ grep -o '^[^#]*' inventory
```

Example 16. DEMO: Installing Automation Hub and Controller

1. Run **lab start** to ensure everything is ready for the demo (takes 1-2 minutes usually).

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

2. Copy **Demo** inventory with files already setup

```
[student@workstation ~]$ cd ~/aap2.2-bundle/

[student@workstation aap2.2-bundle]$ cp
~/Github/DO467_Demo/Demos/CH11/Mesh/inventory  .
```

Review Inventory Settings

It is possible to strip out all comments from inventory and just review the settings using the **grep -o '[^#]'*** command.

Listing 54. Inventory Settings

```
[student@workstation aap2.2-bundle]$  grep -o '^[^#]*' inventory

[automationcontroller]
controller.lab.example.com
control2.lab.example.com
[automationcontroller:vars]
node_type=control
web_server_ssl_cert=/home/student/certs/{{ inventory_hostname
}}.crt
web_server_ssl_key=/home/student/certs/{{ inventory_hostname
}}.key
[execution_nodes]
exec1.lab.example.com peers=automationcontroller
exec2.lab.example.com peers=automationcontroller
exec3.lab.example.com peers=hop1.lab.example.com
hop1.lab.example.com peers=automationcontroller node_type=hop
[automationhub]
hub.lab.example.com
[automationcatalog]
[database]
db.lab.example.com
[sso]
[all:vars]
admin_password='redhat'
pg_host='db.lab.example.com'
```

```
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='redhat'
pg_sslmode='prefer'
registry_url='hub.lab.example.com'
registry_username='admin'
registry_password='redhat'
receptor_listener_port=27199
automationhub_admin_password='redhat'
automationhub_pg_host='db.lab.example.com'
automationhub_pg_port=5432
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='redhat'
automationhub_pg_sslmode='prefer'
automationcatalog_pg_host=''
automationcatalog_pg_port=5432
automationcatalog_pg_database='automationservicescatalog'
automationcatalog_pg_username='automationservicescatalog'
automationcatalog_pg_password=''
custom_ca_cert=/home/student/certs/classroom-ca.pem
automationhub_ssl_cert=/home/student/certs/hub.lab.example.com.crt
automationhub_ssl_key=/home/student/certs/hub.lab.example.com.key
postgres_use_ssl=True
postgres_ssl_cert=/home/student/certs/db.lab.example.com.crt
postgres_ssl_key=/home/student/certs/db.lab.example.com.key
sso_keystore_password=''
sso_console_admin_password=''
```

3. Run the setup script to generate the Topology

```
[student@workstation aap2.2-bundle]$ ./setup.sh -- --tags generate_dot_file
```

Listing 55. Installing graphviz

```
[student@workstation aap2.2-bundle]$ sudo dnf install graphviz
[sudo] password for student:
```

Listing 56. Converting DOT file to Image

```
dot -Tjpg mesh-topology.dot -o graph-topology.jpg
```

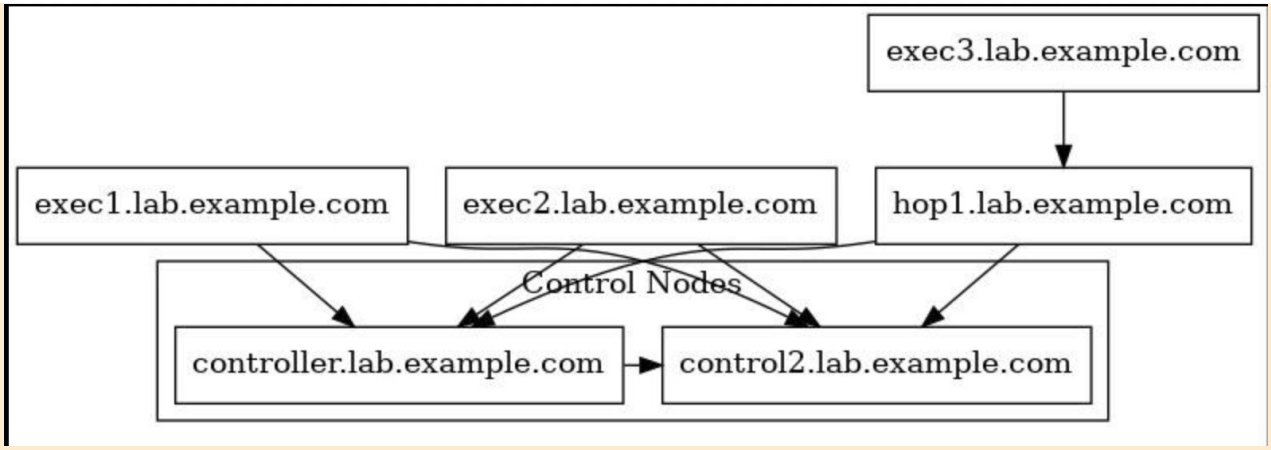

Figure 49. Mesh Topology from Inventory

4. Become **root** and run the setup (*Ignore Preflight Errors*)

```
[student@workstation aap2.2-bundle]$ sudo -i
[sudo] password for student:
```

Listing 57. Installing Automation Mesh Changes (*This takes ~ 10-15 minutes*)

```
[root@workstation ~]# cd ~student/aap2.2-bundle/

[root@workstation aap2.2-bundle]# ./setup.sh -e ignore_preflight_errors=true
```

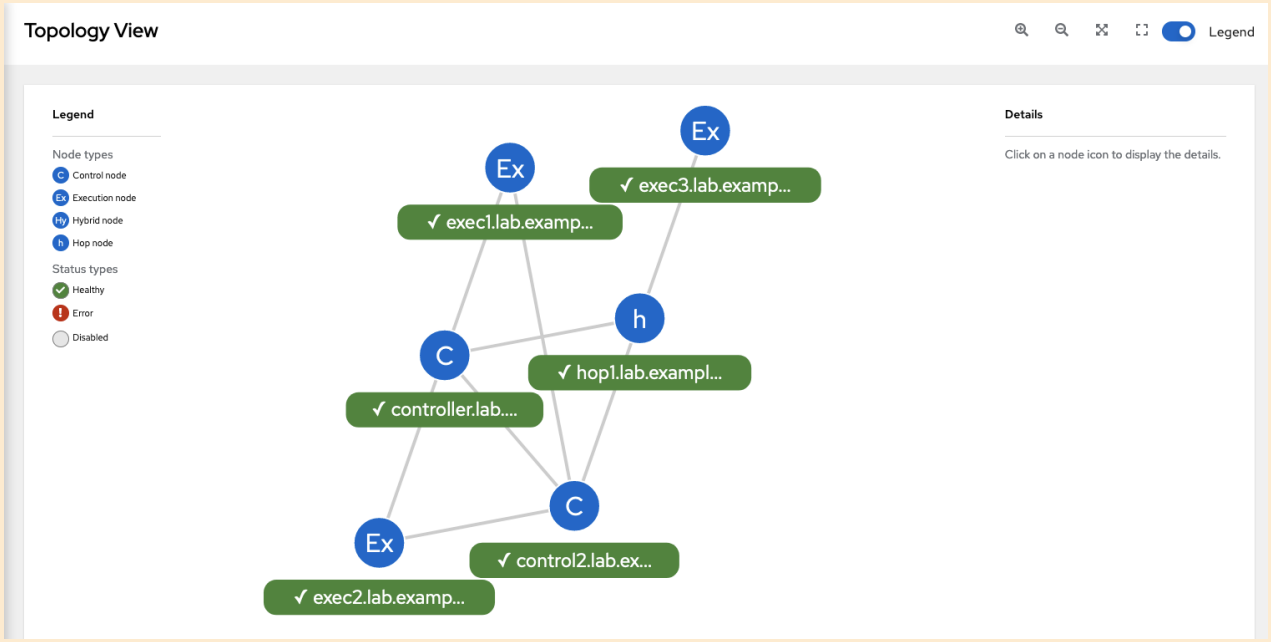

Figure 50. Mesh Topology from WebUI

11.2. Deploying Distributed Execution with Automation Mesh

11.2.1. Configuring Automation Mesh

The Ansible Automation Mesh network is also controlled and configured by the inventory file which is used by the installation script.

Listing 58. Inventory File with More Advanced Options

```
[automationcontroller] ①
controller.lab.example.com

[execution_nodes] ②
exec1.lab.example.com

[automationcontroller:vars] ③
peers=execution_nodes ④
node_type=control ⑤
```

① Adds controller nodes. By default, any host here is considered a hybrid node **node_type=hybrid** in which they are both control and execution nodes.

② Specifies execution nodes or hops. By default, hosts specified here are **node_type=execution**

③ Group variables. These can be set individually and overridden at the host group level.

④ **peers=execution_nodes** means that the **automationcontroller** group is directly peered with all nodes in the **execution_nodes** group

⑤ **node_type=control** specifies nodes in the **automationcontroller** group are all control nodes. Another option can be **hybrid** in which the controllers are both control and execution nodes.

Inventory File Settings and Roles

In most instances, things appear in the inventory file by default and can be changed based on values or removing comments. However, there are some options and items no present and added to the inventory by default, but can still be added as part of the modified installation and configuration.

Overriding Variables

It is important to note that while inventory variables can be set in the inventory file used with the installation script, these variables can be overridden for each individual host.

11.2.1.1. Creating Instance Groups

The **controlplane** instance group is created automatically for hosts in the **automationcontroller** group and the **default** instance group is created for all hosts in the **execution_nodes** group sections of the inventory file. Additionally, the installer automatically creates instance groups for any **instance_group_** prefixes in the inventory file.

Adding Nodes to the Automation Mesh

Adding new mesh nodes consists of updating the appropriate sections in the inventory file and running the installation script again.

Removing Nodes from the Automation Mesh

Removing nodes can be done by updating the appropriate entries in the inventory file and appending **node_state=deprovision** and running the installation script again.

Automation Controller Section

The **node_state=deprovision** can't be the first variable in the **automationcontroller** section. If that is required for the first entry, the order must be changed in inventory to accomodate this requirement.

11.2.2. Visualizing Automation Mesh Topology

The inventory file controls all setup and communication of Ansible Automation Mesh. As of AAP 2.2, there is a topology viewer added to the AAP 2.2 WebUI in **Administration** ⇒ **Topology View**. More importantly, **graphviz** is also supported as part of the **setup.sh** installation script and is capable of generating a topology graphic at the command line.

Generating the Mesh Topology

1. Install the **graphviz** package

```
[student@workstation ~]$ sudo dnf install graphviz
```

2. Run the **setup.sh** with the **--tag generat_dot_file**

```
[student@workstation aap2.2-bundle]$ ./setup.sh -- --tags generate_dot_file
```

3. Convert the **.dot** to a JPEG image

```
[student@workstation aap2.2-bundle]$ dot -Tjpg mesh-topology.dot -o graph-topology.jpg
```

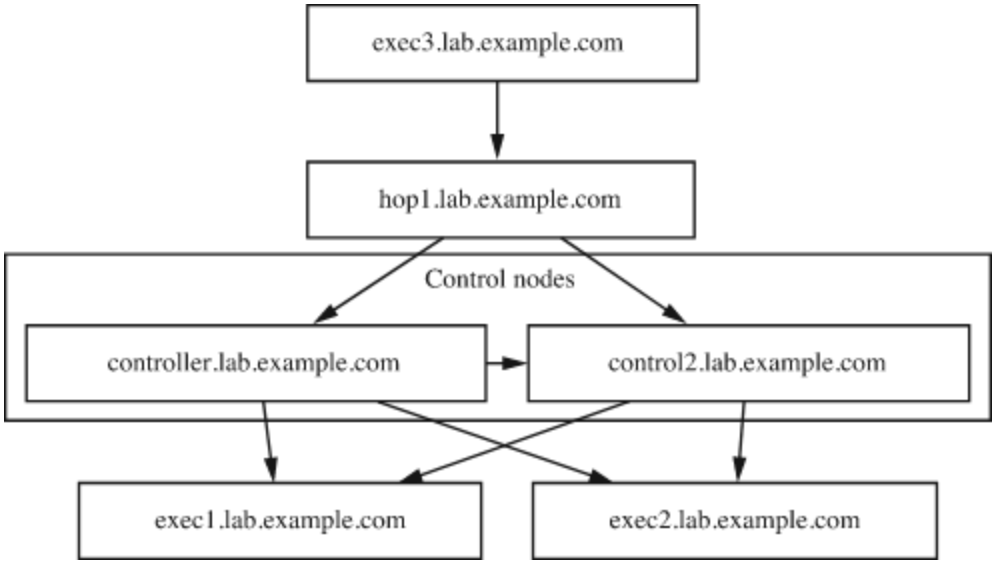

Figure 51. Sample Topology View from Installation Inventory File

Topology Image Arrows

Even though arrows are shown for the connections between nodes, they don't actually indicate direction or network flow. Thus they should be ignored as they can be misleading.

11.2.3. Automation Mesh Design Patterns

Automation mesh is flexible and can adapt to meet the needs of the environment.

11.2.4. Validation Checks

Installer script runs validation checks

- Hosts belong to single groups
- Hosts aren't peered to non-existen nodes
- Host isn't peered to itself
- Host doesn't have inbound/outbound connection to the same nodes
- Execution nodes have clear path to the control plane

11.3. Managing Distributed Execution with Automation Mesh

11.3.1. Managing Instance Groups in Automation Controller

The **controlplane** instance group is used to synchronize project content and perform various maintenance tasks. The **default** instance group contains the execution nodes for any user jobs not specifying which instance group to use.

Creating Instance Groups

The **inventory** file can be modified with **instance_group_<IG_Name>** to specify one or more instance groups.

controlplane and default Instance Groups

You cannot delete the **controlplane** and **default** instance groups.

11.3.1.1. Creating Instance Groups

Administration ⇒ Instance Groups ⇒ Add ⇒ Add instance group

11.3.1.2. Assigning Execution Nodes to an Instance Group

Administration ⇒ Instance Groups ⇒ <IG Name Link>

Click the **Instances ⇒ Associate** and select all the execution nodes that you want in the instance group.

Running a Health Check on the Nodes

Automation Controller monitors the health of instances. Manually running a health check select one or more instances and click **Health Check**.

Disassociating a Node from an Instance Group

Dissociating a node removes the node from an instance group. This can be done when you need to remove nodes from a cluster or place in a different instance group.

Administration ⇒ Instance Groups ⇒ <IG Name Link> ⇒ Instances then select nodes you want to remove and click **Disassociate** to confirm.

Disassociating Nodes

Disassociating nodes from instance groups doesn't remove from the cluster. In order to remove from the cluster (mesh) you need to have **node_state=deprovision** in the inventory file and run the **setup.sh** again to properly remove the node.

11.3.2. Assigning Default Instance Groups to Inventories and Job Templates

Inventories and job templates can be configured to use specific instance groups by default. This will inform controller to launch hobs on these instance groups and can control which execution nodes will execute and run the playbooks.

Configuring an Inventory to Use Instance Groups

Resources ⇒ **Inventories** ⇒ **Edit Inventory** then search for and select instance group to use.

Configure a Job Template to Use Instance Groups

Resources ⇒ **Templates** ⇒ **Edit Template** and the search for and select the instance group to use.

Running a Job Template with Instance Groups

Jobs always run in instance groups. The **Details** page can display the instance group used for running a job.

Resources ⇒ **Templates** ⇒ **Launch Job Template**

After the job completes, look at the details page and you should see the name of the instance group that executed the job.

11.3.3. Testing the Resilience of Automation Mesh

Testing Control Plane Resilience

Ensure project and assets are synchronized and up-to-date. Then execute a run of the job sync job. Look at the node performing the sync of the project. After this has completed, **Administration** ⇒ **Instances** ⇒ **Execution Node** ⇒ **Disable** and then repeat project sync. This should still pass on a different execution node.

11.3.3.1. Testing Execution Plane Resilience

Ensure project and assets are synchronized and up-to-date. Then execute a run of the job template. Look at the node performing the execution of the playbook. After this has completed, **Administration** ⇒ **Instances** ⇒ **Execution Node** ⇒ **Disable** and then **Relaunch Job** to relaunch the job. This should still run the playbook on a different execution node.

11.3.4. Monitoring Automation Mesh from the Web UI

Administration ⇒ **Topology View**

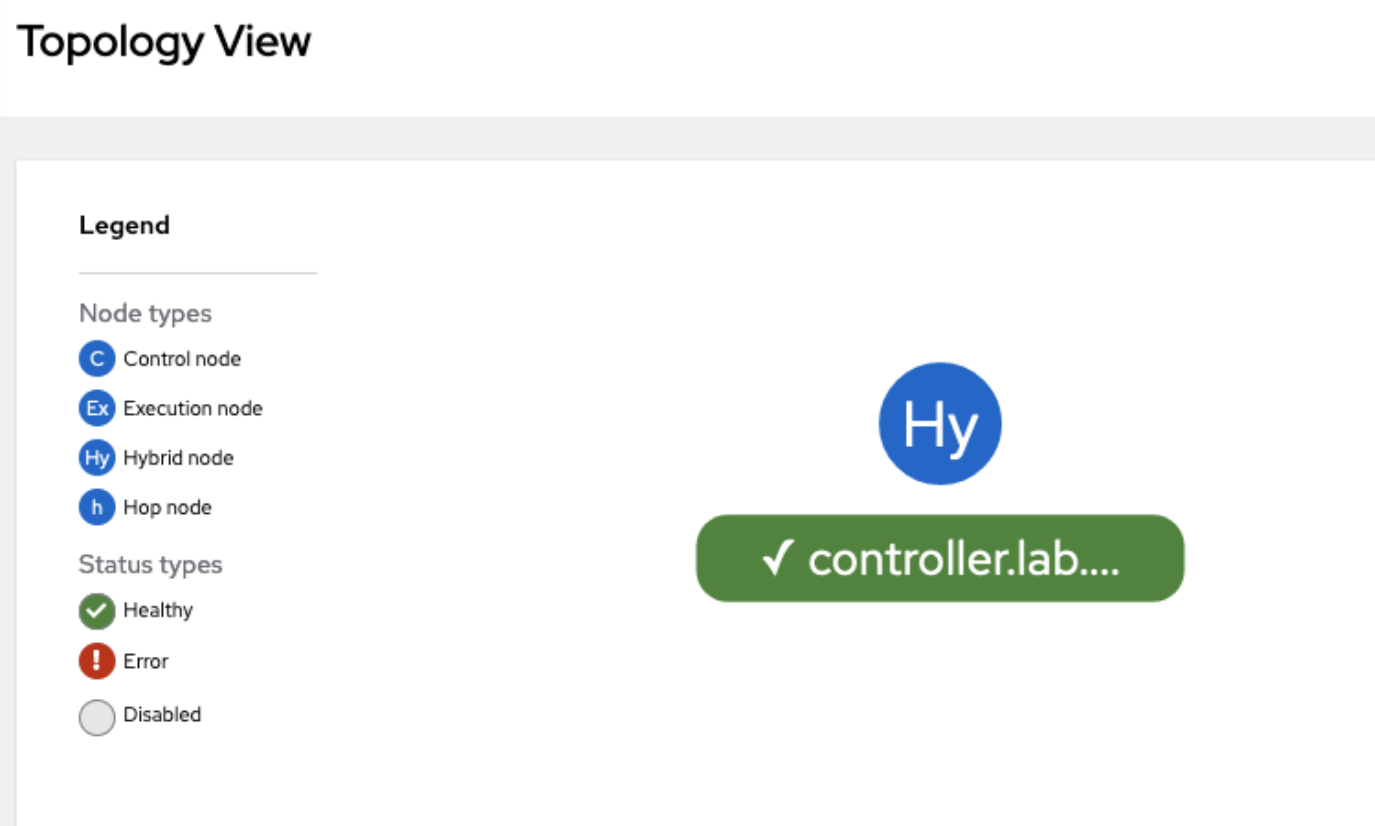

Figure 52. Single Automation Controller - Hybrid

- Healthy nodes are marked green with a checkmark
- Unavailable nodes are red with an exclamation point **!**
- Disabled nodes are grey with a circle

11.3.5. Monitoring Automation Mesh from the Command Line

Listing Nodes and Instance Groups

The **awx-manage** command can provide management of Ansible Controller. The **awx-manage list_instances** will list configured instances for Ansible Automation Mesh.

Listing 59. Using awx-manage

```
awx-manage list_instances
```

11.3.5.1. Monitoring Automation Mesh Using the receptorctl Command

The **receptorctl** command can test communication on the Ansible Automation Mesh network.

receptorctl Sub-Commands

- **receptorctl status** - Status of entire automation mesh
- **receptorctl ping** - Connectivity between current node and another node in mesh

- **receptorctl tracert** - Determines route and communication latency between current node and another node in the mesh.

Using the receptorctl Command

In order to use the **receptorctl** commands, you **MUST** specify the **receptor** sock to test connectivity or get general status.

Listing 60. Status

```
receptorctl --socket /var/run/awx-receptor/receptor.sock status
```

Listing 61. Ping Test

```
receptorctl --socket /var/run/awx-receptor/receptor.sock ping
exec2.lab.example.com
```

Listing 62. Tracert Test

```
receptorctl --socket /var/run/awx-receptor/receptor.sock traceroute
exec3.lab.example.com
```

11.3.6. DEMO: Using Automation Mesh

Example 17. DEMO: Verifying Execution Nodes and Automation Mesh

1. Run playbook to Create Instance Group

```
[student@workstation Instance_Groups]$ ansible-navigator run
Create_Instance_Group.yml
```

2. Run playbook to Job using the Instance Group

```
[student@workstation Instance_Groups]$ ansible-navigator run
Create_Job_Template_IG.yml
```

3. Open Automation Controller WebUI and verify the instance group was created.

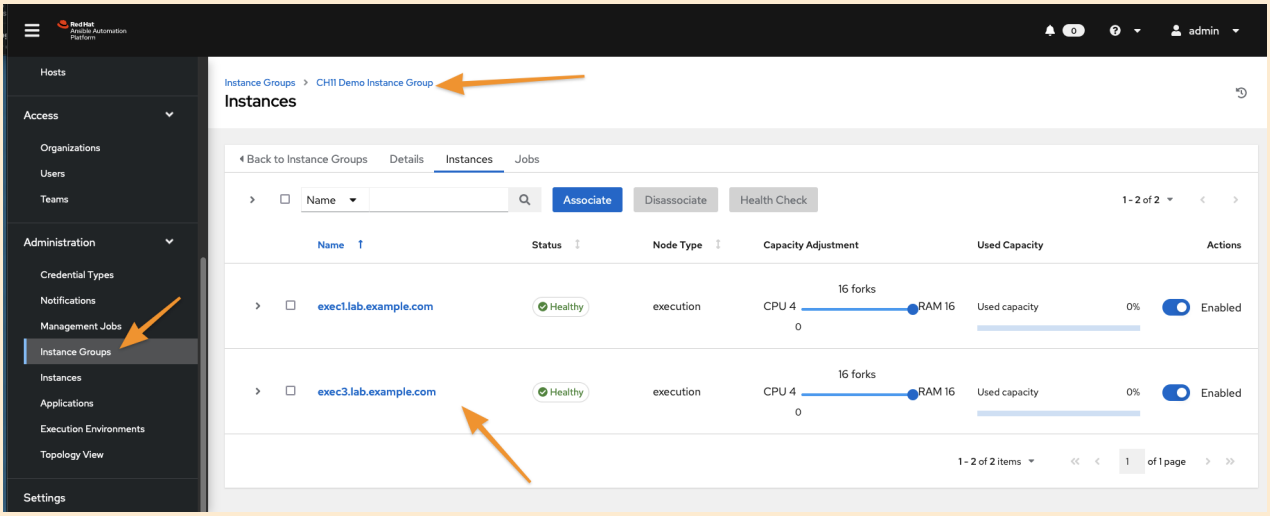

Figure 53. Verifying Instance Group Contains EXEC1 and EXEC3 nodes

4. Verify that the Job Template was created and it has the instance group selected.

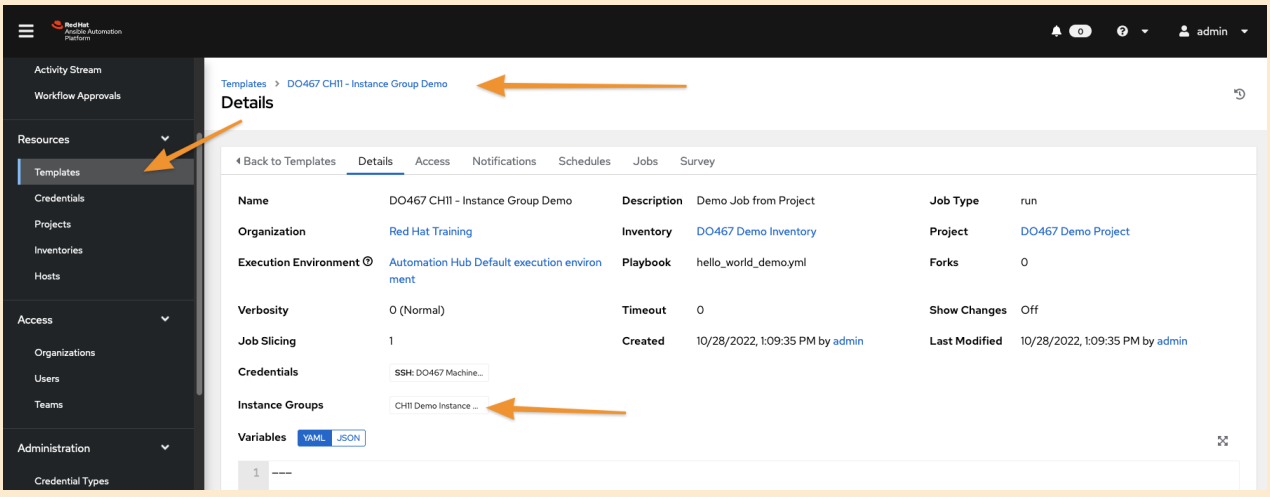

Figure 54. Verifying Instance Group Assigned to Job Template

5. Run the job and verify which execution nodes executed the job.

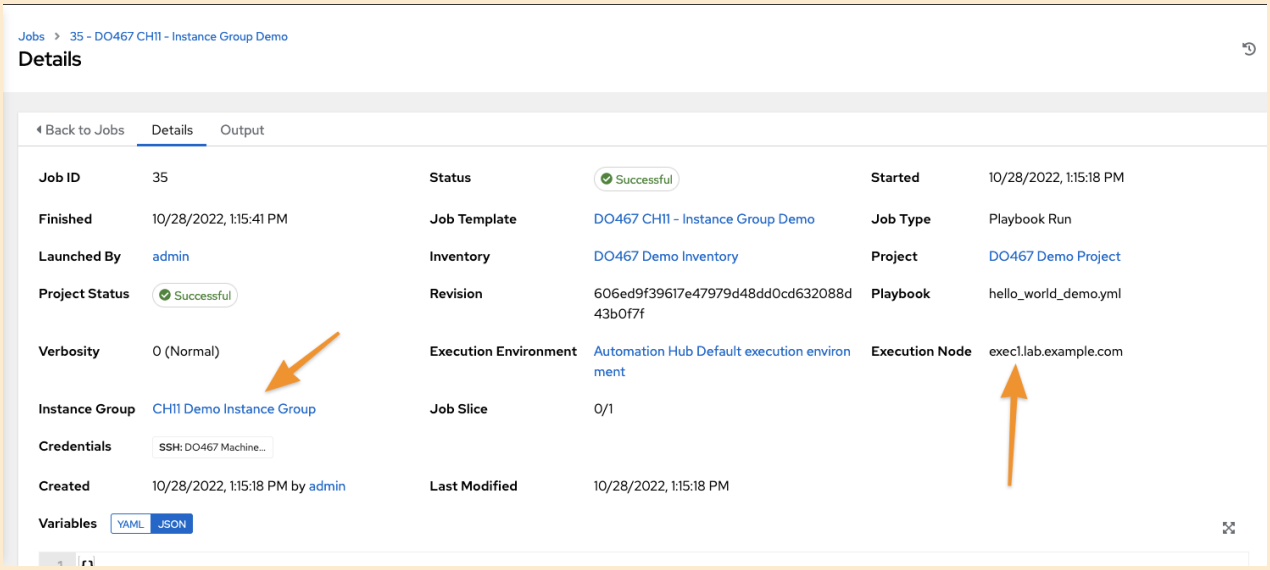

Figure 55. Verifying Execution Node Used on a Job

6. Navigate to **Instances** and turn off the Execution node that the job ran on previously.

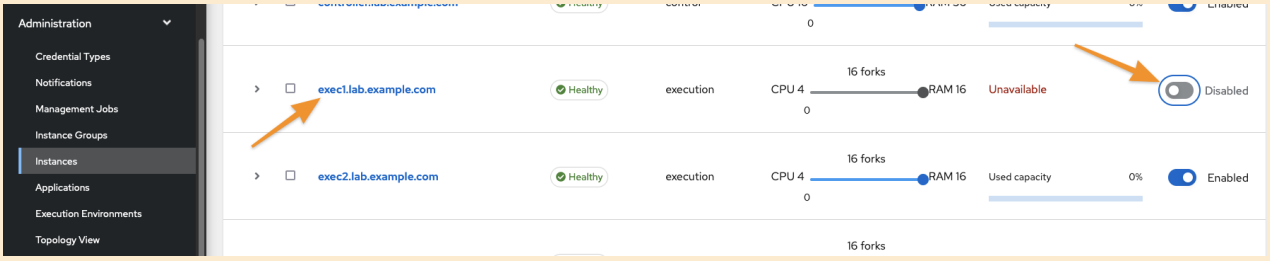

Figure 56. Disable an Execution Node from Instance Group

7. Launch job again and verify that it did run and it switched to a new Execution Node automatically.

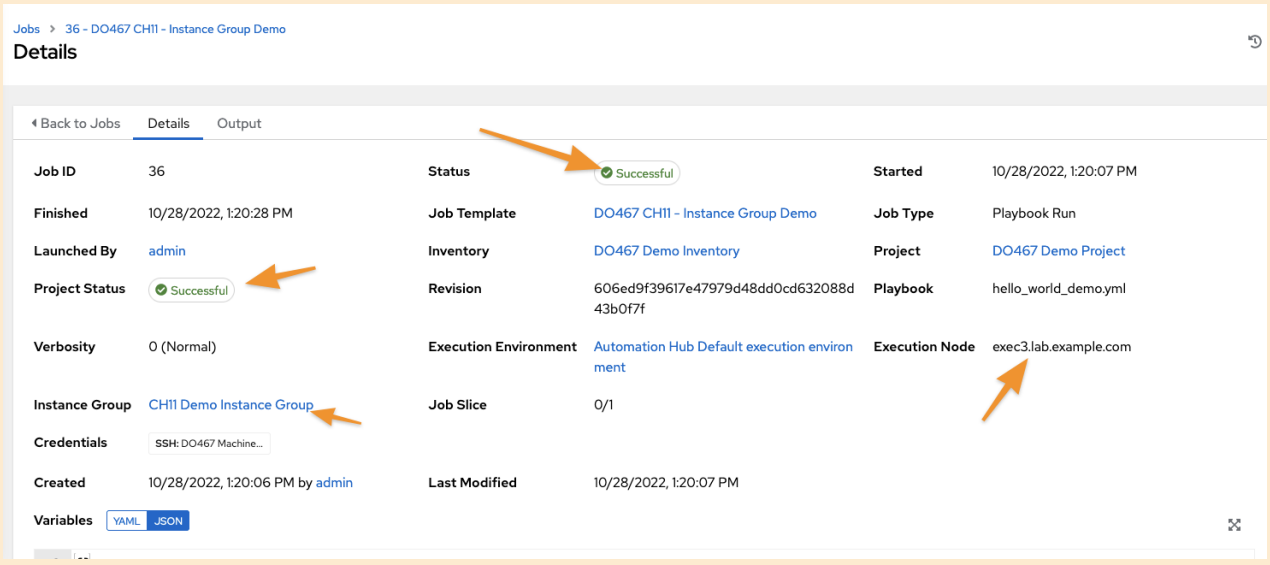

Figure 57. Verification that Instance Group still works

View Instance Groups from CLI with awx-manage

1. Run the **awx-manage list_instances** from Controller as the **root** user.

Listing 63. Instances

```
[root@controller ~]#
[root@controller ~]# awx-manage list_instances
[controlplane capacity=52 policy=100%]
    control2.lab.example.com capacity=16 node_type=control version=4.2.0
heartbeat="2022-10-28 16:22:32"
    controller.lab.example.com capacity=36 node_type=control version=4.2.0
heartbeat="2022-06-14 15:47:18"

[default capacity=24 policy=100%]
    exec1.lab.example.com capacity=4 node_type=execution version=ansible-runner-
2.2.0 heartbeat="2022-11-07 20:12:56"
    exec2.lab.example.com capacity=16 node_type=execution version=ansible-runner-
2.2.0 heartbeat="2022-10-28 16:22:32"
    exec3.lab.example.com capacity=4 node_type=execution version=ansible-runner-
2.2.0 heartbeat="2022-11-03 15:19:03"

[CH11 Demo Instance Group capacity=8]
    exec1.lab.example.com capacity=4 node_type=execution version=ansible-runner-
2.2.0 heartbeat="2022-11-07 20:12:56"
    exec3.lab.example.com capacity=4 node_type=execution version=ansible-runner-
2.2.0 heartbeat="2022-11-03 15:19:03"

[ungrouped capacity=0]
    hop1.lab.example.com node_type=hop heartbeat="2022-10-28 16:22:34"
```

Use receptorctl for Mesh Troubleshooting Tasks

1. Use **receptorctl** to get basic network socket information

```
[root@controller ~]# receptorctl --socket /var/run/awx-receptor/receptor.sock
status
Node ID: controller.lab.example.com
Version: 1.2.3
System CPU Count: 4
System Memory MiB: 5747

Connection                 Cost
hop1.lab.example.com       1
control2.lab.example.com   1
exec1.lab.example.com      1
exec2.lab.example.com      1
```

```
Known Node                  Known Connections
control2.lab.example.com    controller.lab.example.com: 1 exec1.lab.example.com: 1
exec2.lab.example.com: 1 hop1.lab.example.com: 1
controller.lab.example.com control2.lab.example.com: 1 exec1.lab.example.com: 1
exec2.lab.example.com: 1 hop1.lab.example.com: 1
exec1.lab.example.com       control2.lab.example.com: 1 controller.lab.example.com:
1
exec2.lab.example.com       control2.lab.example.com: 1 controller.lab.example.com:
1
exec3.lab.example.com       hop1.lab.example.com: 1
hop1.lab.example.com        control2.lab.example.com: 1 controller.lab.example.com:
1 exec3.lab.example.com: 1

Route                       Via
control2.lab.example.com    control2.lab.example.com
exec1.lab.example.com       exec1.lab.example.com
exec2.lab.example.com       exec2.lab.example.com
exec3.lab.example.com       hop1.lab.example.com
hop1.lab.example.com        hop1.lab.example.com

Node                        Service   Type       Last Seen             Tags
exec3.lab.example.com       control   StreamTLS  2022-11-08 11:34:35   {'type':
'Control Service'}
exec2.lab.example.com       control   StreamTLS  2022-11-08 11:33:37   {'type':
'Control Service'}
controller.lab.example.com control   StreamTLS  2022-11-08 11:34:35   {'type':
'Control Service'}
exec1.lab.example.com       control   StreamTLS  2022-11-08 11:34:27   {'type':
'Control Service'}
control2.lab.example.com    control   StreamTLS  2022-11-08 11:34:33   {'type':
'Control Service'}
hop1.lab.example.com        control   StreamTLS  2022-11-08 11:34:34   {'type':
'Control Service'}

Node                        Secure Work Types
exec3.lab.example.com       ansible-runner
exec2.lab.example.com       ansible-runner
controller.lab.example.com local, kubernetes-runtime-auth, kubernetes-incluster-
auth
exec1.lab.example.com       ansible-runner
control2.lab.example.com    local, kubernetes-runtime-auth, kubernetes-incluster-
auth
```

2. Perform a ping test to ensure a Node is reachable

```
[root@controller ~]#  receptorctl --socket /var/run/awx-receptor/receptor.sock
ping  exec2.lab.example.com
Reply from exec2.lab.example.com in 401.409µs
Reply from exec2.lab.example.com in 309.066µs
Reply from exec2.lab.example.com in 568.982µs
```

Listing 64. Ping on Node with Hop

```
[root@controller ~]#  receptorctl --socket /var/run/awx-receptor/receptor.sock
ping  exec3.lab.example.com
Reply from exec3.lab.example.com in 848.953µs
Reply from exec3.lab.example.com in 808.675µs
Reply from exec3.lab.example.com in 1.202818ms
```

3. Look at **traceroute** statistics to determine additional information.

```
[root@controller ~]#  receptorctl --socket /var/run/awx-receptor/receptor.sock
traceroute  exec2.lab.example.com
0: controller.lab.example.com in 51.511µs
1: exec2.lab.example.com in 697.249µs
```

Listing 65. Traceroute on Node with Hop

```
[root@controller ~]#  receptorctl --socket /var/run/awx-receptor/receptor.sock
traceroute  exec3.lab.example.com
0: controller.lab.example.com in 217.462µs
1: hop1.lab.example.com in 665.44µs
2: exec3.lab.example.com in 704.891µs
```

Specifying Socket with receptorctl

In order for the **receptorctl** to work, you must communicate with it properly over the ***receptor.sock*** which is the special socket connection established as part of the service. It is specified using the following syntax.

```
receptorctl --socket /var/run/awx-receptor/receptor.sock ①
```

① The **--socket /var/run/awx-receptor/receptor.sock** directive specifies the **socket** and the absolute path of the **receptor.sock**.

Appendix A: DO467 Exam Objectives

The **EX467** exam objectives are listed here: https://www.redhat.com/en/services/training/ex467-red-hat-certified-specialist-managing-automation-ansible-automation-platform-exam?section=Objectives. For convenience, a snapshot of the objectives have been placed in this guide.

- **Install Ansible Automation Platform**
  - Install Private Automation Hub
  - Install automation controller
  - Configure automation controller after installation (Multiple chapters for credentials, EEs, users, etc.)

Chapter 1, Section 2

GE: Installing Automation Controller and Private Automation Hub (p24)

Inventory Sections

- **[authomationcontroller]**
  - Contains FQDN for Controller
- **[automationhub]**
  - Contains the FQDN for the PVT Hub
- **[database]**
  - Contains FQDN for the Database
    - admin_password
    - pg_host
    - pg_password
    - registry_url
    - registry_username
    - registry_password

Other Passwords and Settings

There are a few other settings for Automation Hub needed in the installation file.

- **automationhub_admin_password**
- **automationhub_pg_host**
- **automationhub_pg_password**

Certificates

- **custom_ca_cert**
- **web_server_ssl_cert**
- **web_server_ssl_key**
- **automationhub_ssl_cert**
- **automationhub_ssl_key**
- **postgres_use_ssl**
- **postgres_ssl_cert**
- **postgres_ssl_key**

Listing 66. Sample Working Inventory File

```
[automationcontroller]
controller.lab.example.com
[automationcontroller:vars]
peers=execution_nodes
[execution_nodes]
[automationhub]
hub.lab.example.com
[automationcatalog]
[database]
db.lab.example.com
[sso]
[all:vars]
admin_password='redhat'
pg_host='db.lab.example.com'
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='redhat'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL
registry_url='hub.lab.example.com'
registry_username='admin'
registry_password='redhat'
receptor_listener_port=27199
automationhub_admin_password='redhat'
automationhub_pg_host='db.lab.example.com'
automationhub_pg_port=5432
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='redhat'
automationhub_pg_sslmode='prefer'
automationcatalog_pg_host=''
automationcatalog_pg_port=5432
automationcatalog_pg_database='automationservicescatalog'
automationcatalog_pg_username='automationservicescatalog'
automationcatalog_pg_password=''
custom_ca_cert=/home/student/certs/classroom-ca.pem
web_server_ssl_cert=/home/student/certs/controller.lab.example.com.crt
web_server_ssl_key=/home/student/certs/controller.lab.example.com.key
automationhub_ssl_cert=/home/student/certs/hub.lab.example.com.crt
automationhub_ssl_key=/home/student/certs/hub.lab.example.com.key
postgres_use_ssl=True
postgres_ssl_cert=/home/student/certs/db.lab.example.com.crt
postgres_ssl_key=/home/student/certs/db.lab.example.com.key
sso_keystore_password=''
sso_console_admin_password=''
```

Installing within Classroom Environment

Since there aren't enough resources, you will need to perform installation with the **setup.sh** script using **-e ignore_preflight_errors=true**.

Listing 67. Example Installation

```
[root@workstation aap2.2-bundle]# ./setup.sh -e
ignore_preflight_errors=true
```

- **Configure Private Automation Hub**
  - Upload content collections to Private Automation Hub
  - Upload execution environments to Private Automation Hub

Chapter 1, Section 3 (p 32)

GE: Initial Configuration of Automation Controller and Private Automation Hub (p43)

Multiple Ways of Loading EEIs

If images are local to the system, **podman** can be used to tag and push easily. Skopeo and other tools can be used and the there are Ansible tools to publish as well if you configure them. Choose the method easiest for you.

Listing 68. Podman

```
podman push image
```

Loading Collections

Collections must be built using a **namespace**. Before uploading collections into Automation Hub, the namespace must be created. After the collection has been uploaded, it will also need to be approved.

Collections ⇒ Approval

- **Manage access to Private Automation Hub**
  - Create groups and assign different permissions to groups
  - Create users and assign them to groups
  - Configure user types

Chapter 2, Section 3 (p73)

GE: Creating and Managing Users and Groups in Private Automation Hub (p79)

- **Manage access for automation controller**
  - Create automation controller users and teams
  - Associate users with teams associations
  - Configure organization roles
  - Configure user types

Chapter 2, Section 1 (p54) - Users

GE: Creating and Managing Automation Controller Users (p60)

Chapter 2, Section 2 (p63) - Teams

GE: Managing Automation Controller Access with Teams (p68)

- **Manage inventories and credentials**
  - Create host groups
  - Assign systems to host groups
  - Configure access to inventories
  - Configure inventory variables
  - Create and configure machine credentials to access hosts

Chapter 3, Section 1 (p94)

GE: Creating a Static Inventory (p102)

Chapter 3, Section 2 (p106)

GE: Creating Machine Credentials for Access to Inventory Hosts (p114)

- **Manage automation controller projects**
  - Create projects
  - Create job templates
  - Control user access to job templates
  - Launch jobs
  - Create a job template survey
  - Use variables from a survey in a Jinja template
  - Schedule jobs and manage notifications

Chapter 4, Section 1 (p124)

GE: Creating a Project for Ansible Playbooks (p131)

Chapter 4, Section 2 (p134)

GE: Creating Job Templates and Launching Jobs (p141)

Chapter 5, Section 2 (p162)

GE: Creating Job Template Surveys to Set Variables for Jobs (p167)

Chapter 5, Section 3 (p171)

GE: Scheduling Jobs and Configuring Notifications (p178)

- **Manage automation controller workflows**
  - Create workflow templates
  - Launch workflow jobs
  - Configure workflow jobs for automatic approval

Chapter 6, Section 1 (p192)

GE: Creating Workflow Job Templates and Launching Workflow Jobs (p201)

Chapter 6, Section 2 (p207)

GE: Requiring Approvals in Workflow Jobs (p212)

Documentation References

https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/1.0/html/getting_started_with_automation_services_catalog/configuring_approval

- **Manage advanced inventories**
  - Import external static inventories
  - Filter hosts with Smart Inventories

Chapter 7, Section 1 (p238)

GE: Importing External Static Inventories (p241)

Chapter 7, Section 3 (p255)

GE: Filtering Hosts with Smart Inventories (p259)

- **Work with the automation controller API**
  - Understand how to use an API script to obtain required parameters for launching jobs
  - Write an API scriptlet to launch a job

Chapter 8, Section 3 (p300)

GE: Launching Jobs with the Automation Controller API (p317)

- **Install and configure automation mesh**
  - Install automation mesh
  - Create and manage instance groups
  - Assign default instance groups to controller inventories or templates
  - Run controller job templates in a specific instance group

Chapter 11, Section 2 (p413)

GE: Deploying Distributed Execution with Automation Mesh (p420)

Chapter 11, Section 3 (p425)

GE: Managing Distributed Execution with Automation Mesh (p434)

- **Back up Ansible Ansible Automation Platform**

- Back up an instance of automation controller
- Back up an instance of Ansible Private Hub

Chapter 9, Section 2 (p347)

GE: Backing up and Restoring Red Hat Ansible Automation Platform (p351)

Appendix B: Examining Ansible Configuration Options

This section uses `ansible-config` to discover and investigate configuration options and to determine which options have been modified from the default settings.

B.1. Viewing Configuration Options

If you want to find out what options are available in the configuration file, use the `ansible-config list` command. It will display an exhaustive list of the available configuration options and their default settings. This list may vary depending on the version of Ansible that you have installed and whether you have any additional Ansible plugins on your control node.

Each option displayed by `ansible-config list` will have a number of key-value pairs associated with it. These key-value pairs provide information on how that option works. For example, the option `ACTION_WARNINGS` displays the following key-value pairs:

Key	Value	Purpose
`description`	[By default Ansible will issue a warning when received from a task action (module or action plugin). These warnings can be silenced by adjusting this setting to False.]	Describes what this configuration option is for.
`type`	`boolean`	What the type is for the option: `boolean` means true-false value.
`default`	`true`	The default value for this option.
`version_added`	`2.5`	The version of Ansible that added this option, for backward compatibility.
`ini`	`{ key: action_warnings, section: defaults }`	Which section of the INI-like inventory file contains this option, and the name of the option in the configuration file (`action_warnings`, in the `defaults` section).
`env`	`ANSIBLE_ACTION_WARNINGS`	If this environment variable is set, it will override any setting of the option made in the configuration file.

B.2. Determining Modified Configuration Options

When working with configuration files, you might want to find out which options have been set to values which are different from the built-in defaults.

You can do this by running the `ansible-config dump -v --only-changed` command. The `-v` option displays the location of the `ansible.cfg` file used when processing the command. The `ansible-config` command follows the same order of precedence mentioned previously for the `ansible` command. Output will vary depending on the location of the `ansible.cfg` file and which directory the `ansible-config` command is ran from.

In the following example, there is a single ansible configuration file located at `/etc/ansible/ansible.cfg`. The `ansible-config` command is first ran from student's home directory, then from a working directory with the same results:

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles',
u'/usr/share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles',
u'/usr/share/ansible/roles']
```

However, if you have a custom `ansible.cfg` file in your working directory, the same command will display information based on where it is ran from and the relative `ansible.cfg` file.

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles',
u'/usr/share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ cat ansible.cfg
[defaults]
inventory = ./inventory
remote_user = devops

[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
Using /home/student/workingdirectory/ansible.cfg as config file
DEFAULT_HOST_LIST(/home/student/workingdirectory/ansible.cfg) =
[u'/home/student/workingdirectory/inventory']
DEFAULT_REMOTE_USER(/home/student/workingdirectory/ansible.cfg) = devops
```

ansible-config(1) man page

Configuration file: Ansible Documentation

Appendix C: References and Additional Information

Ansible Docs/Tips and Tricks

- **Installing Software and other Packages**: https://ansible-tips-and-tricks.readthedocs.io/en/latest/os-dependent-tasks/installing_packages/
- **Ansible Tips and Tricks (Examples)**: https://github.com/nfaction/ansible-tips-and-tricks/wiki
- **Ansible Product Demos**: https://github.com/ansible/product-demos
- **Ansible Workshops**: https://github.com/ansible/workshops/tree/devel/provisioner
- **Red Hat CoP - Automation Good Practices**:
  - https://redhat-cop.github.io/automation-good-practices/
  - https://github.com/redhat-cop/automation-good-practices/
- **Ansible Controller Collection**: https://console.redhat.com/ansible/automation-hub/repo/published/ansible/controller/docs?keywords=

Ansible KB Articles and Solutions

- **How Do I Perform Security Patching / OS Package Upgrades On Ansible Tower/Automation Controller Nodes Without Breaking Any Ansible Tower/Automation Controller Functionality ?**: https://access.redhat.com/solutions/4566711

Ansible Filters and Collections

- **Using filters to manipulate data (Jinja2 Templating)**: https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html
- **Community General**: https://docs.ansible.com/ansible/latest/collections/community/general/index.html

Ansible Blogs and Articles

- **When localhost isn't what it seems in Red Hat Ansible Automation Platform 2**: https://www.ansible.com/blog/when-localhost-isnt-what-it-seems-in-red-hat-ansible-automation-platform-2

Ansible Execution Environments

- **Execution Environments**: https://docs.ansible.com/automation-controller/4.2.0/html/userguide/execution_environments.html#ee-mount-options