Comprehensive, hands-on training that solves real world problems

Automation with Ansible II: Ansible Tower

Student Workbook (ROLE)

DO409-AT3.1-en-1-20170609

AUTOMATION WITH ANSIBLE II: ANSIBLE TOWER

Ansible Tower 3.1 DO409
Automation with Ansible II: Ansible Tower
Edition 1 20170609 20170609

Authors: Chen Chang, Fabien Cambi, Artur Glogowski, Ricardo da Costa
Editor: Steven Bonneville

Our thanks for feedback from Timothy Appnel, Graham Mainwaring, Bill Nottingham, David Federlein, and Nikhil Jain.

Document Conventions vii
Notes and Warnings vii

Introduction ix
Automation with Ansible II: Ansible Tower ix
Orientation to the Classroom Environment x
Internationalization xiii

1. Installing Ansible Tower and Describing Ansible Tower's Architecture 1
Introduction to Ansible Tower 2
Quiz: Introduction to Ansible Tower 6
Installing Ansible Tower 8
Guided Exercise: Installing Ansible Tower 13
Orientation to the Ansible Tower Web Interface 15
Guided Exercise: Orientation to the Ansible Tower Web Interface 24
Quiz: Installing Ansible Tower and Describing Ansible Tower's Architecture 27
Summary 29

2. Creating Users and Teams for Role-Based Access Control 31
Creating and Managing Ansible Tower Users 32
Guided Exercise: Creating and Managing Ansible Tower Users 38
Managing Users Efficiently with Teams 41
Guided Exercise: Managing Users Efficiently with Teams 44
Lab: Creating Users and Teams for Role-Based Access Control 49
Summary 53

3. Creating and Managing Inventories and Credentials 55
Creating a Static Inventory 56
Guided Exercise: Creating a Static Inventory 65
Creating Machine Credentials for Access to Inventory Hosts 69
Guided Exercise: Creating Machine Credentials for Access to Inventory Hosts 76
Lab: Creating and Managing Inventories and Credentials 79
Summary 83

4. Managing Projects for Provisioning with Ansible Tower 85
Managing Ansible Project Materials Using Git 86
Guided Exercise: Managing Ansible Project Materials Using Git 94
Creating a Project for Ansible Playbooks and Roles 99
Guided Exercise: Creating a Project for Ansible Playbooks and Roles 107
Creating Job Templates and Launching Jobs 110
Guided Exercise: Creating Job Templates and Launching Jobs 118
Lab: Managing Projects for Provisioning with Ansible Tower 120
Summary 126

5. Constructing Advanced Job Workflows 127
Creating Job Template Surveys to Set Variables for Jobs 128
Guided Exercise: Creating Job Template Surveys to Set Variables for Jobs 135
Creating Workflow Job Templates and Launching Workflow Jobs 138
Guided Exercise: Creating Workflow Job Templates and Launching Workflow Jobs 144
Configuring and Receiving Automatic Notification of Job Success and Failure 147
Guided Exercise: Configuring and Receiving Automatic Notification of Job Success and Failure 150
Lab: Constructing Advanced Job Workflows 154
Summary 161

6. Updating Inventories Dynamically and Comparing Inventory Members 163
Creating and Updating Dynamic Inventories 164
Guided Exercise: Creating and Updating Dynamic Inventories 173
Monitoring Configuration Drift with Scan Jobs 175
Guided Exercise: Monitoring Configuration Drift with Scan Jobs 181
Lab: Updating Inventories Dynamically and Comparing Inventory Members 183
Summary 187

7. Performing Maintenance and Routine Administration of Ansible Tower 189
Performing Basic Troubleshooting of Ansible Tower 191
Guided Exercise: Performing Basic Troubleshooting of Ansible Tower 196
Configuring TLS/SSL for Ansible Tower 202
Guided Exercise: Configuring TLS/SSL for Ansible Tower 205
Performing Command-Line Management with **`tower-manage`** 207
Guided Exercise: Performing Command-Line Management with **`tower-manage`** 209
Backing Up and Restoring an Ansible Tower Installation 212
Guided Exercise: Backing Up and Restoring an Ansible Tower Installation 216
Launching Jobs with the Ansible Tower API and **`tower-cli`** 219
Guided Exercise: Launching Jobs with the Ansible Tower API and **`tower-cli`** 231
Quiz: Performing Maintenance and Routine Administration of Ansible Tower 237
Summary 239

8. Comprehensive Review: Provisioning and Managing Systems using Ansible Tower 241
Lab: Restoring Ansible Tower from Backup 242
Lab: Adding Users and Teams 245
Lab: Creating a Custom Dynamic Inventory 251
Lab: Configuring Job Templates 257
Lab: Configuring Workflow Job Templates, Surveys, and Notifications 266
Lab: Testing the Prepared Environment 273

Document Conventions

Notes and Warnings

Note

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

Important

"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.

Warning

"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.

References

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

Introduction

Automation with Ansible II: Ansible Tower

Automation with Ansible II: Ansible Tower (DO409) is designed for IT professionals who use Ansible and need to centrally manage their Ansible projects in a way that scales to large teams and complex enterprise installations, using Ansible Tower.

Objectives

- Deploy and use Ansible Tower to manage their existing Ansible projects, playbooks, and roles
- Use the visual dashboard to centrally launch, control, and monitor Ansible jobs
- Configure users and teams and use them to control access to systems, projects, and other resources through role-based access controls
- Automatically schedule Ansible jobs and update the host inventory
- Perform basic maintenance and administration of the Ansible Tower installation
- Use the Tower API to launch jobs from existing templates

Audience

- Linux system administrators, cloud administrators, and network administrators interested in centrally managing Ansible projects and playbook execution at scale with Ansible Tower.

Prerequisites

- Successfully completed "Automation with Ansible" (DO407) or has a solid familiarity with Ansible, Ansible playbooks, and roles
- RHCSA in RHEL or equivalent Linux system administration skills recommended

Orientation to the Classroom Environment

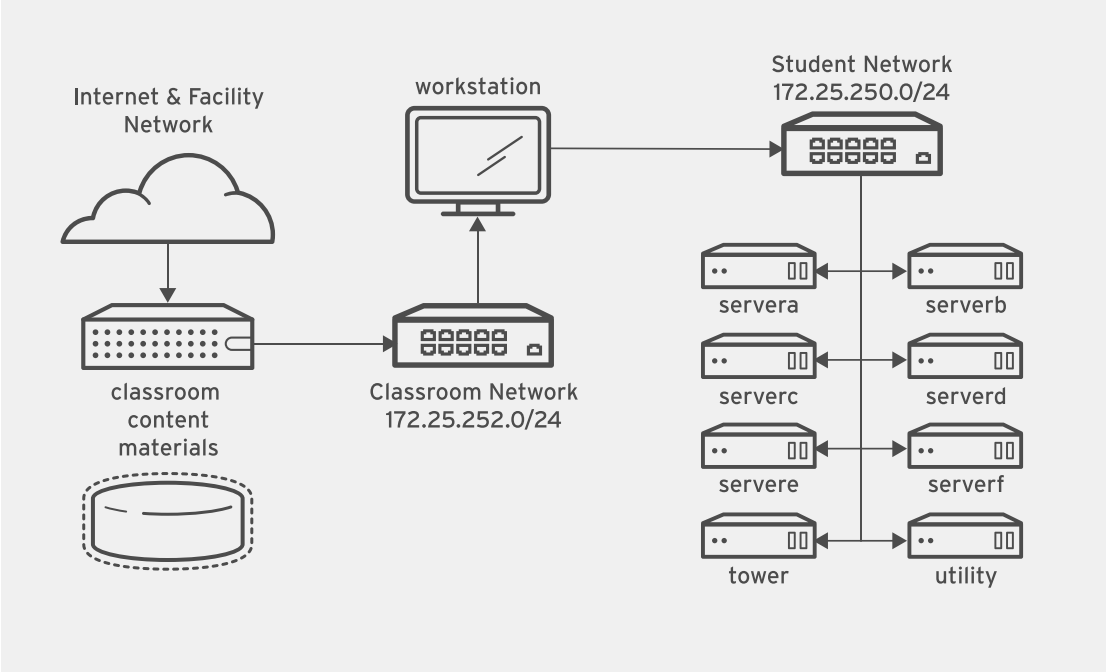

Figure 0.2: Classroom Environment

In this course, the main computer system used for hands-on learning activities is **`workstation`**. Eight other machines will also be used by students for these activities. These are **`tower`**, **`utility`**, **`servera`**, **`serverb`**, **`serverc`**, **`serverd`**, **`servere`**, **`serverf`**. All nine of these systems are in the **`lab.example.com`** DNS domain.

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

Classroom Machines

Machine name	IP addresses	Role
tower.lab.example.com	172.25.250.8	Host used for Ansible Tower
utility.lab.example.com	172.25.250.9	Host used for IPA
workstation.lab.example.com	172.25.250.254	Graphical workstation used to connect to and manage Ansible Tower
servera.lab.example.com	172.25.250.10	Host managed with Ansible Tower
serverb.lab.example.com	172.25.250.11	Host managed with Ansible Tower
serverc.lab.example.com	172.25.250.12	Host managed with Ansible Tower
serverd.lab.example.com	172.25.250.13	Host managed with Ansible Tower

Machine name	IP addresses	Role
servere.lab.example.com	172.25.250.14	Host managed with Ansible Tower
serverf.lab.example.com	172.25.250.15	Host managed with Ansible Tower

One additional function of **workstation** is that it acts as a router between the network that connects the student machines and the classroom network. If **workstation** is down, other student machines will only be able to access systems on the student network.

There are several systems in the classroom that provide supporting services. Two servers, **content.example.com** and **materials.example.com** are sources for software and lab materials used in hands-on activities. Information on how to use these servers will be provided in the instructions for those activities.

Controlling your station
The top of the console describes the state of your machine.

Machine States

State	Description
none	Your machine has not yet been started. When started, your machine will boot into a newly initialized state (the desk will have been reset).
starting	Your machine is in the process of booting.
running	Your machine is running and available (or, when booting, soon will be.)
stopping	Your machine is in the process of shutting down.
stopped	Your machine is completely shut down. Upon starting, your machine will boot into the same state as when it was shut down (the disk will have been preserved).
impaired	A network connection to your machine cannot be made. Typically this state is reached when a student has corrupted networking or firewall rules. If the condition persists after a machine reset, or is intermittent, please open a support case.

Depending on the state of your machine, a selection of the following actions will be available to you.

Machine Actions

Action	Description
Start Station	Start ("power on") the machine.
Stop Station	Stop ("power off") the machine, preserving the contents of its disk.
Reset Station	Stop ("power off") the machine, resetting the disk to its initial state. **Caution: Any work generated on the disk will be lost**.
Refresh	Refresh the page will re-probe the machine state.
Increase Timer	Adds 15 minutes to the timer for each click.

The Station Timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to 60 minutes when your machine is started.

The timer operates as a "dead man's switch," which decrements as your machine is running. If the timer is winding down to 0, you may choose to increase the timer.

Internationalization

Language support

Red Hat Enterprise Linux 7 officially supports 22 languages: English, Assamese, Bengali, Chinese (Simplified), Chinese (Traditional), French, German, Gujarati, Hindi, Italian, Japanese, Kannada, Korean, Malayalam, Marathi, Odia, Portuguese (Brazilian), Punjabi, Russian, Spanish, Tamil, and Telugu.

Per-user language selection

Users may prefer to use a different language for their desktop environment than the system-wide default. They may also want to set their account to use a different keyboard layout or input method.

Language settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the **Region & Language** application. Run the command **`gnome-control-center region`**, or from the top bar, select **(User)** › **Settings**. In the window that opens, select **Region & Language**. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications, including **`gnome-terminal`**, started inside it. However, they do not apply to that account if accessed through an **`ssh`** login from a remote system or a local text console (such as **`tty2`**).

Note

A user can make their shell environment use the same **`LANG`** setting as their graphical environment, even when they log in through a text console or over **`ssh`**. One way to do this is to place code similar to the following in the user's **`~/.bashrc`** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
  | sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, or other languages with a non-Latin character set may not display properly on local text consoles.

Individual commands can be made to use another language by setting the **`LANG`** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
```

```
jeu. avril 24 17:55:01 CDT 2014
```

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to check the current value of **LANG** and other related environment variables.

Input method settings
GNOME 3 in Red Hat Enterprise Linux 7 automatically uses the **IBus** input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The **Region & Language** application can also be used to enable alternative input methods. In the **Region & Language** application's window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

Once more than one input method is configured, the user can switch between them quickly by typing **Super**+**Space** (sometimes called **Windows**+**Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may find also this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary-shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.

Note

Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl**+**Shift**+**U**, followed by the code point. After **Ctrl**+**Shift**+**U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.

For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl**+**Shift**+**U**, then **03bb**, then **Enter**.

System-wide default language settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, *root* can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it will display the current system-wide locale settings.

To set the system-wide language, run the command **localectl set-locale LANG=*locale***, where *locale* is the appropriate **$LANG** from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in **/etc/locale.conf**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from **Region & Language** and clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the login screen will also adjust the system-wide default language setting stored in the **/etc/locale.conf** configuration file.

Important

Local text consoles such as **tty2** are more limited in the fonts that they can display than **gnome-terminal** and **ssh** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a local text console. For this reason, it may make sense to use English or another language with a Latin character set for the system's text console.

Likewise, local text consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through **localectl** for both local text virtual consoles and the X11 graphical environment. See the **localectl**(1), **kbd**(4), and **vconsole.conf**(5) man pages for more information.

Language packs

When using non-English languages, you may want to install additional "language packs" to provide additional translations, dictionaries, and so forth. To view the list of available langpacks, run **yum langavailable**. To view the list of langpacks currently installed on the system, run **yum langlist**. To add an additional langpack to the system, run **yum langinstall *code***, where *code* is the code in square brackets after the language name in the output of **yum langavailable**.

References

locale(7), **localectl**(1), **kbd**(4), **locale.conf**(5), **vconsole.conf**(5), **unicode**(7), **utf-8**(7), and **yum-langpacks**(8) man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in **localectl** can be found in the file **/usr/share/X11/xkb/rules/base.lst**.

Language Codes Reference

Language Codes

Language	$LANG value
English (US)	en_US.utf8
Assamese	as_IN.utf8
Bengali	bn_IN.utf8
Chinese (Simplified)	zh_CN.utf8
Chinese (Traditional)	zh_TW.utf8
French	fr_FR.utf8
German	de_DE.utf8
Gujarati	gu_IN.utf8
Hindi	hi_IN.utf8
Italian	it_IT.utf8
Japanese	ja_JP.utf8
Kannada	kn_IN.utf8
Korean	ko_KR.utf8
Malayalam	ml_IN.utf8
Marathi	mr_IN.utf8
Odia	or_IN.utf8
Portuguese (Brazilian)	pt_BR.utf8
Punjabi	pa_IN.utf8
Russian	ru_RU.utf8
Spanish	es_ES.utf8
Tamil	ta_IN.utf8
Telugu	te_IN.utf8

redhat. TRAINING

CHAPTER 1

INSTALLING ANSIBLE TOWER AND DESCRIBING ANSIBLE TOWER'S ARCHITECTURE

Overview	
Goal	Explain what Ansible Tower is and demonstrate a basic ability to navigate and use its web user interface.
Objectives	• Describe the architecture, use cases, and installation requirements of Ansible Tower. • Install a new Ansible Tower on a single node using the setup.sh script. • Navigate and describe the Ansible Tower web user interface, and successfully launch a job using the demo job template, project, credential, and inventory.
Sections	• Introduction to Ansible Tower (and Quiz) • Installing Ansible Tower (and Guided Exercise) • Orientation to the Ansible Tower Web Interface (and Guided Exercise)
Quiz	• Installing Ansible Tower and Describing Ansible Tower's Architecture

Introduction to Ansible Tower

Objectives

After completing this section, students should be able to describe the architecture, use cases, and installation requirements of Ansible Tower.

Why Ansible Tower?

As an enterprise's experience with Ansible matures, it often finds additional opportunities for leveraging Ansible to simplify and improve IT operations. The same Ansible playbooks utilized by operations teams to deploy production systems can also be used to deploy identical systems in earlier stages of the software development lifecycle. When automated with Ansible, complex production support tasks typically handled by skilled engineers can easily be delegated to and resolved by entry-level technicians.

However, sharing an existing Ansible infrastructure to scale IT automation across an enterprise can present some challenges. While properly written Ansible playbooks can be leveraged across teams, Ansible does not provide any facilities for managing their shared access. Additionally, though playbooks may allow for the delegation of complex tasks, their execution may require highly privileged and guarded administrator credentials.

IT teams often vary in their preferred tool sets. While some may prefer the direct execution of playbooks, other teams may wish to trigger playbook execution from existing continuous integration and delivery tool suites. In addition, those that traditionally work with GUI-based tools may find Ansible's pure command-line interface intimidating and awkward.

Ansible Tower overcomes many of these problems by providing a framework for running and managing Ansible efficiently on an enterprise scale. Tower eases the administration involved with sharing an Ansible infrastructure while maintaining organization security by introducing features such as a centralized web interface for playbook management, *role-based access control (RBAC)*, and centralized logging and auditing. Its REST API ensures that Tower integrates easily with an enterprise's existing workflows and tool sets. Tower's API and notification features make it particularly easy to associate Ansible playbooks with other tools such as Jenkins, CloudForms, or Red Hat Satellite, to enable continuous integration and deployment. It provides mechanisms to enable centralized use and control of machine credentials and other secrets without exposing them to end users of Ansible Tower.

Ansible Tower Architecture

Ansible Tower is a Django web application designed to run on a Linux server as an on-premise, self-hosted solution which layers on top of an enterprise's existing Ansible infrastructure.

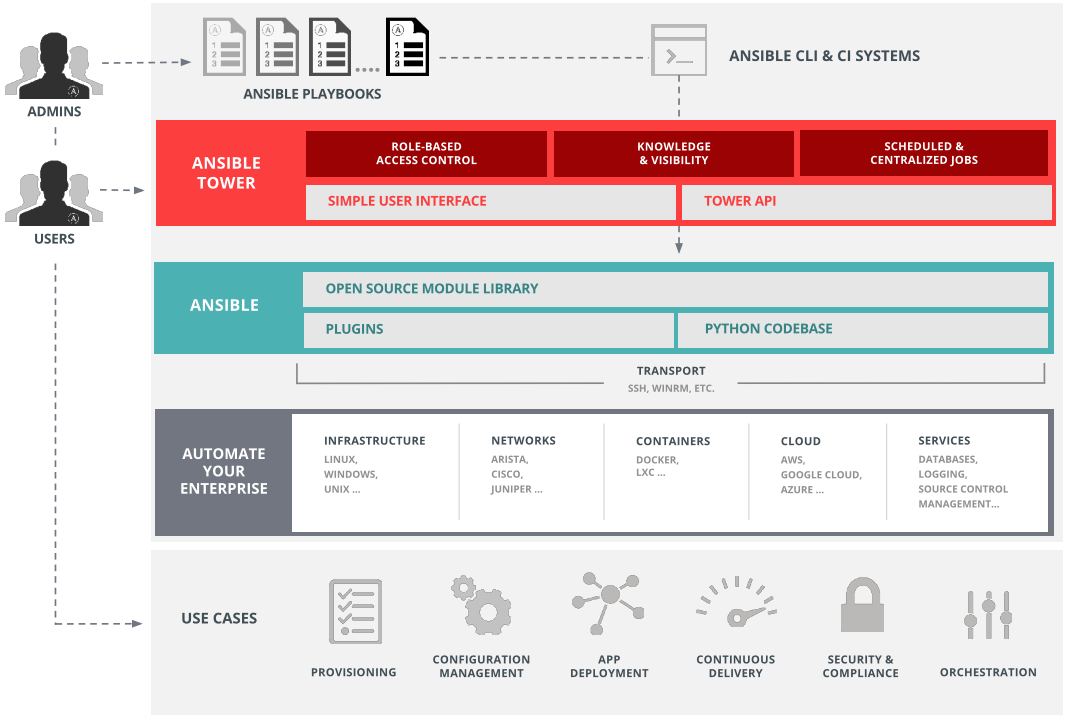

Figure 1.1: Ansible Tower architecture

Users interface with their enterprise's underlying Ansible infrastructure through either Tower's web interface or its RESTful API. Tower's web interface is a graphical interface wrapper which performs its actions by executing calls against the Tower API. Any action available through the Tower web interface can therefore also be performed through Tower's RESTful API. The RESTful API is essential for those users looking to integrate Ansible with existing software tools and processes.

Tower stores its data in a PostgreSQL back-end database and makes use of the RabbitMQ messaging system. Versions of Ansible Tower prior to version 3.0 also relied on a MongoDB database. This dependency has since been removed and data is now stored solely in a PostgreSQL database.

Depending on an enterprise's needs, Ansible Tower can be implemented using one of the following architectures.

Single Machine with Integrated Database

All Tower components, the web front-end, RESTful API back-end, and PostgreSQL database, reside on a single machine. This is the standard architecture.

Single Machine with Remote Database

Tower web front-end and RESTful API back-end are installed on a single machine while the PostgreSQL database is installed remotely on another server on the same network. The remote database can be hosted on a server with an existing PostgreSQL instance outside the management of Tower. Another option is to have the Tower installer create a Tower-managed PostgreSQL instance on the remote server and populate it with the Tower database.

High Availability Multi-Machine Cluster
Older Tower versions offered a redundant, active-passive Tower architecture consisting of a single active node and one or more inactive nodes. Starting with Tower 3.1, this architecture is now replaced by an active-active, high-availability cluster consisting of multiple active Tower nodes.

Each node in the cluster hosts the Tower web front-end and RESTful API back-end, and can receive and process requests. In this cluster architecture, the PostgreSQL database is hosted on a remote server. The remote database can reside either on a server with an existing PostgreSQL instance outside the management of Tower or on a server with a Tower-managed PostgreSQL instance created by the Tower installer.

Note

This course focuses on the most straightforward architecture to deploy, a single Ansible Tower server with an integrated database.

Ansible Tower features

Two types of license are available for Ansible Tower: basic and enterprise. Enterprise license offers access to all Tower features. Basic license offers access to only a subset of Tower's features and does not include many enterprise-level options, such as system tracking, logging aggregation, and clustering.

The following are just some of the many features offered by Ansible Tower for controlling, securing, and managing Ansible in an enterprise environment.

Visual Dashboard
The Tower web interface opens into a dashboard screen which provides a summary view of an enterprise's entire Ansible environment. The Tower Dashboard allows administrators to easily see the current status of hosts and inventories, as well as the results of recent job executions.

Role-based Access Control (RBAC)
Tower utilizes a Role-Based Access Control (RBAC) system which maintains security while streamlining user access management. It simplifies the delegation of user access to Tower objects such as Organizations, Projects, and Inventories.

Graphical Inventory Management
Tower offers users the ability to create inventory groups and add inventory hosts through its web interface. Inventories can also be updated from an external inventory source such as from public cloud providers, local virtualization environments, and an organization's custom *configuration management database (CMDB)*.

Job Scheduling
Tower offers users the ability to schedule playbook execution and updates from external data sources either on a one-time basis or to be repeated at regular intervals. This allows routine tasks to be performed unattended and is especially useful for tasks such as backup routines which should ideally be executed during operational off-hours.

Real-Time and Historical Job Status Reporting
When playbook executions are initiated in Tower, the web interface displays the playbook's output and execution results in real time. Results of previously executed jobs and scheduled job runs are also made available by Tower.

Push-Button Automation
Ansible simplifies IT automation while Tower takes it a step further by enabling user self-service. Tower's streamlined web interface, coupled with the flexibility of its RBAC system, allows administrators to safely delegate complex tasks as single click-of-a-button routines.

Remote Command Execution
Tower makes the on-demand flexibility of Ansible's ad-hoc commands available through its remote command execution feature. User permissions for remote command execution is enforced using Tower's RBAC system.

Credential Management
Tower centrally manages the credentials which are used for authentication purposes: to do things like running Ansible plays on managed hosts, synchronizing information from dynamic inventory sources, and importing Ansible project content from version control systems. It encrypts the passwords or keys provided so that they can not be retrievable by Tower users. Users can be granted the ability to use or replace these credentials without actually exposing them to the user.

Centralized Logging and Auditing
All playbook and remote command executions initiated on Tower are logged. This provides the ability to audit when each job was executed and by whom. In addition, Tower offers the ability to integrate its log data into third party logging aggregation solutions, such as Splunk and Sumologic.

Integrated Notifications
Tower Notifications can be used to signal when Tower job executions succeed or fail. Notifications can be delivered using many different protocols, including email, Slack, and HipChat.

Multi-Playbook Workflows
Complex operations often involve the serial execution of multiple playbooks. Tower's multi-playbook workflows allow users to chain together multiple playbooks to facilitate the execution of complex routines involving provisioning, configuration, deployment, and orchestration. An intuitive workflow editor also helps to simplify the modeling of multi-playbook workflows.

System Tracking
Tower can be configured to routinely scan managed hosts and record their states. The collected data can be used to audit system changes over time. Additionally, this feature can be used to compare and detect differences between systems.

RESTful API
Tower's RESTful API exposes every Tower feature available through Tower's web interface. The API's browsable format makes it self-documenting and simplifies lookup of API usage information.

References

Ansible Tower Administration Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/administration

Quiz: Introduction to Ansible Tower

Choose the correct answer(s) to the following questions:

1. Which of the following three features are provided by Ansible Tower? (Choose three.)

    a. Role-based access control
    b. Playbook creator wizard
    c. Centralized logging
    d. RESTful API

2. Which of the following two enhancements are ***additions*** to Ansible provided by Ansible Tower? (Choose two.)

    a. Playbook development
    b. Remote execution
    c. Multi-play workflows
    d. Monitoring
    e. Version Control
    f. Graphical inventory management

3. Which of the following three architectures are supported by the Tower installer? (Choose three.)

    a. Single machine with integrated database
    b. Single machine with an external database hosted on a separate server on the network.
    c. High-availability, multi-machine cluster with an external database
    d. Active/passive redundancy multi-machine with an external database

4. Which of the following two features are not provided by a basic Ansible Tower license? (Choose two.)

    a. Tower dashboard
    b. Job scheduling
    c. System tracking
    d. Role-based access control
    e. Logging aggregation

Solution

Choose the correct answer(s) to the following questions:

1. Which of the following three features are provided by Ansible Tower? (Choose three.)

   a. Role-based access control
   b. Playbook creator wizard
   c. Centralized logging
   d. RESTful API

2. Which of the following two enhancements are ***additions*** to Ansible provided by Ansible Tower? (Choose two.)

   a. Playbook development
   b. Remote execution
   c. Multi-play workflows
   d. Monitoring
   e. Version Control
   f. Graphical inventory management

3. Which of the following three architectures are supported by the Tower installer? (Choose three.)

   a. Single machine with integrated database
   b. Single machine with an external database hosted on a separate server on the network.
   c. High-availability, multi-machine cluster with an external database
   d. Active/passive redundancy multi-machine with an external database

4. Which of the following two features are not provided by a basic Ansible Tower license? (Choose two.)

   a. Tower dashboard
   b. Job scheduling
   c. System tracking
   d. Role-based access control
   e. Logging aggregation

Installing Ansible Tower

Objectives

After completing this section, students should be able to install a new Ansible Tower on a single node using the **`setup.sh`** script.

Installation Requirements

Ansible Tower can be installed and is supported on 64-bit x86_64 versions of Red Hat Enterprise Linux 7, CentOS 7, Ubuntu 14.04 LTS, and Ubuntu 16.04 LTS. The following are the requirements for installing Ansible Tower on a Red Hat Enterprise Linux 7 system. Details may vary slightly for other supported operating systems.

Operating System
For Red Hat Enterprise Linux installations, the Ansible Tower 3.1 server is supported on systems running Red Hat Enterprise Linux 7.2 or later on the 64-bit x86_64 processor architecture.

Web Browser
A currently supported version of Mozilla Firefox or Google Chrome is required for connecting to the Ansible Tower web interface. Other HTML5-compliant web browsers may work but are not fully tested or supported.

Memory
A minimum of 2 GB of RAM is required on the Tower host. 4 GB or more is recommended.

The actual memory requirement is dependent upon the maximum number of hosts Ansible Tower is expected to configure in parallel. This is managed by the **`forks`** configuration parameter in the job template or system configuration. Red Hat recommends that 4 GB of memory be available for each 100 forks. A deeper discussion on Ansible Tower memory requirements and how they are influenced by job concurrency settings can be found in the *Ansible Tower User Guide*, in the "Job Concurrency" section of the "Jobs" chapter, at *http://docs.ansible.com*.

Disk Storage
At least 20 GB of hard disk space is required for Ansible Tower. In order for the Ansible Tower installation to succeed, 10 GB of this space must be available for the **`/var`** directory.

This minimum disk storage requirement does not include storage required for Tower's System Tracking feature. If this feature is enabled, calculate additional storage requirement based on the number of hosts tracked, scans performed, and the amount of data collected. A formula to estimate this is available in the "Requirements" chapter of the *Ansible Tower Installation and Reference Guide*.

Ansible
Installation of Ansible Tower is performed by executing a shell script that runs an Ansible playbook. Older versions of Ansible Tower required that the latest stable version of Ansible was installed prior to installation, but the current installation program will automatically attempt to install Ansible and its dependencies if they are not already present.

Red Hat Enterprise Linux 7 users must enable the **`extras`** repository.

If the RPM package for Ansible (or DEB for Ubuntu) is already installed on the system, the Tower installer will not attempt to reinstall it. If you have installed Ansible on the server using the

Python package manager **pip**, the Tower installer *will* attempt to reinstall Ansible. For Tower to work correctly, the latest stable version of Ansible should be installed using your distribution's package manager.

SELinux
Ansible Tower supports the **targeted** SELinux policy, which can be set to enforcing mode, permissive, or disabled. Other SELinux policies are not supported.

Managed clients
The installation requirements above apply to the Ansible Tower server, not to the machines it manages with Ansible. Those systems should meet the usual requirements for machines managed with the version of Ansible installed on the Ansible Tower server.

Ansible Tower Licensing and Support

Administrators interested in evaluating Ansible Tower can obtain a trial license at no cost. Instructions on how to get started are available at *https://www.ansible.com/tower-trial*.

Administrators interested in progressing beyond trial licensing can choose from three types of Tower subscriptions:

- Self-Support

  Targeted at small deployments, this includes a basic Tower subscription, with maintenance and upgrades of the software but no technical support or service level agreement (SLA). Some "enterprise" features of Tower are not included. Versions supporting up to 250 managed nodes are available, larger deployments should look at the Enterprise subscriptions.

- Enterprise: Standard

  The Enterprise: Standard edition includes an enterprise Tower subscription with entitlement to all Tower features and 8x5 technical support. Pricing is based on the number of nodes managed. Ansible Playbook support is included, which at the time of writing includes help with runtime execution problems in Tower, assistance with errors and tracebacks, and limited recommended practice guidance on the current or previous minor release of Ansible.

- Enterprise: Premium

  The Enterprise: Premium edition also includes an enterprise Tower subscription with software maintenance and upgrades and all Tower features, but with entitlement to 24x7 technical support. Pricing is based on the number of nodes managed, and Ansible Playbook support is included.

Detailed information on Ansible Tower pricing and support is available at *http://www.ansible.com*.

Licenses for Tower Components
Tower makes use of various software components, some of which may be open source. Licenses for each of these specific components are provided under the **/usr/share/doc/ansible-tower** directory.

Ansible Tower Installers

Two different installation packages are available for Ansible Tower.

The standard **setup** Ansible Tower installation program can be downloaded from **http://releases.ansible.com/ansible-tower/setup/**. The latest version of Ansible Tower for

RHEL 7 is always be located at **http://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.el7.tar.gz**. This archive is smaller, but requires Internet connectivity to download Ansible Tower packages from various package repositories.

A different, bundled installer for RHEL 7 is available at **http://releases.ansible.com/ansible-tower/setup-bundle/ansible-tower-setup-bundle-latest.el7.tar.gz**. This archive includes an initial set of RPM packages for Ansible Tower so that it may be installed on systems disconnected from the Internet. Those systems still need to be able to get software packages for Red Hat Enterprise Linux 7 and the RHEL 7 Extras channel from reachable sources. This may be preferred by administrators in higher security environments. This installation method is not currently available for Ubuntu.

Installing Tower

The following is the procedure for using the bundled installer to install Ansible Tower on a single RHEL 7.3 system with access to the Red Hat Enterprise Linux 7 Extras repository. The exercise after this section will go over this in more detail.

1. As the **root** user, download the Ansible Tower setup bundle to the system.

2. Extract the Tower setup bundle and change into the directory containing the extracted contents.

```
[root@towerhost ~]# tar xzf ansible-tower-setup-bundle-3.1.1-1.el7.tar.gz
[root@towerhost ~]# cd ansible-tower-setup-bundle-3.1.1-1.el7
```

3. In that directory, the **inventory** file needs to be edited in order to set passwords for the Ansible Tower **admin** account (**admin_password**), the PostgreSQL database user account (**pg_password**), and the RabbitMQ messaging user account (**rabbitmq_password**).

```
[tower]
localhost ansible_connection=local

[database]

[all:vars]
admin_password='myadminpassword'

pg_host=''
pg_port=''

pg_database='awx'
pg_username='awx'
pg_password='somedatabasepassword'

rabbitmq_port=5672
rabbitmq_vhost=tower
rabbitmq_username=tower
rabbitmq_password='and-a-messaging-password'
rabbitmq_cookie=cookiemonster

# Needs to be true for fqdns and ip addresses
rabbitmq_use_long_name=false
```

Warning

These passwords can be anything and should be set to something secure. The **admin** user has full control of the Tower server, and the ports for PostgreSQL and the RabbitMQ service are exposed to external hosts by default.

4. Run the Ansible Tower installer by executing the **setup.sh** script.

```
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh
[warn] Will install bundled Ansible
Loaded plugins: langpacks, search-disabled-repos
Examining bundle/repos/epel/ansible-2.2.1.0-1.el7.noarch.rpm:
 ansible-2.2.1.0-1.el7.noarch
Marking bundle/repos/epel/ansible-2.2.1.0-1.el7.noarch.rpm to be installed
... Output omitted ...
The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-02-27-10:52:44.log
```

5. Once the installer has completed successfully, connect to the Ansible Tower system using a web browser. You should be redirected to an HTTPS login page.

   The web browser may generate a warning regarding a self-signed HTTPS certificate presented by the Ansible Tower website. Replacing the default self-signed HTTPS certificate for the Ansible Tower web interface with a properly CA-signed certificate is discussed later in this course.

6. Log in to the Ansible Tower web interface as the Tower administrator using the **admin** account and the password you set in the installer's **inventory** file.

7. Once successfully logged into the Tower web interface for the first time, you are prompted to enter a license and accept the end user license agreement. Enter the Ansible Tower license provided by Red Hat and accept the end user license agreement.

At this point, the **admin** user is presented with the Tower dashboard. The next section provides an orientation to the Tower interface in more detail.

References

Ansible Tower Quick Installation Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/quickinstall/

Ansible Tower Installation and Reference Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/installandreference/

Ansible Tower User Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/userguide/

Ansible Tower Administration Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/administration/

Guided Exercise: Installing Ansible Tower

In this exercise, you will install a single-node Ansible Tower instance on **tower**.

Outcomes
You should be able to successfully install Ansible Tower and its license, resulting in a running Ansible Tower server.

Before you begin
Run **lab tower-install setup** on **workstation** to configure **tower.lab.example.com** with a Yum repository containing package dependencies from the Red Hat Enterprise Linux and Extras repositories required for Ansible Tower installation.

```
[student@workstation ~]$ lab tower-install setup
```

Steps

1. Download the Ansible Tower setup bundle to the **tower** system.

   1.1. Log in to the **tower** system as the **root** user.

   ```
   [student@workstation ~]$ ssh root@tower
   ```

   1.2. Download the Ansible Tower setup bundle using the **curl** command.

   ```
   [root@tower ~]# curl -O -J http://content.example.com/ansible-tower3.1/x86_64/
   dvd/setup-bundle/ansible-tower-setup-bundle.el7.tar.gz
   ```

2. Extract the Tower setup bundle. Change into the directory containing the extracted contents.

   ```
   [root@tower ~]# tar xzf ansible-tower-setup-bundle.el7.tar.gz
   [root@tower ~]# cd ansible-tower-setup-bundle-3.1.1-1.el7
   ```

3. Set the passwords for the Ansible Tower administrator account, database user account, and messaging user account to **redhat** by modifying their respective entries in the **inventory** file used by the Tower installer playbook.

   ```
   [root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# grep password inventory
   admin_password='redhat'
   pg_password='redhat'
   rabbitmq_password='redhat'
   ```

4. Run the Ansible Tower installer by executing the **setup.sh** script. The script may take up to 30 minutes to complete. Ignore the errors in the script output as they are related to verification checks performed by the installer playbook.

   ```
   [root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh
   [warn] Will install bundled Ansible
   Loaded plugins: langpacks, search-disabled-repos
   ```

```
Examining bundle/repos/epel/ansible-2.2.1.0-1.el7.noarch.rpm:
 ansible-2.2.1.0-1.el7.noarch
Marking bundle/repos/epel/ansible-2.2.1.0-1.el7.noarch.rpm to be installed
... Output omitted ...
The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-02-27-10:52:44.log
```

5. Once the installer has completed successfully, exit the console session on the **tower** system.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# exit
```

6. Launch the Firefox web browser from **workstation** and connect to your Ansible Tower at **https://tower.lab.example.com**. Firefox warns you that the Ansible Tower server's security certificate is not secure. Add and confirm the security exception for the self-signed certificate.

7. Log in to the Tower web interface as the Tower administrator using the **admin** account and the **redhat** password.

8. Once you have successfully logged in to the Tower web interface for the first time, you are prompted to enter a license and accept the end user license agreement.

   Upload the Ansible Tower license and accept the end user license agreement.

   8.1. On **workstation**, download the Ansible Tower license provided at **http://materials.example.com/tower/install/Ansible-Tower-license.txt**.

   8.2. In the Tower web interface, click **BROWSE** and then select the license file downloaded earlier.

   8.3. Select the checkbox next to **I agree to the End User License Agreement** to indicate acceptance.

   8.4. Click **SUBMIT** to submit the license and accept the license agreement.

Orientation to the Ansible Tower Web Interface

Objectives

After completing this section, students should be able to navigate and describe the Ansible Tower web user interface, and successfully launch a job using the demo job template, project, credential, and inventory.

Using Ansible Tower

This section provides an overview of how to use the Ansible Tower web interface to launch a job with an example Ansible playbook, an inventory, and some access credentials for the machines in the inventory. Along the way, it provides an orientation to the web interface itself.

The basic idea is that Tower is configured with a number of Ansible *projects* that contain playbooks. It is also configured with a number of Ansible *inventories* and the necessary machine *credentials* to log in to inventory hosts and escalate privileges. A *job template* is set up by an administrator which specifies which playbook from which project should be run on the hosts in a particular inventory using particular machine credentials. A *job* happens when a user runs a playbook on an inventory by launching a job template.

Tower Dashboard

Upon successful login to the Tower web interface, users are presented with the Tower Dashboard, the main control center for Ansible Tower.

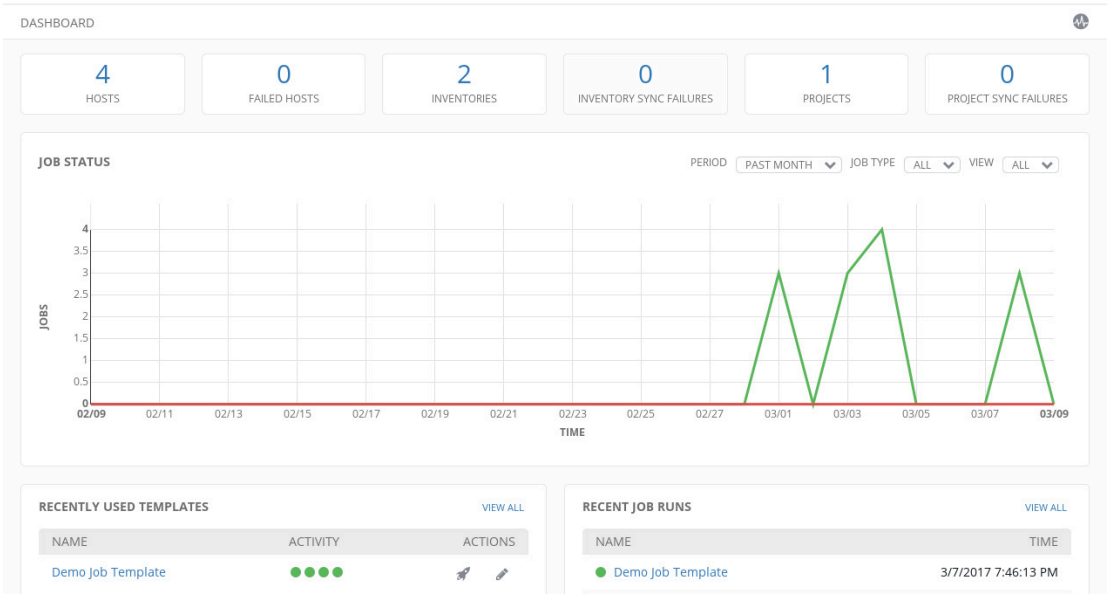

Figure 1.2: Tower Dashboard

This dashboard screen is composed of four reporting sections:

Summary

Across the top of the dashboard is a summary report of the status of managed hosts, inventories, and Ansible projects. Clicking on a cell in the summary section takes the user to the detailed dashboard screen for the reported metric.

Job Status
A *job* is an attempted run of a playbook by Ansible Tower. This section provides a graphical display of the number of successful and failed jobs over time. This graph can be adjusted in several ways:

- The **PERIOD** dropdown menu can be used to change the time window for the plotted graph between either the most recent day, week, or month.
- The **JOB TYPE** dropdown menu can be used to select which job types to include on the graph.
- The **VIEW** drop-down menu can be used to choose between graphing all job status, only failed job, or only successful jobs.

Recently Used Templates
This section reports a listing of job templates which was recently used for the execution of jobs.

- For each job template used, the results of each associated job run is indicated under the **ACTIVITY** column by a colored dot, with green indicating success and red indicating failure.
- Under the **ACTIONS** column are controls for the use and modification of the job template.
- Clicking the **VIEW ALL** link displays all job templates, not just the ones which have recently been used for job execution.

Recent Job Runs
This section provides a listing of recently executed jobs along with their date and time of execution. Each job run listed is preceded by a colored dot which represents the outcome of the run. A green dot represents a successful run while a red dot represents a failed run.

Quick Navigation Links

In the upper left portion of the Tower web interface is a collection of navigation links to commonly used Tower resources. The Ansible Tower icon links users back to the Tower Dashboard page. The Projects, Inventories, Templates, and Jobs links direct to the administrative screen for each of these four Tower resources.

Figure 1.3: Quick navigation links

- *Projects*: In Tower, a project represents a collection of related Ansible playbooks.
- *Inventories*: As in Ansible, inventories in Ansible Tower contain a collection of hosts to be managed.
- *Templates*: The template resource defines the parameters which are to be used for the execution of an Ansible playbook by Ansible Tower.
- *Jobs*: A job represents Tower's execution of an Ansible playbook against an inventory of hosts.

Administration Tool Links

Across the upper-right portion of the Tower web interface are links to various Tower administration tools.

Figure 1.4: Administration tool links

Account configuration
The current user account name is displayed as a link. Clicking the link directs users to their account configuration page, where they can modify their username, password, or account user type.

Settings
The gear icon links users to the **Settings** menu. This menu provides access to administrative interfaces for various Tower resources.

My View
To the right of the **Settings** icon is a report icon, which links to Tower's **My View** screen. This screen reports on the same information as the Tower Dashboard but only for the jobs executed by the logged in user.

View Documentation
The book icon to the right of the **My View** represents the **View Documentation** link. Clicking on this link opens the online Ansible Tower documentation web site in a new window.

Log Out
The power icon link to the right of the **View Documentation** icon is used to log out of the Tower web interface.

Tower Settings

As previously mentioned Tower's Settings menu can be accessed by clicking the Settings icon in the administration toolbar. The Settings menu provides access to the following management interfaces for Tower resources which are not accessible through the quick navigation links.

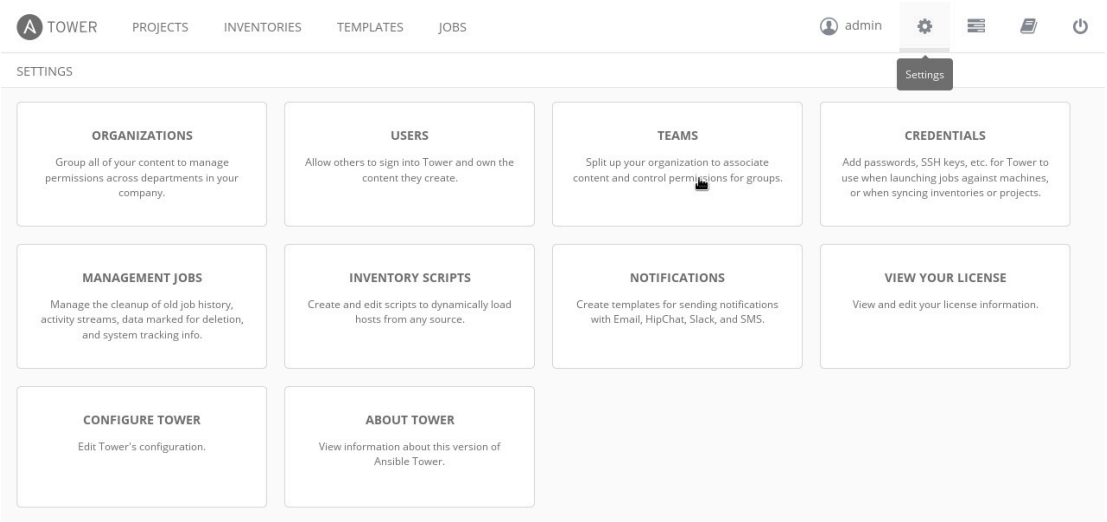

Figure 1.5: Settings

Organizations
This interface is used for managing organization entities within Tower. An organization represents a logical collection of other Tower resources, such as Teams, Projects, and Inventories.

Organizations are often used for departmental separation within an enterprise. An organization is the highest level at which Tower's role-based access control system can be applied.

Users
This interface allows for Tower user management. Users are granted access to Tower and then assigned roles which determine their access to Tower resources.

Teams
Tower Teams can be administered through this interface. Teams represent a group of Users. Like Users, teams can also be assigned roles for access to Tower resources.

Credentials
The management of Credentials is performed through this interface. Credentials are authentication data used by Tower to do such things as logging in to managed hosts to run plays, synchronizing inventory data from external sources, and downloading updated project materials from version control systems.

Management Jobs
This interface is used to administer system jobs which perform cleanup of metrics and activity information stored by Tower operations.

Inventory Scripts
This interface manages scripts used for the generation and update of dynamic inventories from external sources, such as cloud providers and configuration management databases (CMDBs)

Notifications
Administration of Notification Templates is conducted through this interface. These templates define the set of configuration parameters needed to generate notifications using a variety of message delivery tools, such as email, IRC, and HipChat.

View Your License
This interface provides details of the installed license and can also be used to perform administrative licensing tasks such as license installation and upgrade.

Configure Tower
Configurable options related to authentication, job execution, and the Tower web interface are performed through this interface.

About Tower
Information regarding the version of Tower installed can be displayed using this link.

General Controls

In addition to the navigational and administrative controls previously outlined, some additional controls are used throughout the Tower web interface.

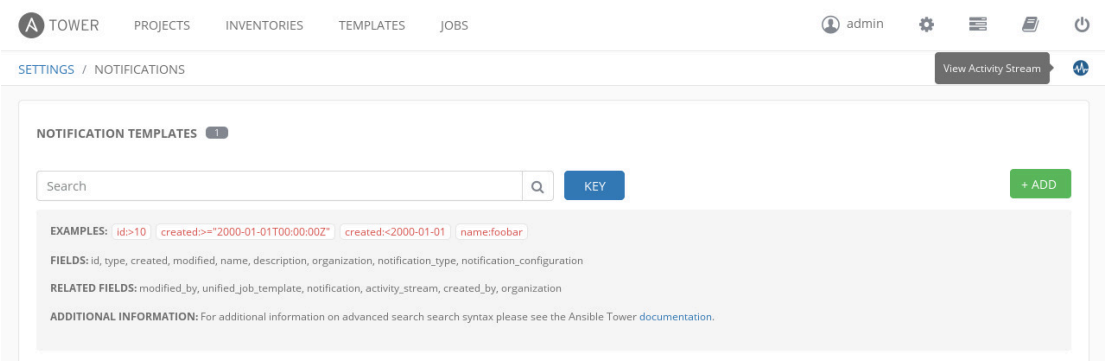

Figure 1.6: Key

Breadcrumb Navigation Links

As a user navigates through the Tower web interface, a set of breadcrumb navigation links is created just below the quick navigation links in the upper left corner of the screen. This series of links not only make clear the path which leads to each screen but also provides a quick way to return to a previous screens in the navigated path.

Activity Streams

On most screens in the Tower web interface, there is a **View Activity Stream** link under the Administrative Tools links at the top right portion. This link can be used to produce a page reporting activities which have occurred and are related to the screen that the button is clicked on. For example, when the Activity Streams button is clicked on the Projects screen, the information regarding the time, executor, and the nature of project-related activities is displayed.

Search Fields

Throughout the Tower web interface are search fields which can be used to search or filter through data sets. Each search field accepts search criteria which can be used to narrow down the search result.

Key

A guide detailing the correct syntax of the specific criteria for each search field can be displayed by clicking the **Key** button located next to the search field.

The **Key** button is also used to define the options provided by other input fields within Tower such as dropdown lists.

Configuration of a Job Template

When installed, Ansible Tower is preconfigured with a demonstration Job Template which can be used as an example to see how a Job Template is constructed and to test the operation of the Tower. The following discussion looks at the components that make up this example. The exercise that follows this section will provide a more detailed hands-on walk through.

1. Under **INVENTORIES**, a **Demo Inventory** has been configured. This is a static inventory with no hostgroups and one host, **`localhost`**. Clicking on that host in the inventory reveals that it has the inventory variable **`ansible_connection: local`** set.

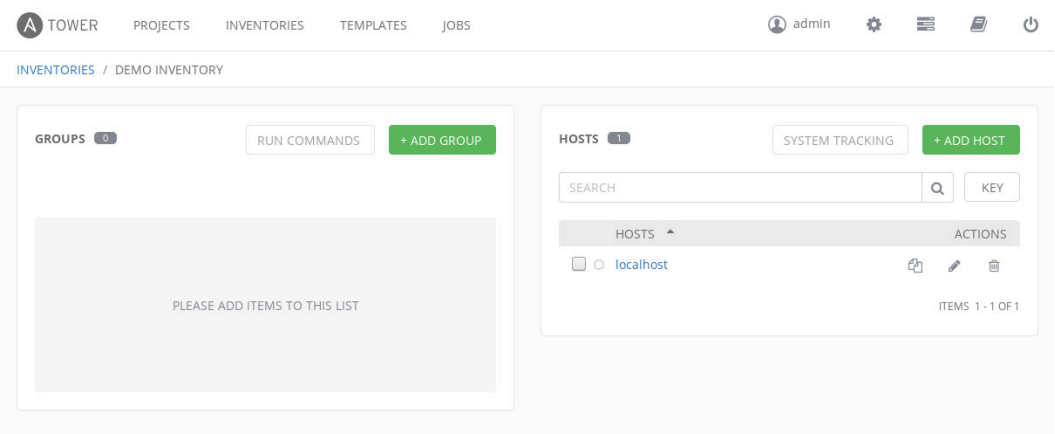

Figure 1.7: Demo Inventory

2. Under Settings (the gear icon), there is a **CREDENTIALS** link. A machine credential named **Demo Credential** has been created which contains information that could be used to authenticate access to machines in an inventory.

3. Under **PROJECTS**, a **Demo Project** has been configured. This Project is configured to get Ansible project materials, including playbooks, from a local directory on the Tower system.

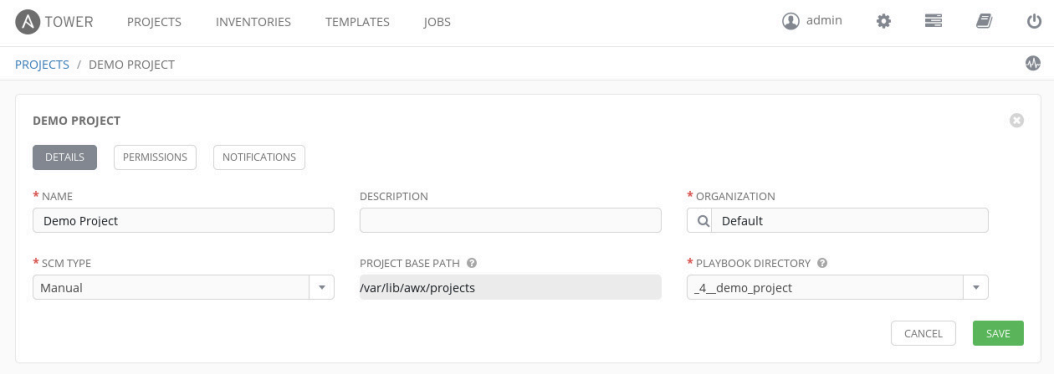

Figure 1.8: Demo Project

4. Under **TEMPLATES**, a **Demo Job Template** has been configured. This Job Template is configured as a normal playbook run (**Job Type** is **Run**), using the **`hello_world.yml`** playbook from **`Demo Project`**.

   It runs on the machines in the **`Demo Inventory`**, using the **`Demo Credential`** to authenticate to those machines. Privilege escalation will not be enabled (**`hello_world.yml`** doesn't need it). Were it needed, the **`Demo Credential`** would need to provide the necessary information. (In fact, because the Job Template is not using privilege escalation and is running only on machines using **`ansible_connection: local`**, there is very little information needed in the **`Demo Credential`**.)

   No extra variables are set (analogous to the **`-e`** or **`--extra-vars`** option of an **`ansible-playbook`** command).

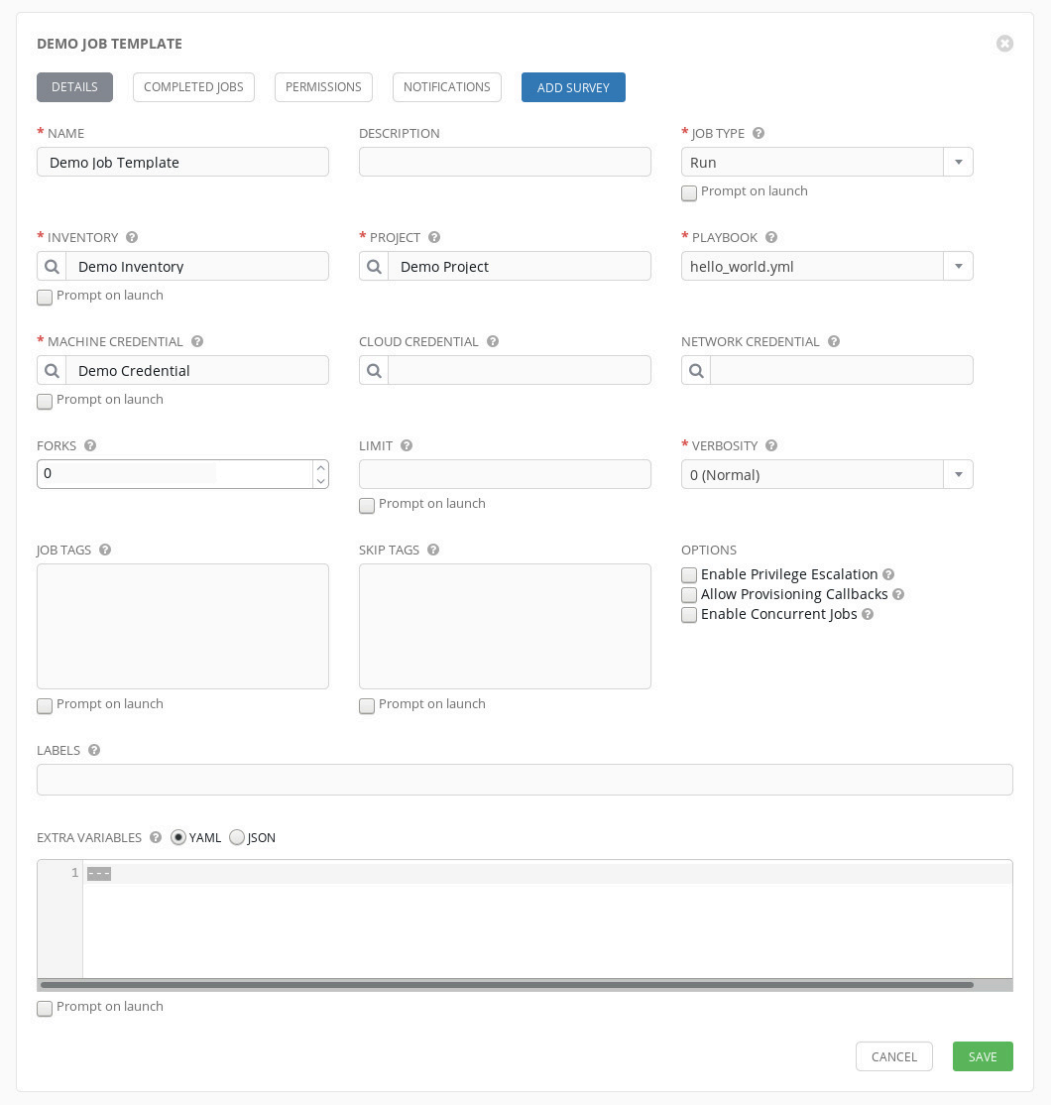

Figure 1.9: Demo Job Template

Launching a Job

Once a Job Template is configured, it can be used to launch Jobs repeatedly using the same parameters. A Job Template is somewhat like a canned **`ansible-playbook`** command complete with options and arguments that's been written down or is in a shell history. When the Job Template is used to launch a Job, it's like running that **`ansible-playbook`** command from a shell prompt.

The following discussion looks at what happens when the **`Demo Job Template`** is used to launch a Job. The exercise following this section also covers this in a more detailed hands-on manner.

1. Under **TEMPLATES**, the **Demo Job Template** should be listed. Across from the name of the Job Template is a rocket icon ("Start a job using this template"). Clicking that icon launches the job using the settings in the Job Template.

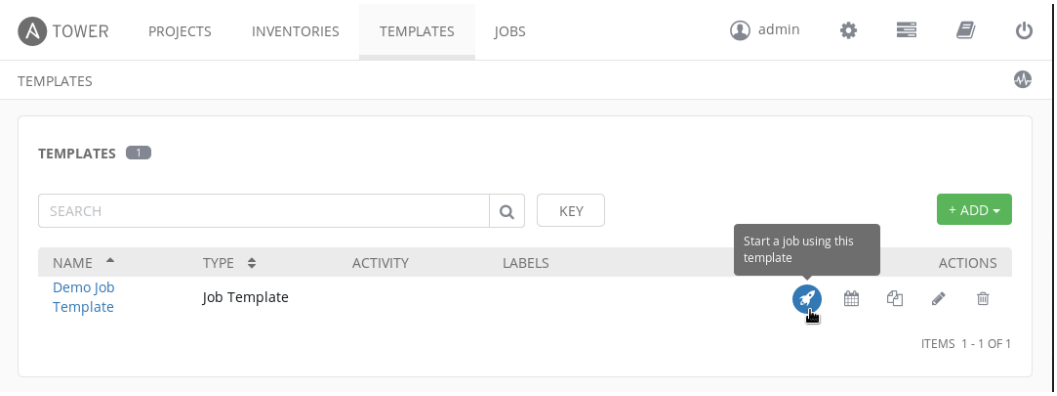

Figure 1.10: Launching a Job

2. As the Job runs, a new status page opens that provides real-time information about the progress of the Job. The **DETAILS** pane provides basic information about the job being run: when it was launched, who launched it, what Job Template, Project, and Inventory were used, and so on. While running, the job status is **Pending**.

3. As the Job executes, the status page also includes the output from the job run in a job output pane. The job run output resembles the output generated when the **`ansible-playbook`** command is executed on an Ansible playbook. This can be used for troubleshooting purposes. In the example screenshot that follows, the play and tasks in the playbook ran successfully.

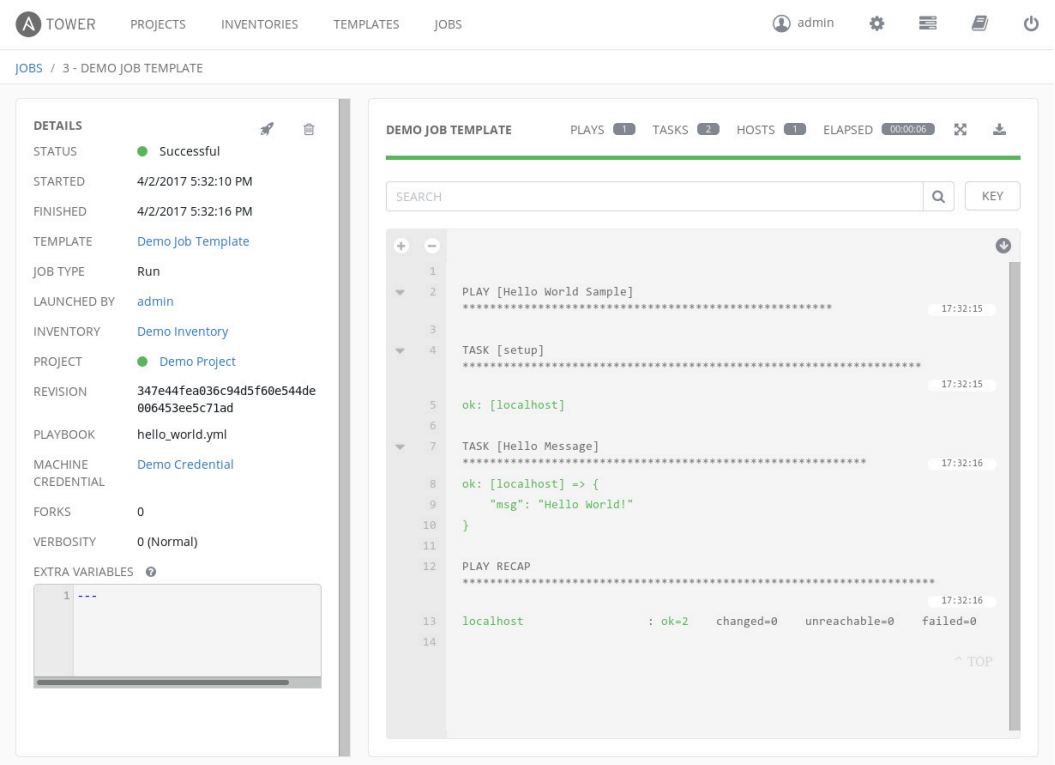

Figure 1.11: Example Job output

4. After the Job completes, the Tower Dashboard (reachable by clicking on the **TOWER** link at the upper left corner of the web interface) has a link to the status page for the Job run under **RECENT JOB RUNS**. The other statistics on the Tower Dashboard are also updated.

5. Under **JOBS**, all the Jobs that have run on the Tower are listed. Clicking on the name of the Job listed on this page also displays the status page logged for that Job.

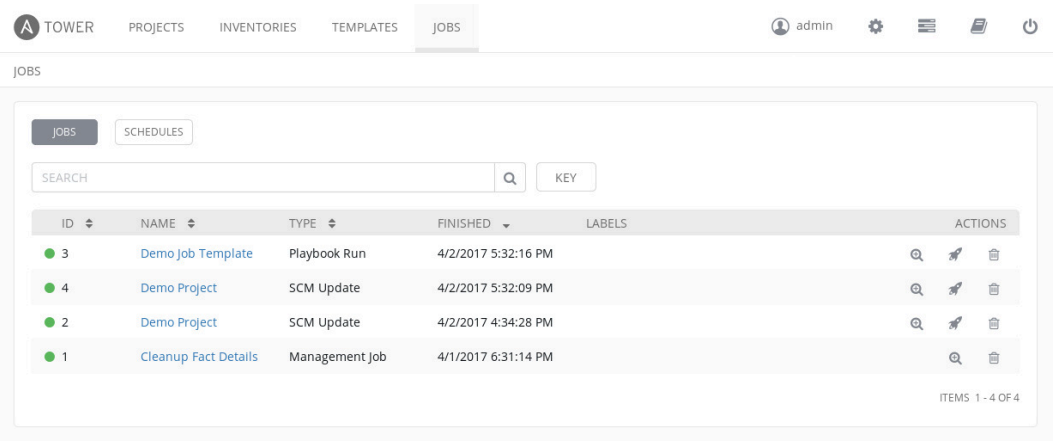

Figure 1.12: Example JOBS screen

References

Ansible Tower Quick Setup Guide for Ansible Tower 3.1.1

http://docs.ansible.com/ansible-tower/3.1.1/html/quickstart/

Ansible Tower User Guide for Ansible Tower 3.1.1

http://docs.ansible.com/ansible-tower/3.1.1/html/userguide/

Guided Exercise: Orientation to the Ansible Tower Web Interface

In this exercise, you will navigate through the Ansible Tower web interface and launch a job.

Outcomes
You should be able to browse through and interact with the Project, Inventory, Credential, and Template screens in the Ansible Tower web interface to successfully launch a job.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab tower-webui setup**. This setup script configures the **tower** virtual machine for the exercise.

```
[student@workstation ~]$ lab tower-webui setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Identify the Project that was created during the Ansible Tower installation and determine its source.

   2.1. Click **PROJECTS** in the quick navigation menu to display the list of Projects. You should see a Project named **Demo Project**, which was created during the Ansible Tower installation.

   2.2. Click the **Demo Project** link to view the details of the Project.

   2.3. Look at the values of the **SCM TYPE**, **PROJECT BASE PATH**, and **PLAYBOOK DIRECTORY** fields to determine the origin of the Project. You should see that **Demo Project** was obtained from the directory, **_4__demo_project**, which resides under the **/var/lib/awx/projects** local directory on the Tower system.

3. Browse the Inventory that was created during the Ansible Tower installation and determine its managed hosts.

   3.1. Click **INVENTORIES** in the quick navigation menu to display the list of known Inventories. You should see an Inventory named **Demo Inventory**, which was created during the Ansible Tower installation.

   3.2. Click the **Demo Inventory** link to view the details of the Inventory. Under the **HOSTS** section, you should see that the Inventory is composed of just one host, **localhost**.

4. View the details of the Credential that was created during the Ansible Tower installation.

   4.1. Click the **Settings** icon in the administration toolbar to display the list of administrative interfaces.

   4.2. Click **CREDENTIALS** to view the list of Credentials.

4.3. Click the **Demo Credential** link to view the details of the Credential. You should see that the Credential is a machine credential that uses the username **admin**.

5. Identify the Job Template that was created during the Ansible Tower installation. Click **TEMPLATES** in the quick navigation menu to display the list of existing Job Templates. You should see a Job Template named **Demo Job Template**, which was created during the Ansible Tower installation.

6. Determine the Inventory, Project, and Credential utilized by the Job Template, **Demo Job Template**.

   6.1. Click the **Demo Job Template** link to view the details of the Job Template.

   6.2. Look at the **INVENTORY** field. You should see that the Job Template uses the **Demo Inventory** Inventory.

   6.3. Look at the **PROJECT** field. You should see that the Job Template uses the **Demo Project** Project.

   6.4. Look at the **PLAYBOOK** field. You should see that the Job Template executes a **hello_world.yml** playbook.

   6.5. Look at the **MACHINE CREDENTIAL** field. You should see that the Job Template uses the **Demo Credential** Credential.

7. Launch a job using the Job Template, **Demo Job Template**.

   7.1. Exit the template details view by clicking the **TEMPLATES** link in the breadcrumb navigation menu near the top of the screen.

   7.2. On the **TEMPLATES** screen, click the rocket icon under the **ACTIONS** column of the **Demo Job Template** row. This launches a job using the parameters configured in the **Demo Job Template** template and redirects you to the job details screen. As the job executes, the details of the job execution as well as its output is displayed.

8. Determine the outcome of the job execution.

   8.1. When the job has completed successfully, the **STATUS** value changes to **Successful**.

   8.2. Review the output of the job execution to determine which tasks were executed. You should see that the **msg** module was used to successfully display a "**Hello World!**" message

9. Review the changes to the dashboard reflecting the job execution.

   9.1. Click **TOWER** in the upper-left corner of the screen to return to the dashboard.

   9.2. Review the **JOB STATUS** graph. The green line on the graph indicates the number of recent successful job executions.

   9.3. Review the **RECENT JOB RUNS** section. This section provides a list of the jobs recently executed as well as their results. The **Demo Job Template** entry indicates that the

Job Template was used to execute a job. The green dot at the beginning of the entry indicates the successful completion of the executed job.

Quiz: Installing Ansible Tower and Describing Ansible Tower's Architecture

Choose the correct answer(s) to the following questions:

1. Which three of the following statements are a component of Ansible Tower 3.1 architecture? (Choose three).

   a. Django web application
   b. MongoDB database
   c. PostgreSQL database
   d. RabbitMQ messaging system

2. Which three of the following statistics are reported by the Tower Dashboard? (Choose three).

   a. Number of jobs executed
   b. Number of credentials stored
   c. Status of executed jobs
   d. Number of hosts managed

3. Which of the following is not a requirement for Tower 3.1 installation?

   a. An existing installation of Ansible
   b. SELinux "targeted" policy
   c. Minimum of 2GB of RAM
   d. Minimum of 20GB of hard disk space

4. Which two of the following statements regarding the bundled Tower installer are incorrect? (Choose two).

   a. Includes additional software dependencies needed by Tower.
   b. Requires access to third party repositories in order to meet software dependencies.
   c. Requires access to Red Hat Enterprise Linux 7 Extras repository for software dependencies.
   d. Supports installations on systems running Ubuntu

5. Which of the following Tower resources cannot be accessed through the quick navigation links?

   a. Projects
   b. Inventories
   c. Notifications
   d. Templates

Solution

Choose the correct answer(s) to the following questions:

1. Which three of the following statements are a component of Ansible Tower 3.1 architecture? (Choose three).

   a. Django web application
   b. MongoDB database
   c. PostgreSQL database
   d. RabbitMQ messaging system

2. Which three of the following statistics are reported by the Tower Dashboard? (Choose three).

   a. Number of jobs executed
   b. Number of credentials stored
   c. Status of executed jobs
   d. Number of hosts managed

3. Which of the following is not a requirement for Tower 3.1 installation?

   a. An existing installation of Ansible
   b. SELinux "targeted" policy
   c. Minimum of 2GB of RAM
   d. Minimum of 20GB of hard disk space

4. Which two of the following statements regarding the bundled Tower installer are incorrect? (Choose two).

   a. Includes additional software dependencies needed by Tower.
   b. Requires access to third party repositories in order to meet software dependencies.
   c. Requires access to Red Hat Enterprise Linux 7 Extras repository for software dependencies.
   d. Supports installations on systems running Ubuntu

5. Which of the following Tower resources cannot be accessed through the quick navigation links?

   a. Projects
   b. Inventories
   c. Notifications
   d. Templates

Summary

In this chapter, you learned:

- Ansible Tower 3.1 is a centralized management solution for Ansible projects.
- Ansible Tower is offered with two installer options. The standard installer downloads packages from network repositories. The bundled installer includes third party software dependencies.
- Job Templates are prepared commands to execute Ansible playbooks. Important components of a job template include an inventory, machine credential, Ansible project and playbook.
- A Job is launched from a Job Template, and represents a single run of a playbook on an inventory of machines.
- Tower Dashboard provides summaries on the status of hosts, inventories, projects, and executed jobs.
- Quick navigation links at the top of the Tower web interface provide access to commonly used Tower resources.
- The **Settings** menu in the Tower web interface provides access to the management interfaces of Tower resources which are not accessible through the quick navigation links.

CHAPTER 2

CREATING USERS AND TEAMS FOR ROLE-BASED ACCESS CONTROL

Overview	
Goal	Create User accounts and organize them into Teams, used in conjunction with Role-Based Access Control to manage administration and access to Organization resources in Ansible Tower
Objectives	• Create new Users in the web user interface and explain the difference between System Administrator, Auditor, and Normal User types of user • Create new Teams in the web user interface, assign existing Users to them, and explain the differences between the Admin, Member, and Read roles which can be assigned to Users in a Team
Sections	• Creating and Managing Ansible Tower Users (and Guided Exercise) • Managing Users Efficiently with Teams (and Guided Exercise)
Lab	• Creating Users and Teams for Role-Based Access Control

Creating and Managing Ansible Tower Users

Objectives

After completing this section, students should be able to:

- Create new Users in the web user interface
- Explain the difference between System Administrator, System Auditor, and Normal User types of user

Role-Based Access Controls

Different people using an Ansible Tower installation need to have different levels of access. Some may simply need to run existing job templates against a preconfigured inventory of machines. Others need to be able to modify particular inventories, job templates, and playbooks. Still others may need access to change anything in the Tower installation.

The Tower interface has a built-in administrative user, **`admin`**, that has superuser access to the entire Tower configuration. In order to make it easier to manage individual access to Inventories, Credentials, Projects, and Job Templates, User accounts can be set up for each person using the Tower installation.

The permissions Users have to make changes to Tower objects are managed through Role-Based Access Controls (RBAC). Users are assigned *roles* that grant permissions which define precisely who can see, change, or delete an object in Tower. Roles can also be managed collectively by granting them to a Team, which is a collection of Users. All Users in the Team inherit the Team's roles.

Roles determine whether Users and Teams can see, use, change, or delete objects such as Inventories and Projects.

Organizations

An Ansible Tower *Organization* is a logical collection of Teams, Projects, and Inventories. Since all Users must be assigned to an Organization, a brief discussion of Organizations is necessary before looking at User configuration.

One of the benefits of implementing Tower is the ability to share Ansible roles and playbooks across departmental or functional boundaries within an enterprise. For example, an operations group of an organization may already have Ansible roles for provisioning production web, database, and application servers, which the developers group should use to provision servers to prepare their development environment. Tower makes it easier for different users and groups to use existing roles and playbooks.

However, for very large deployments it can be useful to categorize large numbers of Users, Teams, Projects, and Inventories under one umbrella Organization. It may be that certain departments do not deploy to certain inventories of hosts, or run certain playbooks. By using Organizations, a collection of Users and Teams can be configured to work with only the Tower resources that they're expected to use.

As part of the Tower installation, a **`Default`** Organization is created. The following is the procedure for creating additional Organizations using the Tower web interface.

1. Log in to the Tower web interface as the Tower **admin** User.
2. Click on the **Settings** icon and then the **ORGANIZATIONS** link.
3. Click on the **+ADD** button to create a new Organization.
4. In the provided fields, enter a name for the new Organization and, if desired, an optional description.

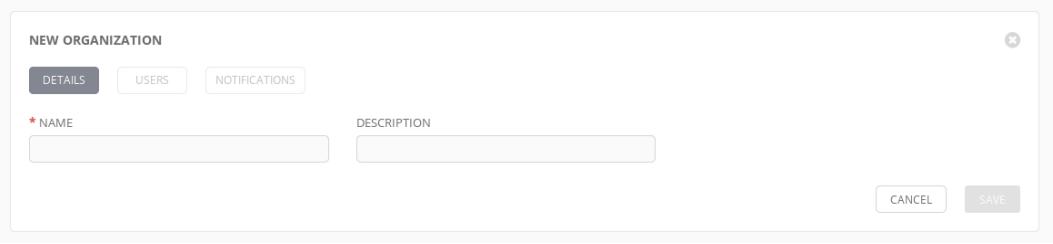

Figure 2.1: New Organization

5. Click **SAVE** to finalize the creation of the new Organization.

Important

Users of Tower with a Self-Support subscription only have the **Default** Organization available and are not able to add additional Organizations. Users of Self-Support subscriptions should not delete the default Organization and try to add a new Organization, or their Tower configuration will break.

For Self-Support deployments of Tower, all objects that need to be assigned to an Organization should be assigned to **Default**.

User Types

By default, an **admin** User account is created by the Tower installer which has full control of the Tower installation. Using the special **admin** account, a Tower administrator can log into the Tower web interface and create additional Users within Tower.

Tower Users can be created as one of three *user types*.

System Administrator
The **System Administrator** user type (also known as *Superuser*) provides a User unrestricted access to perform any action within the entire Tower installation. **System Administrator** is a special *singleton role* which has read-write permission on all objects in all Organizations on the Tower.

The **admin** User created by the installer also has the System Administrator singleton role and should therefore only be used by the Tower administrator.

System Auditor
The **System Auditor** user type also has a special singleton role, which has read-only access to the entire Tower installation.

Normal User

`Normal User` is the standard user type. It has no special roles assigned initially, and starts with extremely limited access. It isn't assigned any singleton roles, and is only assigned roles associated with the Organization of which the User is a member.

Creating Users

A User logged in as the Tower **`admin`** User can create new Tower Users by executing the following procedure.

1. Click on the **Settings** icon and then the **USERS** link.
2. Click on the **+ADD** button to create a new User.
3. Enter the first name, last name, and email address for the new User into the **FIRST NAME**, **LAST NAME**, and **EMAIL** fields, respectively.
4. In the **USERNAME** field, specify a unique username for the new User.
5. Click the magnifying glass icon next to the **ORGANIZATION** field to display a list of Organizations within Tower. Select an Organization from the list and click **SAVE**.
6. Enter the desired password for the new User into the **PASSWORD** and **CONFIRM PASSWORD** fields.
7. Select a user type for the new User.
8. Click **SAVE** to finalize the creation of the new User.

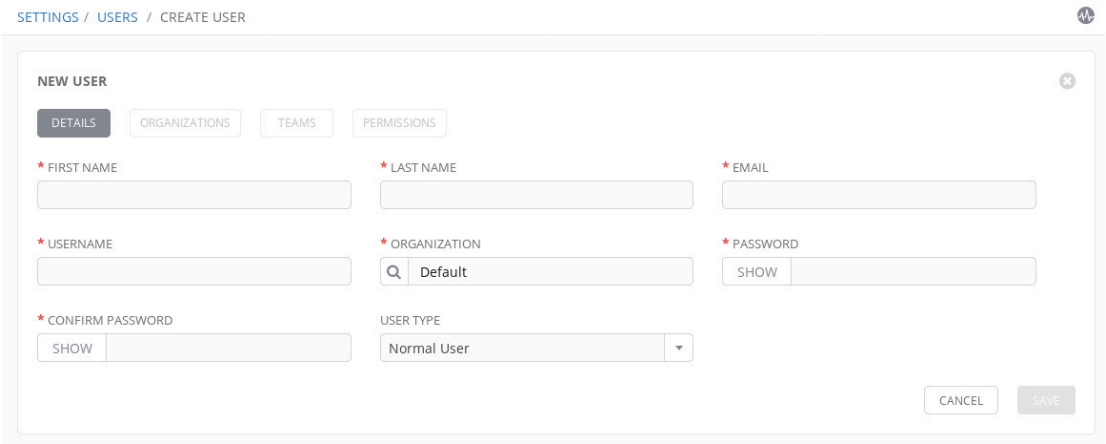

Figure 2.2: New User

Editing Users

After a User is created, its properties can be edited using the **Edit User** screen by executing the following procedure.

1. Click the **Settings** icon and then the **USERS** link.
2. Click on the link for the User to edit.
3. Make the changes to the desired fields.

4. Click **SAVE** to finalize the modification of the new User.

Organization Roles

Newly created Users inherit specific roles from their Organization, based on their user type. Additional roles can be assigned to a User after it is created in order to grant permissions to view, use, or change additional Tower objects. An Organization is itself one of these objects. There are four roles that users can be assigned on an Organization:

Admin
When assigned the **`Admin`** role on an Organization, a User gains the ability to manage all aspects of that Organization, including reading and changing the Organization, and adding and removing Users and Teams from the Organization.

Auditor
When assigned the **`Auditor`** role on an Organization, a User gains read-only access to the Organization.

Member
When assigned the **`Member`** role on an Organization, a User gains read permission to the Organization. The Organization **`Member`** role only provides a User the ability to view the list of Users who are members of the Organization and their assigned Organization roles.

Unlike the Organization **`Admin`** and **`Auditor`** roles, the **`Member`** role does not provide Users permissions to any of the resources the Organization contains, such as Teams, Credentials, Projects, Inventories, Job Templates, Work Templates, and Notifications.

Read
When assigned the **`Read`** role on an Organization, a User gains read permission to the Organization only. The Organization **`Read`** role only provides a User the ability to view the list of Users who are members of the Organization and their assigned Organization roles.

Unlike the Organization **`Admin`** and **`Auditor`** roles, the **`Read`** role does not inherit roles on any of the resources the Organization contains, such as Teams, Credentials, Projects, Inventories, Job Templates, Work Templates, and Notifications.

The Organization object cannot be assigned roles on Tower resources. Therefore, a User that has the **`Member`** role on an Organization only has access to the Organization object and inherits no other permissions as a result of this role. Consequently, a User that has the **`Member`** role on an Organization is the equivalent of a User that has the **`Read`** role on an Organization.

Important

Users with the **System Administrator** singleton role inherit the **`Admin`** role on every Organization within Tower.

Users with the **System Auditor** singleton role inherit the **`Auditor`** role on every Organization within Tower.

A User created with the Normal User user type is assigned a **`Member`** role on the Organization they were assigned to at the time of the User's creation in Tower. Other roles can be added later, including additional **`Member`** roles on other Organizations.

Managing User Organization Roles

The **Edit User** screen only allows changes to the Organization that the User is placed in and the User's user type. Since the user types designated to Users determine their inherited Organization roles, modifying a User's Organization and user type can impact their Organization role.

However, full administration of a User's role in an Organization requires the following procedure.

1. Log in to the Tower web interface as the Tower **admin** User or a User with the **Admin** role on the Organization being modified.
2. Click on the **Settings** icon and then the **ORGANIZATIONS** link.
3. Click on the **USERS** link under the organization being managed.
4. A list of Users who have been granted roles to the Organization is listed.
5. If a User does not appear on the list and needs to be granted a role in the Organization, or if a User exists and needs additional Organization roles added, use the following procedure.

   5.1. Click **+ADD**.

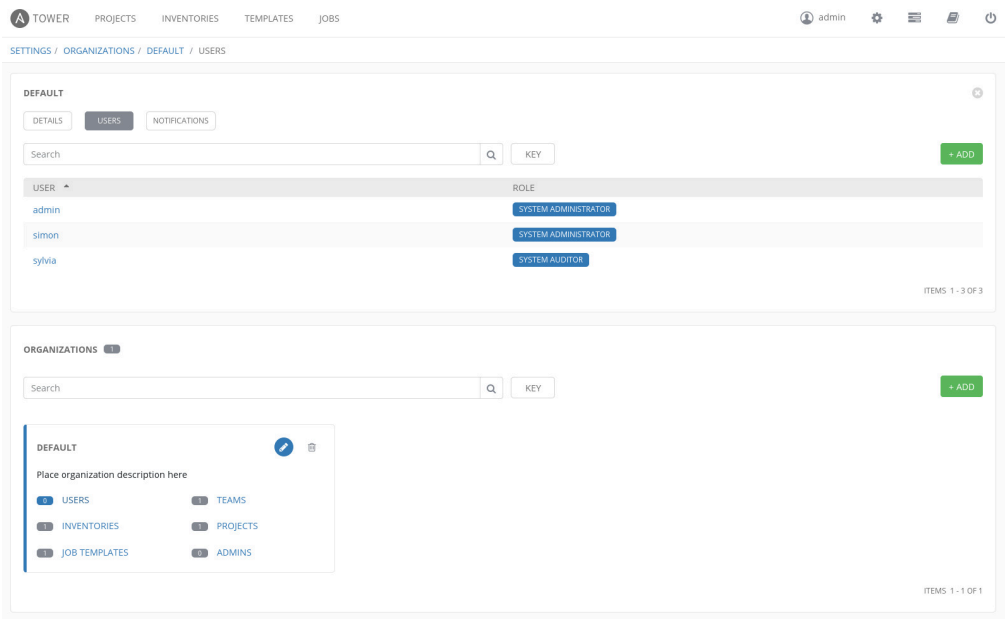

Figure 2.3: Assigning Organization Roles to a User

   5.2. In the **ADD USERS** screen, check the box next to the desired User.

   5.3. Click the **SELECT ROLES** drop-down and select the desired Organization role for the User. This step can be repeated multiple times to add multiple roles for a User.

Note

For a list of each role's definition, click the **KEY** button.

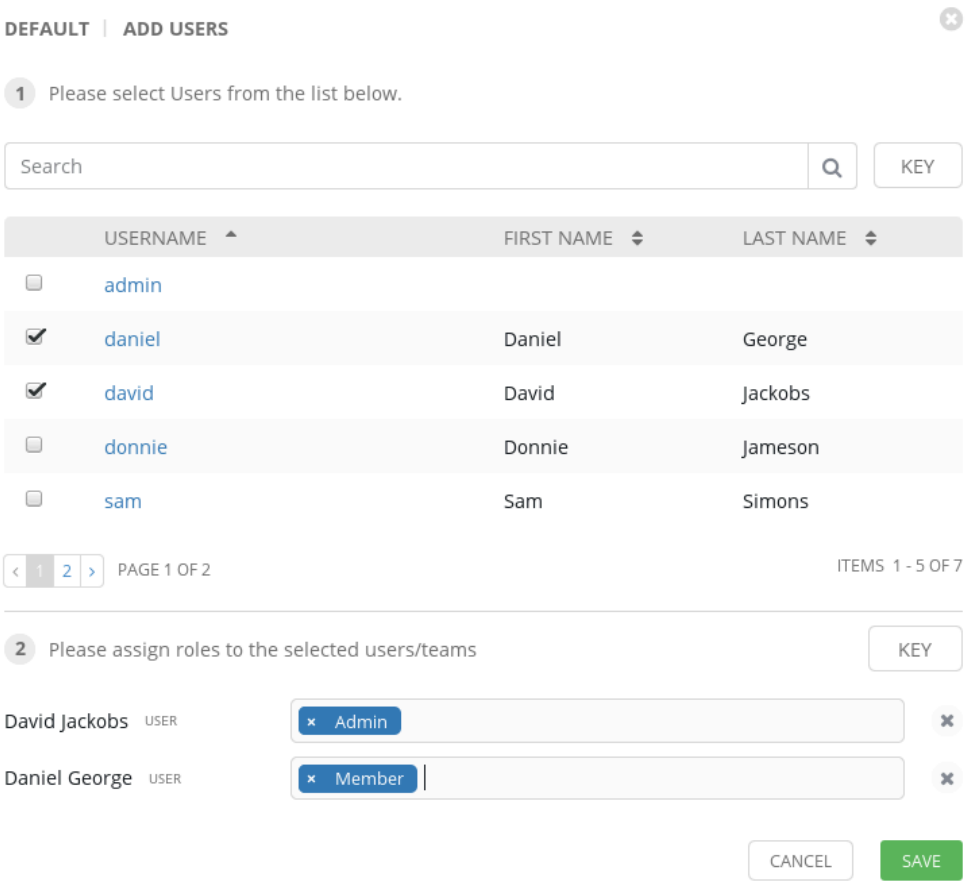

Figure 2.4: Setting Roles on Organization "Add Users" Screen

5.4. Click **SAVE** to assign roles to the User for the Organization.

6. To remove existing roles from a User, click the **X** preceding to the role to be deleted.

References

Ansible Tower User Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Creating and Managing Ansible Tower Users

Outcomes
You should be able to create User accounts and understand the difference between their types.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Steps

1. Log in to the Ansible Tower web interface running on the **`tower`** system using the **`admin`** account and the **`redhat`** password.

2. Create a User, **`sam`**, as a Normal User.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   2.2. Click **USERS** to manage Users.

   2.3. Click **+ADD** to add a new User.

   2.4. On the next screen, fill in the details as follows:

Field	**Value**
FIRST NAME	Sam
LAST NAME	Simons
EMAIL	sam@lab.example.com
USERNAME	sam
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

   2.5. Click **SAVE** to create the new User.

3. Verify the permissions for the newly created **`sam`** User.

   3.1. Click **PERMISSIONS** to see the User's permissions.

   3.2. As you can see, **`sam`** is simply a Member of the **`Default`** Organization.

   3.3. Click the **Log Out** icon in the top right corner to log out and then log back in as **`sam`** with password of **`redhat123`**.

   3.4. Open the **SETTINGS** page by clicking the gear icon in the top right.

3.5. You can see the User's access is limited to the **ORGANIZATIONS**, **USERS**, **TEAMS**, **CREDENTIALS**, **INVENTORY SCRIPTS**, **VIEW YOUR LICENSE**, and **ABOUT TOWER** interfaces.

3.6. Click **USERS** to manage Users. As you can see, it is not possible for **sam** to add new Users.

3.7. Click the **Log Out** icon to log out of the Tower web interface.

4. Create a User, **sylvia**, as a System Auditor.

4.1. Log in to the Tower web interface as the **admin** User with the **redhat** password.

4.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

4.3. Click **USERS** to manage Users.

4.4. Click **+ADD** to add a new User.

4.5. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Sylvia
LAST NAME	Simons
EMAIL	sylvia@lab.example.com
USERNAME	sylvia
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	System Auditor

4.6. Click **SAVE** to create the new User.

5. Verify the permissions for the newly created **sylvia** User.

5.1. Click **PERMISSIONS** to see the User's permissions.

5.2. As you can see, **sylvia** is a Member of the **Default** Organization and has the role of a System Auditor.

5.3. Log out and log back in as **sylvia** with password of **redhat123**.

5.4. Open the **SETTINGS** page by clicking the gear icon in the top right.

5.5. You can see the User has access to all interfaces on the **SETTINGS** page.

5.6. Click **USERS** to manage Users. As you can see, it is not possible for **sylvia** to add new Users.

5.7. Click the **Log Out** icon to log out of the Tower web interface.

6. Create a User, **simon**, as a System Administrator.

   6.1. Log in to the Tower web interface as the **admin** User with the **redhat** password.

   6.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

   6.3. Click **USERS** to manage Users.

   6.4. Click **+ADD** to add a new User.

   6.5. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Simon
LAST NAME	Stephens
EMAIL	simon@lab.example.com
USERNAME	simon
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	System Administrator

   6.6. Click **SAVE** to create the new User.

7. Verify the permissions for the newly created **simon** User.

   7.1. Click **PERMISSIONS** to see the User's permissions. This time the permissions explicitly say "*SYSTEM ADMINISTRATORS HAVE ACCESS TO ALL PERMISSIONS*".

   7.2. Log out and log back in as **simon** with password of **redhat123**.

   7.3. Open the **SETTINGS** page by clicking the gear icon in the top right.

   7.4. You can see that **simon** has access to all interfaces on the **SETTINGS** page.

   7.5. Click **USERS** to manage Users. As you can see, **simon** has the rights to create new Users.

   7.6. Click the **Log Out** icon to log out of the Tower web interface.

Managing Users Efficiently with Teams

Objectives

After completing this section, students should be able to:

- Create new Teams in the web user interface and assign existing Users to them
- Explain the differences between the Admin, Member, and Read roles, which can be assigned to Users in a Team

Teams

Teams are groups of Users. Teams make managing roles on Tower objects such as Inventories, Projects, and Job Templates, more efficient than managing them for each user separately. Users, who are members of a Team, inherit the roles assigned to that Team.

Rather than assigning the same roles to multiple Users, a Tower administrator can assign roles to the Team representing a group of Users.

In Ansible Tower, Users exist as objects at a Tower level. A User can have roles in multiple Organizations. A Team belongs to exactly one Organization, but a Tower **`System Administrator`** can assign the Team roles on resources belonging to other Organizations.

Important

A Team belongs to a particular Organization, but it can't be assigned roles on an Organization object, like individual Users can. To manage an Organization object, roles must be assigned to specific Users directly.

Teams can be assigned roles on resources which belong to a particular Organization, such as Projects, Inventories, or Job Templates.

Creating Teams

Teams are created within each Organization using the following procedure.

1. Log in to the Tower web interface as the Tower **`admin`** User or as a User assigned the **`admin`** role for the Organization in which the new Team is to be created.
2. Click the **Settings** icon and then the **TEAMS** link.
3. Click the **+ADD** button.
4. On the **New Team** screen, enter a name for the new Team in the **NAME** field.
5. If desired, enter a description in the **DESCRIPTION** field.
6. For the required **ORGANIZATION** field, click the magnifying glass icon to get a list of the Organizations within Tower. In the **SELECT ORGANIZATION** screen, select the Organization to create the Team in, and then click **SAVE**.
7. Click **SAVE** to finalize the creation of the new Team.

Team Roles

Once a Team has been created, Users can be added to the Team. A User's relationship to a Team is defined by which role the User is assigned for the Team.

A User can be assigned one or more of the following Team roles:

admin
The Team **admin** role grants Users full control of the Team. Users with this Team role can manage the Team's Users and their associated Team roles. Users with Team **admin** roles can also manage the Team's roles on resources for which the Team has been assigned **admin** role.

member
The Team **member** role allows Users to inherit roles on Tower resources that have been granted to the Team. It also grants Users the ability to see the Team's Users and associated Team roles.

read
The Team **read** role gives Users the ability to see the Team's Users and their associated Team roles. A User assigned a Team **read** role does not inherit roles that the Team has been granted on Tower resources.

Important

Users with Team **admin** roles can only manage the Team's roles on a resource when the resource also grants the Team **admin** role on itself. Just because a Team **admin** can manage Team membership, it does not imply that the Team **admin** has any rights to manage roles on objects to which the team has access.

For example, for a User to grant a Team the **use** role for a Project, the User needs to have the **admin** role for both the Team and the Project.

Adding Users to a Team

Once a Team has been created, Users can be added to the Team. To add Users to a Team in an Organization, execute the following procedure:

1. Log in to the Tower web interface as the Tower **admin** User or a User assigned the **admin** role for the Organization to which the team belongs.
2. Click the **Settings** icon and then the **ORGANIZATIONS** link.
3. Click the **TEAMS** link under the Organization which the Team belongs to.
4. On the **Teams** screen, click the name of the Team to add the User to.
5. On the Team details screen, click the **USERS** button.
6. Click the **+ADD** button.

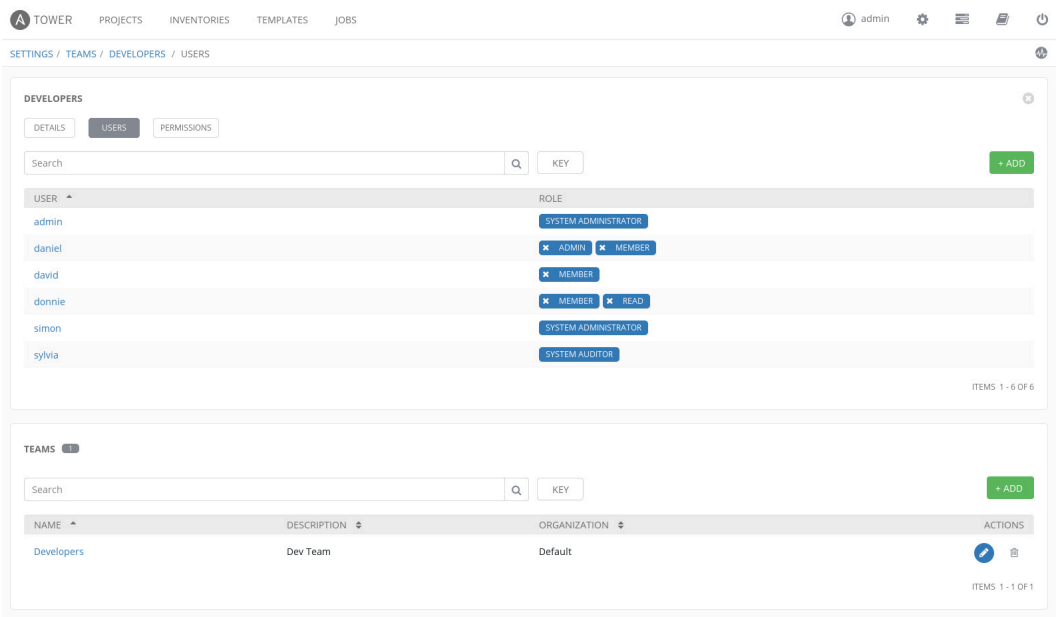

Figure 2.5: Assigning Team Roles to a User

7. On the **ADD USERS** screen, select one or more Users to add to Team. Then, click the **SELECT ROLES** drop-down menu next to each User and select the Team role to assign to the User.

Note

For a list of each role's definition, click the **Key** button.

8. Click **SAVE** to finalize the addition of the User to the Team.

Important

Any User can be added to a Team. A User does not need to have a role with an Organization in order to be assigned a role on a Team within that Organization.

References

Further information is available in the Teams section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Managing Users Efficiently with Teams

Outcomes
You should be able to organize Users into Teams and understand the different User Roles.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Perform the following steps using the **`Default`** Organization.

Steps
1. Create a new Team called **`Developers`**.

   1.1. Log in to the Tower web interface as the **`admin`** User with the **`redhat`** password.

   1.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

   1.3. Click **TEAMS** to manage Teams.

   1.4. Click **+ADD** to add a new Team.

   1.5. On the next screen, fill in the details as follows:

Field	Value
NAME	Developers
DESCRIPTION	Dev Team
ORGANIZATION	Default

   1.6. Click **SAVE** to create the new Team.

2. Create a User which will be the administrator for the newly created **`Developers`** Team.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   2.2. Click **USERS** to manage Users.

   2.3. Click **+ADD** to add a new User.

   2.4. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Daniel
LAST NAME	George
EMAIL	daniel@lab.example.com
USERNAME	daniel
ORGANIZATION	Default

PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

Note

We are creating an administrator of the **Developers** Team, not a System Administrator. This is why the User Type is set here to **Normal User**.

2.5. Click **SAVE** to create the new User.

3. Assign the **Admin** role on the **Developers** Team to the **daniel** User.

   3.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   3.2. Click **TEAMS** to manage Teams.

   3.3. Click the link for the **Developers** Team created previously.

   3.4. Click **USERS** to manage the Users assigned to the Team.

   3.5. Click **+ADD** to add a new User to the Team.

   3.6. In the first section of the screen, check the box next to **daniel** to select this User. This adds **Daniel George** to the list of selected Users in the second section below.

   3.7. In the second section, click on the empty field next to the User and assign the **Admin** role by selecting it from the drop-down list.

   3.8. Click **SAVE**.

4. Create a User, **donnie**, to be granted **Read** role to the **Developers** Team.

   4.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   4.2. Click **USERS** to manage Users.

   4.3. Click **+ADD** to add a new User.

   4.4. On the next screen, fill in the details as follows:

Field	**Value**
FIRST NAME	Donnie
LAST NAME	Jameson
EMAIL	donnie@lab.example.com
USERNAME	donnie
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123

USER TYPE	Normal User

4.5. Click **SAVE** to create the new User.

5. Assign the **donnie** User the **Read** role for the **Developers** Team.

5.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

5.2. Click **TEAMS** to manage Teams.

5.3. Click the link for the **Developers** Team you created previously.

5.4. Click **USERS** to manage the Team's Users.

5.5. Click **+ADD** to add a new User to the Team.

5.6. In the first section of the screen, check the box next to **donnie** to select this User. This adds **Donnie Jameson** to the list of selected Users in the second section below.

5.7. In the second section, click on the empty field next to the User and assign the **Read** role by selecting it from the drop-down list.

5.8. Click **SAVE** and then click **Log Out** icon in the upper right corner to exit the Tower web interface.

6. Create a User, **david**, to be granted **Member** role to the **Developers** Team.

6.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

6.2. Click **USERS** to manage Users.

6.3. Click **+ADD** to add a new User.

6.4. On the next screen, fill in the details as follows:

Field	**Value**
FIRST NAME	David
LAST NAME	Jackobs
EMAIL	david@lab.example.com
USERNAME	david
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

6.5. Click **SAVE** to create the new User.

7. Assign the **Member** role on the **Developers** Team to the **david** User.

7.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

7.2. Click **TEAMS** to manage Teams.

7.3. Click the link for the **Developers** Team you created previously.

7.4. Click **USERS** to manage the Team's Users.

7.5. Click **+ADD** to add a new User to the Team.

7.6. In the first section on the screen, check the box next to **david** to select this User.

7.7. This adds **David Jackobs** to the list of selected Users in the second section below.

7.8. In the second section, click on the empty field next to the User and assign the **Member** role by selecting it from the drop-down list.

7.9. Click **SAVE** and then click the **Log Out** icon to exit the Tower web interface.

8. Verify the permissions for the **daniel** User. Log in as the **daniel** User and assign the **sam** User **Member** role to the **Developers** Team.

8.1. Log in to the Tower web interface as **daniel** with a password of **redhat123**.

8.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

8.3. Click **TEAMS** to manage Teams.

8.4. Click the link for the **Developers** Team.

8.5. Click **USERS** to manage the Users assigned to the Team. As you can see, **daniel** can add and manage role assignments for his Team.

8.6. Click **+ADD**.

8.7. In the first section of the screen, check the box next to **sam** to select the User. This adds **Sam Simons** to the list of selected Users in the second section below.

8.8. In the second section, click on the empty field next to the User and assign it the **Member** role by selecting it from the drop-down list.

8.9. Click **SAVE** and then click the **Log Out** icon to exit the Tower web interface.

9. Verify the permissions for the **donnie** User.

9.1. Log in to the Tower web interface as **donnie** with a password of **redhat123**.

9.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

9.3. Click **TEAMS** to manage Teams.

9.4. Click the link for the **Developers** Team you created previously. The User, **donnie**, can view the settings for the Team that she has been assigned the **Read** role to.

9.5. Click **USERS** and you should see that **donnie** sees all Users that are part of the **Developers** Team (including the System Auditor and System Administrators) but has no rights to manage Users or User roles in that Team.

9.6. Click the **Log Out** icon to exit the Tower web interface.

10. Verify the permissions for the **david** User.

    10.1. Log in to the Tower web interface as **david** with a password of **redhat123**.

    10.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

    10.3. Click **TEAMS** to manage Teams.

    10.4. Click the link for the **Developers** Team you created previously.

    10.5. Click **USERS** and you should see that, like **donnie**, **david** sees all Users that are part of the **Developers** Team, including the System Auditor and System Administrators, but has no rights to manage Users or User roles in that Team.

    10.6. Click the **Log Out** icon to exit the Tower web interface.

Lab: Creating Users and Teams for Role-Based Access Control

Outcomes
You should be able to create User accounts and organize them into Teams.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Steps

1. Add two Users as Normal Users to the **`Default`** organization with the following information:

	User 1	**User 2**
FIRST NAME	Oliver	Ophelia
LAST NAME	Stone	Dunham
EMAIL	oliver@lab.example.com	ophelia@lab.example.com
USERNAME	oliver	ophelia
ORGANIZATION	Default	Default
PASSWORD	redhat123	redhat123
CONFIRM PASSWORD	redhat123	redhat123
USER TYPE	Normal User	Normal User

2. Create a Team called **`Operations`** in the **`Default`** Organization.

3. Add **`Oliver`** as a **`Member`** to the **`Operations`** Team. Add **`Ophelia`** as an **`Admin`** to the **`Operations`** Team.

4. Run the command **`lab org-review grade`** on **`workstation`** to grade your exercise.

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

Solution

Outcomes
You should be able to create User accounts and organize them into Teams.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Steps
1. Add two Users as Normal Users to the **`Default`** organization with the following information:

	User 1	**User 2**
FIRST NAME	Oliver	Ophelia
LAST NAME	Stone	Dunham
EMAIL	oliver@lab.example.com	ophelia@lab.example.com
USERNAME	oliver	ophelia
ORGANIZATION	Default	Default
PASSWORD	redhat123	redhat123
CONFIRM PASSWORD	redhat123	redhat123
USER TYPE	Normal User	Normal User

1.1. Log in to the Tower web interface as the **`admin`** User with the **`redhat`** password.

1.2. Click the gear icon in the top right to open the **SETTINGS** page.

1.3. Click **USERS** to manage Users.

1.4. Click **+ADD** to add a new User.

1.5. On the next screen, fill in the details as follows:

Field	**Value**
FIRST NAME	Oliver
LAST NAME	Stone
EMAIL	oliver@lab.example.com
USERNAME	oliver
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

1.6. Click **SAVE** to create the new User.

1.7. Scroll down the next screen and click **+ADD** to add another User.

1.8. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Ophelia
LAST NAME	Dunham
EMAIL	ophelia@lab.example.com
USERNAME	ophelia
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

1.9. Click **SAVE** to create the new User.

2. Create a Team called **`Operations`** in the **`Default`** Organization.

   2.1. Click the gear icon in the top right to open the **SETTINGS** page.

   2.2. Click **TEAMS** to manage Teams.

   2.3. Click **+ADD** to add a new Team.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Operations
DESCRIPTION	Ops Team
ORGANIZATION	Default

   2.5. Click **SAVE** to create the **`Operations`** Team.

3. Add **`Oliver`** as a **`Member`** to the **`Operations`** Team. Add **`Ophelia`** as an **`Admin`** to the **`Operations`** Team.

   3.1. On the **OPERATIONS** Team editor screen, click **USERS** to manage the Team's Users.

   3.2. Click **+ADD** to add a new User to the Team.

   3.3. In the first section of the screen, check the boxes next to **`oliver`** and **`ophelia`** to select those Users.

   3.4. This adds **`Oliver Stone`** and **`Ophelia Dunham`** to the list of selected Users in the second section below.

   3.5. In the second section, assign the **`Member`** role to **`Oliver Stone`** and the **`Admin`** role to **`Ophelia Dunham`**.

   3.6. Click **SAVE**.

Important

You will get a "*Invalid search term entered.*" error message but the assignments are still made. Click **DETAILS** and then **USERS** to display the results.

3.7. You can verify on the next screen that **oliver** has been assigned the role of **Member** and that **ophelia** has been assigned the role of **Admin**.

4. Run the command **lab org-review grade** on **workstation** to grade your exercise.

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

Summary

In this chapter, you learned:

- Organizations are logical collections of Users, Teams, Projects, and Inventories.
- Users can be created as one of three user types: **System Administrator**, **System Auditor**, and **Normal User**.
- **System Administrator** and **System Auditor** are singleton roles that grant read-write and read-only access to all Tower objects, respectively.
- Users can be assigned one of four Organization roles: **`admin`**, **`auditor`**, **`member`**, and **`read`**.
- A Team is a group of Users and can be used to simplify assignment of roles to Users on Tower resources.
- Users can be assigned one of three Team roles: **`admin`**, **`member`**, and **`read`**.
- The role a User has on an object determines what the User can do to that object.

CHAPTER 3

CREATING AND MANAGING INVENTORIES AND CREDENTIALS

Overview	
Goal	Create inventories of machines to manage and set up credentials, allowing Ansible Tower to run jobs on those systems
Objectives	• Create a static inventory and assign users and teams permissions to use the inventory. • Create a machine credential to invoke Ansible on hosts under your management; assign roles to an SSH private key for users or teams.
Sections	• Creating a Static Inventory (and Guided Exercise) • Creating Machine Credentials for Access to Inventory Hosts (and Guided Exercise)
Lab	• Creating and Managing Inventories and Credentials

Creating a Static Inventory

Objectives

After completing this section, students should be able to:

- Create a static inventory and assign users and teams permissions to use the inventory.

Ansible Inventory

Ansible playbooks run against an *inventory* of hosts and groups of hosts. Ansible can select parts or all of the inventory as target systems to manage through task execution. Without Ansible Tower, Ansible normally defines this inventory as a static file or dynamically from external sources through a script. This section of the course will focus on static inventory configuration.

As a review, look at the following Ansible static inventory file. It defines four hosts. The host **`london1.example.com`** is a member of no groups. The hosts **`raleigh1.example.com`** and **`atlanta1.example.com`** are members of the host group **`southeast`**. The host **`boston1.example.com`** is a member of the host group **`northeast`**. The host group **`east`** is a group of groups, including the **`southeast`** and **`northeast`** host groups.

```
london1.example.com

[southeast]
raleigh1.example.com
atlanta1.example.com

[northeast]
boston1.example.com

[east:children]
southeast
northeast
```

Two implicit groups also exist. The group **`all`** includes all the hosts in the inventory. The group **`ungrouped`** includes all hosts that are not in a group other than **`all`**, which in the previous example would be only **`london1.example.com`**.

Inventories are managed as objects in Ansible Tower, and can be configured in a couple of different ways. This section looks at how to set up a static inventory and use RBAC to control access to it by using the Tower web interface.

Creating an Inventory in Ansible Tower

Inventories are managed as objects in Ansible Tower. Each Organization may have many inventories available. When Job Templates are created, they can be configured to use a specific Inventory belonging to the Organization. Which Users on the Tower can use an Inventory object is determined by the roles they have on the Inventory.

This section looks at how to use Tower's web user interface to create a new Inventory object from scratch, using the preceding sample inventory file as a model. Later sections will cover other methods of configuring an inventory.

The following procedure illustrates how the example Ansible inventory previously shown can be implemented as a static Inventory resource named **`Mail Servers`** assigned to the **`Default`** Organization.

1. Log in as a User which is assigned the **`admin`** role for the **`Default`** Organization.

2. Click the **INVENTORIES** quick navigation link at the top of the screen.

3. Create a new Inventory. On the **INVENTORIES** screen, click **+ADD**.

   On the **NEW INVENTORY** screen, the **NAME** of the Inventory and its **ORGANIZATION** are required. For this example, use **`Mail Servers`** for **NAME** and click the magnifying glass icon to select the **`Default`** organization. Click **SAVE**.

4. Add the host **`london1.example.com`** to the Inventory.

   On the **MAIL SERVERS** detail screen, click **+ADD HOST**, enter **`london1.example.com`** for the **HOST NAME**, and then click **SAVE**.

5. Add the **`east`** group to the Inventory.

   On the **MAIL SERVERS** Inventory detail screen, click the **+ADD GROUP** button. On the **CREATE GROUP** screen, enter **`east`** in the **NAME** field. Click the **SAVE** button on the **CREATE GROUP** screen to finalize the addition of the group to the Inventory.

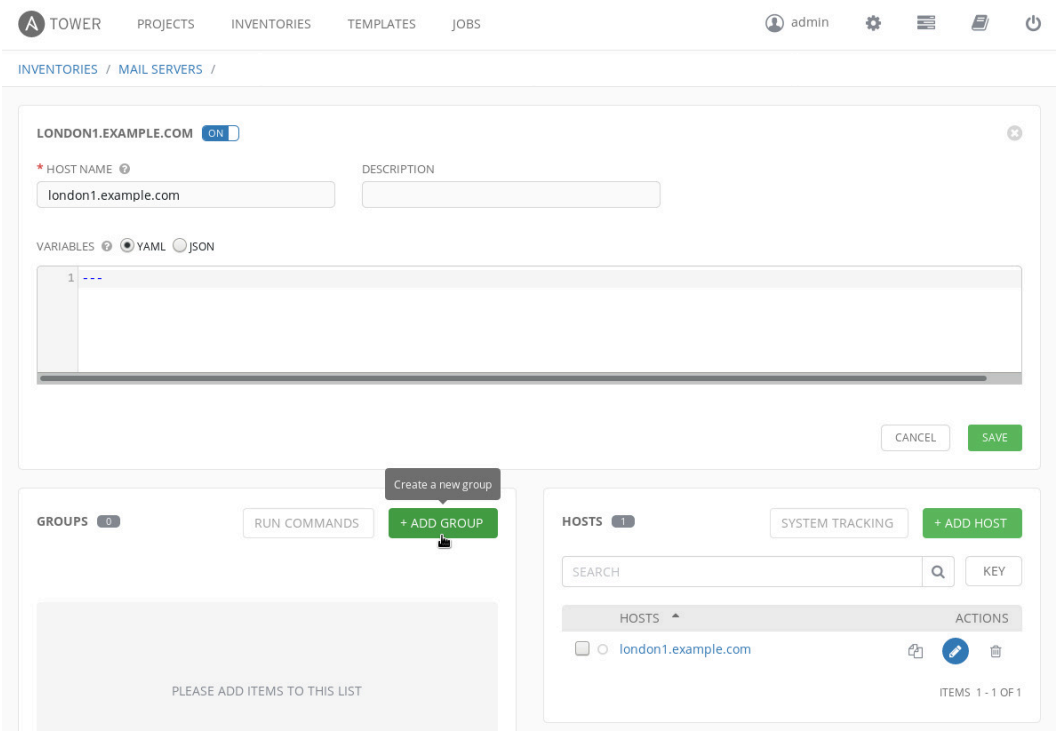

Figure 3.1: New host group

6. Add the **`northeast`** and **`southeast`** groups as child groups of the **`east`** group just added to the Inventory.

On the **MAIL SERVERS** Inventory detail screen, click the **east** group link.

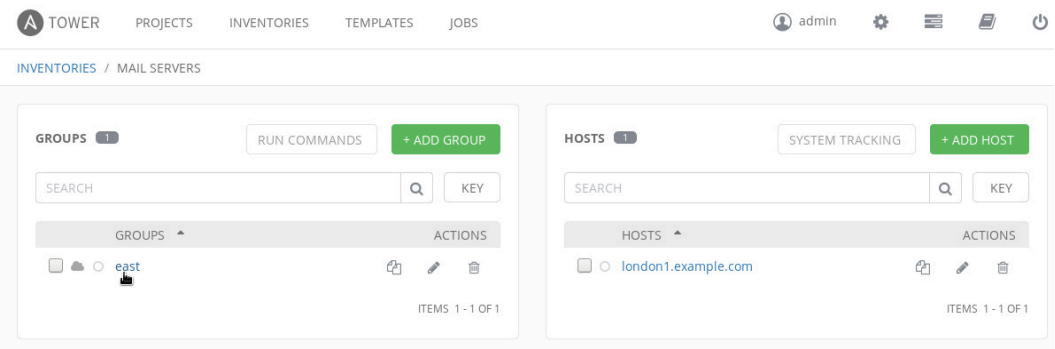

Figure 3.2: Editing a host group

On the **EAST** group details screen, click the **+ADD GROUP** button. On the **CREATE GROUP** screen, enter **`southeast`** in the **NAME** field. Click the **SAVE** button on the **CREATE GROUP** screen to finalize the addition of the child group to the Inventory.

On the **EAST** group details screen, click the **+ADD GROUP** button again and repeat for the **`northeast`** group.

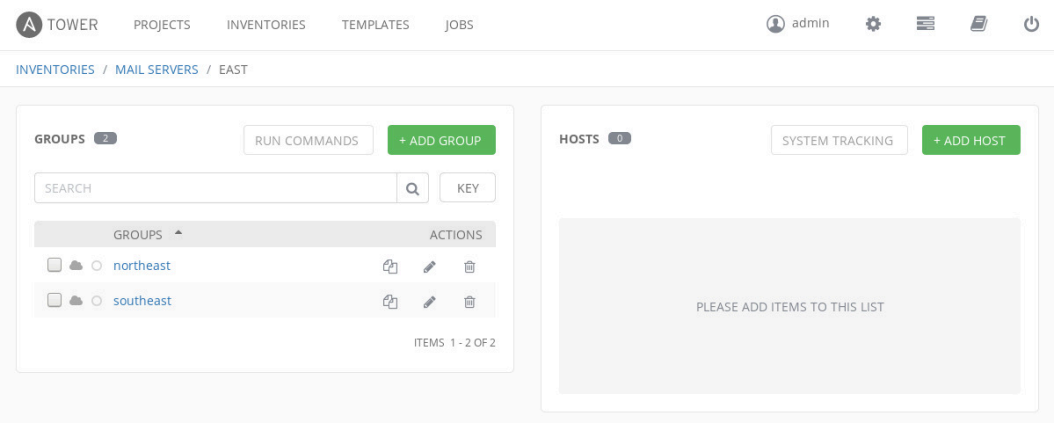

Figure 3.3: New child host groups

7. Add the **`raleigh1.example.com`** and **`atlanta1.example.com`** hosts to the **`southeast`** child group just added to the Inventory.

   On the **EAST** group details screen, click the **southeast** group link.

   On the **SOUTHEAST** group details screen, click the **+ADD HOST** button. On the **CREATE HOST** screen, enter **`raleigh1.example.com`** in the **NAME** field. Click the **SAVE** button on the **CREATE HOST** screen to finalize the addition of the host to the child group.

   On the **SOUTHEAST** group details screen, click the **+ADD HOST** button again, and repeat the process to add the **`atlanta1.example.com`** host.

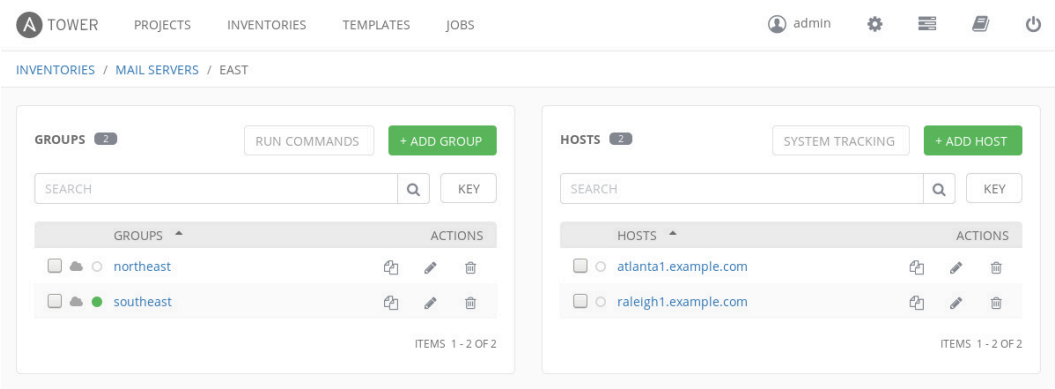

Figure 3.4: Partially configured child groups

8. Add the **`boston1.example.com`** host to the **`northeast`** child group.

   On the **EAST** group details screen, click the **northeast** group link.

   On the **NORTHEAST** group details screen, click the **+ADD HOST** button. On the **CREATE HOST** screen, enter **`boston1.example.com`** in the **NAME** field. Click the **SAVE** button on the **CREATE HOST** screen to finalize the addition of the host to the child group.

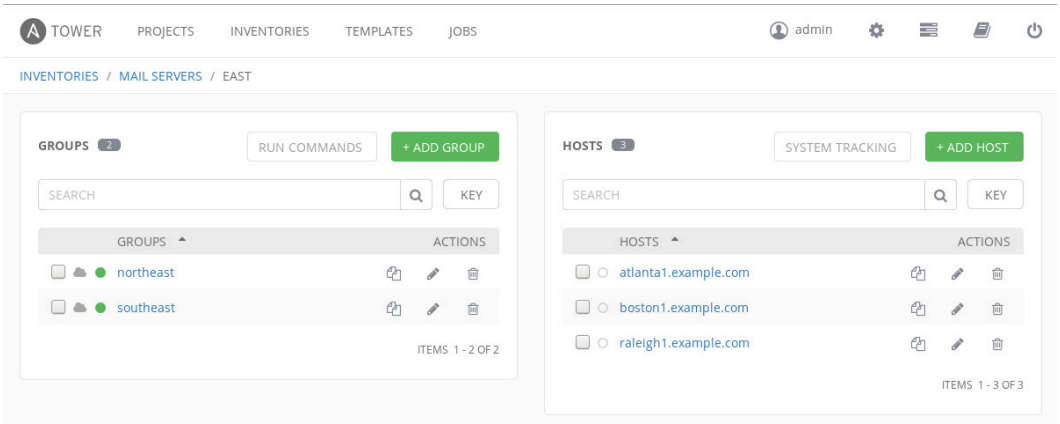

Figure 3.5: Completed host group detail screen

9. The completed **`Mail Servers`** inventory screen looks like the following:

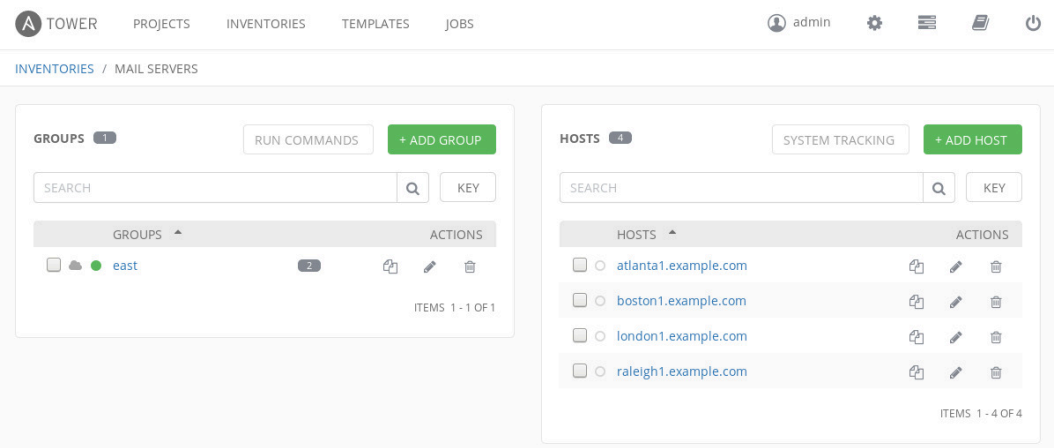

Figure 3.6: Completed inventory detail screen

All four hosts in the inventory are shown: the one in **`northeast`**, the two in **`southeast`**, and the ungrouped one. This provides an immediate overview of all the hosts in the inventory.

The **`east`** host group is also shown. The **2** to the right of its name indicates it has two children. Clicking on **`east`** displays a detail page showing the **`northeast`** and **`southeast`** child groups and all the host members of **`east`** and its children, providing an overview of just the machines in that group in the inventory.

The document stack icon can be used to move a host or host group to another host group or to the top level of the inventory. It can also be used to create a copy of that host or host group.

If you click on a trash can icon next to a host, it is deleted from all host groups in the inventory.

If you click on a trash can icon next to a host group, you are given a choice before the host group is deleted:

- Delete all the children belonging to the host group.
- *Promote* the children belonging to the host group to its parent object. For example, when deleting **`east`**, its child groups could be promoted to top level groups in the inventory.

Inventory Roles

A previous section covered how RBAC roles can be assigned to allow Users and Teams to view and manage Teams and Organizations. Roles can also be assigned to allow them to view and manage other objects such as Inventories.

Users and Teams can be assigned the ability to read, use, or manage an Inventory by assigning them appropriate roles for that Inventory. In some cases, instead of directly assigning a role to the User or Team, they may inherit a role granting them permissions to work with an Inventory indirectly.

The following is the list of available roles for an Inventory.

admin
The Inventory **admin** role grants Users full permissions over an Inventory. These permissions include deletion and modification of the Inventory. In addition, this role also grants permissions associated with the Inventory roles **use**, **adhoc**, and **update**, which is discussed shortly.

use
The Inventory **use** role grants Users the ability to use an Inventory in a Job Template resource. This controls which Inventory is used to launch Jobs using the Job Template's playbook. Using an Inventory in a Job Template is discussed in detail in a later chapter.

adhoc
The Inventory **adhoc** role grants Users the ability to use the Inventory to execute ad hoc commands. Using Ansible Tower for ad hoc command execution is discussed in detail in a later chapter.

update
The Inventory **update** role grants Users the ability to update a dynamic inventory from its external data source. This is discussed in more detail later in this course.

read
The Inventory **read** role grants Users the ability to view the contents of an Inventory.

When an Inventory is first created, it is only accessible by Users who have either the **admin** or **auditor** roles for the Organization to which the Inventory belongs. All other access must be specifically configured.

This is done by assigning Users and Teams appropriate roles as discussed above. The Inventory must be created and saved before Users and Teams can be assigned roles on it.

Roles are assigned through the **PERMISSIONS** section of the Inventory's editor screen (accessible through the pencil icon next to its name). The following procedure details the steps to assign Users and Teams roles on an Inventory.

1. Log in as a User with **admin** role on the Organization in which the Inventory was created.
2. Click **INVENTORIES** in the quick navigation links to display the Organization's Inventory list.
3. Click the pencil icon for the Inventory to enter the Inventory's editor screen.
4. On the Inventory's editor screen, click the **PERMISSIONS** button to enter the permissions editor.

   A list of Users and Teams that already have been assigned roles, and what those roles are, will appear here. If there is an **X** next to the role, it can be clicked to remove that role on the Inventory for that User.
5. Click the **+ADD** button to add permissions.
6. In the **ADD USERS / TEAMS** selection screen, click either **USERS** or **TEAMS** and then select the check boxes next to the Users or Teams which will be assigned new roles.
7. Under **Please assign roles to the selected users/teams**, click the **SELECT ROLES** drop-down list and select the desired Inventory role for each User or Team.

   A list of Inventory roles and their definitions is available by clicking **KEY**.

8. Click **SAVE** to finalize the new roles.

Important

Permissions for Inventories can also be added through the User or Team management screens under Tower's **Settings** (the gear icon) interface. However, those interfaces currently do not provide the Inventory **`read`** role as an option for assignment.

Configuring Inventory Variables

Ansible supports *inventory variables* that apply values to variables on particular hosts (*host variables*) or to host groups (*group variables*).

If you are using Ansible by itself, the recommended way to set inventory variables is to use **`host_vars`** and **`group_vars`** directories in the same working directory as the inventory file. These directories contain YAML files named after the host or host group they represent, which define variables and values that should apply to that host or host group. Variables can be assigned to all hosts in an inventory by defining them for the special **`all`** group.

For example, the file **`group_vars/southeast`** might define the value of the **`ntp`** variable to **`ntp-se.example.com`** for the host group **`southeast`**:

```
---
ntp: ntp-se.example.com
```

With Ansible Tower, instead of using **`host_vars`** and **`group_vars`** directories, the inventory variables and their values are defined directly in the Inventory object.

On the **INVENTORIES** screen, variables can be set by clicking on the **Edit inventory** (pencil) icon next to the Inventory's name. On the **DETAILS** screen for the Inventory, variables can be set which affect all hosts in the Inventory:

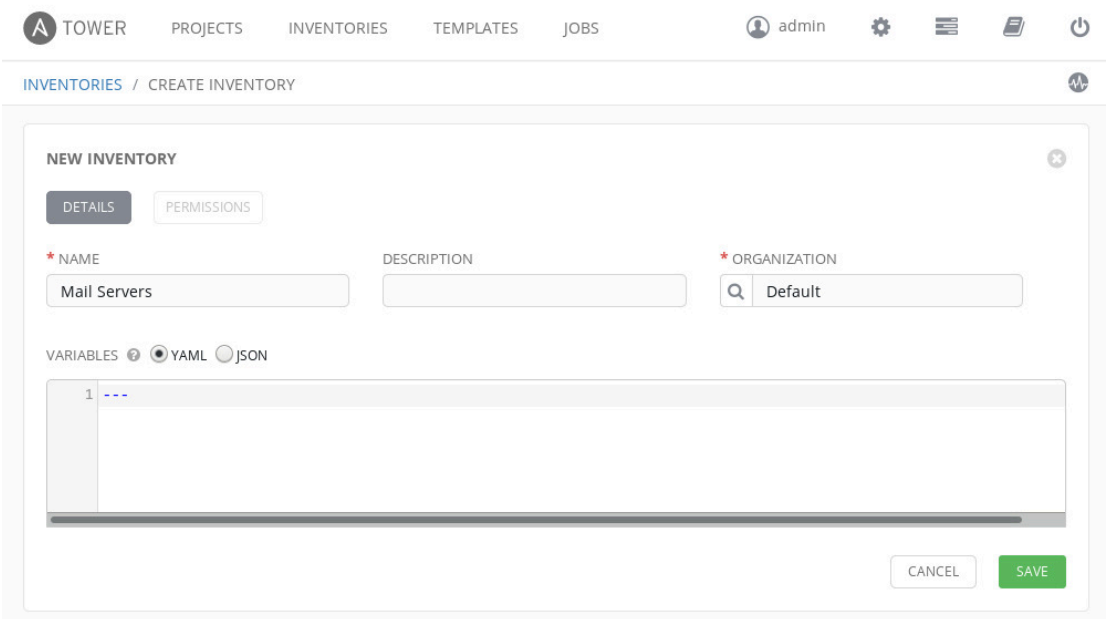

Figure 3.7: Variables for all hosts

When a host group is created within an inventory, group variables can be defined using either YAML or JSON in the **VARIABLES** field on the **CREATE GROUP** screen. They can also be set by clicking on the **Edit group** (pencil) icon next to the host group's name in the Inventory. These variables apply to all hosts which are part of the group:

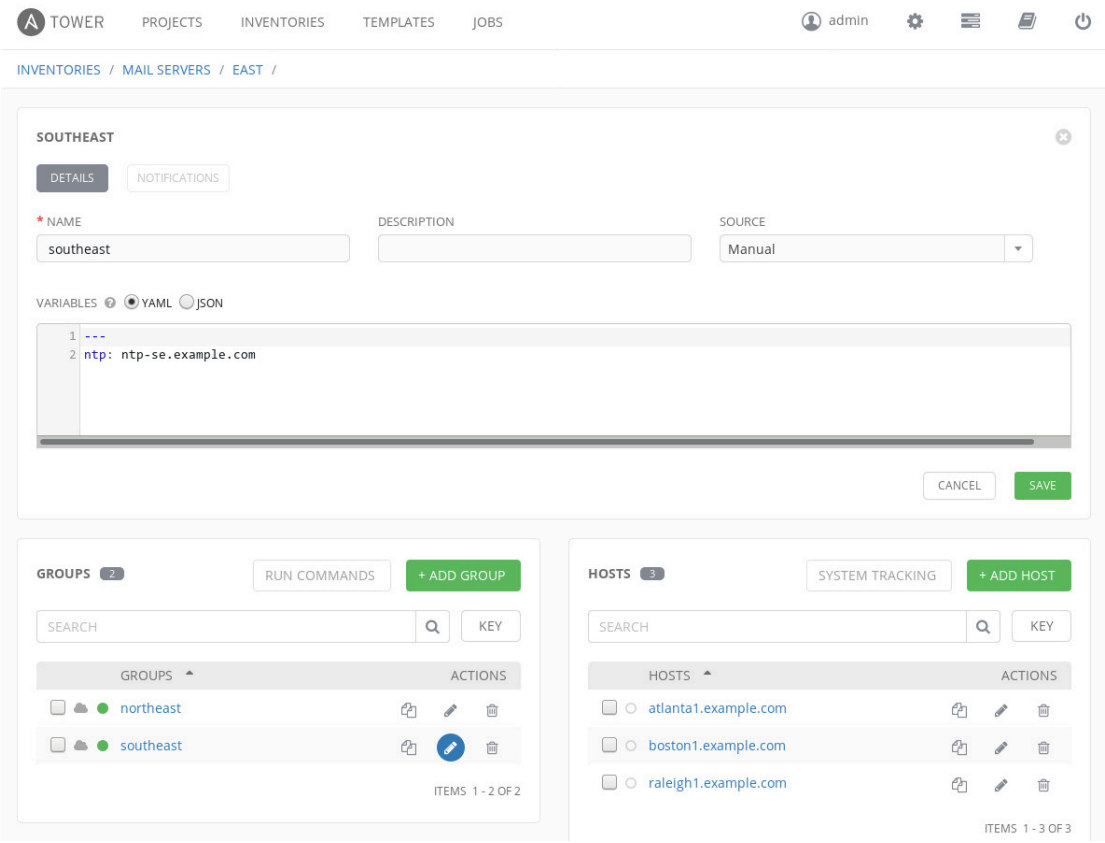

Figure 3.8: Variables for a host group

Likewise, host variables can be defined using either YAML or JSON in the **VARIABLES** field on the **CREATE HOST** screen when an individual host is created within an inventory, or by clicking on the **Edit host** (pencil) icon next to the host's name in the Inventory once created. Variables defined in this manner only apply to the specific host.

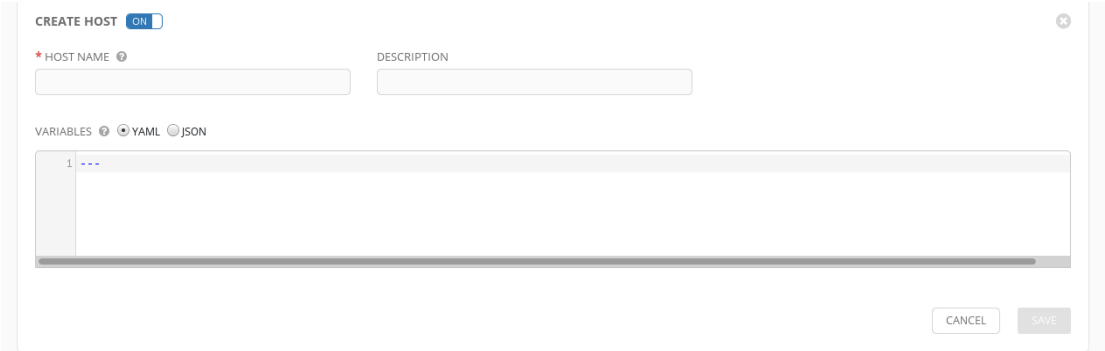

Figure 3.9: Variables for an individual host

Important

Remember that inventory variables can be overridden by variables with a higher precedence. Extra variables defined in a Job Template and playbook variables both have higher precedence than inventory variables.

References

Further information is available in the Inventories section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1 at

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Further information is available in the Built-in Roles section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1 at

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Creating a Static Inventory

In this exercise, you will create static inventories and configure their permissions.

Outcomes
You should be able to create and manage a static Inventory containing hosts and groups and assign appropriate permissions to a Team.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab host-inventory setup**.

```
[student@workstation ~]$ lab host-inventory setup
```

This setup script creates some additional inventories, hosts, and groups needed for this exercise.

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a static Inventory named **Prod**.

   2.1. Open the **INVENTORIES** page by clicking the **INVENTORIES** quick navigation link at the top left of the page.

   2.2. Click the **+ADD** button to add a new Inventory.

   2.3. On the next screen, fill in the details as follows:

Field	Value
NAME	Prod
DESCRIPTION	Production Inventory
ORGANIZATION	Default

   2.4. Click **SAVE** to create the new Inventory. You are redirected to the Inventory detail page.

3. Create the group **prod-servers** in the **Prod** Inventory.

   3.1. Click the **+ADD GROUP** button to add a new group.

   3.2. On the next screen, fill in the details as follows:

Field	Value
NAME	prod-servers
DESCRIPTION	Production servers
SOURCE	Manual

   3.3. Click **SAVE** to create the new group.

4. Add the host, **servere.lab.example.com**, to the group **prod-servers** in the **Prod** Inventory.

   4.1. Click the link for the **prod-servers** group that you just created.

> ## Important
>
> Omitting this step creates a host in the **Prod** Inventory but not in the **prod-servers** group.

   4.2. Click the **+ADD HOST** button to add a new host to the group.

   4.3. On the next screen, fill in the details as follows:

Field	Value
HOST NAME	servere.lab.example.com
DESCRIPTION	Server E

   4.4. Click **SAVE** to add the new host.

5. Assign the **Operations** Team **Admin** role to the **Prod** Inventory.

   5.1. Click the **INVENTORIES** quick navigation link to display the list of available Inventories.

   5.2. On the same line as the **Prod** Inventory entry, click the pencil icon to edit the Inventory.

   5.3. On the next page, click **PERMISSIONS** to manage the **Prod** Inventory's permissions.

   5.4. Click the **+ADD** button to add permissions.

   5.5. Click **TEAMS** to display the list of available Teams.

   5.6. In the first section, check the box next to the **Operations** Team. This causes the Team to display in the second section.

   5.7. In the second section below, select the **Admin** role from the drop-down list.

   5.8. Click **SAVE** to finalize the role assignment. This redirects you to the list of permissions for the **Prod** Inventory, which now shows that all members of the **Operations** Team are assigned the **Admin** role on the **Prod** Inventory.

6. Verify access to the **Prod** Inventory with the User, **oliver**, who belongs to the **Operations** Team.

   6.1. Click the **Log Out** icon in the top right corner to log out. Then log back in as **oliver** with a password of **redhat123**. This User is assigned the **Member** role for the **Operations** Team.

   6.2. Click **INVENTORIES** to display the list of available Inventories.

   6.3. Click the link for the **Prod** Inventory created earlier.

   6.4. Review the contents of the **Prod** Inventory to see the hosts and group it contains.

7. Add the host, **serverf.lab.example.com**, to the **Prod** Inventory while logged in as the User, **oliver**.

   7.1. Click the link for the **prod-servers** group to enter the group.

   7.2. Click the **+ADD HOST** button to add a second host to the group.

   7.3. On the next screen, fill in the details as follows:

Field	Value
HOST NAME	serverf.lab.example.com
DESCRIPTION	Server F

   7.4. Click **SAVE** to create the second host.

8. Assign the **Use** role on the **Test** Inventory to the **Developers** Team.

   8.1. Click the **Log Out** icon in the top right corner to log out and then log back in as **admin** with a password of **redhat**.

   8.2. Click **INVENTORIES** to display the list of available Inventories.

   8.3. On the same line as the **Test** Inventory, click the pencil icon to edit the Inventory.

   8.4. On the next page, click **PERMISSIONS** to manage the Inventory's permissions.

   8.5. Click the **+ADD** button to add permissions.

   8.6. Click **TEAMS** to display the list of available Teams.

   8.7. In the first section, check the box next to the **Developers** Team. This causes the Team to display in the second section underneath the first one.

   8.8. In the second section, select the **Use** role from the dropdown list.

   8.9. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the **Test** Inventory, which now shows that all members of the **Developers** Team are assigned the **Use** role on the **Test** Inventory.

9. Verify access to the **Test** Inventory with the **daniel** User who belongs to the **Developers** Team.

   9.1. Click the **Log Out** icon in the top right corner to log out. Then log back in as **daniel** with a password of **redhat123**. This User is assigned the **Admin** role for the **Developers** Team.

   9.2. Click **INVENTORIES** to display the list of available Inventories.

   9.3. Click the link for the **Test** Inventory.

   9.4. Review the contents of the **Test** Inventory to see the hosts and group it contains. Note that even though **daniel** is an **Admin** of the **Developers** Team, he cannot manage the **Test** Inventory because you only granted the **Use** role for the Inventory to the **Developers** Team.

Note

This is representative of a real world scenario where developers may have access to the list of systems in the testing environment but are not able to modify the list.

9.5. Click the **Log Out** icon in the top right corner to log out of the Tower web interface.

Creating Machine Credentials for Access to Inventory Hosts

Objectives

After completing this section, students should be able to:

- Create a machine credential to invoke Ansible on hosts under your management and assign roles to an SSH private key for users or teams.

Credentials

Credentials are Ansible Tower objects that are used to authenticate to remote systems for various purposes. They may provide secrets such as passwords, SSH keys, or other supporting information needed to successfully access or use a remote resource.

Ansible Tower is responsible for maintaining secure storage for the secrets in these Credential objects. Credential passwords and keys are encrypted before they are saved to the Tower database, and can not be retrieved in clear text from the Tower user interface. Users and Teams can be assigned privileges to use Credentials, but the secrets are not exposed to them. This means that when a User changes Team or leaves the organization, the credentials and systems do not need to be rekeyed. When a Credential are needed by Tower, it decrypts the data internally and passes it to SSH or other program directly.

Important

Once sensitive authentication data is entered into a Credential and encrypted, it can no longer be retrieved in decrypted form through the Tower web interface.

Credential Types

Ansible Tower has a number of different types of Credentials that it can manage:

- *Machine* credentials are used to authenticate playbook logins and privilege escalation for Inventory hosts
- *Network* credentials are used when Ansible network modules are used to manage networking equipment
- *Source Control* (or SCM) credentials are used by Projects to clone and update Ansible project materials from a remote version control system such as Git, Subversion, or Mercurial
- *Inventory* credentials are used to update dynamic inventory information from one of Ansible Tower's built-in dynamic inventory sources. There are separate credential types for each inventory source in question: Amazon Web Services, VMware vCenter, Red Hat Satellite 6, Red Hat Cloud Forms, Google Compute Engine, Microsoft Azure Resource Manager, OpenStack, and so on.

This section focuses on how to set up machine credentials for the hosts in an Inventory with appropriate login and privilege escalation information. Other types of credentials are discussed in more detail elsewhere in the course.

Creating Machine Credentials

Credentials are managed through the **CREDENTIALS** interface under Tower's **Settings** (the gear icon) screen.

Any User can create a Credential, and is considered the *owner* of that Credential. If the Credential is not assigned to an Organization, it is a "private" Credential. This means that only the User that owns the Credential and Users with the Tower System Administrator singleton role can use it, and only they and Users with the Tower System Auditor singleton role can see it.

On the other hand, if the Credential is assigned to an Organization, then it is an "Organization" Credential. Only Tower System Administrators and Users with **admin** on an Organization can create Credentials assigned to an Organization. Users and Teams in the Organization can be granted roles on a Credential assigned to the Organization, to use or manage the Credential.

In summary, the main distinctions between "private" Credentials and those assigned to an Organization are:

- Any User can create a private Credential, but only Tower System Administrators and Users with Organization **admin** can create an Organization Credential
- If a Credential belongs to an Organization, Users and Teams can be granted roles on it and it can be shared. Private Credentials that are not assigned to an Organization can only be used by the owner and the Tower singleton roles, and other Users and Teams can not be granted roles on it.

Important

The Tower **admin** user can assign an Organization to an existing private Credential, converting it into an Organization Credential.

The following procedure details how Machine Credentials are created.

1. Login as a User with the appropriate role assignment. If creating a private Credential, there are no specific role requirements. If creating an Organization Credential, login as a User with **admin** role for the Organization.
2. Click **Settings** (the gear icon) and then click **CREDENTIALS** to enter the Credentials management interface.
3. On the **CREDENTIAL** screen, click **+ADD** to create a new Credential.
4. On the **CREATE CREDENTIAL** screen, enter the required information for the new Credential.

   A **NAME** is required, and the **TYPE** drop-down menu should be used to select **Machine**.

   If the user has Organization **admin** privileges, the **ORGANIZATION** can be set to assign this Credential to an Organization. If the User does not have **admin** privileges, the **ORGANIZATION** field is not present and only private Credentials can be created.
5. For Machine Credentials, additional fields appear in the **TYPE DETAILS** section, as shown in the following illustration:

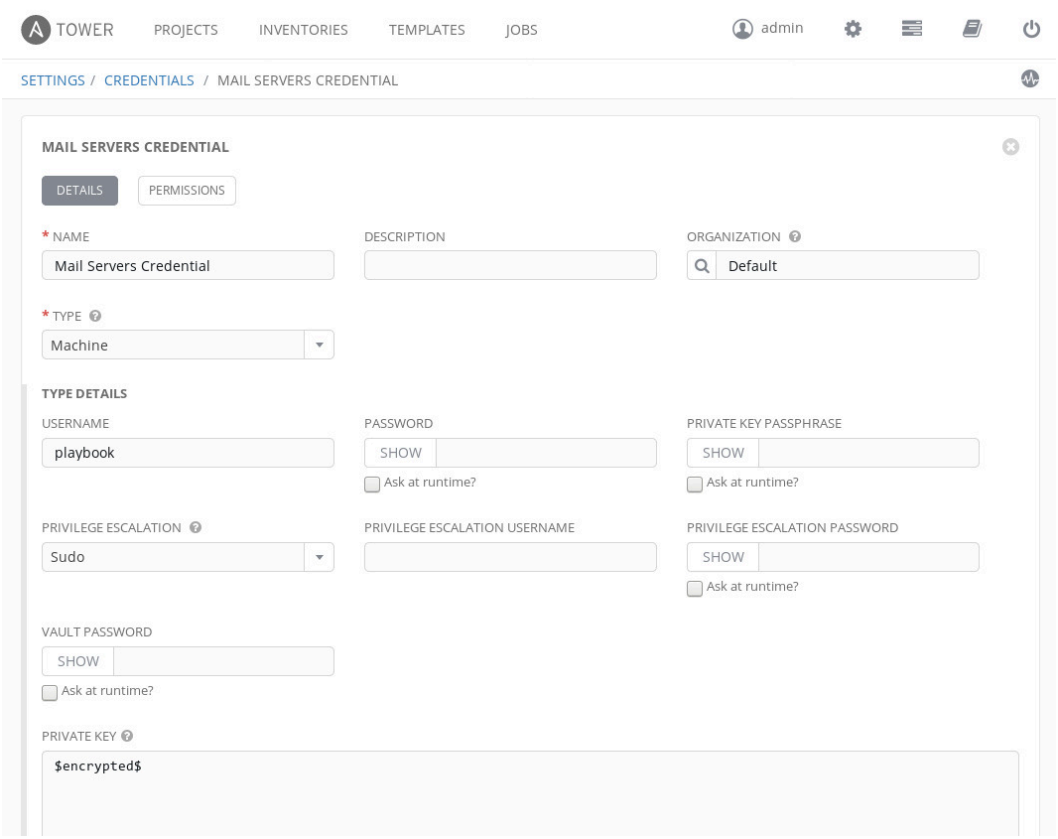

Figure 3.10: New Machine Credential (After Save)

These fields can contain the information needed to authenticate to and escalate privileges on the machines that use this Credential. Many of them are map to settings which might be in an **`ansible.cfg`** file:

- **USERNAME** is the username to log in as on the remote system (**`remote_user`** in **`ansible.cfg`**)
- **PASSWORD** is the SSH password to use for that user. Leave this blank if private key authentication is used.
- The **PRIVATE KEY** field contains the SSH private key needed to log in as **USERNAME** on the machines. It is easier to cut and paste the text from the file rather than to manually type it in. If you are using **Firefox** running in GNOME 3 on a Linux system to administer Tower, you can drag and drop the private key file from the **Files** application window into this field in your web browser window.

  Once the Credential is saved, this will be encrypted by Tower and the field reads **`$encrypted$`**.
- **PRIVATE KEY PASSPHRASE** is the passphrase to use to decrypt the **PRIVATE KEY** if an private key that is itself encrypted by SSH is pasted into that field. Otherwise, this can be blank.

- **PRIVILEGE ESCALATION** is a drop-down menu that chooses what type of privilege escalation, if any, is needed (**become_method**). This affects other fields that may appear.

  For **sudo** privilege escalation, **PRIVILEGE ESCALATION USERNAME** is the privileged user on the remote machine that Ansible should use (**become_user**). **PRIVILEGE ESCALATION PASSWORD** is the **sudo** password to use. This can be blank if no password is needed.

- **VAULT PASSWORD** is the Ansible Vault password that will be needed to decrypt Vault for Job Templates using this Credential.

6. Click the **SAVE** button to finalize the creation of the new Machine Credential.

Editing Machine Credentials

Once Machine Credentials are created, they can be edited, if needed, using the Credential editor interface. The following procedure details how Credentials are modified.

1. Log in as a User with the appropriate role assignment. If editing a private Credential, login as the User who created the Credential. If editing an Organization Credential, login as a User with **admin** role on the Organization Credential.

2. Click the **Settings** icon and then click **CREDENTIALS** to enter the Credentials management interface.

3. On the **CREDENTIAL** screen, click the name of the Credential to edit.

4. On the Credential editor screen, make the necessary changes to the desired Credential properties.

5. Click **SAVE** button to finalize the changes made to the Credential.

Credential Roles

As discussed earlier, private Credentials (Credentials that are not assigned to an Organization) are only accessible to their creators or to Users that have the System Administrator or System Auditor user type. Other users can not be assigned roles on private Credentials.

To assign roles to Credentials, the Credential must have an Organization. Then Users and Teams in that Organization can share that Credential through role assignments.

The following is the list of available Credential roles.

admin
The Credential **admin** role grants Users full permissions on a Credential. These permissions include deletion and modification of the Credential, as well as the ability to use the Credential in a Job Template.

use
The Credential **use** role grants Users the ability to use a Credential in a Job Template. Use of a Credential in a Job Template is discussed later in this course.

read
The Credential **read** role grants Users the ability to view the details of a Credential. This still does not allow them to decrypt the secrets which belong to that Credential through the web interface.

Managing Credential Access

When an Organization Credential is first created, it is only accessible by the owner and Users with either the **admin** or **auditor** role in the Organization in which the Credential was created. Additional access must be specifically configured if desired.

Additional roles cannot be assigned at the time of an Organization Credential's creation, until it is first saved. Then they can be set by editing the Credential.

Roles are assigned through the **PERMISSIONS** section of the Credential editor screen.

The following procedure details the steps for granting permissions to an Organization Credential after its creation.

1. Log in as a User with admin role on the Organization that the Credential was created in.
2. Click the **Settings** icon and then click **CREDENTIALS** to enter the Credentials management interface.
3. Click the name of the Credential to edit in order to enter the Credential editor screen.
4. On the Credential editor screen, click the **PERMISSIONS** button to enter the permissions editor.
5. Click the **+ADD** button to add permissions.
6. In the User and Team selection screen, click either **USERS** or **TEAMS** and then select the checkboxes for the Users or Teams to be assigned roles.
7. Under **Please assign roles for the selected users/teams**, click **KEY** to display the list of Credential roles and their definitions.
8. Click the **SELECT ROLES** drop-down list and select the desired Credential role for each User or Team.
9. Click **SAVE** to finalize the changes to permissions.

Important

Permissions for Credentials can also be added through either the User or Team management screens under Tower's Settings interface. However, those interfaces currently do not provide the Credential **read** role as an option for role assignment.

In general, the **read** role is rarely used directly, but is automatically granted to System Auditors and Organization **auditor** roles by Tower internally. It is expected that direct use of **read** will be deprecated in future versions of Ansible Tower.

Common Credential Scenarios

To help you understand ways in which Credentials are used, her are some common Credential scenarios:

Credentials Protected by Tower, Not Known to Users

One common scenario for the use of Tower Credentials is the delegation of task execution from administrators to Tier 1 support staff.

For example, suppose that the support staff needs to be delegated the ability to run a playbook which ensures a web application has been restarted to restore service when outages occur outside of business hours. The support staff's Credential uses a shared account, **`support`**, and a passphrase-protected private key for SSH authentication to managed hosts. The **`support`** account needs to escalate privileges using **sudo**, with a **sudo** password, in order to run the playbook.

Since the Credential is shared by the support staff's Team, an Organization Credential resource should be created to store the username, SSH private key, and SSH key passphrase needed to authenticate SSH sessions to the managed hosts. The Credential also stores the privilege escalation method, username, and **sudo** password information. Once created, the Credential can be used by the support staff to launch Jobs on the managed hosts without needing to know the SSH key passphrase or **sudo** password.

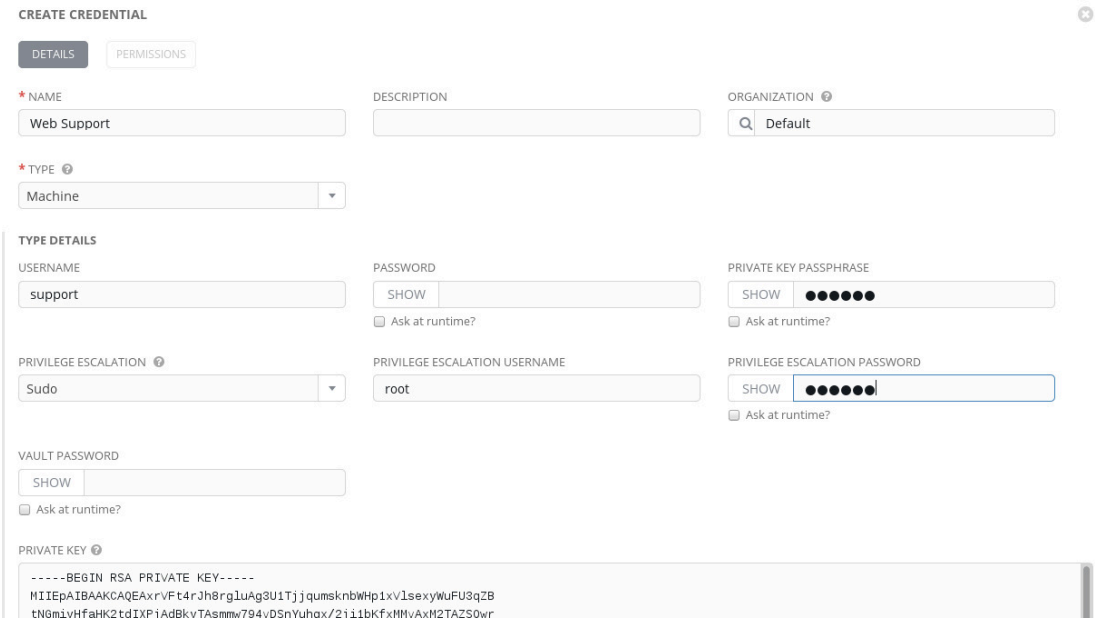

Figure 3.11: Organization Credential for shared account

Credential Prompts for Sensitive Password, Not Stored in Tower

Another scenario is to use Credentials to store username authentication information while still prompting interactively for a sensitive password when the Credential is used.

Suppose that a database administrator wants to run a playbook managed in Tower to execute tasks on a database server which houses sensitive data for the company financials. Due to the highly sensitive nature of the data, the company's financial compliance regulations forbid the storage of the account's password.

A machine Credential is still used to configure the database administrator's authentication to the database server. Since the Credential is not to be shared, a private Credential can be used to store the SSH **USERNAME**. It is also configured to prompt the User for the account's password when the Credential is used by a Job, by selecting the **Ask at runtime?** check box for **PASSWORD**.

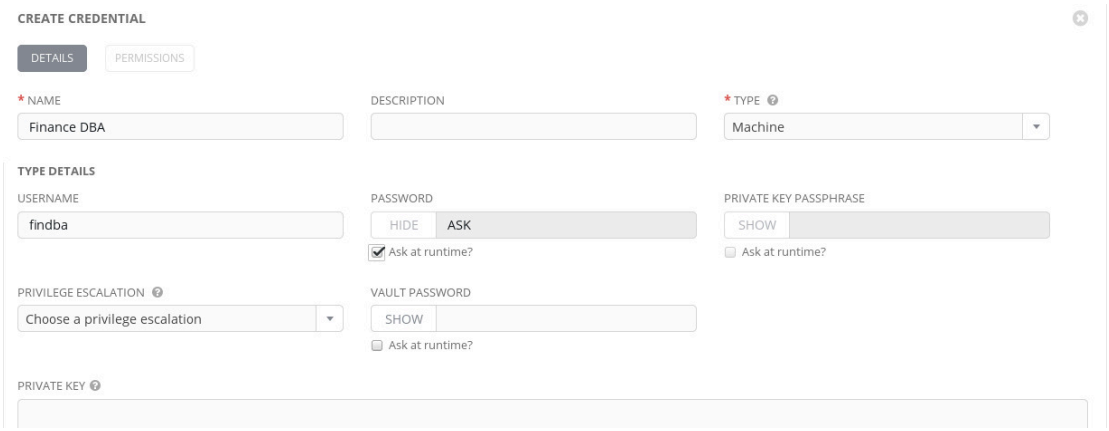

Figure 3.12: Private Credential with password prompting

Important

Ansible Tower has a feature that allows playbooks to be run automatically on a schedule, much like a **cron** job. Credentials configured to prompt interactively for password information at runtime can not be used with scheduled jobs, since Tower can not provide that information without user interaction.

References

Further information is available in the Credentials section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1 at

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Further information is available in the Built-in Roles section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1 at

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Creating Machine Credentials for Access to Inventory Hosts

In this exercise, you will create and assign roles to Credentials.

Outcomes
You should be able to create and manage Machine Credentials to be used against hosts and groups in an Inventory.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Steps

1. Log in to the Ansible Tower web interface running on the **`tower`** system using the **`admin`** account and the **`redhat`** password.

2. Create a new Credential called **`Operations`**.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   2.2. Click **CREDENTIALS** to manage Credentials.

   2.3. Click the **+ADD** button to add a new Credential.

   2.4. On the next screen, fill in the details as follows:

Field	**Value**
NAME	Operations
DESCRIPTION	Operations Credential
ORGANIZATION	Default
TYPE	Machine
USERNAME	root
PASSWORD	redhat

   2.5. Leave the other fields untouched and click **SAVE** to create the new Credential.

3. Assign the **`Operations`** Team the **`Admin`** role on the **`Operations`** Credential.

   3.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   3.2. Click **CREDENTIALS** to manage Credentials.

   3.3. On the same line as the **`Operations`** Credential, click the pencil icon to edit the **`Operations`** Credential.

   3.4. On the next page, click **PERMISSIONS** to manage the permissions for the Credential.

   3.5. Click the **+ADD** button to add permissions.

3.6. Click **TEAMS** to display the list of available Teams.

3.7. In the first section, check the box next to the **Operations** Team. This causes the Team to display in the second section underneath the first one.

3.8. In the second section below, select the **Admin** Role from the drop-down list.

3.9. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the **Operations** Credential which now shows that all members of the **Operations** Team, **oliver** and **ophelia**, are assigned the **Admin** role on the **Operations** Credential.

4. Verify the permissions of the **Admin** role to the **Operations** Credential with the User **oliver**, who belongs to the **Operations** Team.

4.1. Click the **Log Out** icon in the top right corner to log out and sign back in as **oliver** with password of **redhat123**. This User is a **Member** of the **Operations** Team.

4.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

4.3. Click **CREDENTIALS** to display the list of available Credentials.

4.4. Click the link for the **Operations** Credential created earlier. Note how **oliver** can modify the Credential.

5. Assign the **Developers** Team the **Use** role on the **Operations** Credential.

5.1. Click the **Log Out** icon in the top right corner to log out and log back in as **admin** with a password of **redhat**.

5.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

5.3. Click **CREDENTIALS** to display the list of available Credentials.

5.4. On the same line as the **Operations** Credential entry, click the pencil icon to edit the Credential.

5.5. On the next page, click **PERMISSIONS** to manage the permissions.

5.6. Click the **+ADD** button to add permissions.

5.7. Click **TEAMS** to display the list of available Teams.

5.8. In the first section, check the box next to the **Developers** Team. This causes the Team to display in the second section underneath the first one.

5.9. In the second section below, select the **Use** role from the drop-down list.

5.10. Click **SAVE** to finalize the role assignment. This redirects you to the list of permissions for the **Operations** Credential which now shows that all members of the **Developers** Team, **daniel**, **david** and **sam**, are assigned the **Use** role on the **Operations** Credential.

6. Verify the **Use** role for the **Operations** Credential with the User **daniel**, who belongs to the **Developers** Team.

6.1. Click the **Log Out** icon in the top right corner to log out and log back in as **daniel** with **redhat123** as the password. This User has an **Admin** role for the **Developers** Team.

6.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

6.3. Click **CREDENTIALS** to display the list of available Credentials.

6.4. Click the link for the **Operations** Credential created earlier. Note that **daniel** cannot modify the Credential even though he has an **Admin** role for the Team.

6.5. Click the **Log Out** icon in the top right corner to log out of the Tower web interface.

Lab: Creating and Managing Inventories and Credentials

In this lab, you will create and manage permissions to Inventories and Credentials.

Outcomes
You should be able to manage Inventories and Credentials in order for a Team to be able to run a playbook against an inventory.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Steps

1. Log in to the Ansible Tower web interface running on the **`tower`** system using the **`admin`** account and the **`redhat`** password.
2. Create a new Inventory called **`Dev`** within the **`Default`** Organization.
3. Create a group called **`dev-servers`** in the **`Dev`** Inventory.
4. Add two hosts with the host names **`servera.lab.example.com`** and **`serverb.lab.example.com`** to the **`dev-servers`** group.
5. Grant the **`Admin`** role on the **`Dev`** Inventory to the **`Developers`** Team.
6. Create a new Credential, **`Developers`**, with the following information:

Field	Value
NAME	Developers
DESCRIPTION	Developers Credential
ORGANIZATION	Default
TYPE	Machine
USERNAME	student
PASSWORD	student
PRIVILEGE ESCALATION	Sudo
PRIVILEGE ESCALATION USERNAME	root
PRIVILEGE ESCALATION PASSWORD	student

7. Grant the **`Admin`** role on the **`Developers`** Credential to the **`Developers`** Team.
8. Run the command, **`lab host-review grade`**, on **`workstation`** to grade your exercise.

Solution

In this lab, you will create and manage permissions to Inventories and Credentials.

Outcomes
You should be able to manage Inventories and Credentials in order for a Team to be able to run a playbook against an inventory.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a new Inventory called **Dev** within the **Default** Organization.

   2.1. Open the **INVENTORIES** page by clicking the **INVENTORIES** quick navigation link at the top left of the page.

   2.2. Click the **+ADD** button to add a new Inventory.

   2.3. On the next screen, fill in the details as follows:

Field	Value
NAME	Dev
DESCRIPTION	Development Inventory
ORGANIZATION	Default

   2.4. Click **SAVE** to create the new Inventory. You are redirected to the Inventory details page.

3. Create a group called **dev-servers** in the **Dev** Inventory.

   3.1. Click the **+ADD GROUP** button to add a new group.

   3.2. On the next screen, fill in the details as follows:

Field	Value
NAME	dev-servers
DESCRIPTION	Development servers
SOURCE	Manual

   3.3. Click **SAVE** to create the new group.

4. Add two hosts with the host names **servera.lab.example.com** and **serverb.lab.example.com** to the **dev-servers** group.

   4.1. Click the link for the **dev-servers** group you just created.

   4.2. Click the **+ADD HOST** button to add a new host to the group.

4.3. On the next screen, fill in the details as follows:

Field	Value
HOST NAME	servera.lab.example.com
DESCRIPTION	Server A

4.4. Click **SAVE** to create the new host.

4.5. Click the **+ADD HOST** button to add another new host to the group.

4.6. On the next screen, fill in the details as follows:

Field	Value
HOST NAME	serverb.lab.example.com
DESCRIPTION	Server B

4.7. Click **SAVE** to create the new host.

5. Grant the **Admin** role on the **Dev** Inventory to the **Developers** Team.

5.1. Click the **INVENTORIES** quick navigation link to display the list of available Inventories.

5.2. On the same line as the **Dev** Inventory, click the pencil icon to edit the Inventory.

5.3. On the next page, click **PERMISSIONS** to manage the Inventory's permissions.

5.4. Click the **+ADD** button to add permissions.

5.5. Click **TEAMS** to display the list of available Teams.

5.6. In the first section, check the box next to the **Developers** Team. This causes the Team to display in the second section underneath the first one.

5.7. In the second section, select the **Admin** role from the drop-down list.

5.8. Click **SAVE** to finalize the role assignment. You are redirected to the list of permissions for the **Dev** Inventory which now shows that all members of the **Developers** Team are assigned the **Admin** role on the **Dev** Inventory.

6. Create a new Credential, **Developers**, with the following information:

Field	Value
NAME	Developers
DESCRIPTION	Developers Credential
ORGANIZATION	Default
TYPE	Machine
USERNAME	student
PASSWORD	student
PRIVILEGE ESCALATION	Sudo

PRIVILEGE ESCALATION USERNAME	root
PRIVILEGE ESCALATION PASSWORD	student

6.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

6.2. Click **CREDENTIALS** to manage Credentials.

6.3. Click the **+ADD** button to add a new Credential.

6.4. Create a new Credential, **`Developers`**, with the following information:

Field	**Value**
NAME	Developers
DESCRIPTION	Developers Credential
ORGANIZATION	Default
TYPE	Machine
USERNAME	student
PASSWORD	student
PRIVILEGE ESCALATION	Sudo
PRIVILEGE ESCALATION USERNAME	root
PRIVILEGE ESCALATION PASSWORD	student

6.5. Leave the other fields untouched and click **SAVE** to create the new Credential.

7. Grant the **`Admin`** role on the **`Developers`** Credential to the **`Developers`** Team.

7.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

7.2. Click **CREDENTIALS** to manage Credentials.

7.3. On the same line as the **`Developers`** Credential, click the pencil icon to edit the Credential.

7.4. On the next page, click **PERMISSIONS** to manage the Credential's permissions.

7.5. Click the **+ADD** button to add permissions.

7.6. Click **TEAMS** to display the list of available Teams.

7.7. In the first section, check the box next to the **`Developers`** Team. This causes the Team to display in the second section underneath the first one.

7.8. In the second section below, select the **`Admin`** Role from the drop-down list.

7.9. Click **SAVE** to finalize the role assignment. This redirects you to the list of permissions for the **`Developers`** Credential which now shows that the Users, **`daniel`**, **`david`**, and **`sam`**, are assigned the **`Admin`** role on the **`Developers`** Credential.

8. Run the command, **`lab host-review grade`**, on **`workstation`** to grade your exercise.

Summary

In this chapter, you learned:

- Inventory resources are used to manage Ansible inventories of hosts and hostgroups and their inventory variables
- Multiple Inventories can be configured, and roles can be used to manage who can use and administer particular Inventories
- Static inventories can be manually configured through the web interface
- Credentials are used to store authentication information for machines, network devices, source control, and dynamic inventory updates
- Machine credentials are used to allow Tower to authenticate access and privilege escalation on inventory hosts for playbook execution
- Credentials assigned to an Organization can be shared by granting roles to Users and Teams
- Credentials not assigned to an Organization are private to the User who created it and to the Tower singleton roles, and cannot be shared without assigning it to an Organization

CHAPTER 4

MANAGING PROJECTS FOR PROVISIONING WITH ANSIBLE TOWER

Overview	
Goal	Create basic Projects and Job Templates in Ansible Tower, which can be used to run Ansible playbooks in order to provision and configure managed systems
Objectives	• Store and manage Ansible projects intended for use with Ansible Tower in the Git version control system; be able to explain basic Git operations and recommended practices for structure of Ansible projects. • Create a Project that gets Ansible materials from a Git repository using an SCM credential configured with a SSH private key; configure it so a Team can Use and Update the project; explain the differences between the Admin, User, Update, and Read roles for projects • Create a Job Template from a Project to run a playbook, launch a Job using that template, and be able to interpret whether the job succeeded or had failures
Sections	• Managing Ansible Project Materials Using Git (and Guided Exercise) • Creating a Project for Ansible Playbooks and Roles (and Guided Exercise) • Creating Job Templates and Launching Jobs (and Guided Exercise)
Lab	• Managing Projects for Provisioning with Ansible Tower

Managing Ansible Project Materials Using Git

Objectives

- Store and manage Ansible projects intended for use with Ansible Tower in the Git version control system
- Explain basic Git operations and recommended practices for structure of Ansible projects

Infrastructure as Code

One key DevOps concept is the idea of *infrastructure as code*. Rather than managing infrastructure through the manual execution of commands, essential components are built and maintained through automated, programmatic procedures. Ansible projects and their playbooks are key tools which helps implement this.

If Ansible projects are the code which is used to manage the infrastructure, then in accordance with development best practices a *version control system* such as Git should be used to track and control changes.

Version control allows administrators to implement a life cycle for the different stages of their infrastructure code, such as development, QA, and production. By managing infrastructure code with a version control tool, administrators can test their infrastructure code changes in noncritical development and QA environments to minimize surprises and disruptions when deployments are implemented in production environments.

Ansible Tower provides a central location from which Ansible playbooks can be run. It can pull projects containing those playbooks from a Git repository, and can even be configured so that particular Tower projects pull materials from specific branches of a Git repository.

Introducing Git

Git is a *distributed version control system* (DVCS) that allows developers to manage changes to files in a project in a collaborative manner. Each revision of a file is committed to the system. Old versions of files can be restored, and a log of who made the changes is maintained.

Version control systems provide many benefits, including:

- The ability to review and restore old versions of files
- The ability to compare two versions of the same file to identify changes
- A record or log of who made what changes at what time
- Mechanisms for multiple users to collaboratively modify files, resolve conflicting changes, and merge the changes together

Git is a *distributed* version control system. Each developer can start by *cloning* an existing shared project from a *remote repository*. This gives them a complete copy of the original remote repository as their *local repository*. This consists of a local copy of the entire history of the files in the version control system, not just the latest snapshot of the project files.

The developer makes edits in their *working tree*, a checkout of a single snapshot of the project. A set of related changes are then staged and committed to the local repository. At this point, no changes have been made to the shared remote repository.

When the developer is ready to share their work, they can *push* their changes to the remote repository. Alternatively, if their local repository is accessible from the network, they can ask the owner of the remote repository to *pull* the changes from the developer's repository to the remote repository.

Likewise, when a developer is ready to update their local repository with the latest changes to the remote repository, they can pull the changes (fetching them from the remote repository and merging them into their local work).

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

- *Modified*: the copy of the file in the working tree has been edited and is different from the latest version in the repository
- *Staged*: the modified file has been added to a list of changed files to commit as a set, but has not yet been committed
- *Committed*: the modified file has been committed to the local repository

Once the file is committed to the local repository, the commit can be pushed to or pulled by a remote repository.

Figure 4.1: The four areas where Git manages files

Initial Git Configuration

Since Git users frequently modify projects with multiple contributors, Git records the user's name and email address on each of their commits. These values can be defined at a project level, but global defaults can also be set for a user. The **`git config`** command controls these settings. Using it with the **`--global`** option manages the default settings for all Git projects to which the user contributes by saving the settings in their **`~/.gitconfig`** file.

```
[peter@host ~]$ git config --global user.name 'Peter (Star-Lord) Quill'
```

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

If **bash** is the user's shell, another useful but optional thing to do is to configure your prompt to automatically modify itself to report the status of your working tree. The easiest way is to use the **git-prompt.sh** script shipped with the *git* package.

To modify your shell prompt in this way, add the following lines to your **~/.bashrc** file. If your current directory is in a Git working tree, the name of the current Git branch for the working tree is displayed in parentheses. If you have untracked, modified, or staged files that aren't committed in your working tree, the prompt will indicate this:

- `(branch *)` means a tracked file is modified
- `(branch +)` means a tracked file is modified and staged with **git add**
- `(branch %)` means untracked files are in your tree
- Combinations of markers are possible, such as `(branch *+)`

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

The Git Workflow

When working on shared projects, the Git user clones an existing upstream repository with the **git clone** command. The path name or URL provided determines which repository is cloned into the current directory. A working tree is also created so that the directory of files is ready for revisions. Since the working tree is unmodified, it is initially in the clean state.

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

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

Note

Another way to start the Git workflow is to create a new, private project with the **git init** command. A project started in this way has no remote repository at first.

An advanced setup is to use **git init --bare** to create a *bare repository* on a server. This is used only as a remote repository by other developers, and does not have a local working tree and therefore is not usable as a local repository by anyone. The server also needs to be set up to allow users to clone, pull from, and push to the repository using the HTTPS or SSH protocol.

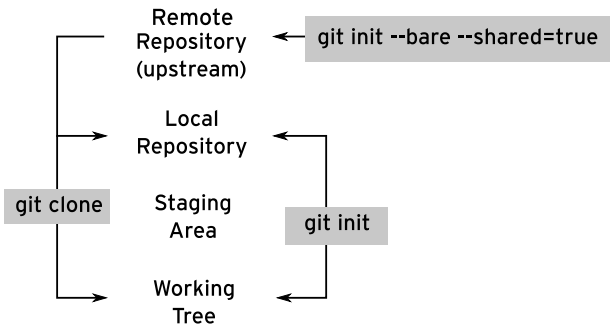

Figure 4.2: Git subcommands used to create a repository

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

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

If a user is working on two changes at the same time, the files can be organized into two commits for better tracking of changes. One set of changes is staged and committed, and then the rest of the changes are staged and committed.

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

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

The **`git commit`** command commits the staged files to the local Git repository. A log message must be provided that explains why the current set of staged files is being saved. Failure to provide a message will abort the commit. Log messages do not have to be long, but they should be meaningful so that they are useful.

Important

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

The **`git commit -a`** command stages and commits *modified* files in one step. However, that command does not include any *untracked* (newly created) files in the directory. When you add a new file, you must explicitly **`git add`** that file to stage it the first time so that it is tracked for future **`git commit -a`** commands.

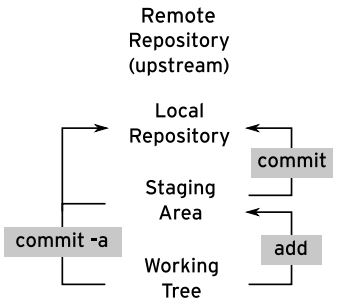

Figure 4.3: Git subcommands that add/update local repository content

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

Before Git pushes will work, the default push method must be defined. The following command sets the default push method to the **`simple`** method. This is the safest option for beginners.

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

The **`git pull`** command fetches commits from the remote repository and to the local repository. It also merges the changes into the files into your working tree.

This command should be run frequently to stay current with the changes that others are making to the project in the remote repository.

Note

An alternative approach to get commits from the remote repository is to use **`git fetch`** to download the changes in the remote repository into a special *tracking branch* of your local repository, then to use **`git merge tracking-branch`** to merge the changes in the tracking branch to your current branch. There are a number of advantages and disadvantages to this approach.

In order to keep this discussion simple, this lesson defers discussing branches in depth, despite their importance in Git, and focus on the **`git pull`** approach.

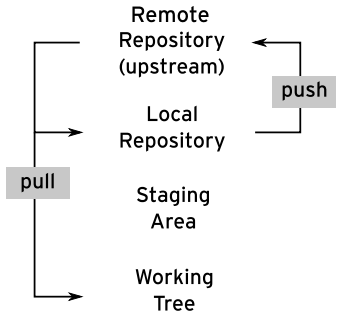

Figure 4.4: Git subcommands that interact with a remote repository

Examining the Git Log

Part of the point of a version control system is to track a history of commits. Each commit is identified by a commit hash. The **`git log`** command displays the commit log messages with the associated ID hashes for each commit. The **`git show`** ***`commit-hash`*** command shows what was in the change set for a particular *commit-hash*. The entire hash doesn't need to be entered with the command, only enough of it to uniquely identify a particular commit in the repository. These hashes can also be used to revert to earlier commits or otherwise explore the version control system's history.

Git Quick Reference

Command	Description
`git clone` ***`URL`***	Clone an existing Git project from the remote repository at *URL* into the current directory.
`git status`	Display the status of modified and staged files in a working tree.
`git add` ***`file`***	Stage a file for the next commit.
`git rm` ***`file`***	Stage removal of a file for the next commit.
`git reset`	Unstage files for the next commit (the opposite of **`git add`**).
`git commit`	Commit the staged files to the local repository with a descriptive message.
`git push`	Push changes in the local repository to the remote repository.
`git pull`	Fetch updates from the remote repository to the local repository and merge them into the working tree.

Important

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

- The local repository was cloned from a remote repository
- Only one local branch is in use.
- The local branch is configured to fetch from and push to a branch on the original remote repository
- Write access is provided to the remote repository, such that **`git push`** works.

Links to more detailed information and tutorials on how to use Git are available in the **References** at the end of this section.

Structuring Ansible Projects in Git

It's highly recommended that Ansible playbooks, roles, and any other details used by an Ansible Tower project are stored in a version control system like Git. This provides an easy way for the administrative team to collaborate on playbook development and share their work with other teams. In addition, the version control system itself provides a record of changes that can be used as an audit trail.

Each project should have its own Git repository. However, projects should not assume they can import roles or content from other projects. One way to import roles which are used by multiple projects would be to have each project use Git submodules (see **`git-submodule`**(1) for more information). This also helps limit the size of the download when a playbook is updated, and helps contributors avoid confusing the code for different projects.

The basic structure of the files in that repository should follow the recommended practices for Ansible documented at *http://docs.ansible.com/ansible/playbooks_best_practices.html*. For example, the directory structure might look something like this:

```
library/             # for custom modules (optional)
filter_plugins/      # for custom filter plugins (optional)

site.yml             # MASTER PLAYBOOK includes other playbooks
webservers.yml       # playbook for webserver tier
dbservers.yml        # playbook for dbserver tier

roles/               # directory for roles
  webserver/            # a particular role
    tasks/
      main.yml            # tasks for the role, can include other files
    defaults/
      main.yml            # default low-priority variables for the role
    templates/
      httpd.conf.j2       # a Jinja2 template used by the role
    files/
      motd                # a file used by the role
    handlers/
      main.yml            # handlers used by the role
    meta/
      main.yml            # role information and dependencies
  ...additional roles...
```

Instead of using **`host_vars`** and **`group_vars`** directories in the project, a better practice would be to store variables with the Inventory object in Tower.

Important

Playbooks must not use the **`vars_prompt`** feature to set variables interactively, because it will not work with Ansible Tower. Instead, you should use Tower's Survey functionality, as discussed later in this course.

Ansible Tower grants Users access to Ansible projects stored in version control systems through the use of Credentials. These credentials allow access to Git SCM sources at a repository level. How these are configured is covered in the next section.

Administrators need to keep in mind that Tower Users have all-or-nothing access to Git repositories. When granted Tower Credentials to access a Git repository, a Tower User will be able to access all of the repository's contents. Therefore, only the playbooks and roles which are to be shared with an intended audience should be kept in the same Git repository.

References

gittutorial(7) and **git**(1) man pages

> Git
> https://git-scm.com

> *Pro Git* by Scott Chacon and Ben Straub (free book)
> https://git-scm.com/book/en/v2

Further information is available in the *GitHub Getting Started Tutorial* at
> http://try.github.io

A useful hands-on tool for learning about branching and merging in Git is at
> http://learngitbranching.js.org

Further information on recommended practices is found in the Best Practices section of the *Ansible Tower User Guide* for Ansible Tower 3.1.1 at
> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

> Best Practices — Ansible Documentation
> http://docs.ansible.com/ansible/playbooks_best_practices.html

Guided Exercise: Managing Ansible Project Materials Using Git

In this exercise, you will configure and use Git to clone a remote repository. You will use the basic **git** commands to make some modifications to and perform verification on files in the resulting local repository. Finally, you will push those changes to the remote repository.

Outcomes
You should be able to use basic **git** commands to manage a version controlled project with Ansible Tower.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab provision-git setup**. This setup script installs the package for Git and create the Git repository needed for the exercise.

```
[student@workstation ~]$ lab provision-git setup
```

Steps

1. Perform basic configuration of Git by setting the following user name, email address, and default push method using **git config**.

```
[student@workstation ~]$ git config --global user.name 'Daniel George'
[student@workstation ~]$ git config --global user.email daniel@lab.example.com
[student@workstation ~]$ git config --global push.default simple
```

2. Check your configuration using **git config**. The output should resemble the following:

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

3. Create and change directory to a new directory called **git-repos**, where you will store your Git repositories.

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

4. Clone and examine the repository called **my_webservers_DEV**.

   4.1. Clone the repository using **git clone**. This creates a directory called **my_webservers_DEV** in your current directory. This new directory contains a playbook intended to setup an Apache web server.

```
[student@workstation git-repos]$ git clone
 git@git.lab.example.com:my_webservers_DEV.git
Cloning into 'my_webservers_DEV'...
```

```
Warning: Permanently added 'git.lab.example.com' (ECDSA) to the list of known
 hosts.
remote: Counting objects: 27, done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 27 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (27/27), done.
Resolving deltas: 100% (5/5), done.
```

4.2. Change directory to the new directory. This is the root directory of the Git repository.

```
[student@workstation git-repos]$ cd my_webservers_DEV
```

4.3. Take a look at the files in that repository using the **tree** command. The **apache-setup.yml** file is the playbook. There are two Jinja2 templates that the playbook uses to build static files. The **httpd.conf.j2** template is used to generate the Apache server configuration. The **index.html.j2** template is used to generate a basic index page to be served by the server.

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

5. Edit the **index.html.j2** template so it displays **HELLO WORLD** at the bottom of the page and then verify your modification.

5.1. Using the **vi** editor, open the **templates/index.html.j2** template file for editing.

```
[student@workstation my_webservers_DEV]$ vi templates/index.html.j2
```

5.2. Append the string **HELLO WORLD** to the end of the file. The **index.html.j2** file should look like this after the modification:

```
{{ apache_test_message }} {{ ansible_distribution }}
 {{ ansible_distribution_version }} <br>
Current Host: {{ ansible_hostname }} <br>
Server list: <br>
{% for host in groups['all'] %}
{{ host }} <br>
{% endfor %}
HELLO WORLD <br>
```

5.3. Save the changes made to the file and then exit from the **vi** editor.

5.4. Use **git status** to check the status of your modifications. Git shows that you have unstaged changes.

```
[student@workstation my_webservers_DEV]$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
```

```
#
#   modified:   templates/index.html.j2
#
no changes added to commit (use "git add" and/or "git commit -a")
```

6. Add the template to the staging area and check the status of your modifications.

   6.1. Use **`git add`** to add the template to the staging area.

```
[student@workstation my_webservers_DEV]$ git add templates/index.html.j2
```

   6.2. Use **`git status`** to check the status of your modifications. Git shows that you have modifications in the staging area which are ready to be committed.

```
[student@workstation my_webservers_DEV]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   templates/index.html.j2
#
```

7. Commit your changes and check the status of your modifications.

   7.1. Use **`git commit`** with a comment of "**`My first commit`**".

```
[student@workstation my_webservers_DEV]$ git commit -m "My first commit"
[master 918ceb7] My first commit
 1 file changed, 1 insertion(+)
```

   7.2. Use **`git status`** to check the status of your modifications. Git shows that there are no more modifications to be either staged or committed on the local repository.

```
[student@workstation my_webservers_DEV]$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

8. Push the changes to the remote repository using **`git push`**.

```
[student@workstation my_webservers_DEV]$ git push
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 427 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To git@git.lab.example.com:my_webservers_DEV.git
   c414fe9..918ceb7  master -> master
```

9. Change directory to the parent directory and clone the repository into a new directory called **`my_webservers_DEV_2`** to verify the changes have been pushed to the remote repository.

```
[student@workstation my_webservers_DEV]$ cd ..
[student@workstation git-repos]$ git clone
 git@git.lab.example.com:my_webservers_DEV.git my_webservers_DEV_2
```

10. Verify that **HELLO WORLD** appears in the template.

```
[student@workstation git-repos]$ grep "HELLO WORLD" my_webservers_DEV_2/templates/
index.html.j2
HELLO WORLD <br>
```

11. Make a change to the remote repository from the second clone.

    11.1. Change directory to the **my_webservers_DEV_2** clone directory.

```
[student@workstation git-repos]$ cd my_webservers_DEV_2
```

    11.2. Using a text editor, remove the **HELLO WORLD** line from the **templates/index.html.j2** file.

    11.3. Add the template to the staging area using **git add**.

```
[student@workstation my_webservers_DEV_2]$ git add templates/index.html.j2
```

    11.4. Commit the changes using **git commit** with the comment **Removed HELLO WORLD**.

```
[student@workstation my_webservers_DEV_2]$ git commit -m "Removed HELLO WORLD"
[master 3a4cb00] Removed HELLO WORLD
 1 file changed, 1 deletion(-)
```

    11.5. Push the changes to the remote repository.

```
[student@workstation my_webservers_DEV_2]$ git push
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 414 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 3 (delta 1)
To git@git.lab.example.com:my_webservers_DEV.git
   918ceb7..3a4cb00  master -> master
```

12. Use **git pull** to pull the latest changes from the remote repository.

    12.1. Change directory to the first clone of the repository, **my_webservers_DEV**.

```
[student@workstation my_webservers_DEV_2]$ cd ../my_webservers_DEV
```

    12.2. Verify that the **HELLO WORLD** line is present in the **index.html.j2** file.

```
[student@workstation my_webservers_DEV]$ grep "HELLO WORLD" templates/
index.html.j2
```

```
HELLO WORLD <br>
```

12.3. Pull the latest changes using **`git pull`**.

```
[student@workstation my_webservers_DEV]$ git pull
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From git.lab.example.com:my_webservers_DEV
   918ceb7..3a4cb00  master     -> origin/master
Updating 918ceb7..3a4cb00
Fast-forward
 templates/index.html.j2 | 1 -
 1 file changed, 1 deletion(-)
```

12.4. Verify that the **`HELLO WORLD`** line is no longer present in the **`index.html.j2`** file.

```
[student@workstation my_webservers_DEV]$ grep "HELLO WORLD" templates/
index.html.j2
[student@workstation my_webservers_DEV]$
```

Creating a Project for Ansible Playbooks and Roles

Objectives

After completing this section, students should be able to:

- Create a Project which gets Ansible materials from a Git repository using an SCM credential configured with an SSH private key
- Configure Project permissions so a Team can use and update it
- Explain the differences between the Admin, Use, Update, and Read roles for projects

Projects

An Ansible project represents at least one playbook and its associated collection of related playbooks and roles. Whether or not Ansible Tower is being used, it is a good practice for these materials to be managed together in a version control system. The design of Ansible Tower assumes that most Ansible projects are managed in a version control system for this reason, and can automatically retrieve updated materials for a project from several commonly used version control systems.

In the Ansible Tower web interface, each Ansible project is represented by a Project resource. The Project is configured to retrieve these materials from a version control system (also referred to by Tower as a *source control management* or SCM system). Tower supports the ability to download and automatically get updates of project materials from SCMs using Git, Subversion, or Mercurial.

Note

It is possible to simply copy projects to the Tower server in a location known as the *Project Base Path*. Configured by **`/etc/tower/settings.py`**, this directory is located by default at **`/var/lib/awx/projects`**.

This is not a recommended practice. Updating the projects then requires manual intervention outside of the Tower interfaces. It also requires that project administrators have direct access to make changes in the operating system environment on the Tower, which reduces the security of the Tower server. It is better to have Tower get project materials from an SCM system.

Creating a Project

The following is the procedure for creating a Project to share a collection of Ansible playbooks and roles managed in an existing Git repository. The hands-on exercise following this section covers this in more detail.

1. Log in to the Tower web interface as a User with **`Admin`** role on an Organization.
2. Click the **PROJECTS** quick navigation link to go to the Projects management screen.

3. Click the **+ADD** button to create the new Project.
4. Enter a unique name for the Project in the **NAME** field.
5. Optionally, enter a description for the Project in the **DESCRIPTION** field.
6. Click the magnifying glass icon next to the **ORGANIZATION** field to display a list of Organizations within Tower. Select an Organization from the list and click **SAVE**.
7. In the **SCM TYPE** dropdown list, select the **Git** option.
8. Under the **SOURCE DETAILS** section, enter the location of the Git repository in the **SCM URL** field.
9. Optionally, in the **SCM BRANCH** field, specify the branch of the repository to obtain the contents from.
10. If authentication is required to access the Git repository, click the magnifying glass icon next to the **SCM CREDENTIAL** field to display a list of available SCM Credentials. Select an SCM Credential from the list and click **SAVE**. The creation of SCM Credentials is discussed later in this section.
11. Lastly, select the desired action to be taken to update the Project against its SCM source. The available options are **`Clean`**, **`Delete on Update`**, and **`Update on Launch`**. These three options are discussed in further detail at the end of this section.
12. Click **SAVE** to finalize the creation of the Project.

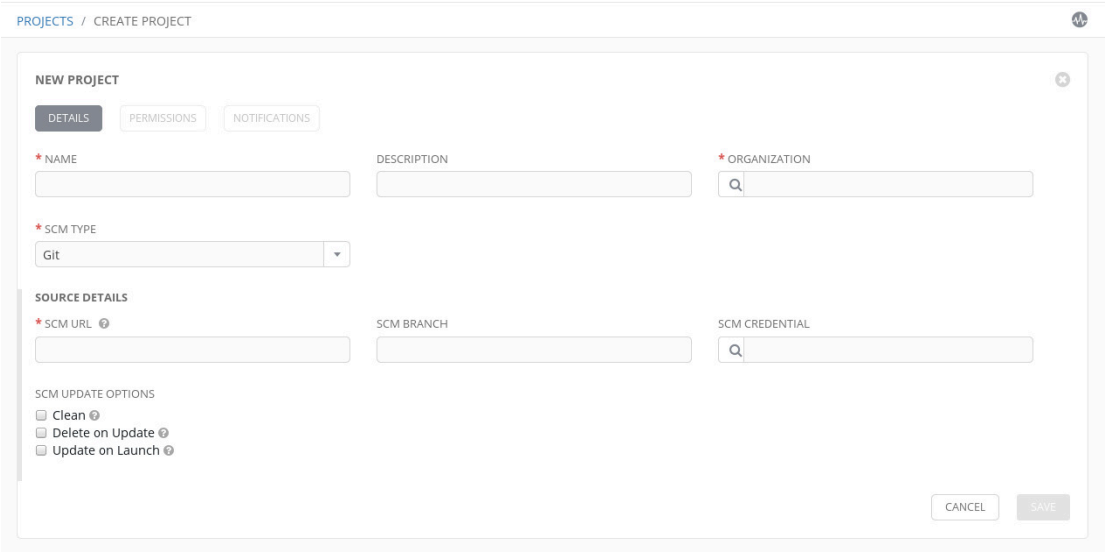

Figure 4.5: New Project

Project Roles

Users are granted permissions to Project resources through being assigned roles on the Project resource. Users can be directly assigned roles, or they can be assigned As always, assignment of roles can be made directly to a User or indirectly through a Team. For example, in order for a User to gain permissions on a specific Project, they must be assigned or inherit a role for that Project.

The following are the list of available Project roles.

Admin
The **Admin** role grants Users full access to a Project. When granted this role on a Project, a User can delete the Project and modify its properties, including its permissions. In addition, this role also grants Users the permissions associated with the Project **Use**, **Update**, and **Read** roles which is discussed later in this section.

Important

Users can only make changes to a Project's properties if they have the **Admin** role on both the Project *and* the Organization to which the Project belongs.

If a User is assigned **Admin** role on a Project but not on the Project's Organization, they are able to modify the Project's permissions but they will not be able to modify any settings on the Project's **DETAILS** page.

Use
The **Use** role grants Users the ability to use a Project in a Template resource. Use of a Project in a Template resource will be discussed in detail in a later section. This role also grants Users the permissions associated with the Project **Read** role.

Update
The **Update** role grants Users the ability to manually update or schedule an update of a Project's materials from its SCM source. This role also grants Users the permissions associated with the Project **Read** role.

Read
The **Read** role grants Users the ability to view the details, permissions, and notifications associated with a Project.

Managing Project Access

When a Project is first created, it is only accessible by Users who have the **Admin** or **Auditor** role in the Project's Organization.

Other access by Users must be specifically configured. Role cannot be assigned when the Project is created, but must be added by editing the Project.

Roles are assigned on the **PERMISSIONS** section of the Project editor screen. The following procedure details the steps to set roles for a Project:

1. Login as a User with **Admin** role for the Organization in which the Project was created.
2. Click **PROJECTS** in the quick navigation links to display the list of Projects.
3. Click the pencil icon for the Project to edit to enter the Project editor screen.
4. On the Project editor screen, click the **PERMISSIONS** button to enter the permissions editor.
5. Click the **+ADD** button to add permissions.
6. In the User and Team selection screen, click either **USERS** or **TEAMS** and then select the users or teams to be granted permissions.

7. Click **KEY** to display the list of Project roles and their definitions.

8. Click the **SELECT ROLES** dropdown list and select the desired Project role for each User or Team.

9. Click **SAVE** to finalize the changes to permissions.

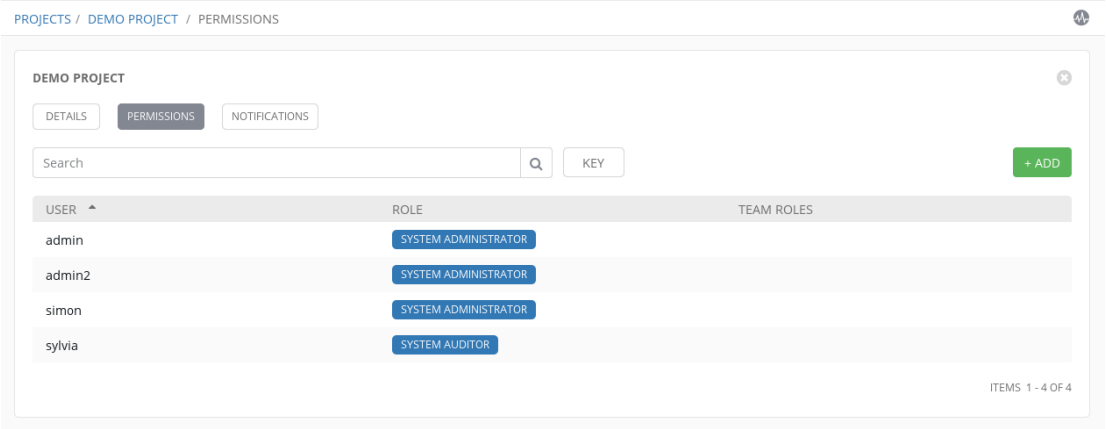

Figure 4.6: Assigning Project Roles to a User

Important

Roles for Projects can also be added through either the User or Team management screens under Tower's Settings interface. However, those interfaces currently do not provide the Project **Read** role as an option for role assignment.

Creating SCM Credentials

A previous section covered the creation of a Machine Credential which stores authentication information used by playbooks to connect to managed hosts and perform tasks on them. Another type of Credential that Ansible Tower can manage are *SCM Credentials* which authenticate to a version control system.

SCM Credentials are used by Ansible Tower to authenticate to Source Control Management (version control) systems in order to update the Ansible project materials used by Projects. SCM Credentials can be used to store user name and password or private key (and private key passphrase) information needed to authenticate against an SCM source.

The following procedure creates an SCM Credential which can be used by Tower to retrieve playbooks and roles from a Git repository to a Project resource in Tower. The hands-on exercise following this section covers this in more detail.

1. Log in as a User with the appropriate role assignment:
   - If creating a private SCM Credential, there are no specific role requirements.
   - If creating an SCM Credential belonging to an Organization, log in as a User with **`Admin`** role for the Organization.

2. Click the **Settings** icon and then click the **CREDENTIALS** link to enter the Credentials management interface.

3. On the **CREDENTIAL** screen, click **+ADD** to create a new Credential.

4. On the **CREATE CREDENTIAL** screen, enter the required information for the new Credential.

   4.1. Enter a unique name for the Credential in the **NAME** field.

   4.2. If creating an Organization Credential, click the magnifying glass next to the **ORGANIZATION** field, select the Organization to create the Credential in, and then click **SAVE**. Skip this step if creating a private Credential.

   4.3. Click the **TYPE** dropdown list and select the **Source Control** Credential type.

5. Once a **`Source Control`** Credential type is selected in the previous step, the appropriate fields will appear in the **TYPE DETAILS** section.

   Enter authentication data into their respective fields. For example, you may need to specify a **USERNAME**. If a password is needed, you must put that in the **PASSWORD** field. If you are using an SSH private key to authenticate, either copy and paste or drag and drop your private key into the **SCM PRIVATE KEY** field. That key can be a passphrase encrypted SSH private key, in which case you can provide that passphrase to Tower in the **PRIVATE KEY PASSPHRASE** field.

6. Click the **SAVE** button to finalize the creation of the new SCM Credential.

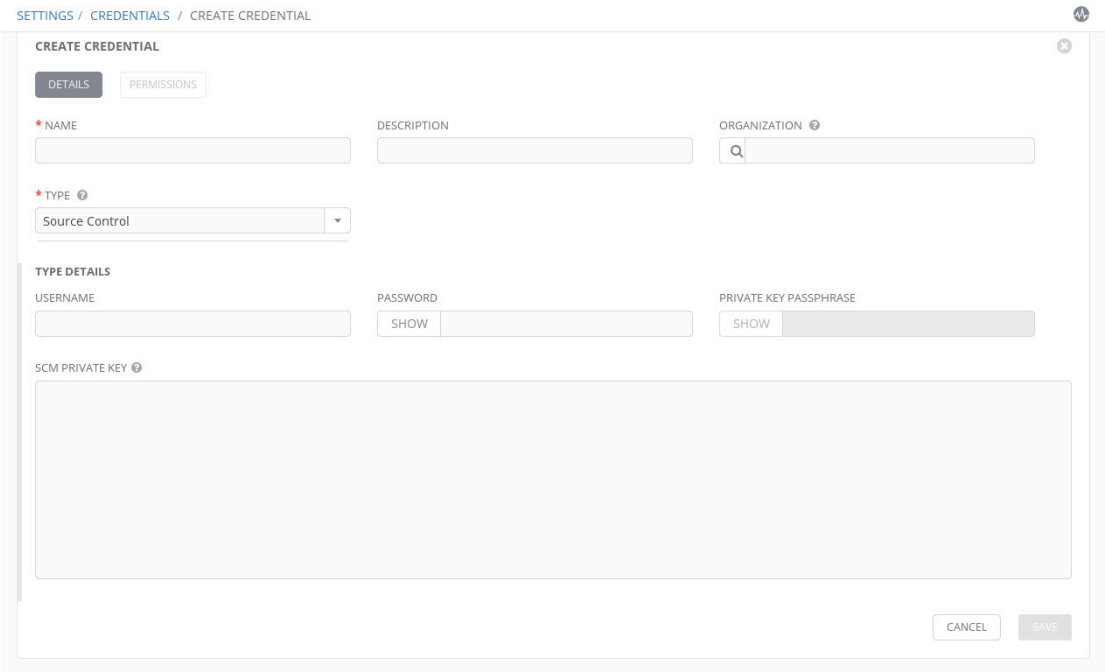

Figure 4.7: New SCM Credential

SCM Credential Roles

Just like Machine Credentials, private SCM Credentials are only usable by their creators and **`System Administrator`** and **`System Auditor`** Users. SCM Credentials assigned to an Organization can be shared with other Users by assigning Users or Teams roles for that credential.

The following is the list of roles available for providing Users access to SCM Credentials.

Admin
The **Admin** role grants Users full permissions over an SCM Credential. These permissions include deletion and modification of the SCM Credential. This role also grants Users permissions associated with the Credential **Use** and **Read** roles.

Use
The **Use** role grants Users the ability to associate an SCM Credential with a Project resource. This role also grants Users the permissions associated with the Credential **Read** role.

Important

The **Use** role does not control whether a User can themselves use the SCM Credential to *update* a Project, only whether they can assign that SCM Credential so that it can then be used by someone who has the **Update** role on the Project.

For example, if an SCM Credential is associated with a Project, any User assigned the **Update** role on the Project can use the associated SCM Credential without being granted **Use** role on the Credential.

Read
The **Read** role grants Users the ability to view the details of an SCM Credential.

Managing Access to SCM Credentials

When an Organization SCM Credential is first created, it is only accessible by Users with either the **Admin** or **Auditor** role in the Organization to which the Credential is assigned. Additional access for other users must be specifically configured.

The assignment of SCM Credential roles to Users or Teams dictates who has permissions to an SCM Credential which belongs to an Organization. These permissions cannot be assigned at the time of the SCM Credential's creation. They are adjusted after its creation by editing the Credential.

Roles are assigned through the **PERMISSIONS** section of the Credential editor screen. The following procedure details the steps for granting permissions to an SCM Credential which has been assigned to an Organization, after the credential's creation.

1. Log in as a User with **Admin** role on the Organization that the SCM Credential belongs to.
2. Click the **Settings** icon and then click the **CREDENTIALS** link to enter the Credentials management interface.
3. Click the name of the SCM Credential to edit in order to enter the Credential editor screen.
4. On the Credential editor screen, click the **PERMISSIONS** button to enter the permissions editor.
5. Click the **+ADD** button to add permissions.
6. In the User and Team selection screen, click either **USERS** or **TEAMS** and then select the users or teams to be granted permissions.

7. Click **KEY** to display the list of Credential roles and their definitions.

8. Click the **SELECT ROLES** dropdown list and select the desired Credential role for each User or Team.

9. Click **SAVE** to finalize the changes to permissions.

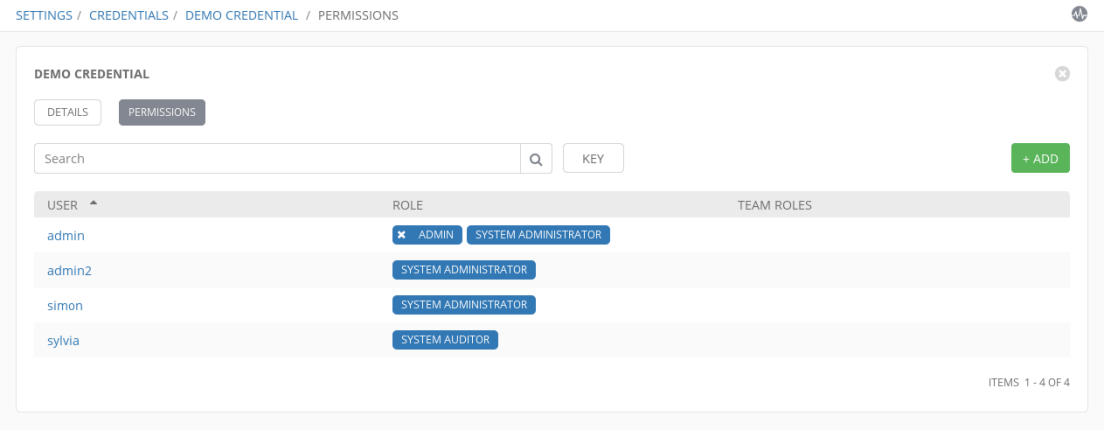

Figure 4.8: Assigning SCM Credential Role to a User

Important

Permissions for SCM Credentials can also be added through either the User or Team management screens under Tower's Settings interface. However, those interfaces currently do not provide the Credential **Read** role as an option for role assignment.

Updating Projects

An SCM Project resource in Tower represents a copy of playbooks and roles obtained from an SCM source. Since modifications to the contents of these playbooks and roles are managed in an external SCM system, their respective counterparts in a Tower Project will need to be updated routinely from the SCM source in order to reflect new changes.

There are several ways to update SCM Project resources in Tower. As previously mentioned, Projects can be configured to update from their SCM sources by choosing one of three SCM update options in the Project's detail screen.

Clean
This SCM update option removes local modifications before performing an update against the SCM source.

Delete on Update
This SCM update option completely removes the local Project repository on Tower before performing an update against the SCM source.

Update on Launch
This SCM update option performs an update against the SCM source prior to each use of the Project in a job run.

In addition to the updates performed as a result of the SCM update option setting in a Project's configuration, a Project can also be manually triggered to update its contents against its SCM source. The following procedure details the steps for manually updating an Project from its SCM source.

1. Log in as a User with **`Update`** role on the Project.
2. Click the **PROJECTS** quick navigation link to enter the Projects management interface.
3. In the table of Projects, a cloud icon will appear under the **ACTIONS** column if the User has **`Update`** role for a given Project.
4. To trigger a manual, immediate update on a Project, click on its cloud icon to initiate an update of its contents against its SCM source.

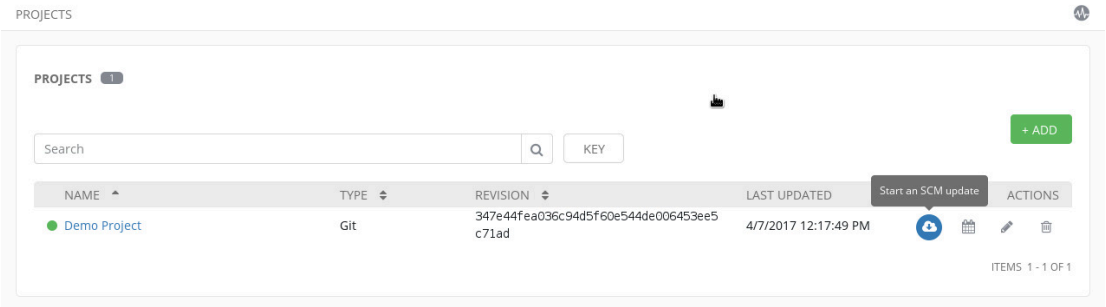

Figure 4.9: Updating SCM Project

Note

The **`Update`** role only dictates whether a User can manually trigger an update of a Project against its SCM source. It does not impact the update behavior configured by the Project's SCM update option. For example, a Project configured with an **`Update on Launch`** SCM update option will still perform updates even when the Project is used by a User who has not been granted the **`Update`** role on the Project.

References

Ansible Tower User Guide for Ansible Tower 3.1.1

http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Creating a Project for Ansible Playbooks and Roles

In this exercise, you will create a new SCM Credential, a new Project, and assign an appropriate role for a Team to be able to use that Project.

Outcomes
You should be able to create a Project which will allow Ansible Tower to leverage an external Git repository containing a playbook.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab provision-project setup**. This setup script ensures that the **workstation** and **tower** virtual machines are started.

```
[student@workstation ~]$ lab provision-project setup
```

Steps

1. Log into the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create the new SCM Credential that will be needed in order to create a new Project.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the Administration tool links.

   2.2. Click **CREDENTIALS** to manage Credentials.

   2.3. Click the **+ADD** button to add a new Credential.

   2.4. On the next screen, fill in the details as follows:

Field	**Value**
NAME	student-git
DESCRIPTION	Student Git Credential
ORGANIZATION	Default
TYPE	Source Control
USERNAME	**git**
SCM PRIVATE KEY	Copy the contents of the **/home/student/.ssh/lab_rsa** private key file on **workstation** into this field

   2.5. Leave the other fields untouched and click **SAVE** to create the new Credential.

3. Create a new Project called "**My Webservers DEV**".

   3.1. Click the **PROJECTS** quick navigation link.

3.2. Click the **+ADD** button to add a new Project.

3.3. On the next screen, fill in the details as follows:

Field	Value
NAME	My Webservers DEV
DESCRIPTION	Development Webservers Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	ssh://git.lab.example.com/home/git/my_webservers_DEV.git
SCM CREDENTIAL	student-git

3.4. Click **SAVE** to create the new Project. This automatically triggers the SCM update of the Project. Ansible Tower uses the values provided in the **SCM URL** and **SCM CREDENTIAL** fields to pull down a local copy of that repository.

4. Observe the automatic SCM update of the project **`My Webservers DEV`**.

4.1. Scroll down the page and wait a couple of seconds. In the list of Projects, there is a status icon left of the Project, **`My Webservers DEV`**. This icon is white at the start, red with an exclamation mark when it fails, and green when it succeeds.

4.2. Click on the status icon to show the detailed status page of the SCM update job. As you can see in the **STANDARD OUT** window, the SCM update job runs like any other Ansible playbook.

4.3. Verify that the **STATUS** of the job in the **RESULTS** section shows **`Successful`**.

5. Give the **`Developers`** Team **`Admin`** role on the Project, **`My Webservers DEV`**.

5.1. Click the **PROJECTS** quick navigation link.

5.2. On the same line as the Project, **`My Webservers DEV`**, click the pencil icon on the right to edit the Project.

5.3. On the next page, click **PERMISSIONS** to manage the Project's permissions.

5.4. Click the **+ADD** button on the right to add permissions.

5.5. Click **TEAMS** to display the list of available Teams.

5.6. In the first section, check the box next to the **`Developers`** Team. This causes the Team to display in the second section underneath the first one.

5.7. In the second section below, select the **`Admin`** role from the drop-down list.

5.8. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Project, **My Webservers DEV**, which now shows that all members of the **Developers** Team are assigned the **Admin** role on the Project.

Creating Job Templates and Launching Jobs

Objectives

After completing this section, students should be able to create a Job Template from a Project to run a playbook, launch a Job using that template and be able to interpret whether the job succeeded or had failures.

Job Templates, Projects, and Inventories

In Tower, Projects represent collections of playbooks while Inventories represent collections of managed hosts that playbooks can be run on. The contents of both Projects and Inventories are made available to Users through the assignment of roles. When granted the **`Use`** role, Users can associate Projects and Inventories with a Job Template resource in Tower.

A Job Template defines the parameters for the execution of an Ansible job. An Ansible job executes a playbook against a set of managed hosts. Therefore, a Job Template must define what Project provides the playbook and what Inventory contains the list of managed hosts.

Additionally, the Job Template must also define the Machine Credential which will be used to authenticate to the managed hosts. Like Projects and Inventories, a User must have Use role assigned to a Machine Credential, to be able to associate it to a Job Template.

Once defined, a Job Template allows for the repeated execution of a job and is therefore ideal for routine execution of tasks. Since the Project, Inventory, and Machine Credential parameters are part of the Job Template definition, the job is ensured to run the same way each time.

Creating Job Templates

Unlike other Tower resources, Job Templates are not created directly in the context of an Organization but rather in the context of a Project within an Organization. A Job Template's indirect relationship to an Organization is determined by the Project that the Job Template is associated with. Therefore, unlike other resources which can only be created by a User with **`Admin`** role on an Organization, any User can create a Job Template as long as they have been granted the **`Use`** role on a Project.

Since a Job Template has to be defined with an Inventory, Project, and Machine Credential, a User can only create a Job Template if they have Use roles assigned to one or more of each of these three Tower resources. The following procedure details how to create a Job Template for running a playbook on a set of managed hosts with a focus on just the mandatory parameters. Optional parameters are discussed later.

1. Log in to the Tower web interface as a User who has been assigned the **`Use`** role for the Inventory, Project, and Machine Credential resources, which will be used to define the new Job Template.

2. Click the **TEMPLATES** quick navigation link to go to the Templates management interface.

3. Click the **+ADD** button and then select **Job Template**.

4. Enter a name for the Job Template in the **NAME** field.

5. Select **Run** as the **JOB TYPE**.

6. Specify the managed hosts that the job will be executed against.
   6.1. Click the magnifying glass icon for the **INVENTORY** field.
   6.2. Select the desired Inventory and then click **SAVE**.
7. Specify the Project which contains the playbook that the job will execute.
   7.1. Click the magnifying glass icon for the **PROJECT** field.
   7.2. Select the desired Project and then click **SAVE**.
8. Specify the playbook that the job will execute.
   8.1. Click the dropdown list for the **PLAYBOOK** field. The list contains all playbooks contained in the Project selected in the previous field.
   8.2. Select the desired playbook.
9. Specify the Machine Credential which should be used to authenticate against the managed hosts.
   9.1. Click the magnifying glass icon for the **MACHINE CREDENTIAL** field.
   9.2. Select the desired Credential and then click **SAVE**.
10. Select the desired setting from the drop-down list for the **VERBOSITY** field. This determines the level of detail that is generated in the output of the job run.
11. Click **SAVE** to finalize creation of the new Job Template.

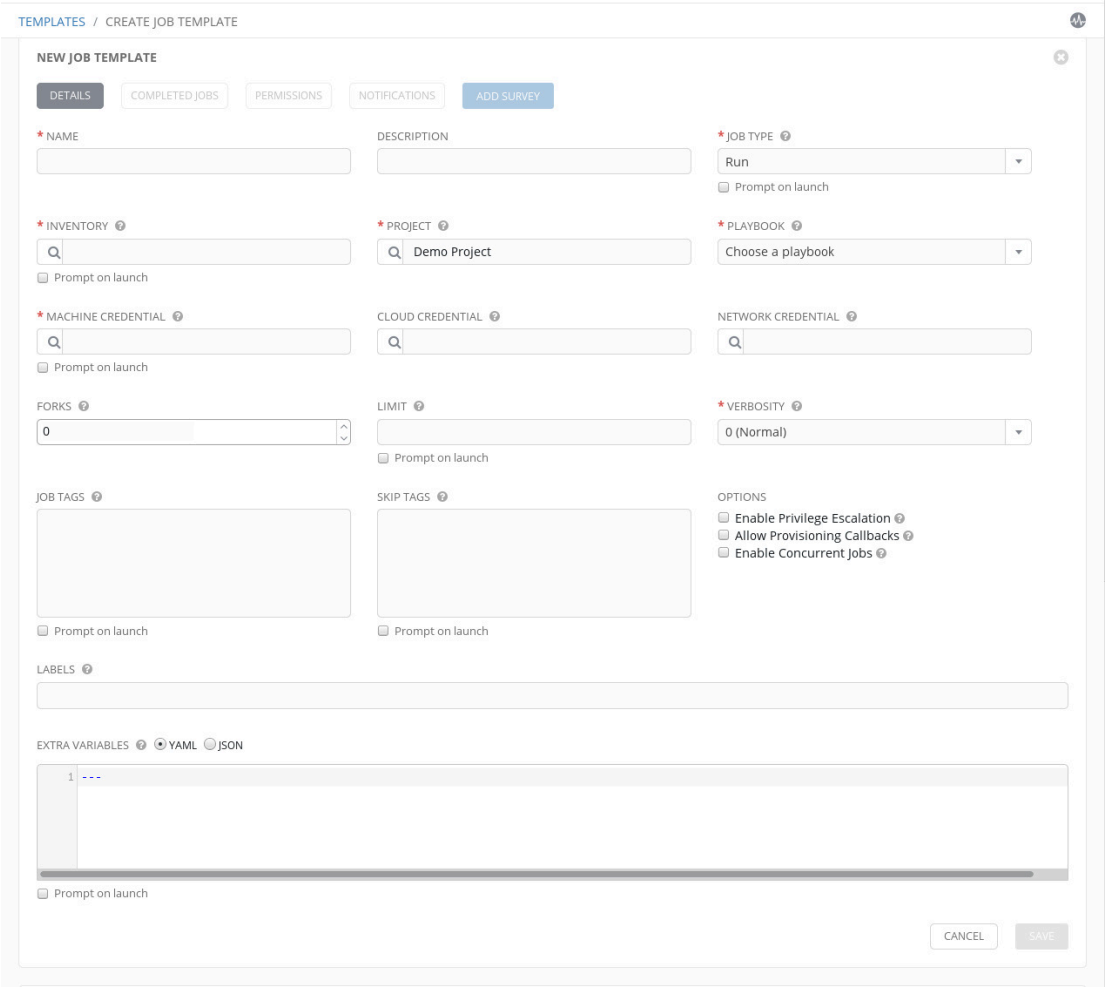

Figure 4.10: New Job Template

Modifying Job Execution

Besides the required parameters discussed previously, a Job Template has various other settings which can be used to configure how job runs are executed.

DESCRIPTION

This field is used to store an optional description of the Job Template.

CLOUD CREDENTIAL

When the playbook used in a Job Template uses a cloud module which require authentication, the Cloud Credential to use is specified in this field.

NETWORK CREDENTIAL

When the playbook used in a Job Template uses a network module which requires authentication, the Network Credential to use is specified in this field.

FORKS

This field is used to specify the number of parallel processes to allow during playbook execution. This is the equivalent of the **`-f`** or **`--fork`** option for the **`ansible-playbook`** command. A value of **`0`** causes the default setting from the Ansible configuration file to be used.

LIMIT
This field is used to further restrict the selection of managed hosts from the complete list in the selected Inventory. Filtering is accomplished by supplying a host pattern as a value for this field. This is the equivalent of the **-l** or **--limit** option for the **ansible-playbook** command.

JOB TAGS
This field accepts a comma separated list of tags which exist in a playbook. Tags are used to identify distinct portions of a playbook. By specifying a list of tags in this field, one can selectively execute only certain portions of a playbook. This is the equivalent of the **-t** or **--tags** option for the **ansible-playbook** command.

SKIP TAGS
This field accepts a comma separated list of tags which exist in a playbook. By specifying a list of tags in this field, one can selectively skip certain portions of a playbook during its execution. This is the equivalent of the **--skip-tags** option for the **ansible-playbook** command.

Enable Privilege Escalation
When enabled, this option causes the playbook to be executed as an administrator. This is the equivalent of the **--become** option for the **ansible-playbook** command.

Allow Provisioning Callbacks
When enabled, this option results in the creation of a provisioning callback URL on Tower which can be used by hosts to request a configuration update using the Job Template.

Enable Privilege Escalation
When enabled, this option allows for multiple, simultaneous executions of this Job Template.

LABELS
Labels are names which can be attached to Job Templates to facilitate grouping or filtering of Job Templates.

EXTRA VARIABLES
Equivalent to the **-e** or **--extra-vars** options for the **ansible-playbook** command, this option can be used to pass extra command line variables to the playbook executed by a job. These extra variables are defined as key/value pairs using either YAML or JSON.

Prompting for Job Parameters

When executing playbooks from the command line using the **ansible-playbook** command, administrators have the flexibility of modifying the playbook execution through the use of command line options. Ansible Tower allows for some of this flexibility by allowing certain parameters in Job Templates to prompt for user input at the time of job execution. This *Prompt on launch* option is available for the following parameters.

- JOB TYPE
- INVENTORY
- MACHINE CREDENTIAL
- LIMIT
- JOB TAGS

- SKIP TAGS
- EXTRA VARIABLES

The flexibility to change job parameters at the time of job execution encourages playbook reuse. For example, rather than creating multiple job templates to run the same playbook on different sets of managed hosts, one only needs a single job template which has the **`Prompt on launch`** option enabled for the Inventory field. When the job is launched, the User executing the job is given the option to specify an Inventory to execute the playbook on. When prompted, Users are only able to select from Inventories that they have been assigned the **`Use`** role on.

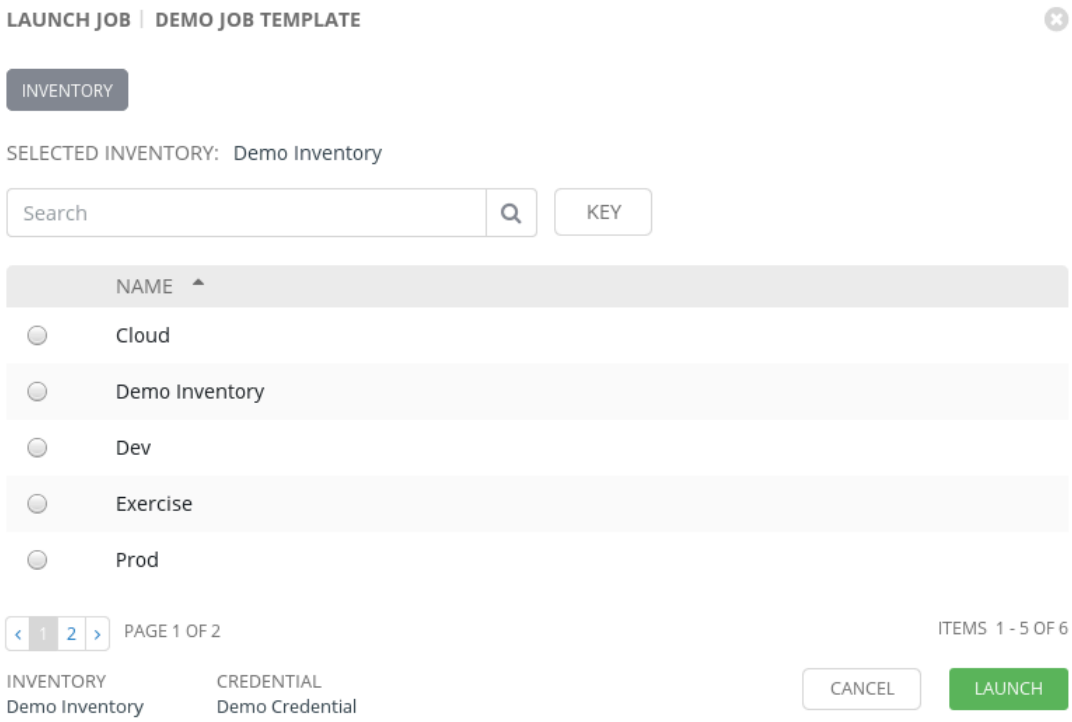

Figure 4.11: Prompting for Inventory on Launch

Job Templates Roles

There are three roles which are used to control User access to Job Templates.

Admin

The **`Admin`** role provides a User the ability to delete a Job Template or edit its properties, including its associated permissions. This role also grants permissions associated with the Job Template **`Execute`** and **`Read`** roles.

Important

The ability to copy a Job Template is not granted by the **`Admin`** role on a Job Template. Permissions to copy a Job Template is only granted to Users who have been granted **`Use`** role on the Project associated with the Job Template being copied.

Execute
The **Execute** role grants Users permission to execute a job using the Job Template. It also grants the Users permission to schedule a job using the Job Template. This role also grants permissions associated with the Job Template **Read** role.

Important

A Job Template makes use of other Tower resources such as Projects, Inventories, and Credentials. For a User to execute a job using a Job Template, they only need to be assigned the **Execute** role on the Job Template and do not need to be assigned **Use** roles to any of these associated Tower resources.

Read
The **Read** role grants Users read only access to view the properties of a Job Template. It also grants access to view other information related to the Job Template, such as the list of jobs executed using the Job Template, as well as its associated permissions and notifications.

Managing Job Template Access

When a Job Template is first created, it is only accessible by the User that created it or Users with either an **Admin** or **Auditor** role in the Organization that the Project was created in. Additional access must be specifically configured if desired.

The assignment of the previously discussed Job Template roles to Users or Teams dictates who has permissions to a Job Template. These permissions cannot be assigned at the time of a Job Template's creation. They are administered after a Job Template has been created by editing the Job Template.

Roles are assigned through the **PERMISSIONS** section of the Job Template editor screen. The following procedure details the steps for granting permissions to a Job Template after its creation.

1. Log in as a User with **Admin** role on the Organization that the Job Template is associated with or as the User who created the Job Template.
2. Click **TEMPLATES** in the quick navigation links to display the list of templates.
3. Click the pencil icon for the Job Template to edit to enter the Job Template editor screen.
4. On the Job Template editor screen, click the **PERMISSIONS** button to enter the permissions editor.
5. Click the **+ADD** button to add permissions.
6. In the User and Team selection screen, click either **USERS** or **TEAMS** and then select the users or teams to be granted permissions.
7. Click **KEY** to display the list of Job Template roles and their definitions.
8. Click the **SELECT ROLES** dropdown list and select the desired Job Template role for each User or Team.
9. Click **SAVE** to finalize the changes to permissions.

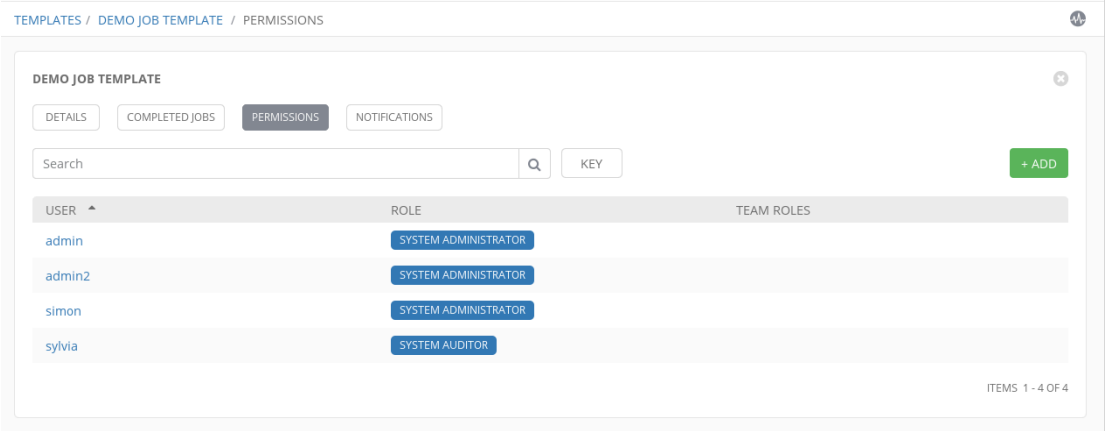

Figure 4.12: Assigning Job Template Role to a User

Launching Jobs

Once a Job Template is created, it can be used to launch a job with the following procedure.

1. Login as a User that possesses the **`Execute`** role on the desired Job Template.
2. Click the **TEMPLATES** quick navigation link to see the list of templates.
3. Locate the desired Job Template in the list of templates and click the rocket icon under the **ACTIONS** column to launch the job.
4. If any of the Job Template parameters have the **`Prompt on launch`** option enabled, you are prompted for input prior to the job execution. Enter the desired input for each parameter prompted for and the click **LAUNCH** to launch the job.

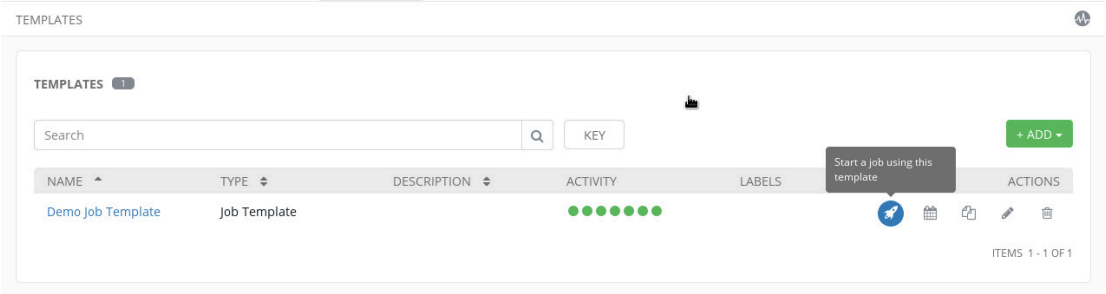

Figure 4.13: Launching a Job

Evaluating Job Run Results

Once a job run has been launched from a Job Template in the Tower web interface, the User will automatically be redirected to the job's detail page. Users can also navigate to this same page by clicking the **JOBS** quick navigation link to see the list of executed jobs and then clicking the link for the job of interest.

The job detail page is divided into two panes. The **DETAILS** pane displays the details of the job's parameters while the job output pane displays the output of the playbook executed by the job.

While the output in the job output pane resembles that which would have been generated by the execution of the playbook on the command line using the **`ansible-playbook`** command,

it also offers several additional features. Across the top of the job output pane is a summary detailing the number of plays and tasks which were executed, the count of hosts which the job was executed against, and also the time it took for the job to execute. Additionally, controls are provided for maximizing this pane to full screen size, as well as for downloading the output of the job execution.

Along the left side of the output section, the **+** and **-** controls can be used to expand or collapse the output for each task in the playbook. Along the right side of the output section are controls for scrolling through the output as well as for jumping to the beginning and end of the output.

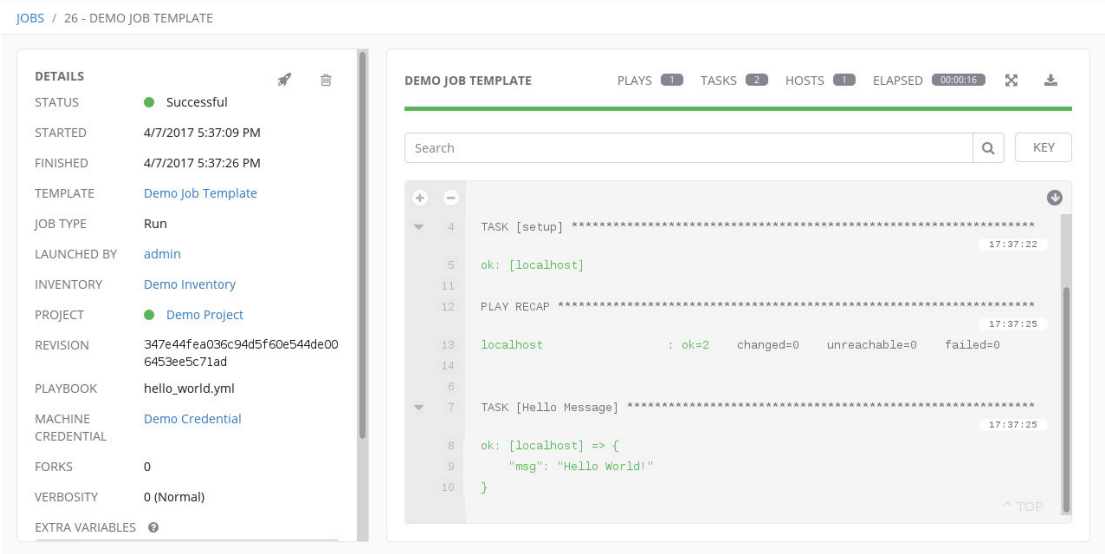

Figure 4.14: Job Run Results

References

Ansible Tower User Guide for Ansible Tower 3.1.1

http://docs.ansible.com/ansible-tower/3.1.1/html/userguide

Guided Exercise: Creating Job Templates and Launching Jobs

In this exercise, you will create a Job Template, assign an appropriate role for a Team to be able to use that Job Template, and launch a Job.

Outcomes
You should be able to create a Job Template and launch a Job from the Tower web interface.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab provision-job setup**. This setup script ensures that the **workstation** and **tower** virtual machines are started.

```
[student@workstation ~]$ lab provision-job setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a new Job Template called **DEV webservers setup**.

   2.1. Click the **TEMPLATES** quick navigation link

   2.2. Click the **+ADD** button to add a new Job Template.

   2.3. From the drop-down list, select **Job Template**.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	DEV webservers setup
DESCRIPTION	Setup apache on DEV webservers
JOB TYPE	Run
INVENTORY	Dev
PROJECT	My Webservers DEV
PLAYBOOK	apache-setup.yml
MACHINE CREDENTIAL	Developers

   2.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

3. Give the **Developers** Team **Admin** role on the Job Template, **DEV webservers setup**.

   3.1. Click **PERMISSIONS** to manage the Job Template's permissions.

   3.2. Click the **+ADD** button on the right to add permissions.

3.3. Click **TEAMS** to display the list of available Teams.

3.4. In the first section, check the box next to **Developers** Team. This causes the Team to display in the second section underneath the first one.

3.5. In the second section below, select the **Admin** role from the drop-down list.

3.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Job Template, **DEV webservers setup**, which now shows that all members of the **Developers** Team are assigned the **Admin** role on the Job Template.

4. Launch a Job using the Job Template, **DEV webservers setup**, as a member of the **Developers** Team.

4.1. Click the **Log Out** icon in the top right corner to logout and log back in as **daniel** with a password of **redhat123**.

4.2. Click the **TEMPLATES** quick navigation link.

4.3. On the same line as the Job Template, **DEV webservers setup**, click the rocket icon on the right to launch the Job. This will redirect you to a detailed status page of the running job.

4.4. Observe the live output of the running job for a minute.

4.5. Verify that the **STATUS** of the Job in the **RESULTS** section displays **Successful**.

5. Verify that the web servers are up and running on **servera.lab.example.com** and **serverb.lab.example.com**.

5.1. Open a web browser and go to **http://servera.lab.example.com**. You should see the following output:

```
This is a test message RedHat 7.3
Current Host: servera
Server list:
servera.lab.example.com
serverb.lab.example.com
```

5.2. Open a web browser and go to **http://serverb.lab.example.com**. You should see the following output:

```
This is a test message RedHat 7.3
Current Host: serverb
Server list:
servera.lab.example.com
serverb.lab.example.com
```

6. Click the **Log Out** icon to log out of the Tower web interface.

Lab: Managing Projects for Provisioning with Ansible Tower

In this exercise, you will create a new Project and Job Template, and assign an appropriate role for a Team to be able to launch a Job.

Outcomes
You should be able to manage Projects and Job Templates in order for a Team to be able to launch a Job.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab provision-review setup**. This setup script creates a new Git repository needed for the exercise.

```
[student@workstation ~]$ lab provision-review setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a new Project called **My Webservers TEST** with the following information:

Field	Value
NAME	My Webservers TEST
DESCRIPTION	Test Webservers Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	ssh://git.lab.example.com/home/git/my_webservers_TEST.git
SCM Credential	student-git

3. Give the **Developers** Team the **Use** role on the Project, **My Webservers TEST**.

4. Create a new Job Template called **TEST webservers setup** with the following information:

Field	Value
NAME	TEST webservers setup
DESCRIPTION	Setup apache on TEST webservers
JOB TYPE	Run
INVENTORY	Test
PROJECT	My Webservers TEST

PLAYBOOK	apache-setup.yml
MACHINE CREDENTIAL	Operations

5. Give the **Developers** Team the **Execute** role on the Job Template, **TEST webservers setup**.

6. Launch a Job using the Job Template, **TEST webservers setup**, as the User, **david**, who is a member of the **Developers** Team.

7. Log out of the Tower web interface and then verify that the web servers are up and running on **serverc.lab.example.com** and **serverd.lab.example.com**.

8. Run the command **lab provision-review grade** on **workstation** to grade your exercise.

Solution

In this exercise, you will create a new Project and Job Template, and assign an appropriate role for a Team to be able to launch a Job.

Outcomes
You should be able to manage Projects and Job Templates in order for a Team to be able to launch a Job.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab provision-review setup**. This setup script creates a new Git repository needed for the exercise.

```
[student@workstation ~]$ lab provision-review setup
```

Steps
1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a new Project called **My Webservers TEST** with the following information:

Field	Value
NAME	My Webservers TEST
DESCRIPTION	Test Webservers Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	ssh://git.lab.example.com/home/git/my_webservers_TEST.git
SCM Credential	student-git

   2.1. Click the **PROJECTS** quick navigation link.

   2.2. Click the **+ADD** button to add a new Project.

   2.3. On the next screen, fill in the details as follows:

Field	Value
NAME	My Webservers TEST
DESCRIPTION	Test Webservers Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	ssh://git.lab.example.com/home/git/my_webservers_TEST.git
SCM Credential	student-git

2.4. Click **SAVE** to create the new Project.

2.5. Verify that the status icon of the SCM update job for the Project, **My Webservers TEST**, becomes green before proceeding.

3. Give the **Developers** Team the **Use** role on the Project, **My Webservers TEST**.

3.1. Click the **PROJECTS** quick navigation link.

3.2. On the same line as the Project, **My Webservers TEST**, click the pencil icon on the right to edit the Project.

3.3. On the next page, click **PERMISSIONS** to manage the Project's permissions.

3.4. Click the **+ADD** button on the right to add permissions.

3.5. Click **TEAMS** to display the list of available Teams.

3.6. In the first section, check the box next to the **Developers** Team. This causes the Team to display in the second section underneath the first one.

3.7. In the second section below, select the **Use** role from the drop-down list.

3.8. Click **SAVE** to make the role assignment.

4. Create a new Job Template called **TEST webservers setup** with the following information:

Field	Value
NAME	TEST webservers setup
DESCRIPTION	Setup apache on TEST webservers
JOB TYPE	Run
INVENTORY	Test
PROJECT	My Webservers TEST
PLAYBOOK	apache-setup.yml
MACHINE CREDENTIAL	Operations

4.1. Click the **TEMPLATES** quick navigation link.

4.2. Click the **+ADD** button to add a new Job Template.

4.3. From the drop-down list, select **Job Template**.

4.4. On the next screen, fill in the details as follows:

Field	Value
NAME	TEST webservers setup
DESCRIPTION	Setup apache on TEST webservers
JOB TYPE	Run
INVENTORY	Test

PROJECT	My Webservers TEST
PLAYBOOK	apache-setup.yml
MACHINE CREDENTIAL	Operations

4.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

5. Give the **Developers** Team the **Execute** role on the Job Template, **TEST webservers setup**.

   5.1. Click **PERMISSIONS** to manage the Job Template's permissions.

   5.2. Click the **+ADD** button on the right to add permissions.

   5.3. Click **TEAMS** to display the list of available Teams.

   5.4. In the first section, click to select the box next to the **Developers** Team. This causes the Team to display in the second section underneath the first one.

   5.5. In the second section below, select the **Execute** role from the drop-down list.

   5.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Job Template, **TEST webservers setup**, which now shows that all members of the **Developers** Team are assigned the **Execute** role on the Job Template.

6. Launch a Job using the Job Template, **TEST webservers setup**, as the User, **david**, who is a member of the **Developers** Team.

   6.1. Click the **Log Out** icon in the top right corner to log out and log back in as **david** with a password of **redhat123**.

   6.2. Click the **TEMPLATES** quick navigation link.

   6.3. On the same line as the Job Template, **TEST webservers setup**, click the rocket icon on the right to launch the Job. This redirects you to a detailed status page of the running job.

   6.4. Observe the live output of the running job for a minute.

   6.5. Verify that the **STATUS** of the job in the **DETAILS** section displays **Successful**.

7. Log out of the Tower web interface and then verify that the web servers are up and running on **serverc.lab.example.com** and **serverd.lab.example.com**.

   7.1. Click the **Log Out** icon in the top right corner to logout of the Tower web interface.

   7.2. Open a web browser and go to **http://serverc.lab.example.com**. You should see the following output:

```
This is a test message RedHat 7.3
Current Host: serverc
Server list:
serverc.lab.example.com
serverd.lab.example.com
```

7.3. Open a web browser and go to **`http://serverd.lab.example.com`**. You should see the following output:

```
This is a test message RedHat 7.3
Current Host: serverd
Server list:
serverc.lab.example.com
serverd.lab.example.com
```

8. Run the command **`lab provision-review grade`** on **`workstation`** to grade your exercise.

Summary

In this chapter, you learned:

- Git is a distributed version control system. Users clone a Git project from a remote repository to a local repository, make changes to the local repository, and, when ready, push their changes to the remote repository to be shared with other users.
- The Project resource is used to represent an Ansible Project. A Project can be populated in different ways. A common method is to retrieve the Project from an SCM source. A Project which uses this method requires an SCM Credential resource to authenticate against SCM sources.
- Job Templates define the parameters for the execution of Ansible jobs. A Job Template needs to be defined with a Project from which to obtain the playbook for execution. It also uses an Inventory to define the list of managed hosts to execute the playbook on. A Machine Credential is used to authenticate against managed hosts.

CHAPTER 5

CONSTRUCTING ADVANCED JOB WORKFLOWS

Overview	
Goal	Use additional features of Job Templates to improve workflows by creating simple job launch forms, templates to launch multiple jobs in sequence, and to report job success or failure through external notification systems
Objectives	• Create a job template survey which helps Tower users launch a job with special playbook variables set in a simple way • Create a Workflow Job Template and launch multiple Ansible jobs as a single workflow • Configure and use Notifications to report job completion success or failure
Sections	• Creating Job Template Surveys to Set Variables for Jobs (and Guided Exercise) • Creating Workflow Job Templates and Launching Workflow Jobs (and Guided Exercise) • Configuring and Receiving Automatic Notification of Job Success and Failure (and Guided Exercise)
Lab	• Constructing Advanced Job Workflows

Creating Job Template Surveys to Set Variables for Jobs

Objectives

After completing this section, students should be able to create a job template survey which helps Tower users launch a job with extra variables set in a simple way.

Managing Variables

Ansible users are encouraged to write playbooks that can be reused in different situations or when deploying to systems that should have slightly different behaviors, configurations, or work in different environments. One easy way to deal with this is to use variables.

Variables can have values set in a number of ways by Ansible, and values can be overridden depending on how they were set. For example, a role can provide a default value for a variable that may in turn be overridden by values set for that variable by the inventory or by the playbook. However, in general it is best to set a value for a variable in exactly one place so that you don't have to think too hard about variable precedence.

When running playbooks using **`ansible-playbook`**, users have a couple of ways to set values for variables interactively. Firstly, they can pass *extra variables* by using the **`-e`** or **`--extra-vars`** to the command. Extra variables always win precedence. Alternatively, the playbook may have a **`vars_prompt`** section that can interactively prompt the user for input during a playbook run. The values set by **`vars_prompt`** variables have a lower precedence than extra variables and can be overridden by various things.

In Ansible Tower, this works a little differently. Extra variables can be set by the Job Template, the user can be prompted for them when launching a Job Template, or they may be set automatically by re-running a Job that was launched with extra variables defined. Playbooks with **`vars_prompt`** questions are not supported by Ansible Tower. The closest replacement for **`vars_prompt`** is the Surveys feature of Ansible Tower, which will be discussed later in this section.

Important

Ansible Tower does not support playbooks which use **`vars_prompt`** to interactively set variables.

Defining Extra Variables

In Ansible Tower, Job Templates can be used to directly set extra variables in two ways:

- Extra variables can be set by entering them in YAML or JSON format in the **EXTRA VARIABLES** field of the Job Template
- If **Prompt on launch** is checked for the **EXTRA VARIABLES** field, the Tower user will be prompted to interactively modify the list of extra variables used when they use the Job Template to launch a job

These extra variables are exactly like the variables specified by the **`-e`** or **`--extra-vars`** options for **`ansible-playbook`**, and their values override any values set for those variables. The values set through extra variables always win precedence.

The following screenshot is an example of extra variables being set in the Job Template:

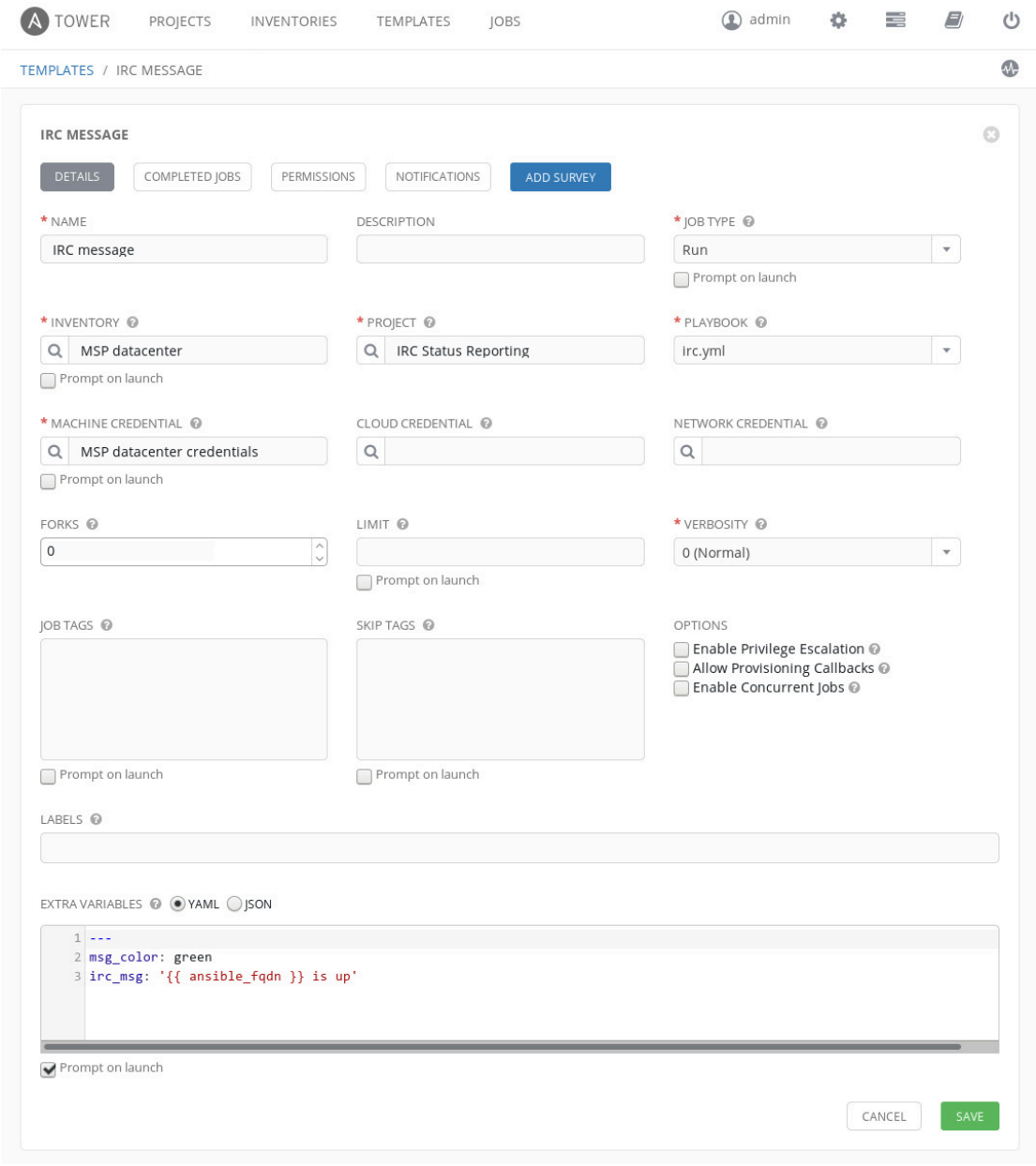

Figure 5.1: Adding Extra Variables to a Job Template

If the **Prompt on Launch** checkbox is set for **EXTRA VARIABLES**, when a Job is launched using the Job Template, a dialog box will open, which allows the Tower user to edit the extra variables for the job:

Figure 5.2: Adjusting Extra Variables on Job Launch

If the resulting Job is later relaunched, the same extra variables used to launch it are used again. The extra variables for a Job cannot be changed when relaunching it. Instead, launch the job from the original Job Template with different extra variables set.

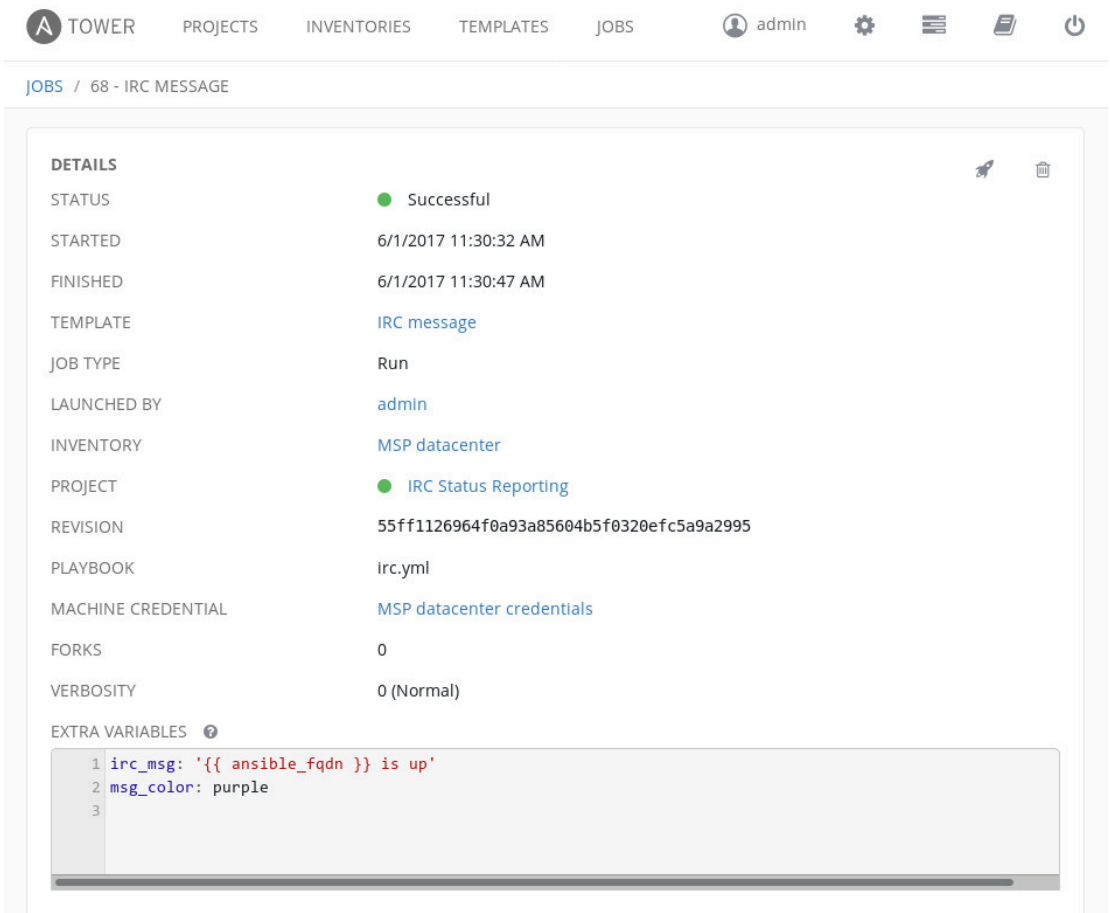

Figure 5.3: Relaunching a Job that Used Extra Variables

Job Template Surveys

Extra variables can be hard to use because the user launching the job needs to understand what variables are available and how they should be used with the Job Template's playbook. *Job Template Surveys* allow the Job Template to display a short form when used to launch a job to ask the user questions that will be used to set values for extra variables.

Prompting for user input offers several advantages over other ways to set extra variables. Users do not need to have detailed knowledge regarding the workings of extra variables or that they are even being used. They also do not need to have knowledge of the names of the extra variables which are used by the playbook.

Since prompts can contain arbitrary text, they can be phrased in a manner which is user-friendly and easily understood by users who may not have detailed knowledge about Ansible.

Important

Surveys set extra variables. In fact, the values set by Surveys for variables override the values set on variables of the same name in any other way. This includes the Job Template's **EXTRA VARIABLES** field or its **Prompt on launch** setting.

This is one way in which Surveys and **`vars_prompt`** are not direct replacements for each other. Variables set through **`vars_prompt`** have a lower precedence than extra variables and can be overridden in a number of ways. Values set by Surveys are extra variables and always win.

User-Friendly Questions

Surveys allow users to be prompted with customized questions. This allows for a much more user-friendly prompting for user input of extra variable values than the **Prompt on launch** method.

Answer Type

Aside from offering a user-friendly prompt, Surveys can also define rules for, and perform validation of, user inputs. User responses to survey questions can be restricted to one of the following seven answer types:

Type	Description
Text	Single line text
Textarea	Multi-line text
Password	Data will be treated as sensitive information.
Multiple Choice (single select)	A list of options from which only one option can be chosen as a response.
Multiple Choice (multiple select)	A list of options from which one or more options can be chosen as a response.
Integer	Integer number
Float	Floating-point decimal number

Answer Length
Rules can also be defined for the size of user responses to survey questions. For the following non-list answer types, a survey can define the minimum and maximum allowable character length for the user response: **Text**, **Textarea**, **Password**, **Integer**, and **Float**.

Default Answer
A default answer can be provided for a question. The question can also be marked as **REQUIRED** which indicates an answer must be provided for the question.

Creating a Job Template Survey

During the creation of a Job Template, the addition of a Survey is not option. A Survey can only be added to a Job Template after the Job Template has been created.

The following procedure illustrates how to add a Survey to an existing Job Template.

1. Click the **TEMPLATES** quick navigation link to see the list of existing Job Templates.
2. Click the pencil icon next to the desired Job Template to edit the Job Template.
3. Click **ADD SURVEY** to enter the Survey creation interface.
4. Add a question to the Survey.
   4.1. In the **PROMPT** field, enter the question to be presented to the user.
   4.2. In the **ANSWER VARIABLE NAME** field, enter the name of the extra variable to assign the user's response to.
   4.3. From the **ANSWER TYPE** drop-down list, select the desired answer type for the user's response.
   4.4. If using a list answer type, define the list by entering one list item per line in the **MULTIPLE CHOICE OPTIONS** field.
   4.5. If using a non-list answer type, optionally specify the minimum and maximum character length for the user's response in the **MINIMUM LENGTH** and **MAXIMUM LENGTH** fields, respectively.
   4.6. If desired, optionally define a default value for the extra variable being surveyed in the **DEFAULT ANSWER** field. This value will be used if no user reasons is entered.
   4.7. Each survey question is mandatory by default as determined by the selection of the checkbox next to the **REQUIRED** field. Deselect this box if the survey question does not require a response.
   4.8. Click the **+ADD** button to add the question to the Survey. A preview of the added question will appear under the **PREVIEW** section of the interface.
5. Repeat the previous step to add additional questions to the Survey. Once all questions have been added, scroll to the top of the screen. In the upper left corner next to **SURVEY** is a toggle button, which determines whether the Survey is enabled or disabled. By default, Surveys are enabled upon creation, so the button is toggled to **ON**. If the Survey should be disabled, click on the button to toggle it to **OFF**.
6. Lastly, click **SAVE** to save the Survey.

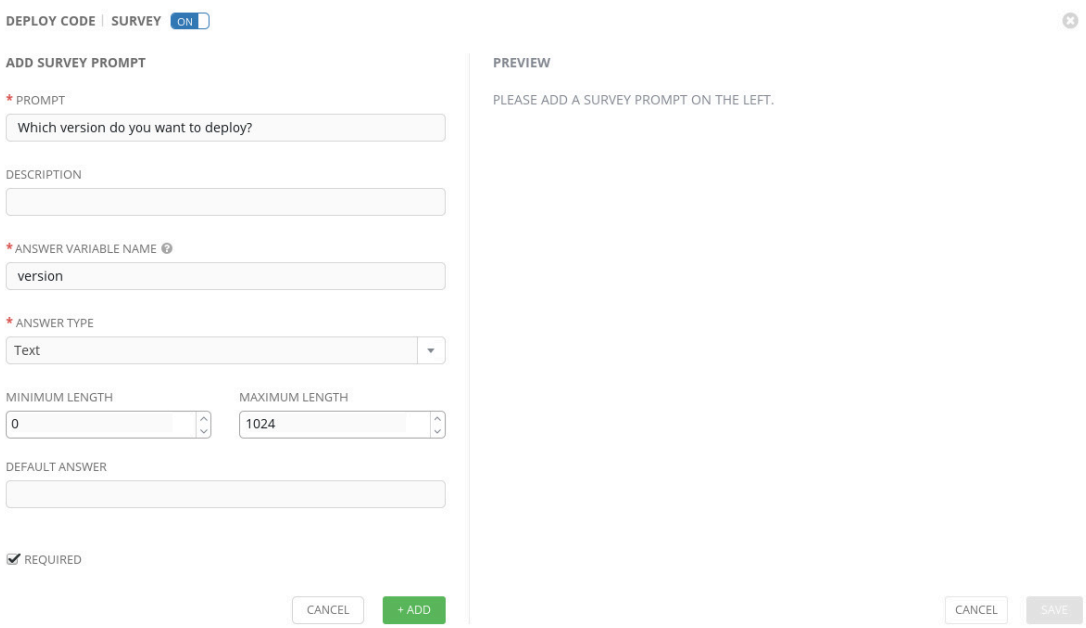

Figure 5.4: Adding Survey to a Job Template

Once a Survey is added to a Job Template, users will be prompted for answers to the Survey's questions in future job executions using the Job Template. If the Job Template has extra variables, or prompts the user to set extra variables, those will be set and prompted for first, and then the Survey will be displayed and answers to those prompts may override extra variables that have already been set.

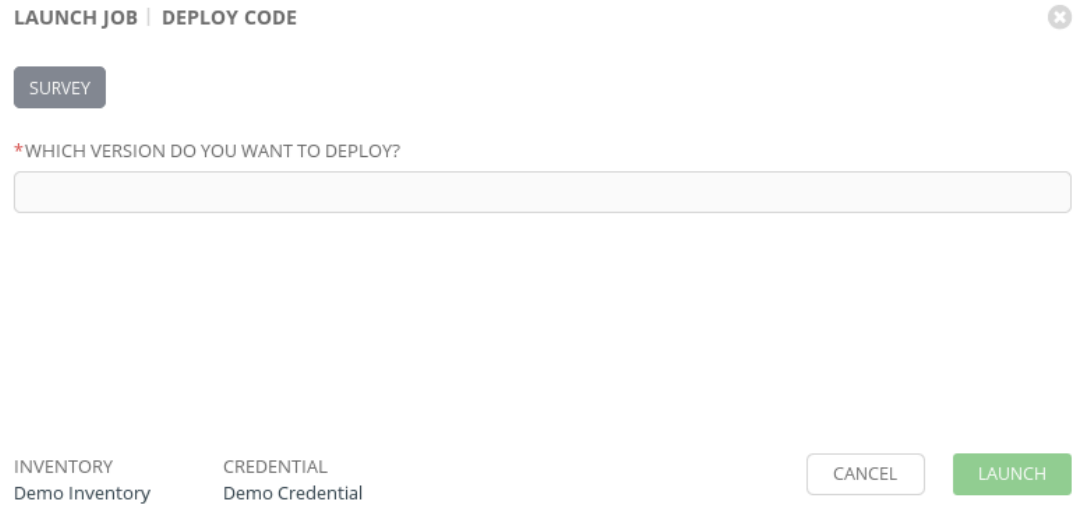

Figure 5.5: Survey Prompt

Note

The Job Template Survey feature is only available with Enterprise-level licenses.

References

Ansible Tower User Guide for Ansible Tower 3.1.1
http://docs.ansible.com/ansible-tower/3.1.1/html/userguide/

Guided Exercise: Creating Job Template Surveys to Set Variables for Jobs

In this exercise, you will add a Survey to an existing Job Template and launch a Job using a Survey.

Outcomes
You should be able to add a Survey to an existing Job Template and launch a Job using a Survey from the Tower web interface.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab project-survey setup** command. This setup script will ensure that the **workstation** and **tower** virtual machines are started.

```
[student@workstation ~]$ lab project-survey setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Add a Survey to the Job Template, **DEV webservers setup**.

   2.1. Click the **TEMPLATES** quick navigation link.

   2.2. On the same line as the Job Template, **DEV webservers setup**, click the pencil icon on the right to edit the Job Template.

   2.3. Click the **ADD SURVEY** button to add a Survey.

   2.4. On the next screen, fill in the details as follows:

Field	Value
PROMPT	What version are you deploying?
DESCRIPTION	This version number will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	deployment_version
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	v1.0
REQUIRED	Checked

   2.5. Click **+ADD** to add that Survey Prompt to the Survey. This displays a preview of your Survey on the right.

Important

Before saving, make sure that the **ON/OFF** switch is set to **ON** at the top of the Survey editor window.

2.6. Click **SAVE** to add the Survey to the Job Template.

3. Modify the **apache-setup** playbook in order to use the variable from the Survey.

3.1. Open a terminal on **workstation** and change directory to the **my_webservers_DEV** Git repository.

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

3.2. Edit the **index.html.j2** template.

```
[student@workstation my_webservers_DEV]$ vi templates/index.html.j2
```

3.3. Append the following line at the bottom of the file.

```
Deployment Version: {{ deployment_version }} <br>
```

3.4. Save the file.

3.5. Add, commit and push the file to the remote repository.

```
[student@workstation my_webservers_DEV]$ git add templates/index.html.j2 && git
 commit -m "Display Deployment Version on index page" && git push
[master 814b7c3] Display Deployment Version on index page
 1 file changed, 1 insertion(+)
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 429 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To git@git.lab.example.com:my_webservers_DEV.git
   d6e363f..814b7c3  master -> master
```

4. Update the local copy of the repository for the Project, **My Webservers DEV**.

4.1. Click the **PROJECTS** quick navigation link.

4.2. On the same line as the Project, **My Webservers DEV**, click the cloud icon on the right to launch an SCM Update of the Project and wait for the status icon to be steady green.

5. Launch a Job using the updated **DEV webservers setup** Job Template as a member of the **Developers** Team.

5.1. Click the **Log Out** icon in the top right corner to log out, and then log back in as **daniel** with a password of **redhat123**.

5.2. Click the **TEMPLATES** quick navigation link.

5.3. On the same line as the Job Template, **DEV webservers setup**, click the rocket icon on the right to launch the Job. This opens the Survey you just created and ask for your input.

5.4. You can leave **v1.0** in the text field and click **LAUNCH** to launch the job. This redirects you to a detailed status page of the running job.

5.5. Observe the live output of the running Job for a minute.

5.6. Verify that the **STATUS** of the job in the **DETAILS** section ends up being **Successful**.

6. Verify that the web servers have been updated on **servera.lab.example.com** and **serverb.lab.example.com**.

   6.1. Open a web browser and go to **http://servera.lab.example.com** and **http://serverb.lab.example.com** in separate tabs. You should see this line at the bottom of both pages:

```
Deployment Version: v1.0
```

7. Click the **Log Out** icon to log out of the Tower web interface.

Creating Workflow Job Templates and Launching Workflow Jobs

Objectives

After completing this section, students should be able to create a Workflow Job Template and launch multiple Ansible jobs as a single workflow.

Workflow Job Templates

As an organization's use of Ansible grows, so does the number of Ansible playbooks. Each playbook typically performs a set of tasks associated with a certain IT function. In a previous section, it was shown that Ansible Tower's Job Templates provide users the ability to execute Ansible playbooks as jobs.

IT processes in enterprise environments usually require multiple operations to happen. For example, server provisioning may involve IP address allocation and DNS record creation by a Networking team, followed by the installation and configuration of the operating system on the server by the Operations team, and then concluded by installation and configuration of the necessary software by the Application Development team. There is a particular *workflow* that must be followed for the process to succeed.

This could be managed in Ansible Tower by having users manually launch multiple jobs in sequence. But the jobs have to be executed in the correct order defined by the workflow for everything to work correctly.

- The Networking jobs have to be executed first.
- The Operations jobs would follow only if the Networking jobs were successfully completed.
- Likewise, the Application Development jobs would then follow only if the Networking and Operations jobs successfully completed.

To make this easier, Ansible Tower provides *Workflow Job Templates*. A Workflow Job Template connects multiple Job Templates into a workflow. When launched, the Workflow Job Template launches a Job using the first Job Template, and depending on whether it succeeds or fails, determines which Job Template to launch next. This allows a sequence of jobs to be launched, and for recovery steps to be taken automatically if a job fails.

Tower's Workflow Job Template has the flexibility to accommodate even complicated IT processes such as the server provisioning cross-functional example provided earlier. It does not simply batch Job Templates in a serial fashion. Using the graphical Workflow Editor, Workflow Job Templates can be created to chain together multiple Job Templates and also incorporate decision-making. This creates complex, branching workflows whose course is guided by the outcomes of each executed job.

Note

The Workflow Job Template feature is only available with Enterprise-level licenses.

Delegating Workflow Job Template
In addition, as a Tower resource, Workflow Job Templates automatically inherit access control features and can therefore be easily delegated while still maintaining necessary security controls. Since users do not need to understand the underlying relationships and decision-making interactions of the Job Templates involved, Workflow Job Templates lend themselves to push-button execution. With Workflow Job Templates, complex IT processes which previously required time and resources from multiple teams can now be automated and simplified down to the click of a button to initiate a Workflow Job.

Workflow Job Template Execution
Being a Tower resource, once a Workflow Job Template is created, it is accessible for execution via the numerous job execution methods offered by Tower. They can be launched by a user in the Tower web interface. Alternatively, they can be launched by a program sending a request through the Tower API. Finally, they can be launched in an automated fashion through scheduled execution.

Surveys
Workflow Job Templates have access to many of the features that have been discussed for Job Templates. Like Job Templates, they can have Surveys added in order to allow users to interactively set extra variables.

Note

When Surveys are added to a Workflow Job Template, the resulting extra variables are accessible by every job executed by the workflow.

Creating Workflow Job Templates

A new Workflow Job Template must be created before a workflow can be defined and associated with it. They can be created with an Organization or without an Organization. Creating a Workflow Job Template within the context of an Organization requires that the user has the **`admin`** role for the Organization. The singleton **`System Administrator`** user type is required in order to create a Workflow Job Template that isn't part of an Organization.

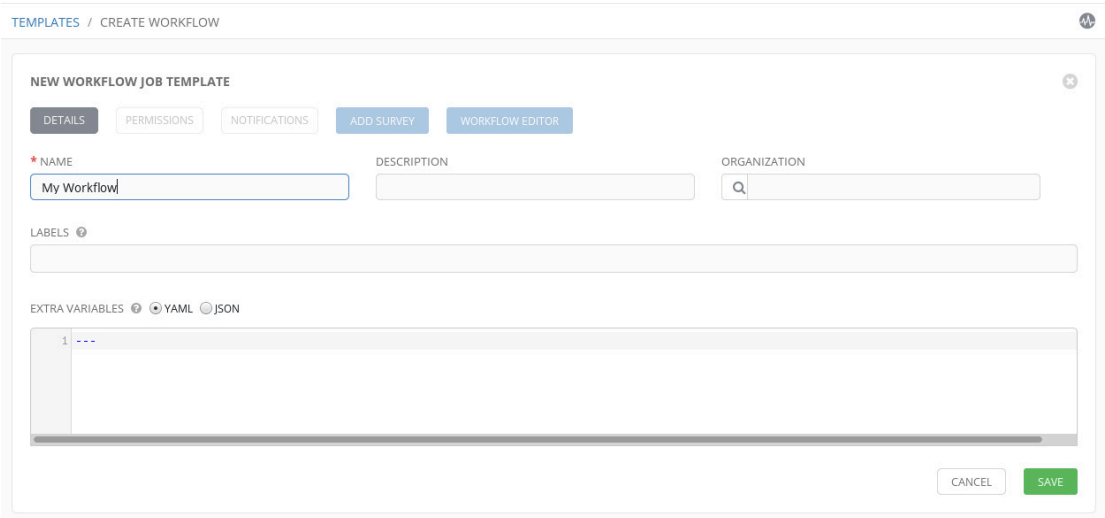

Figure 5.6: Creating a Workflow Job Template

Workflow Job Templates are created in a similar fashion as Job Templates.

1. Click on the **TEMPLATES** quick navigation link to access to Template management interface.
2. Click **+ADD** and select **Workflow Job Template** from the dropdown list.
3. Enter a unique name for the Workflow Job Template in the **NAME**. Optionally enter any desired key/value pairs in the **EXTRA VARIABLES** field.
4. Click **SAVE** to finalize the creation of the Workflow Job Template. Once a Workflow Job Template is created, the Workflow Editor can be used to define an associated workflow.

Using the Workflow Editor

Once a Workflow Job Template is created, the **WORKFLOW EDITOR** becomes clickable in the Workflow Job Template editing screen. Clicking this button launches a the Workflow Editor in a new window. The Workflow Editor is a graphical interface for defining the job templates to incorporate in a workflow as well as the decision tree structure, which should be used to chain the job templates together.

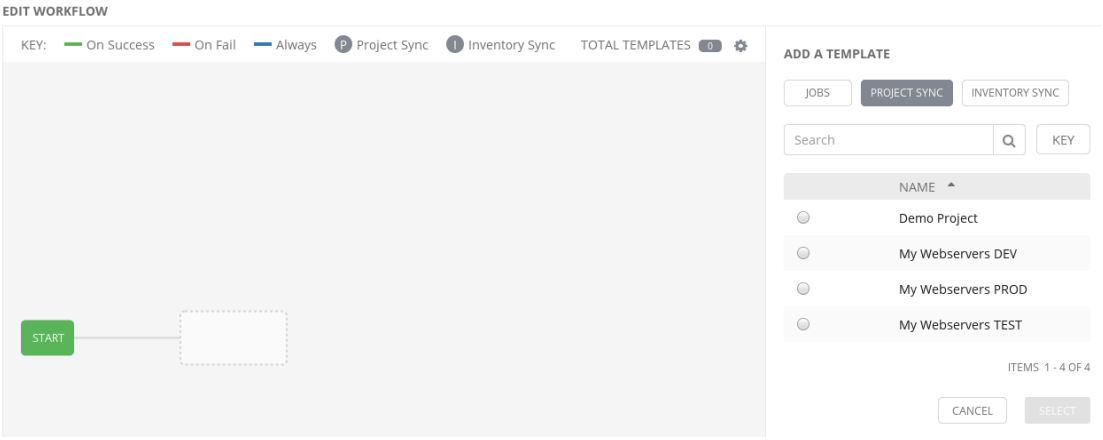

Figure 5.7: Starting a workflow in the Workflow Editor

When the Workflow Editor is launched, it contains just a single **START** node representing the starting point for the execution of the workflow. Clicking **START** initiates the workflow editing process by displaying lists of Tower resources, which can be added as the first step of the workflow. Selecting the desired resource type, the specific resource, and then clicking the **SELECT** button, adds a Tower resource as the first node in the workflow.

In addition to Job Templates, Jobs that synchronize Projects or Inventories can also be incorporated into a workflow. This is useful to ensure that Project and Inventory resources are updated prior to the use of Job Templates depending on them. To make them easier to identify, Project Sync and Inventory Sync nodes are indicated by a **P** or an **I** in the lower left corner of the node, respectively. This notation is explained by the key at the top of the Workflow Editor screen. Job Template nodes are not marked with any special notation since they are the main node type in a workflow.

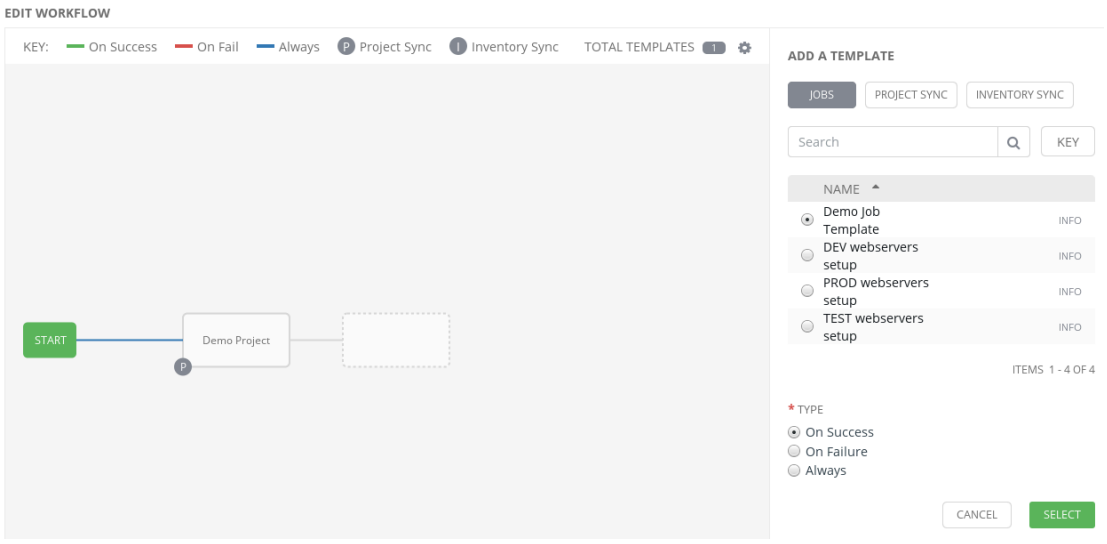

Figure 5.8: Adding On Success node to workflow

Once a resource is added as the first workflow node, hovering over it causes two buttons to appear. The red **-** button is used to delete the node while the green **+** button is used to add a subsequent node. When adding subsequent nodes, the **TYPE** prompt appears in the resource selection panel asking for additional user input once a resource is selected. The following three choices are offered by this prompt to designate the relationship between the new node and the preceding node.

Type	Node Relationship
On Success	Node resource is executed upon successful completion of the actions associated with the previous node.
On Failure	Node resource is executed upon failure of the actions associated with the previous node.
Always	Node resource is executed regardless of the outcome of the actions associated with the previous node.

A node can have more than one or two child nodes. For example, a child node can be added with a parent node association type of **On Success** and another child node can be added with an association type of **On Failure**. This creates a branch in the workflow structure such that one course of action is taken upon the success of an action, while a different course is taken upon failure.

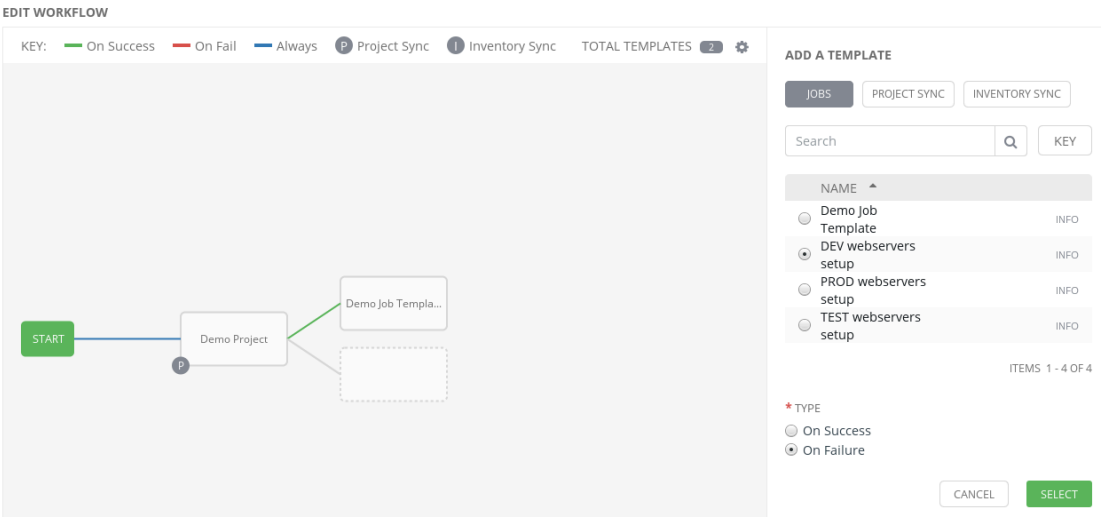

Figure 5.9: Adding multiple child nodes to the workflow

As nodes are added to a workflow, differently colored lines connecting the nodes in the Workflow Editor indicate the relationships between parent and child nodes. A green line indicates an **On Success** type relationship between a parent and child node, while a red line indicates an **On Failure** type relationship. An **Always** type relationship is represented by a blue line.

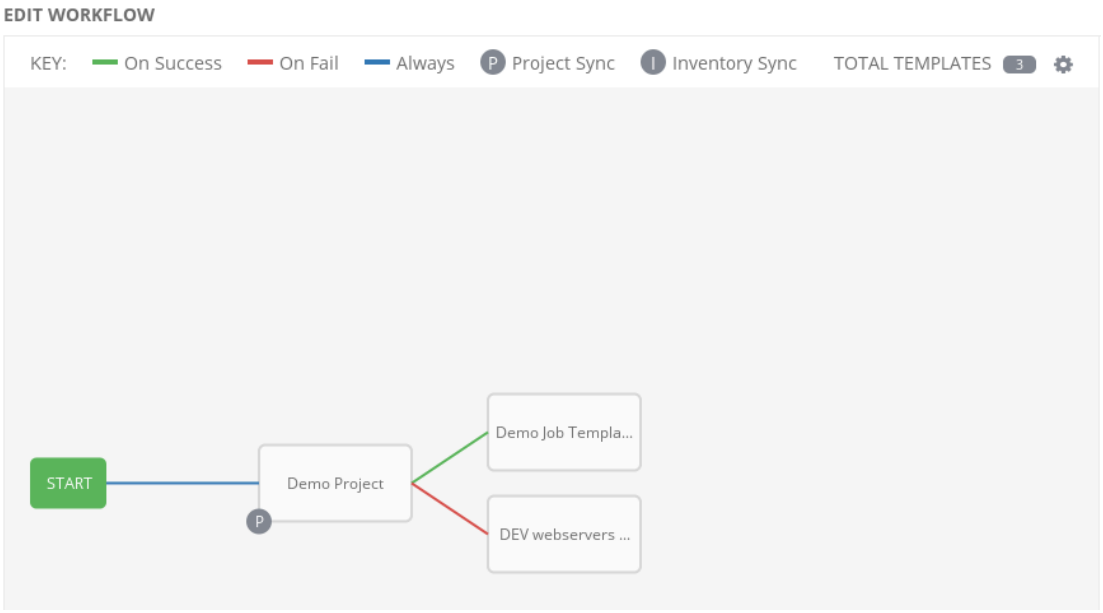

Figure 5.10: Workflow node relationships

Once the entire decision tree structure of the workflow has been created in the Workflow Editor, click **SAVE** to save the workflow.

Launching Workflow Jobs

Like Job Templates, Users need the **execute** role on the Workflow Job Template in order to execute it. When assigned the **execute** role, a User can launch a job through a Workflow Job Template even if they do not have permissions to independently launch the Job Templates it uses.

The procedure to launch a Workflow Job Template is similar to how a Job Template is launched:

1. Click on the **TEMPLATES** quick navigation link to access the Template management interface.

2. Click the Workflow Job Template's rocket icon under the **ACTIONS** column to launch the workflow job.

Evaluating Workflow Job Execution

After a workflow job is launched, Tower redirects the user's browser to the Job Details page for the job being executed. This page consists of two panes. The **DETAILS** pane displays details of the workflow job execution. The workflow progress pane displays the progress of the job through the steps in the workflow.

As each step is completed, its node is outlined in either green or red, indicating the success or failure of the actions associated with that step in the workflow. Progression from one step to another are represented by colored lines indicating the decision responsible for the progression. Green indicates an **`On Success`** progression, while red indicates an **`On Failure`** progression. Blue indicates an **`Always`** progression.

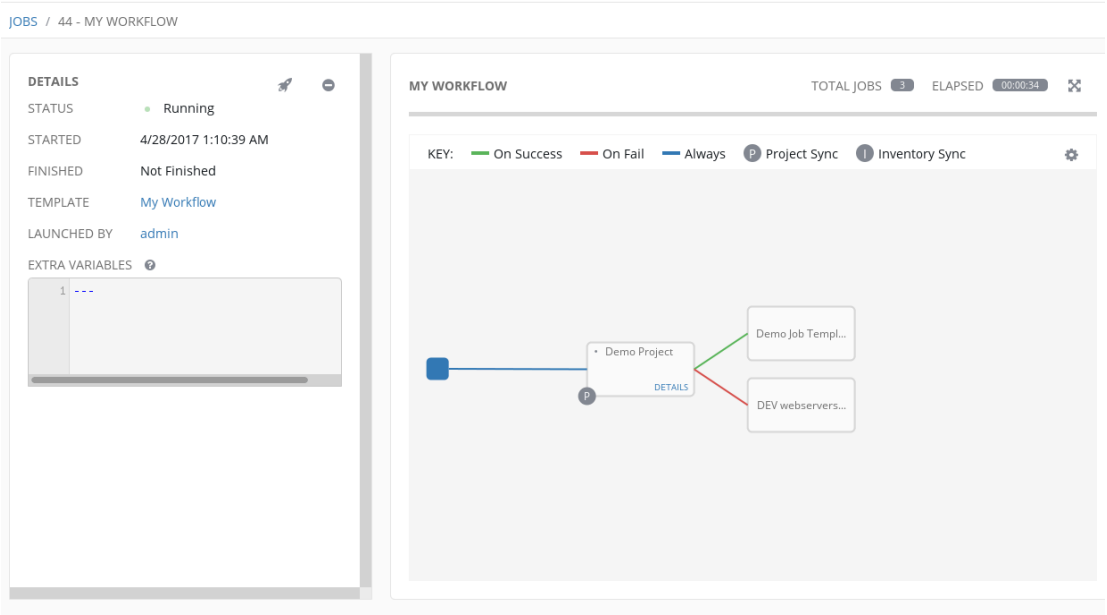

Figure 5.11: Workflow job progress

Details of the workflow job's run can be displayed either during or after the execution. Each node in the workflow diagram representing a currently running job or completed job provides a **DETAILS** link. Clicking this link for a job displays the results and standard output for the job run.

References

Ansible Tower User Guide for Ansible Tower 3.1.1

http://docs.ansible.com/ansible-tower/3.1.1/html/userguide/

Guided Exercise: Creating Workflow Job Templates and Launching Workflow Jobs

In this exercise, you will create a Workflow Job Template that will orchestrate the deployment of the DEV web servers. Upon the successful deployment of the DEV web servers, the Workflow Job Template will carry out the deployment of the TEST web servers.

Outcomes
You should be able to create a Workflow Job Template and launch a Workflow from the Tower web interface.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab project-workflow setup**. This setup script will ensure that the **workstation** and **tower** virtual machines are started.

```
[student@workstation ~]$ lab project-workflow setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a Workflow Job Template called **From Dev to Test**.

   2.1. Click the **TEMPLATES** quick navigation link.

   2.2. Click the **+ADD** button to add a new Workflow Job Template.

   2.3. From the drop-down list, select **Workflow Job Template**.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	From Dev to Test
DESCRIPTION	Deploy to Dev and on success deploy to Test
ORGANIZATION	Default
EXTRA VARIABLES	deployment_version: "v1.1"

   2.5. Click **SAVE** to create the new Workflow Job Template.

3. Configure the Workflow of the **From Dev to Test** Workflow Job Template.

   3.1. Click **WORKFLOW EDITOR** to open the Workflow Editor.

   3.2. Click **START** to add the first action to be performed. This displays in the right panel a list of actions to be performed.

3.3. In the right panel, click **PROJECT SYNC** to display the list of Projects available.

3.4. Select **My Webservers DEV** and click **SELECT**. This links the **START** node with a blue line (always perform) to the node for the Project, **My Webservers DEV**, in the Workflow Editor window.

3.5. Move your mouse over the new node and click on the green **+** button to add an action after the Project Sync of **My Webservers DEV**. This displays a list of actions to be performed in the right panel.

3.6. In the right panel, make sure to be in the **JOBS** section and select the Job Template, **DEV webservers setup**.

3.7. In the **TYPE** section below, select **On Success** and click **SELECT**. This links the node for the Project, **My Webservers DEV**, to a new node for the Job Template, **DEV webservers setup**, in the Workflow Editor window. The green color of the link indicates that the progression only takes place upon success of the first step.

3.8. Move your mouse over the new node and click on the green **+** button to add an action after the Job Template, **DEV webservers setup**.

3.9. In the right panel, click **PROJECT SYNC** to display the list of Projects available.

3.10. Select **My Webservers TEST**. In the **TYPE** section below, select **On Success** and click **SELECT**.

3.11. Move your mouse over the new node and click on the green **+** button to add an action after the Project Sync of **My Webservers TEST**.

3.12. In the right panel, make sure to be in the **JOBS** section and select the Job Template, **TEST webservers setup**. In the **TYPE** section below, select **On Success** and click **SELECT**.

3.13. Click **SAVE** to save the Workflow Job Template.

4. Launch a job using the Workflow Job Template, **From Dev to Test**.

4.1. Click the **TEMPLATES** quick navigation link.

4.2. On the same line as the Workflow Job Template, **From Dev to Test**, click the rocket icon on the right to launch the Workflow. This redirects you to a detailed status page of the running Workflow.

4.3. Observe the running Jobs of the Workflow. You can click on the **DETAILS** link of each running Job to see a more detailed live output of each Job.

4.4. Verify that the **STATUS** of the Workflow in the **DETAILS** section of the page is **Successful**.

5. Verify that the webservers have been updated on **servera.lab.example.com**, **serverb.lab.example.com**, **serverc.lab.example.com** and **serverd.lab.example.com**.

5.1. Open a web browser and go to **http://servera.lab.example.com**, **http://serverb.lab.example.com**, **http://serverc.lab.example.com** and **http://serverd.lab.example.com** in separate tabs. You should see this line at the bottom of each page:

```
Deployment Version: v1.1
```

Configuring and Receiving Automatic Notification of Job Success and Failure

Objectives

After completing this section, students should be able to configure and use Notifications to report job completion success or failure.

Reporting Job Execution Results

One of the benefits of using Ansible Tower to manage an enterprise's Ansible infrastructure is centralized logging and auditing. When a job is executed, details regarding the job execution are logged in the Tower database. This database can be referenced by users at a later time to determine the historical results of past job executions.

Historical job execution details are helpful to administrators for confirming the success and failure of scheduled and delegated job executions. While the availability to retrieve historical job execution details is helpful, for jobs related to critical functions, administrator likely desire immediate notification of a job's success or failure.

Tower offers a feature for the immediate alerting of job execution results. To use this feature, administrators create Notification Templates which define how notifications are to be sent. Tower supports many mechanisms to send notifications. Some are based on open protocols such as email and IRC, while others are based on proprietary solutions, such as HipChat and Slack.

Notification Templates

A *Notification Template* is defined in the context of an Organization. Once it has been created, the Notification Template can be used to send notifications of the results of jobs run by Ansible Tower for that Organization.

A Notification Template defines the mechanism for how a notification is to be sent. Supported mechanisms include:

- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC

Creating Email Notification Templates

Notification Templates are created through the **NOTIFICATIONS** interface, accessible from the **SETTINGS** screen. Depending on the selected notification mechanism, the **TYPE DETAILS** section of the Notification Template interface prompts for different user inputs.

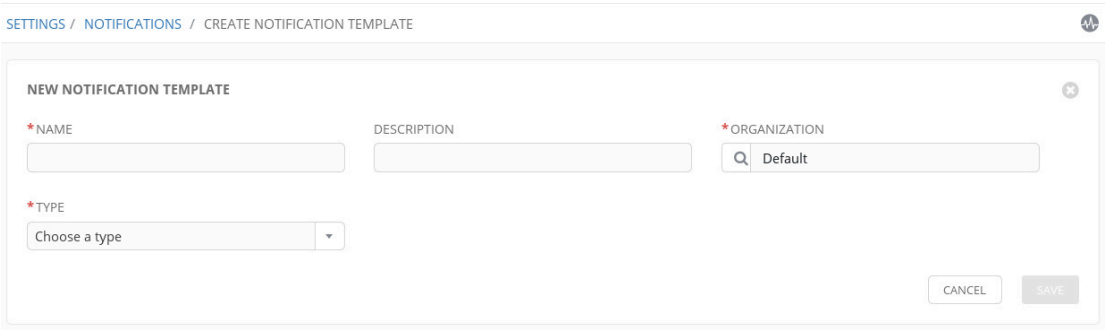

Figure 5.12: Creating notifications

The following steps are used to create an **`Email`** type Notification Template.

1. In the **NOTIFICATIONS** interface, click **+ADD** to create a new Notification Template.
2. Enter a unique name for the new Notification Template in the **NAME** field.
3. In the **ORGANIZATION** field, specify the Organization within which to create the Notification Template.
4. In the **TYPE** drop-down list, select **Email** as the mechanism to be used by the Notification Template for generating notifications. Once the **TYPE** drop-down list selection is made, type-specific user input fields appear under the **TYPE DETAILS** section.
5. In the **SENDER EMAIL** field, specify the sending email address to be used in composing the notification email.
6. In the **RECIPIENT LIST** field, specify the email addresses of the recipients for the notification email, one on each line.
7. In the **PORT** field, specify the port to connect to on the SMTP host to relay the notification email.
8. If desired, populate the optional fields provided for configuring SMTP authentication and for enabling secure transport.
9. Click **SAVE** to finalize the creation of the new Notification Template.

Enabling Job Result Notification

Once a Notification Template has been created, it becomes available for use by certain Tower resources that are part of the Notification Template's Organization, such as Job Templates.

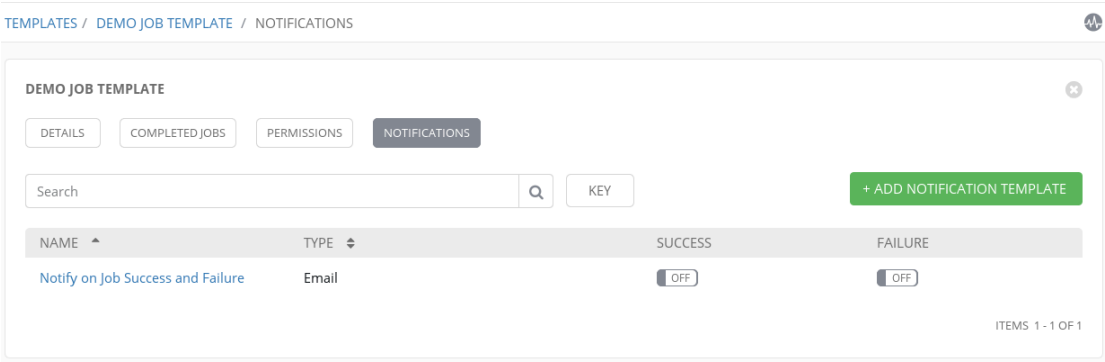

Figure 5.13: Enabling notifications on a Job Template

The following steps enable notifications for a Job Template.

1. Click the **TEMPLATES** quick navigation link to display the list of templates.
2. Click the name of the Job Template to enable notification for and then click **NOTIFICATIONS**.
3. In the **NOTIFICATIONS** interface for the selected Job Template, a list of the Notification Templates for the Organization is displayed.
4. Each listed Notification Template has controls for toggling success and failure notifications. Toggle the **SUCCESS** and **FAILURE** switches to achieve the desired notification configuration for the Job Template.

Note

Notification Templates can also be used to enable notifications for system jobs triggered by Project and Inventory resources to synchronize their data.

References

Ansible Tower User Guide for Ansible Tower 3.1.1

> http://docs.ansible.com/ansible-tower/3.1.1/html/userguide/

Guided Exercise: Configuring and Receiving Automatic Notification of Job Success and Failure

In this exercise, you will create an email Notification Template and use it for an existing Job Template. Then you will launch the Job and check the result of the notification.

Outcomes
You should be able to create a Notification Template and use it with a Job Template to generate email notifications of job results.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab project-notification setup**. This setup script will setup the SMTP server.

```
[student@workstation ~]$  lab project-notification setup
```

Steps

1. Log on to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Add a Notification Template to the Job Template, **DEV webservers setup**.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   2.2. Click **NOTIFICATIONS** to manage Notification Templates.

   2.3. Click **+ADD** to add a Notification Template.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Notify on Job Success and Failure
DESCRIPTION	Sends an email to notify the status of the Job
ORGANIZATION	Default
TYPE	Email
HOST	localhost
SENDER EMAIL	system@tower.lab.example.com
RECIPIENT LIST	student@tower.lab.example.com
PORT	25

   2.5. Leave all the other fields untouched and click **SAVE** to save the Notification Template. You will be redirected to the list of Notification Templates.

3. Validate the Notification Template.

   3.1. On the same line as the Notification Template, **Notify on Job Success and Failure**, click the bell icon on the right to test the Notification process.

   3.2. Wait several seconds. You should see a green notification pop-up saying "Notify on Job Success and Failure: Notification sent." This validates that the notification has been sent.

4. Configure the Notification Template to be used by the Job Template, **DEV webservers setup**.

   4.1. Click the **TEMPLATES** quick navigation link.

   4.2. On the same line as the Job Template, **DEV webservers setup**, click the pencil icon on the right to edit the Job Template.

   4.3. Click **NOTIFICATIONS** to manage Notifications for the Job Template.

   4.4. On the same line as the Notification Template, **Notify on Job Success and Failure**, set both **ON/OFF** switches for **SUCCESS** and **FAILURE** to **ON**.

5. Verify that the **DEV webservers setup** Job triggers an email after completion.

   5.1. Open a terminal and connect to the **tower** VM.

```
[student@workstation ~]$ ssh tower
Last login: Thu Apr 20 11:33:22 2017 from workstation.lab.example.com
[student@tower ~]$
```

   5.2. Using the **tail** command, read incoming messages to the local mailbox file of the student user. You should see the email that has been sent by the Test Notification launched from a previous step.

```
[student@tower ~]$ tail -f /var/mail/student
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
Subject: Tower Notification Test 1 https://tower.lab.example.com
From: system@tower.lab.example.com
To: student@tower.lab.example.com
Date: Thu, 20 Apr 2017 15:48:40 -0000
Message-ID: <20170420154840.1795.89530@tower.lab.example.com>

Ansible Tower Test Notification 1 https://tower.lab.example.com
```

   5.3. Go back to the web interface and click the **TEMPLATES** quick navigation link.

   5.4. On the same line as the Job Template, **DEV webservers setup**, click the rocket icon on the right to launch the Job.

   5.5. In the Survey window, enter **v1.2** for the deployment version and click **LAUNCH**.

   5.6. Go to your terminal running the **tail** command and wait. After about one minute, you should see this type of successful Job completion notification email arrive:

```
From system@tower.lab.example.com  Thu Apr 20 12:00:40 2017
Return-Path: <system@tower.lab.example.com>
X-Original-To: student@tower.lab.example.com
Delivered-To: student@tower.lab.example.com
Received: from tower.lab.example.com (localhost [IPv6:::1])
 by tower.lab.example.com (Postfix) with ESMTP id 32C64401FAD
 for <student@tower.lab.example.com>; Thu, 20 Apr 2017 12:00:40 -0400 (EDT)
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
Subject: Job #32 'DEV webservers setup' succeeded on Ansible Tower:
 https://tower.lab.example.com/#/jobs/32
From: system@tower.lab.example.com
To: student@tower.lab.example.com
Date: Thu, 20 Apr 2017 16:00:40 -0000
Message-ID: <20170420160040.1796.95491@tower.lab.example.com>

Job #32 had status successful on Ansible Tower, view details at https://
tower.lab.example.com/#/jobs/32

{
    "status": "successful",
    "credential": "Developers",
    "name": "DEV webservers setup",
    "started": "2017-04-20T15:59:58.272958+00:00",
    "extra_vars": "{\"deployment_version\": \"v1.2\"}",
    "traceback": "",
    "friendly_name": "Job",
    "created_by": "admin",
    "project": "My Webservers DEV",
    "url": "https://tower.lab.example.com/#/jobs/32",
    "finished": "2017-04-20T16:00:39.469360+00:00",
    "hosts": {
        "serverb.lab.example.com": {
            "skipped": 0,
            "ok": 6,
            "changed": 1,
            "dark": 0,
            "failed": false,
            "processed": 1,
            "failures": 0
        },
        "servera.lab.example.com": {
            "skipped": 0,
            "ok": 6,
            "changed": 1,
            "dark": 0,
            "failed": false,
            "processed": 1,
            "failures": 0
        }
    },
    "playbook": "apache-setup.yml",
    "limit": "",
    "id": 32,
    "inventory": "Dev"
}
```

5.7. Exit the console session on the **tower** system.

```
[student@tower ~]$ exit
```

6. Verify that the web servers have been updated successfully on **servera.lab.example.com** and **serverb.lab.example.com**.

   6.1. Open a web browser and go to **http://servera.lab.example.com** and **http://serverb.lab.example.com** in separate tabs. You should see this line at the bottom of each page:

```
Deployment Version: v1.2
```

Lab: Constructing Advanced Job Workflows

In this exercise, you will create a new Workflow Job Template that will orchestrate the deployment of the TEST web servers. Upon successful completion, the Workflow Job Template will then proceed with the deployment of the PROD web servers. This Workflow Job Template will use a Survey to solicit user input and will send an email notification detailing whether the Workflow completed successfully or not.

Outcomes

You should be able to create a Workflow Job Template with an associated Survey and Notification Template and then launch a job from the Tower web interface using the Workflow Job Template.

Before you begin

You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab project-review setup**. This setup script will create all the necessary objects (Git repository, Project, and Job Template) for use with the Prod Inventory.

```
[student@workstation ~]$  lab project-review setup
```

Steps

1. Log on to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a Workflow Job Template called **From Test to Prod** with the following information.

Field	Value
NAME	From Test to Prod
DESCRIPTION	Deploy to Test and on success deploy to Prod
ORGANIZATION	Default
EXTRA VARIABLES	deployment_version: "v1.3"

3. Configure the Workflow Job Template, **From Test to Prod**, so that it contains the following steps.

   - Sync the Project, **My Webservers TEST**.
   - Upon success of the previous step, launch the Job Template, **TEST webservers setup**.
   - Upon success of the previous step, sync the Project, **My Webservers PROD**.
   - Upon success of the previous step, launch the Job Template, **PROD webservers setup**

4. Add a Survey to the Workflow Job Template, **From Test to Prod**, with the following information.

Field	Value
PROMPT	What version are you deploying?
DESCRIPTION	This version number will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	deployment_version
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	v1.0
REQUIRED	Checked

5. Activate both the **SUCCESS** and **FAILURE** Notifications for the Workflow Job Template, **From Test to Prod**, using the existing Notification Template, **Notify on Job Success and Failure**.

6. Launch a Workflow using the **From Test to Prod** Workflow Job Template. When prompted by the Survey, enter **v1.3** for the deployment version.

7. Verify that the Workflow triggers an email notification after completion.

8. Verify that the web servers have been updated on **serverc.lab.example.com**, **serverd.lab.example.com**, **servere.lab.example.com**, and **serverf.lab.example.com**.

9. Run the command **lab project-review grade** on **workstation** to grade your exercise.

Solution

In this exercise, you will create a new Workflow Job Template that will orchestrate the deployment of the TEST web servers. Upon successful completion, the Workflow Job Template will then proceed with the deployment of the PROD web servers. This Workflow Job Template will use a Survey to solicit user input and will send an email notification detailing whether the Workflow completed successfully or not.

Outcomes
You should be able to create a Workflow Job Template with an associated Survey and Notification Template and then launch a job from the Tower web interface using the Workflow Job Template.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab project-review setup**. This setup script will create all the necessary objects (Git repository, Project, and Job Template) for use with the Prod Inventory.

```
[student@workstation ~]$  lab project-review setup
```

Steps
1. Log on to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Create a Workflow Job Template called **From Test to Prod** with the following information.

Field	Value
NAME	From Test to Prod
DESCRIPTION	Deploy to Test and on success deploy to Prod
ORGANIZATION	Default
EXTRA VARIABLES	deployment_version: "v1.3"

   2.1. Click the **TEMPLATES** quick navigation link.

   2.2. Click the **+ADD** button to add a new Workflow Job Template.

   2.3. From the drop-down list, select **Workflow Job Template**.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	From Test to Prod
DESCRIPTION	Deploy to Test and on success deploy to Prod
ORGANIZATION	Default

2.5. Click **SAVE** to create the new Workflow Job Template.

3. Configure the Workflow Job Template, **From Test to Prod**, so that it contains the following steps.

   - Sync the Project, **My Webservers TEST**.
   - Upon success of the previous step, launch the Job Template, **TEST webservers setup**.
   - Upon success of the previous step, sync the Project, **My Webservers PROD**.
   - Upon success of the previous step, launch the Job Template, **PROD webservers setup**

   3.1. Click **WORKFLOW EDITOR** to open the Workflow Editor.

   3.2. Click **START** to add the first action to be performed. This displays a list of actions to be performed in the right panel.

   3.3. In the right panel, click **PROJECT SYNC** to display the list of Projects available. Select **My Webservers TEST** and click **SELECT**. In the Workflow Editor window, this links the **START** node to the node for the Project, **My Webservers TEST**, with a blue line, indicating that this step will always be performed.

   3.4. Move your mouse over the new node and click on the green **+** button to add an action after the Project Sync of **My Webservers TEST**. This displays a list of actions to be performed in the right panel.

   3.5. In the right panel, make sure to be in the **JOBS** section and select the Job Template, **TEST webservers setup**. In the **TYPE** section below, select **On Success** and click **SELECT**.

   In the Workflow Editor window, this links the node for the Project, **My Webservers TEST**, to the node for the Job Template, **TEST webservers setup**, with a green line, indicating that this progression will only be performed if the Project Sync step is successful.

   3.6. Move your mouse over the new node and click on the green **+** button to add an action after the Job Template, **TEST webservers setup**.

   3.7. In the right panel, click **PROJECT SYNC** to display the list of Projects available. Select **My Webservers PROD** and click **SELECT**.

   3.8. Move your mouse over the new node and click on the green **+** button to add an action after the Project Sync of **My Webservers PROD**.

   3.9. In the right panel, make sure to be in the **JOBS** section and select the Job Template, **PROD webservers setup**. In the **TYPE** section below, select **On Success** and click **SELECT**.

   3.10. Click **SAVE** to save the Workflow Job Template.

4. Add a Survey to the Workflow Job Template, **From Test to Prod**, with the following information.

Field	Value
PROMPT	What version are you deploying?
DESCRIPTION	This version number will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	deployment_version
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	v1.0
REQUIRED	Checked

4.1. Click the **ADD SURVEY** button to add a Survey.

4.2. On the next screen, fill in the details as follows:

Field	Value
PROMPT	What version are you deploying?
DESCRIPTION	This version number will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	deployment_version
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	v1.0
REQUIRED	Checked

4.3. Click **+ADD** to add the Survey Prompt to the Survey. This displays a preview of your Survey on the right.

Important

Before saving, make sure that the **ON/OFF** switch is set to **ON** at the top of the Survey editor window.

4.4. Click **SAVE** to add the Survey to the Workflow Job Template.

5. Activate both the **SUCCESS** and **FAILURE** Notifications for the Workflow Job Template, **`From Test to Prod`**, using the existing Notification Template, **`Notify on Job Success and Failure`**.

5.1. Click **NOTIFICATIONS** to manage notifications for the Workflow Job Template, **`From Test to Prod`**.

5.2. On the same line as the Notification Template, **Notify on Job Success and Failure**, set both **ON/OFF** switches for **SUCCESS** and **FAILURE** to **ON**.

6. Launch a Workflow using the **From Test to Prod** Workflow Job Template. When prompted by the Survey, enter **v1.3** for the deployment version.

6.1. Scroll down to the **TEMPLATES** section.

6.2. On the same line as the Workflow Job Template, **From Test to Prod**, click the rocket icon on the right to launch the Workflow. This opens the Survey you just created and ask for your input.

6.3. Type **v1.3** in the text field and click **LAUNCH** to launch the Workflow. This redirects you to a detailed status page of the running Workflow.

6.4. Observe the running Jobs of the Workflow. You can click on the **DETAILS** link of a running or completed Job to see a more detailed live output of the Job.

7. Verify that the Workflow triggers an email notification after completion.

7.1. Open a terminal and connect to the **tower** VM.

```
[student@workstation ~]$ ssh tower
Last login: Thu Apr 20 11:33:22 2017 from workstation.lab.example.com
[student@tower ~]$
```

7.2. View incoming messages to the local mailbox file of the **student** user using the **tail** command. You should see this type of successful Workflow Job completion notification email arrive:

```
[student@tower ~]$ tail -f /var/mail/student
[...output omitted...]
From system@tower.lab.example.com  Thu Apr 20 18:06:10 2017
Return-Path: <system@tower.lab.example.com>
X-Original-To: student@tower.lab.example.com
Delivered-To: student@tower.lab.example.com
Received: from tower.lab.example.com (localhost [IPv6:::1])
 by tower.lab.example.com (Postfix) with ESMTP id AAB30401FB3
 for <student@tower.lab.example.com>; Thu, 20 Apr 2017 18:06:10 -0400 (EDT)
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
Subject: Workflow Job #50 'From Test to Prod' succeeded on Ansible Tower:
 https://tower.lab.example.com/#/workflows/50
From: system@tower.lab.example.com
To: student@tower.lab.example.com
Date: Thu, 20 Apr 2017 22:06:10 -0000
Message-ID: <20170420220610.2290.67752@tower.lab.example.com>

Workflow job summary:

- node #9 spawns job #51, "My Webservers TEST", which finished with status
 successful.
- node #10 spawns job #52, "TEST webservers setup", which finished with status
 successful.
- node #11 spawns job #54, "My Webservers PROD", which finished with status
 successful.
```

```
- node #12 spawns job #55, "PROD webservers setup", which finished with status
 successful.
```

7.3. Exit the console session on the **tower** system.

```
[student@tower ~]$ exit
```

8. Verify that the web servers have been updated on **serverc.lab.example.com**, **serverd.lab.example.com**, **servere.lab.example.com**, and **serverf.lab.example.com**.

8.1. Open a web browser and go to **http://serverc.lab.example.com**, **http://serverd.lab.example.com**, **http://servere.lab.example.com**, and **http://serverf.lab.example.com** in separate tabs. You should see this line at the bottom of each page:

```
Deployment Version: v1.3
```

9. Run the command **lab project-review grade** on **workstation** to grade your exercise.

Summary

In this chapter, you learned:

- Surveys can be added to Job Surveys to enable push-button automation by prompting for validated user input with user-friendly questions.
- Workflow Job Templates can mimic the decision tree behind complicated processes by chaining together multiple Job Templates while establishing decision-making relationships between them.
- Notification Templates can be used to generate notifications of job execution success and failure results. Tower supports the use of many different mechanisms for generating notifications.

CHAPTER 6

UPDATING INVENTORIES DYNAMICALLY AND COMPARING INVENTORY MEMBERS

Overview	
Goal	Use advanced techniques to work with inventories, including dynamic generation of inventories from centralized information sources and monitoring of hosts in an inventory for configuration deviations or differences
Objectives	• Create a dynamic inventory that uses a custom inventory script to set hosts and hostgroups • Detect differences between systems in an inventory by using scheduled scan jobs and System Tracking
Sections	• Creating and Updating Dynamic Inventories (and Guided Exercise) • Monitoring Configuration Drift with Scan Jobs (and Guided Exercise)
Lab	• Updating Inventories Dynamically and Comparing Inventory Members

Creating and Updating Dynamic Inventories

Objectives

After completing this section, students should be able to create a dynamic inventory that uses a custom inventory script to set hosts and hostgroups.

Dynamic Inventories

When Ansible runs a playbook, it uses an *inventory* to help determine against which hosts the plays should be run. Both Ansible and Ansible Tower make it easy to set up a *static inventory* of hosts which are explicitly specified in the inventory by the administrator.

However, these static lists require manual administration in order to keep them up to date. This can be inconvenient or challenging, especially when an organization wants to run the playbooks against hosts which are dynamically created in a virtualization or cloud computing environment.

In that scenario, it is useful to use a *dynamic inventory*. Dynamic inventories are scripts that, when run, dynamically determine which hosts and host groups should be in the inventory based on information from some external source. These can include the API for cloud providers, Cobbler, LDAP directories, or other third party CMDB software. Using a dynamic inventory is a recommended practice in a large and fast changing IT environment, where systems are frequently deployed, tested and then removed.

By default, Ansible Tower comes with built-in dynamic inventory support for a number of external inventory sources (or *cloud inventory sources*), including:

- Amazon EC2
- Google Compute Engine
- Microsoft Azure Resource Manager
- VMware vCenter
- Red Hat Satellite 6
- Red Hat CloudForms
- OpenStack

In addition, it is possible to use custom dynamic inventory scripts in Ansible Tower to access inventory information from other sources.

The remainder of this section looks at three examples of dynamic inventory configuration in Ansible Tower. The first looks at the built-in support for OpenStack. The second briefly examines the built-in support for getting information from a Red Hat Satellite 6 server. The third investigates how custom dynamic inventory scripts may be used.

OpenStack Dynamic Inventories

Cloud technologies like OpenStack bring many changes to the server lifecycle. Hosts come and go over time, and they can be created and started by external applications. Maintaining

an accurate static inventory file is challenging over any length of time. Therefore, having the inventory update dynamically based on information provided directly from OpenStack is very helpful.

The basic process for configuring dynamic inventories using any of the built-in *cloud sources* is similar for each of them:

1. Create a Credential to authenticate to the cloud data source you intend to use, using a credential type which matches the source
2. Create a new Inventory to provide dynamic inventory information
3. In that new Inventory, create a Group which has its **Source** set to one of the built-in dynamic inventory sources (instead of **Manual**). It should also use the new Credential to authenticate to that source. You may also want to set other options, like having it automatically **Update on Launch**.
4. Update the Group in the Inventory for the first time.

This process works for OpenStack dynamic inventories. This is an example of how to create credentials for use by an OpenStack dynamic inventory:

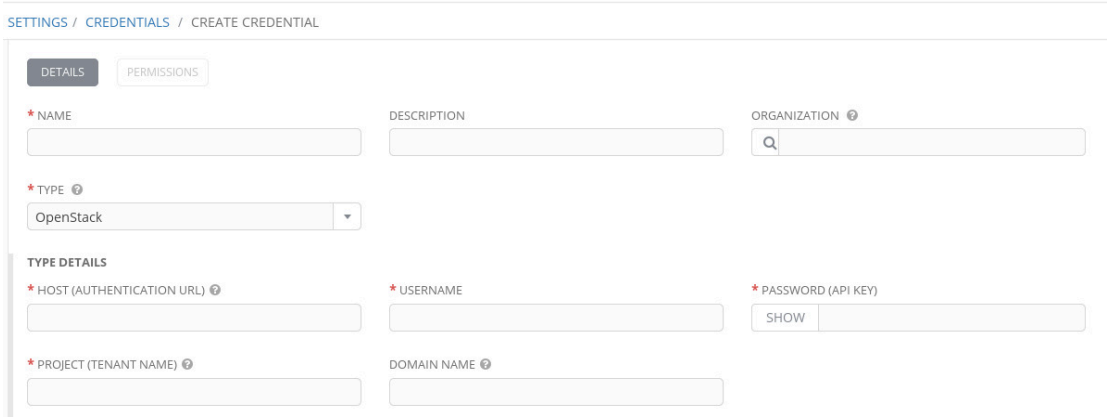

Figure 6.1: Cloud credentials creation

As you can see in this screenshot, there are a number of items you need to provide. You've used some of these with other objects in Ansible Tower: **Name**, **Description** and **Organization**. The only new item is the **Type** of the credentials. You have to choose the appropriate one for the product you are using. In this example, it is **OpenStack**.

An OpenStack Credential needs some additional information:

- **Host** - the authentication URL of the host to authenticate with, for example *https://demo.lab.example.com/v2.0/*
- **Username** - the user who can access the required resources
- **Password** for that user
- **Project** - the name of the project (tenant) you want to use
- **Domain Name** - needed only with Keystone v3, defines administrative boundaries

Those newly created credentials are going to be used by the Ansible Tower Inventory synchronization mechanism. After creating the credentials, you can switch to the **INVENTORIES** link. As you can see in the example below, you need to create a new Inventory:

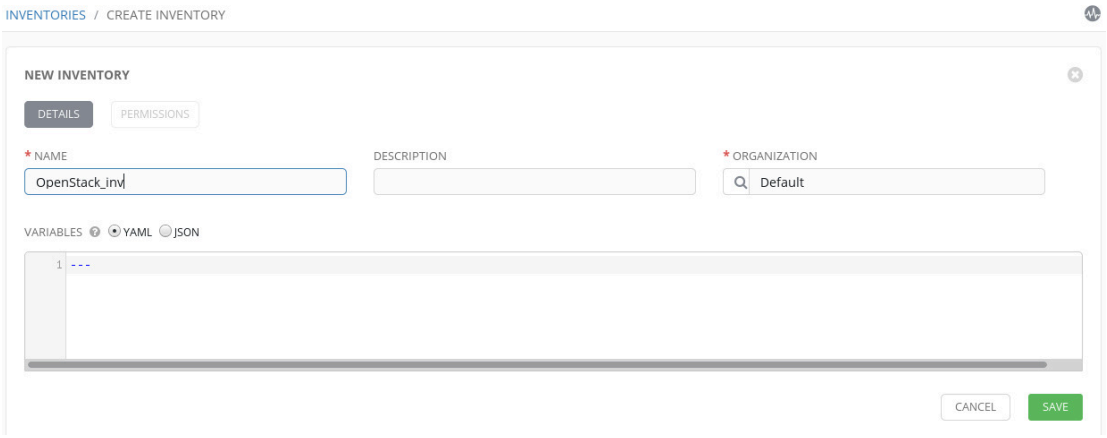

Figure 6.2: Cloud inventory creation

That new Inventory needs a unique **NAME** and you need to assign it to an existing **ORGANIZATION**. When the new Inventory configuration is saved, you will be able to create a new Group. Ansible Tower uses that Group in conjunction with the existing OpenStack scripts and the previously created credentials. The screenshot below shows an example of such a Group:

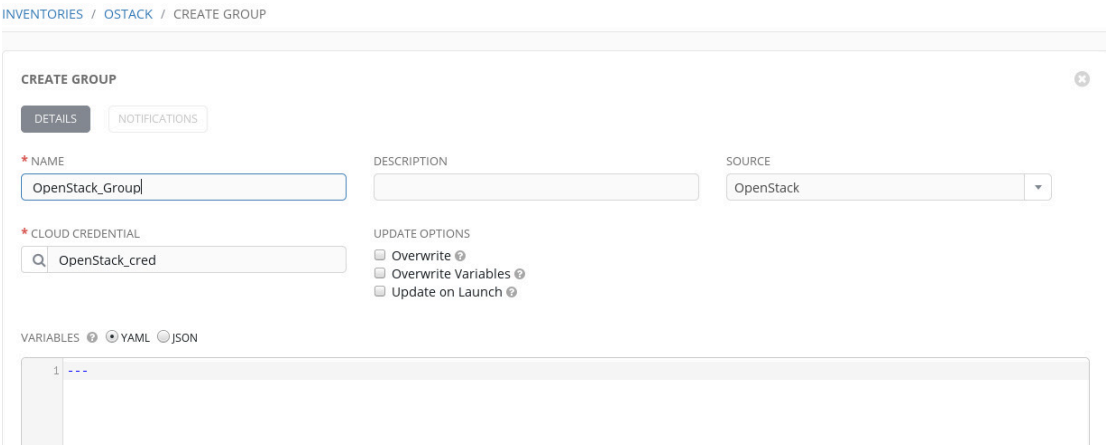

Figure 6.3: Cloud group creation

This new Group uses the built-in Ansible Tower support for OpenStack as **SOURCE** and the new credentials for your OpenStack environment as **CLOUD CREDENTIAL**. There are three **UPDATE OPTIONS** available to choose:

- **Overwrite** - when this option is activated, the Inventory update process deletes all child groups and hosts from the local Inventory, not found on the external source. By default it is not active, it means that all child hosts and groups not found on the external source remains untouched.
- **Overwrite Variables** - when not activated, a merge combining local variables with those found on the external source, is performed. Otherwise, when activated all variables not found on the external source is removed.

- **Update on Launch** - when activated, each time a job runs using this Inventory, a refresh of this Inventory from the external source is performed, before the job tasks are executed.

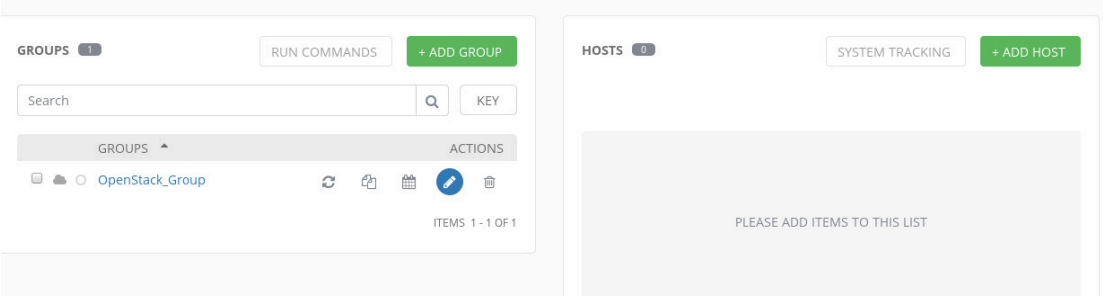

Figure 6.4: New external Inventory source created

This screenshot shows the newly created Group within the new Inventory. The small cloud icon left of the Group name shows the status of the dynamic inventory synchronization for that group. Since it is gray, currently no status is available. You can start the synchronization process manually by clicking the two-arrows icon. When the synchronization finishes, the status of the cloud icon changes to green if synchronization has been successful, or red if it failed.

After a successful synchronization with the external source, you can review the child Groups and Hosts which have been created in Tower using the information from the external source. The following screenshot shows an example of child Groups and hosts created by a dynamic inventory using a cloud source:

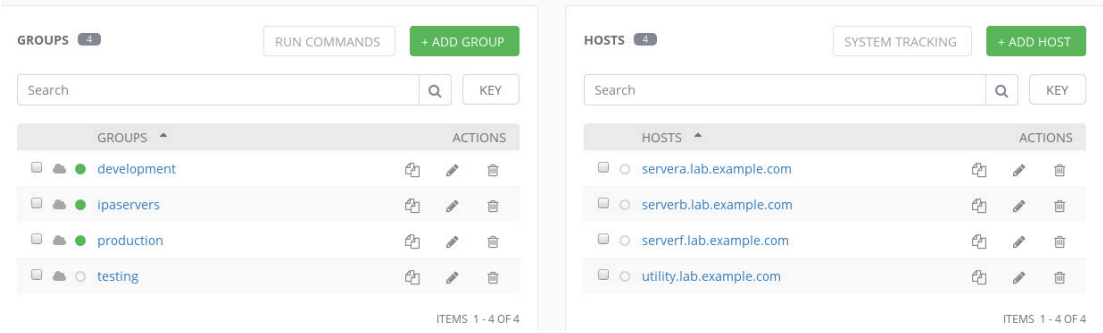

Figure 6.5: Child groups and hosts created with the use of an external source

The child groups contain the hosts visible on the **HOSTS** lists. You can review each child group by clicking on the group name. This displays the content of that group, giving you a list of the hosts associated with that group. This Inventory is updated every time you synchronize it with the external source. That action can be performed manually, scheduled using the Tower mechanisms or performed automatically each time a job runs using that Inventory.

Red Hat Satellite 6 Dynamic Inventories

Another example of a built-in dynamic inventory uses information about hosts that are registered with a **Red Hat Satellite 6** server. Red Hat Satellite is a system management tool that enables you to deploy, configure, and maintain your systems across physical, virtual, and cloud environments.

Workflow for deployment and configuration of a new bare-metal server using Red Hat Satellite 6 in conjunction with Ansible Tower might look like this:

1. The new server uses some combination of PXE, DHCP, and TFTP to boot from the network or a boot ISO to prepare for either an unattended installation from the Satellite server or installation through Satellite's Discovery service

2. The new server performs a Kickstart installation from materials provided by the Satellite server and registers itself to the Satellite server

3. The new server appears in the dynamic inventory generated from Red Hat Satellite information once it's registered. Ansible Tower can now be used to launch various jobs using that inventory to ensure that the new server is provisioned correctly

Note

The *Provisioning Callbacks* feature of Ansible Tower is particularly useful for triggering initial provisioning jobs when a new server is deployed. More information on this feature is available in the documentation at *http://docs.ansible.com/ansible-tower/latest/html/userguide/job_templates.html#provisioning-callbacks/*.

Configuration of an Ansible Tower dynamic inventory using Red Hat Satellite 6 is very similar to the OpenStack scenario.

The first step is to create a new Credential. The **Type** of the Credential in this case will be **Red Hat Satellite 6**. It will require three pieces of additional information:

- **Satellite 6 URL** - the URL for the Satellite server, such as "*https://satellite.example.com*"
- **Username** - for a user on the Satellite server
- **Password** - password for the Satellite user

The next step is to create a Group in an Inventory that will be used to synchronize inventory data from the Satellite server. Just like the OpenStack example, the group needs a **NAME**, but the **SOURCE** should be set to **Red Hat Satellite 6**. The Group's **CLOUD CREDENTIAL** should be set to the Credential you just created for Satellite. Just like the OpenStack dynamic inventory configuration, the three **UPDATE OPTIONS** (**Overwrite**, **Overwrite Variables**, and **Update on Launch**) may be selected as desired.

The last step is to synchronize the Group with the Red Hat Satellite inventory source, in exactly the same way as discussed in the section on the OpenStack inventory source. When the synchronization finishes, all information gathered from the external source are visible in Ansible Tower web interface as child groups and hosts associated with those groups.

Custom Dynamic Inventory Scripts

Ansible allows you to write custom scripts to generate a dynamic inventory. While Ansible Tower offers built-in support for a number of dynamic inventory sources, custom dynamic inventory scripts can still be used with Ansible Tower.

Writing or Obtaining Custom Inventory Scripts

Ansible Tower supports custom inventory scripts written in any dynamic language installed on the Ansible Tower server. This should include Python and Bash at a minimum. These scripts run as the **awx** user and have limited access to the Tower server. The script must start with an appropriate shebang line (for example, **`#!/usr/bin/python`** for a Python script).

Many examples of custom inventory scripts for use with various external sources have been contributed by the community the Git repository for Ansible at *https://github.com/ansible/ansible/tree/devel/contrib/inventory/*.

If you want to write your own custom inventory script, information is available at *Developing Dynamic Inventory Sources* [http://docs.ansible.com/ansible/dev_guide/developing_inventory.html] in the *Ansible Developer Guide*. When the dynamic inventory script is called with the **`--list`** option, it must output the inventory in JSON format.

This is example output from a custom dynamic inventory script:

```
{
   "databases" : {
      "vars" : {
         "example_db" : true
      },
      "hosts" : [
         "db1.demo.example.com",
         "db2.demo.example.com"
      ]
   },
   "webservers" : [
      "web1.demo.example.com",
      "web2.demo.example.com"
   ],
   "boston" : {
      "children" : [
         "backup",
         "ipa"
      ],
      "vars" : {
         "example_host" : false
      },
      "hosts" : [
         "server1.demo.example.com",
         "server2.demo.example.com",
         "server3.demo.example.com"
      ]
   },
   "backup" : [
      "server4.demo.example.com"
   ],
   "ipa" : [
      "server5.demo.example.com"
   ]
}
```

As you can see in the preceding example, each group may contain a list of hosts, potential child groups, possible group variables or a list of hosts.

Note

When called with the option **`--host hostname`**, the script must print a JSON hash/dictionary of the variables for the specified host (potentially an empty JSON hash or dictionary if there are no variables provided).

Optionally, if the **`--list`** option returns a top level element called **`_meta`**, it is possible to return all host variables in one script call, which improves script performance. In that case, **`--host`** calls will not be made.

See the previously referenced documentation in the *Developing Dynamic Inventory Sources* section of the *Ansible Developer Guide* for more information.

Using Custom Inventory Scripts in Ansible Tower

Once you have created or downloaded the appropriate custom inventory script, you need to import it into Ansible Tower and configure the Inventory. Below is a procedure how to accomplish this task:

To upload a custom inventory script into Tower, go to **Settings**. From the available options, choose **INVENTORY SCRIPTS**.

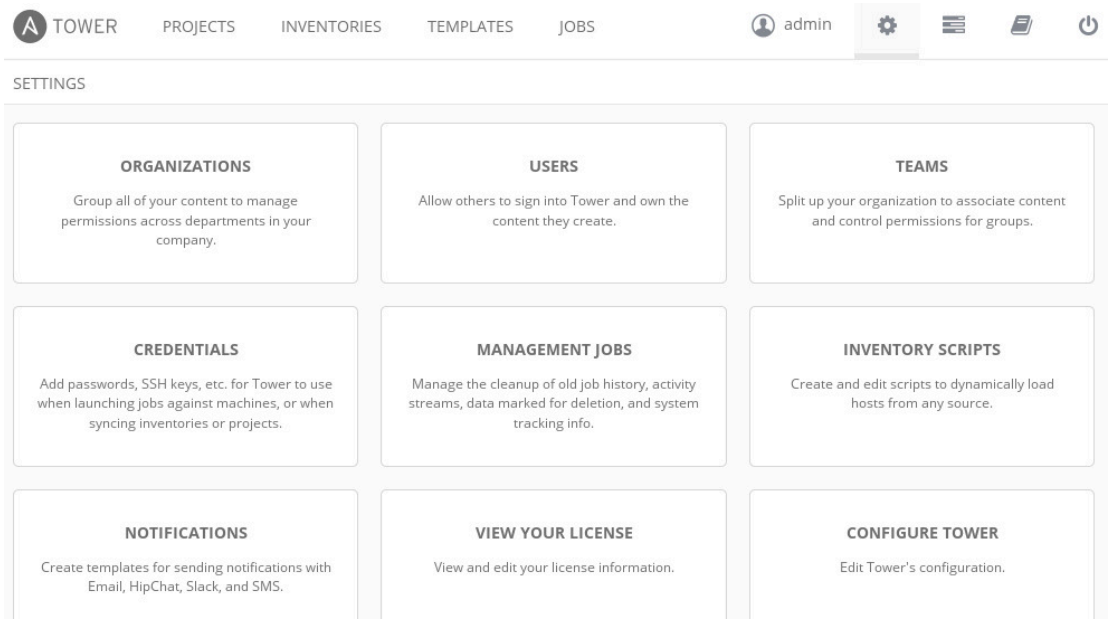

Figure 6.6: Ansible Tower Settings

You can now add a new custom inventory script by clicking the **+ADD** button.

Define a new name for the custom inventory script in the **NAME** field, and copy and paste the actual script in to the **CUSTOM SCRIPT** text box. Click the **SAVE** button.

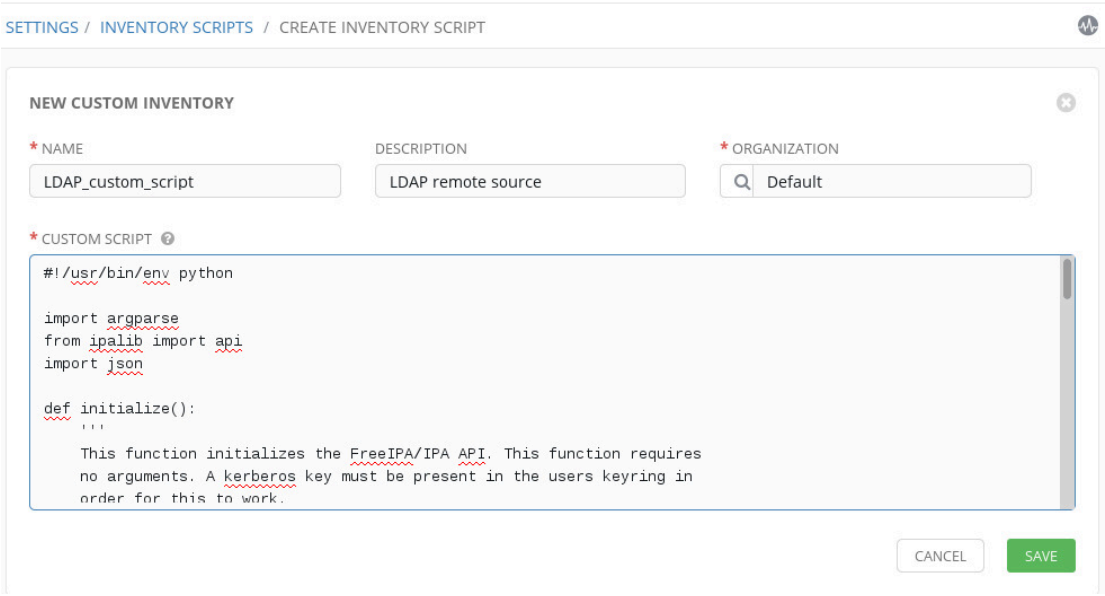

Figure 6.7: Adding new custom inventory script

Once the dynamic inventory script has been installed in Ansible Tower, you configure it just like any of the built-in dynamic inventories:

1. Create a new Group in an Inventory for the dynamic inventory. Set its **SOURCE** to **Custom Script** and **CUSTOM INVENTORY SCRIPT** to the name you set for the custom script that you just imported into Ansible Tower.

2. Synchronize the Group with the inventory source, as discussed for the OpenStack and Red Hat Satellite 6 dynamic inventory sources.

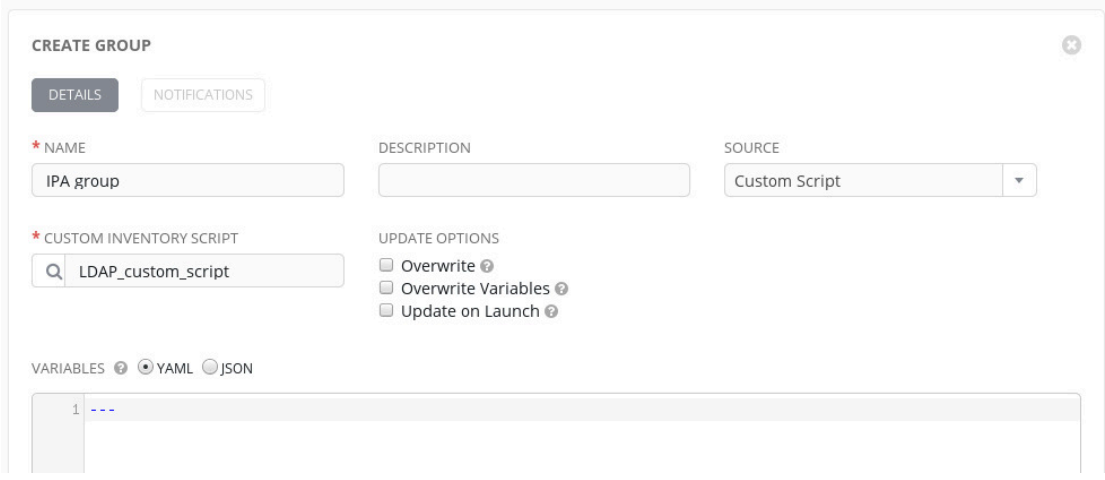

Figure 6.8: Adding new Inventory Group for custom dynamic inventory

References

Further information on configuring dynamic inventories in Ansible Tower can be found in the *Ansible Tower User Guide*:

> Inventories
> https://docs.ansible.com/ansible-tower/3.1.1/html/userguide/inventories.html

Further information on writing dynamic inventory scripts is available in the *Ansible Developer Guide*:

> Developing Dynamic Inventory Sources
> http://docs.ansible.com/ansible/dev_guide/developing_inventory.html

Guided Exercise: Creating and Updating Dynamic Inventories

In this exercise, you will add a custom inventory script and use it to manage a Dynamic Inventory managed on an IPA server.

Outcomes
You should be able to add a custom inventory script and use it to populate a Dynamic Inventory in Tower.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab data-inventory setup**. This setup script will prepare the IPA server for this exercise.

```
[student@workstation ~]$ lab data-inventory setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Add the **ldap-freeipa.py** custom inventory script to Tower.

   2.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   2.2. Click **INVENTORY SCRIPTS** to manage custom inventory scripts.

   2.3. Click **+ADD** to add a custom inventory script.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	ldap-freeipa.py
DESCRIPTION	Dynamic Inventory for IPA Server
ORGANIZATION	Default

   2.5. Copy the contents of the **ldap-freeipa.py** script located at **http://materials.example.com/classroom/ansible/ipa-setup/ldap-freeipa.py** into the **CUSTOM SCRIPT** field.

   2.6. Click **SAVE** to add the custom inventory script.

3. Create a new Inventory called **Dynamic Inventory**.

   3.1. Click the **INVENTORIES** quick navigation link.

   3.2. Click **+ADD** to create a new Inventory.

3.3. On the next screen, fill in the details as follows:

Field	Value
NAME	Dynamic Inventory
DESCRIPTION	Dynamic Inventory from IPA server
ORGANIZATION	Default

3.4. Click **SAVE** to create the Inventory. This redirects you to the Inventory details page.

4. Create a Group called **`Dynamic Group`** within the Inventory.

4.1. Click **+ADD GROUP** to add a Group to the Inventory.

4.2. On the next screen, fill in the details as follows:

Field	Value
NAME	Dynamic Group
DESCRIPTION	Dynamic Group from IPA server
SOURCE	Custom Script
CUSTOM INVENTORY SCRIPT	ldap-freeipa.py

4.3. Under the **UPDATE OPTIONS** section, check the checkbox next to the **Overwrite** option.

4.4. Click **SAVE** to create the Group.

5. Update the Dynamic Inventory.

5.1. Scroll down to the **GROUPS** section and click the **Start sync process** icon on the same line as the group, **`Dynamic Group`**.

5.2. When the update of the Inventory has completed click on **Dynamic Group**.

5.3. Observe that **`Dynamic Group`** contains four Groups: **`development`**, **`ipaservers`**, **`production`**, and **`testing`**. Each of these groups contains hosts.

Monitoring Configuration Drift with Scan Jobs

Objective

After completing this section, students should be able to detect differences between systems in an inventory by using scheduled scan jobs and System Tracking.

Scan Jobs and System Tracking

One challenge in system administration is to ensure servers that are supposed to have identical configurations continue to have identical configurations. Systems can be inappropriately changed through local modifications, or software updates can be applied that cause additional but unexpected updates to be applied. *Configuration drift* happens when two systems diverge from a common configuration, or a single system diverges from an approved configuration.

Minor changes and differences in configuration that happen through configuration drift can eventually result in unexpected problems or downtime. This is one reason why the logs of Ansible playbook runs are important, because they can record when a task run by the playbook results in an unexpected change to a managed host. It can also be useful to directly compare how the configuration of a system changes over time, or how the configurations of two systems differ.

Ansible Tower supports *scan jobs*, which collect information about managed hosts and stores it in the Tower database. The results of the scan jobs can be compared using Ansible Tower's *System Tracking* feature. System Tracking can compare two scan runs of the same host from different dates, or a scan run of two hosts on the same date.

Scan Jobs

Creating a Scan Job Template

Scan Jobs are launched using a special Job Template type, a *Scan Job Template*. You can use this Job Template type to run a special type of Ansible playbook, that scans your managed hosts and sends back the gathered system facts to Tower, where you can use this information to compare multiple systems with each other or to review the changes made on a particular machine in a given time period.

The first step to set up a Scan Job Template is to create a new Job Template. Click the **TEMPLATES** quick navigation link to go to the management interface for Templates.

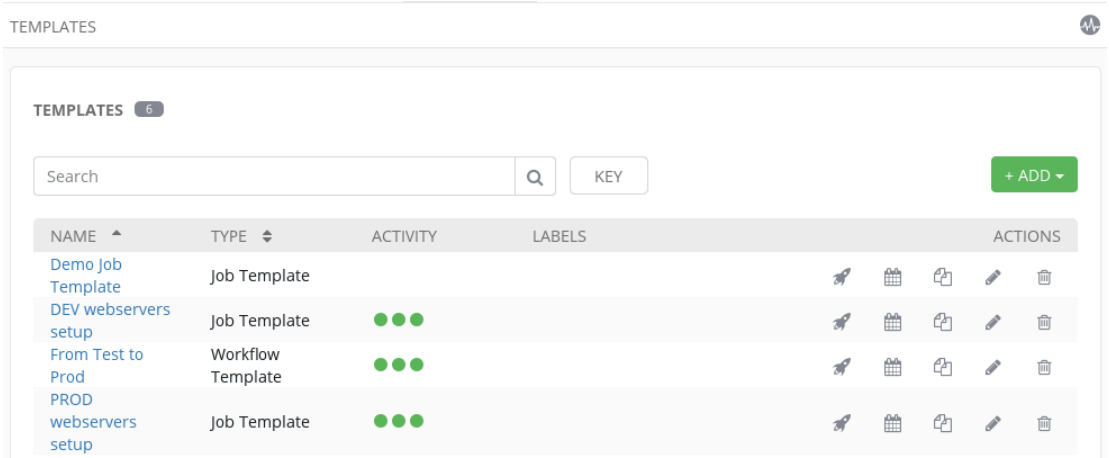

Figure 6.9: List of available Templates

Click the **+ADD** button and select **Job Template**. A **CREATE JOB TEMPLATE** screen should be displayed that looks something like the following screenshot:

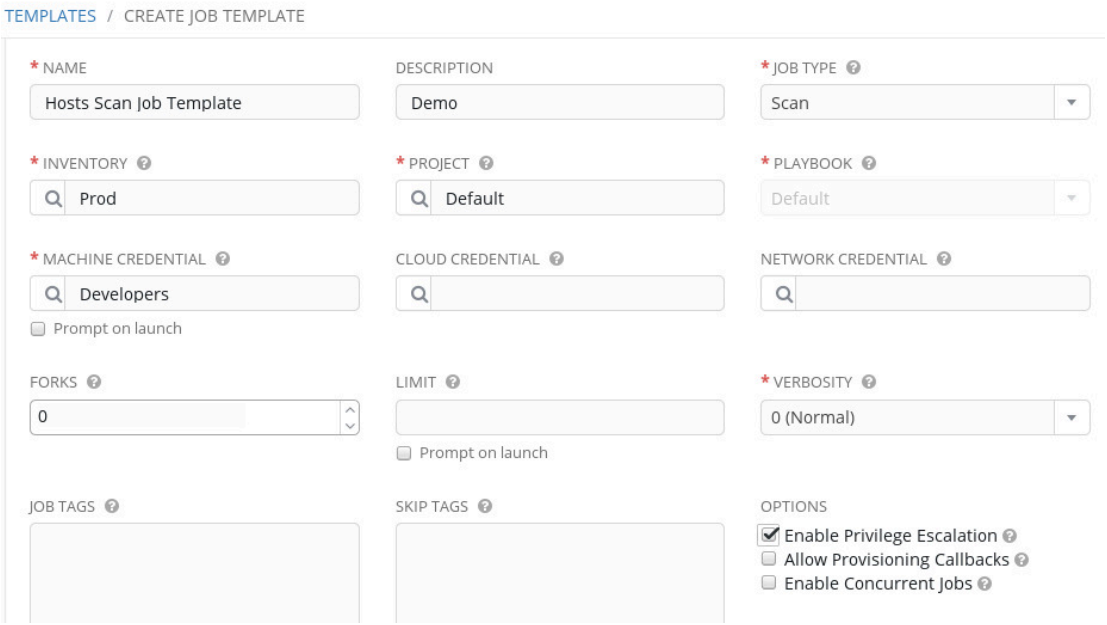

Figure 6.10: Creating new Scan Template

Enter a name for the Job Template in the **NAME** field. You may find it helpful to include the word **scan** in the name of the Scan Job Template to make it easier to find Scan Job Templates in the list of available templates.

Select **Scan** as the **JOB TYPE**.

Click the magnifying glass icon for the **INVENTORY** field. From the list of available Inventories choose the one containing the hosts you want to launch scan jobs on, and then click **SAVE**.

The **PROJECT** is initially set to **Default**. For a basic scan of system facts, that Project has a predefined playbook available. If you change the Project, you must specify a custom playbook to collect facts.

Note

It is possible to create special custom scan job templates that use your own playbooks to collect facts. For more information, see the section *Custom Scan Job Templates* [http://docs.ansible.com/ansible-tower/latest/html/userguide/job_templates.html#custom-scan-job-templates] in the *Ansible Tower User Guide*.

Choose the **MACHINE CREDENTIAL** needed to run the scan playbook on hosts in the specified Inventory.

Choose the desired **VERBOSITY** for the job output.

When the Job Template configuration is complete, click **SAVE**.

At the time of writing, this is a list of managed host operating systems that were supported for default scan jobs according to the *Ansible Tower User Guide*:

- Red Hat Enterprise Linux 5, 6, and 7
- CentOS 5, 6, and 7
- Ubuntu 12.04, 14.04, and 16.04
- Oracle Enterprise Linux 6 and 7
- SuSE Enterprise Linux 11 and 12
- Debian 6 and 7
- Fedora 23, 24, and 25
- Amazon Linux 2016.03

Important

The preceding list is probably not comprehensive and is likely to be outdated by the time you take this course. For example, the authors of this course know of no specific reason why scan jobs would not be supportable on Fedora 26 (although there may be one).

Scan jobs may require that software packages have been installed on the managed host, or that certain initial configuration changes have been made. For example, the scan job's Ansible modules will require that Python scripts can be run on the managed host, and those may depend on specific Python packages.

Launching a Scan Job

A scan job is launched from a Scan Job Template exactly like any other Job is launched from a Job Template, by using the buttons in the last **ACTIONS** column to the right. When you launch the Template, Tower automatically opens the **JOBS** page, displaying real time output from the scan job's playbook.

Scheduling Scan Jobs

Scan Jobs Templates can be launched as scheduled jobs just like regular Job Templates. To be able to compare configuration drift between servers over time, Ansible Tower needs to have historic scan job data available. Scheduling regular scan jobs will allow you to collect the data needed to track this configuration drift.

You can schedule a Scan Job by clicking on the calendar icon ("Schedule future job template runs") for the Scan Job Template on the **TEMPLATES** page. Click the **+ADD** button to set a schedule for a Scan Job.

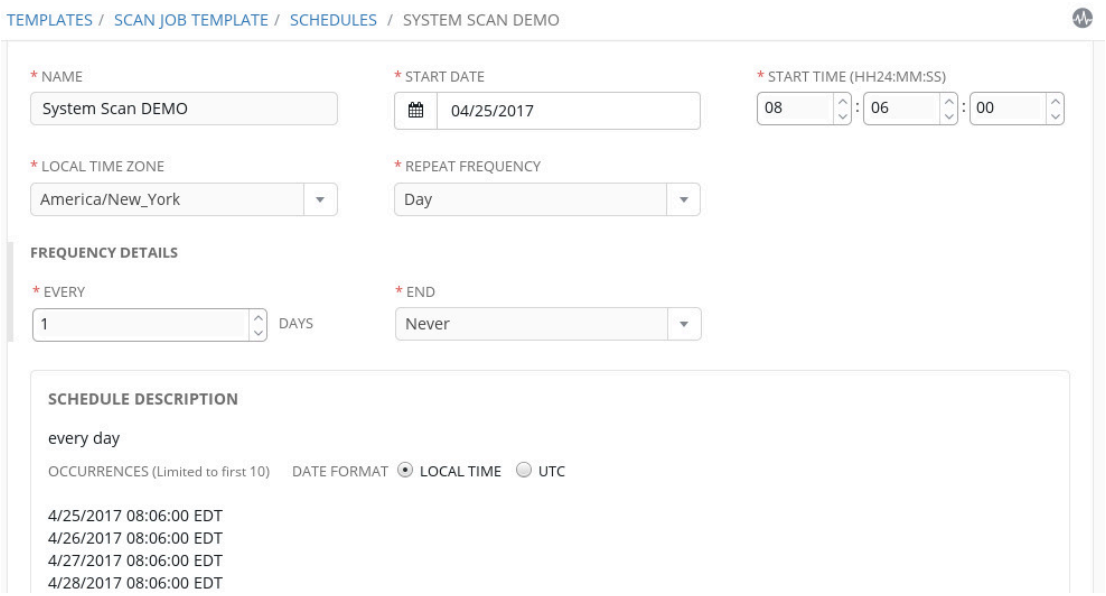

Figure 6.11: Scheduling Scan Template

Specify the requested details in the required fields:

- **NAME** - the name of the schedule
- **START DATE** - the date the job schedule should start
- **START TIME** - the time the associated Job will be launched
- **LOCAL TIME ZONE** - the **`tzselect`** command-line utility can be used to determine your local time zone in this format
- **REPEAT FREQUENCY** - a drop-down menu to select the units for the interval: **None (run once)**, **Minute**, **Hour**, **Day**, **Week**, **Month** or **Year**

  Depending on the chosen frequency, you might need to provide additional information (for example, to launch a job every two days, or on the first Sunday of every month).

When finished, click the **SAVE** button to save the schedule. You can always deactivate or activate a schedule using the **ON/OFF** button.

Note

It probably doesn't make any sense to schedule scan jobs more frequently than once a day. The System Tracking interface does not anticipate more frequent runs, and it would also rapidly increase the size of the database and place extra load on the managed hosts and Ansible Tower.

System Tracking

System Tracking uses the historical data recorded by regularly launched scan jobs to help administrators track changes and configuration drift on managed hosts over time.

System Tracking is performed from the **INVENTORIES** management interface. From the list of available Inventories, click on the name of the inventory containing the host or hosts you would like to analyze.

To compare two scan jobs that run at different times on a single host, in the Inventory's **HOSTS** section, select the checkbox next to the hosts's name and click the **SYSTEM TRACKING** button.

The dates for the scan jobs to compare are selected from the fields **COMPARE LATEST FACTS COLLECTED ON OR BEFORE** and **TO LATEST FACTS COLLECTED ON OR BEFORE**. The results of the comparisons are presented in three categories: **PACKAGES** (shows differences between packages), **SERVICES** (shows differences between configured services), and **ANSIBLE** (shows differences between Ansible facts). You can access each of them by clicking the appropriate button.

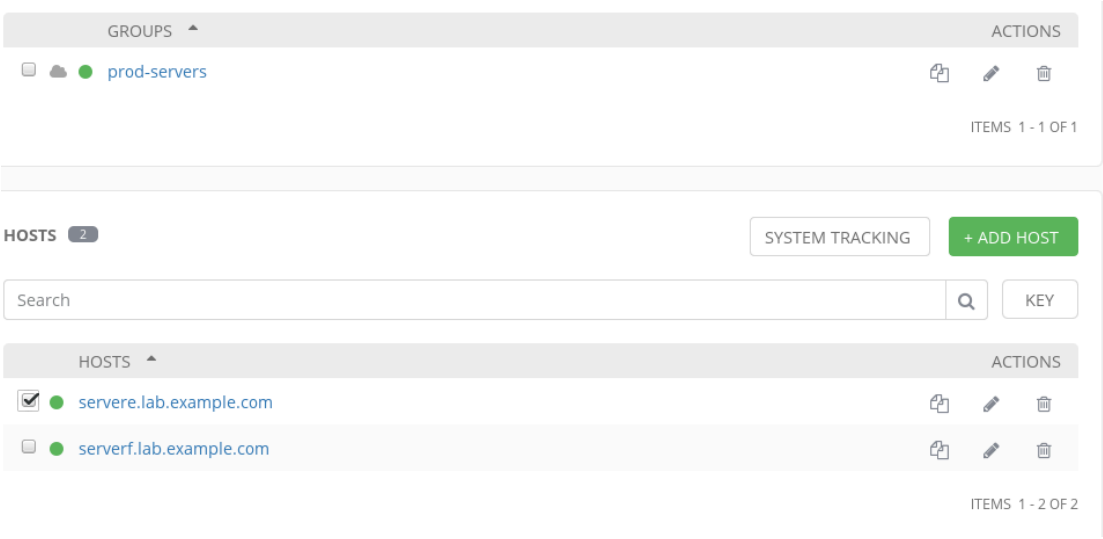

Figure 6.12: System Tracking

Alternatively, it is possible to compare the latest gathered facts between two hosts in the same Inventory on or before a particular date. To do this, in the Inventory's **HOSTS** section, select the checkboxes next to the two hosts that you want to compare, and click the **SYSTEM TRACKING** button.

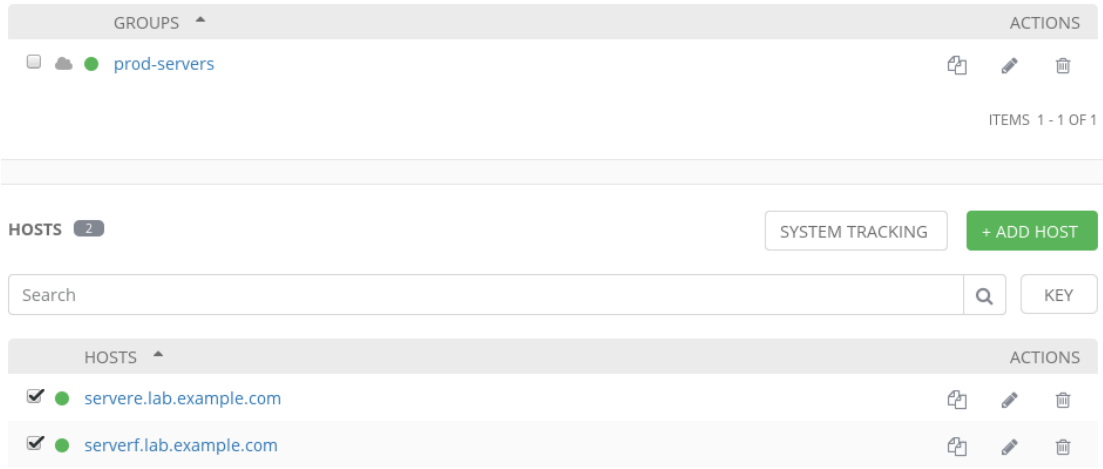

Figure 6.13: Choosing hosts for System Tracking

The System Tracking results page is very similar to the one used to examine two scan job runs on a single host on two dates. In this case, the date of the scan job to use is determined by a single **COMPARE LATEST FACTS COLLECTED ACROSS BOTH HOSTS ON OR BEFORE** field, and there is a column of output for each machine being compared. The example screenshot that follows illustrates this:

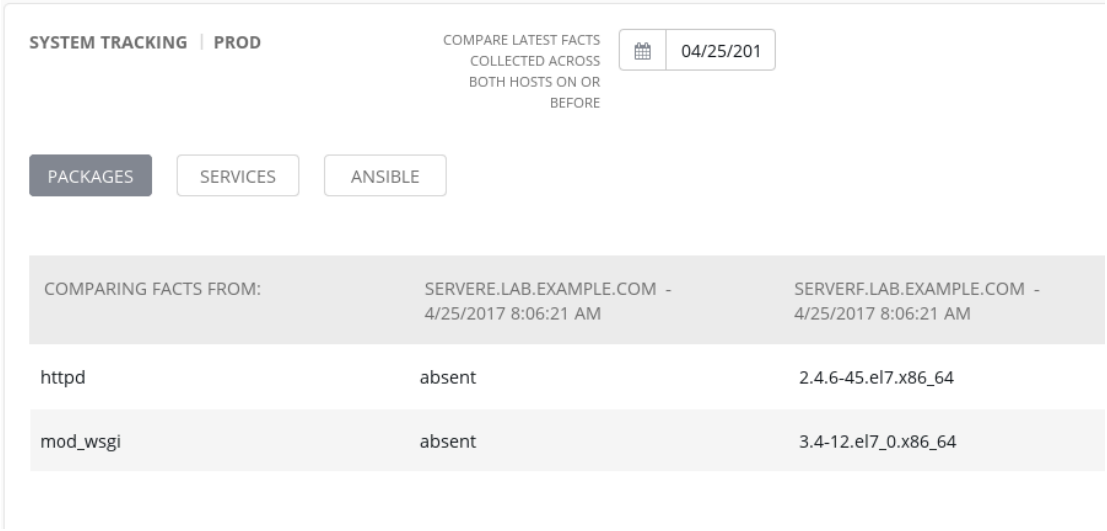

Figure 6.14: Differences between hosts

As you can see in the example, there are differences between those two systems being compared: the server **`servere.lab.example.com`** was missing two packages compared to the list of installed packages on server **`serverf.lab.example.com`**.

You can use System Tracking to compare lists of installed packages, configured services, or other facts gathered by Ansible. For example, the Ansible facts include the amount of memory available or used by the systems.

References

Further information on Scan Job Templates is available in the *Ansible Tower User Guide*:

> https://docs.ansible.com/ansible-tower/3.1.1/html/userguide/job_templates.html#scan-job-templates

Further information on System Tracking is also available in the *Ansible Tower User Guide*:

> https://docs.ansible.com/ansible-tower/3.1.1/html/userguide/inventories.html#system-tracking

Guided Exercise: Monitoring Configuration Drift with Scan Jobs

In this exercise, you will create a new Scan Job Template and use it to compare the drift between two machines.

Outcomes
You should be able to create a Scan Job Template, run it, and compare the configuration drift between machines in the same Inventory.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **`tower`** system.

Log in as the student user on **`workstation`** and run **`lab data-scan setup`**. This setup script will remove the *httpd* package on **`servera`** to simulate a configuration drift from the other servers.

```
[student@workstation ~]$ lab data-scan setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **`tower`** system using the **`admin`** account and the **`redhat`** password.

2. Create new Job Template called **`Scan ALL servers`**.

   2.1. Click the **TEMPLATES** quick navigation link.

   2.2. Click the **+ADD** button to add a new Job Template.

   2.3. From the drop-down list, select **Job Template**.

   2.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Scan ALL servers
DESCRIPTION	Scan all my web servers
JOB TYPE	Scan
INVENTORY	Dynamic Inventory
MACHINE CREDENTIAL	Operations

   2.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

3. Launch a Job using the Job Template, **`Scan ALL servers`**.

   3.1. Scroll down the page to the **TEMPLATES** section and on the same line as the Job Template, **`Scan ALL servers`**, click the rocket icon on the right to launch the Job. This redirects you to a detailed status page of the running job.

3.2. Wait about a minute and then verify that the **STATUS** of the job in the **DETAILS** section is **`Successful`**.

4. Observe the configuration drift between **`servera`** and **`serverb`**.

4.1. Click the **INVENTORIES** quick navigation link.

4.2. Click **Dynamic Inventory** to view the contents the Dynamic Inventory you created in the previous section.

4.3. In the **HOSTS** pane, check the checkboxes next to the hosts **`servera.lab.example.com`** and **`serverb.lab.example.com`**.

4.4. Click **SYSTEM TRACKING**. This will redirect you to the **SYSTEM TRACKING** page.

4.5. Observe the fact that the packages, *httpd* and *mod_wsgi*, display **`absent`** under the column, **SERVERA.LAB.EXAMPLE.COM**, whereas you can see their respective package versions under the column, **SERVERB.LAB.EXAMPLE.COM**.

Lab: Updating Inventories Dynamically and Comparing Inventory Members

In this exercise, you will synchronize a Dynamic Inventory and compare the drift between two of its servers.

Outcomes
You should be able to synchronize a Dynamic Inventory that uses a custom inventory script. You should also be able to run and compare the results of two scan job runs.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab data-review setup**. This setup script will modify the inventory on the IPA server and make server modifications necessary for the exercise.

```
[student@workstation ~]$ lab data-review setup
```

Steps
1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.
2. Synchronize the Inventory, **Dynamic Inventory**.
3. Observe the new configuration of the hosts inside the Inventory and identify which hosts are in which Group in this updated Inventory.
4. Determine how **servere**'s current package and service configuration differs from that of **serverf**.
5. Resolve the package and service configuration drift on **servere** using **serverf** as the reference.
6. Verify that **servere** no longer has a configuration drift from that of **serverf**.
7. Click the **Log Out** icon to log out of the Tower web interface.
8. Run **lab data-review grade** on **workstation** to grade your work.

Solution

In this exercise, you will synchronize a Dynamic Inventory and compare the drift between two of its servers.

Outcomes
You should be able to synchronize a Dynamic Inventory that uses a custom inventory script. You should also be able to run and compare the results of two scan job runs.

Before you begin
You should have an Ansible Tower instance installed and configured from an earlier exercise running on the **tower** system.

Log in as the **student** user on **workstation** and run **lab data-review setup**. This setup script will modify the inventory on the IPA server and make server modifications necessary for the exercise.

```
[student@workstation ~]$ lab data-review setup
```

Steps

1. Log in to the Ansible Tower web interface running on the **tower** system using the **admin** account and the **redhat** password.

2. Synchronize the Inventory, **Dynamic Inventory**.

   2.1. Click the **INVENTORIES** quick navigation link.

   2.2. Click **Dynamic Inventory** to view the contents of the Inventory.

   2.3. On the same line as the Group, **Dynamic Group**, click the **Start sync process** icon on the right to launch the job. The green cloud status icon will move while the job is running and will stay green and steady when the sync is finished.

3. Observe the new configuration of the hosts inside the Inventory and identify which hosts are in which Group in this updated Inventory.

   3.1. Click the Group, **Dynamic Group**. Click the Group, **production**.

   3.2. Observe the list of hosts in the Group, **production**. It now contains **servera.lab.example.com** and **serverb.lab.example.com**.

   3.3. Go back to the Group, **Dynamic Group**, by clicking the **Dynamic Group** breadcrumb navigation link. Click on the Group, **development**.

   3.4. Observe the list of hosts in the Group, **development**. It now contains **servere.lab.example.com** and **serverf.lab.example.com**.

4. Determine how **servere**'s current package and service configuration differs from that of **serverf**.

   4.1. Click the **TEMPLATES** quick navigation link.

   4.2. Scroll down the page to the same line as the Job Template, **Scan ALL servers**, and click the rocket icon on the right to launch the job. It redirects you to a detailed status page of the running job.

4.3. Wait several seconds and then verify that the **STATUS** of the job in the **DETAILS** section is **Successful**.

4.4. Click the **Inventories** quick navigation link. Click **Dynamic Inventory** to view the contents of the Inventory. Click the Group, **Dynamic Group**, and then click the Group, **development**.

4.5. In the **HOSTS** pane, check the boxes next to **servere.lab.example.com** and **serverf.lab.example.com** and then click **SYSTEM TRACKING**.

4.6. Make sure that you are in the **PACKAGES** section, and observe that the package *wget* displays **absent** under the column **SERVERE.LAB.EXAMPLE.COM**, whereas you can see its version under the column **SERVERF.LAB.EXAMPLE.COM**.

4.7. Click **SERVICES** and observe the fact that the service **httpd** displays **stopped** under the column **SERVERE.LAB.EXAMPLE.COM** whereas it shows **running** under the column **SERVERF.LAB.EXAMPLE.COM**.

5. Resolve the package and service configuration drift on **servere** using **serverf** as the reference.

5.1. Log in to **servere** as the **student** user.

```
[student@workstation ~]$ ssh servere
Last login: Thu May  4 16:20:54 2017 from tower.lab.example.com
[student@servere ~]$
```

5.2. Install the *wget* package using the password, **student**, when prompted by **sudo**.

```
[student@servere ~]$ sudo yum install wget -y
[sudo] password for student: student
Loaded plugins: langpacks, search-disabled-repos
Resolving Dependencies
--> Running transaction check
... Output Omitted ...
Installed:
  wget.x86_64 0:1.14-13.el7

Complete!
```

5.3. Start and enable the **httpd** service using the password, **student** if prompted by **sudo**.

```
[student@servere ~]$ sudo systemctl start httpd && sudo systemctl enable httpd
[sudo] password for student: student
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service
 to /usr/lib/systemd/system/httpd.service.
```

6. Verify that **servere** no longer has a configuration drift from that of **serverf**.

6.1. Click the **TEMPLATES** quick navigation link.

6.2. On the same line as the Job Template **Scan ALL servers**, click the rocket icon on the right to launch the job. This redirects you to a detailed status page of the running job.

6.3. Wait several seconds and verify that the **STATUS** of the job in the **DETAILS** section is **`Successful`**.

6.4. Click the **INVENTORIES** quick navigation link. Click **Dynamic Inventory** to view the contents of the Inventory.

6.5. In the list of hosts in the right pane, select the checkboxes next to **`servere.lab.example.com`** and **`serverf.lab.example.com`** and click **SYSTEM TRACKING**.

6.6. Verify that both the **PACKAGES** and **SERVICES** sections of the report display the text **`THE TWO FACT SCANS WERE IDENTICAL FOR THIS MODULE.`**

7. Click the **Log Out** icon to log out of the Tower web interface.

8. Run **`lab data-review grade`** on **`workstation`** to grade your work.

Summary

In this chapter, you learned:

- How to create a dynamic inventory that uses a custom inventory script to set hosts and hostgroups.
- How to detect differences between systems in an inventory by using scheduled scan jobs and System Tracking.

CHAPTER 7

PERFORMING MAINTENANCE AND ROUTINE ADMINISTRATION OF ANSIBLE TOWER

Overview	
Goal	Perform routine maintenance and administration on Ansible Tower and have a basic familiarity with the command line tools and the Ansible Tower API
Objectives	• Identify the locations of Ansible Tower log files and use information in them for basic troubleshooting. • Replace the default TLS/SSL certificate used by Ansible Tower with a provided certificate appropriate for the server's hostname. • Reset the admin password, or import an existing inventory file, using the tower-manage utility. • Backup and restore the Ansible Tower system and its databases and configuration settings. • Launch a job from an existing template using the tower-cli utility and the Tower API.
Sections	• Performing Basic Troubleshooting of Ansible Tower (and Guided Exercise) • Configuring TLS/SSL for Ansible Tower (and Guided Exercise) • Performing Command-Line Management with tower-manage (and Guided Exercise) • Backing Up and Restoring an Ansible Tower Installation (and Guided Exercise) • Launching Jobs with tower-cli and the Ansible Tower API (and Guided Exercise)

Overview	
Quiz	• Performing Maintenance and Routine Administration of Ansible Tower

Performing Basic Troubleshooting of Ansible Tower

Objectives

After completing this section, students should be able to:

- Identify the locations of Ansible Tower log files and use information in them for basic troubleshooting

Ansible Tower Components

Ansible Tower is a web application made up of a number of cooperating processes and services. Four main network services are enabled, which start the rest of the components of Ansible Tower:

- *nginx* provides the web server that hosts the Tower application and supports the web interface and the API
- *PostgreSQL* is the database that stores most Tower data, configuration, and history
- *supervisord* is a process control system that itself manages the various components of the Tower application to do things like schedule and run jobs, listen for callbacks from running jobs, and so on
- *rabbitmq-server* is an AMQP message broker that is used to support signaling for the Tower application components

A fifth component also used by Ansible Tower is the **`memcached`** memory object caching daemon, which is used as a local caching service.

These network services communicate with each other using normal network protocols. For a normal self-contained Tower server, the main ports that need to be exposed outside the system are 80/tcp and 443/tcp, to allow clients to access the web interface and API.

However, the other services may also expose ports to external clients unless specifically protected. For example, the PostgreSQL service listens for connections from anywhere on 5432/tcp, and the RabbitMQ server **`beam`** is listening for connections on 5672/tcp, 15672/tcp, and 25672/tcp. If only the local Tower services need to be able to connect to these ports, it may be desirable to block access to them using the local firewall.

Warning

This is one reason why setting good passwords for the PostgreSQL and RabbitMQ services in the **`inventory`** used to install Tower is important. These services can be contacted by Internet clients directly by default, and weak passwords may leave them vulnerable to remote attack.

Starting, Stopping and Restarting Tower

Ansible Tower comes with a **/usr/bin/ansible-tower-service** script, which can start, stop, restart, and give the status of the major Tower services, including the database and message queue components.

```
[root@tower ~]# ansible-tower-service status
Showing Tower Status
● postgresql-9.4.service - PostgreSQL 9.4 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-9.4.service; enabled; vendor
 preset: disabled)
...output omitted...
Status of node rabbitmq@localhost ...
[{pid,1554},
 {running_applications,
...output omitted...
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset:
 disabled)
   Active: active (running) since czw 2017-04-13 03:57:56 EDT; 3h 58min ago
...output omitted...
● supervisord.service - Process Monitoring and Control Daemon
   Loaded: loaded (/usr/lib/systemd/system/supervisord.service; enabled; vendor preset:
 disabled)
   Active: active (running) since czw 2017-04-13 03:57:57 EDT; 3h 58min ago
```

To access the list of available options, issue the **ansible-tower-service** command, without any options:

```
[root@tower ~]# ansible-tower-service
Usage: /usr/bin/ansible-tower-service {start|stop|restart|status}
```

This is an example of restarting the Tower infrastructure:

```
[root@tower ~]# ansible-tower-service restart
Restarting Tower
Redirecting to /bin/systemctl stop  postgresql-9.4.service
Stopping rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl stop  nginx.service
Redirecting to /bin/systemctl stop  supervisord.service
Redirecting to /bin/systemctl start  postgresql-9.4.service
Starting rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl start  nginx.service
Redirecting to /bin/systemctl start  supervisord.service
```

Supervisord Components

supervisord is a process control system often used to control Django-based applications. It is used to manage and monitor long running processes or daemons, and to automatically restart them as needed. In Ansible Tower, **supervisord** manages important components of the Tower application itself.

You can use the **supervisorctl status** to see the list of Tower processes controlled by **supervisord** service:

```
[root@tower ~]# supervisorctl status
exit-event-listener                      RUNNING   pid 15382, uptime 0:34:22
tower-processes:awx-callback-receiver    RUNNING   pid 15387, uptime 0:34:22
tower-processes:awx-celeryd              RUNNING   pid 15389, uptime 0:34:22
```

```
tower-processes:awx-celeryd-beat          RUNNING   pid 15388, uptime 0:34:22
tower-processes:awx-channels-worker       RUNNING   pid 15383, uptime 0:34:22
tower-processes:awx-daphne                RUNNING   pid 15386, uptime 0:34:22
tower-processes:awx-fact-cache-receiver   RUNNING   pid 15385, uptime 0:34:22
tower-processes:awx-uwsgi                 RUNNING   pid 15384, uptime 0:34:22
```

As you can see in the preceding output, **supervisord** controls a number of processes owned by the **awx** user. One of them is the **awx-celeryd** daemon, which is used as a real time, distributed message passing task and job queue.

Ansible Tower Configuration and Log Files

Configuration Files

The main configuration files for Ansible Tower itself are kept in the **/etc/tower** directory. These include settings files for the Tower application which are outside the PostgreSQL database, the TLS certificate for **nginx** and other key files.

Perhaps the most important of these files for the Tower application is the **/etc/tower/settings.py** file, which specifies the locations for job output, project storage, and other directories.

The other individual services may have service-specific configuration files elsewhere on the system, such as the **/etc/nginx** files used by the web server.

Log Files

The log files for the Ansible Tower application itself are stored in one of two centralized locations:

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

Tower server errors are logged in the **/var/log/tower/** directory. Some key files in the **/var/log/tower/** directory include:

- **/var/log/tower/tower.log**, the main log for the Tower application
- **/var/log/tower/setup*.log**, which are logs of runs of the **setup.sh** script to install, backup, or restore the Tower server
- **/var/log/tower/task_system.log**, which logs various system housekeeping tasks (such as the removal of the record of old job runs)

The **/var/log/supervisor/** directory stores log files for services, daemons and applications managed by **supervisord**. The **supervisord.log** in this directory is the main log file for the service that controls all of these daemons. The other files contain log information about the activity of those daemons.

Ansible Tower is also capable of sending detailed logs to external log aggregation services. Log aggregation can offer insight into Tower technical trends or usage. The data can be used to monitor for anomalies, analyze events, and correlate events. Splunk, Elastic stack/logstash (formerly ELK), Loggly, and Sumologic are all log aggregation and data analysis systems that can be used with Tower.

More information on how to configure such services is located in the Ansible Tower Administration Guide at *https://docs.ansible.com/ansible-tower/3.1.1/html/administration/*.

Important

This discussion has focused on looking at the log files to troubleshoot problems with the Tower server itself.

If you are having errors running playbooks which don't appear to be related to actual errors in Tower's configuration, don't forget to look at the output of your launched jobs in Tower's web interface or the API.

Other Ansible Tower Files

A number of other key files for Ansible Tower are kept in the **`/var/lib/awx`** directory. Some of the contents of this directory include:

- **`/var/lib/awx/public/static`** - for static root directory (this is the location of your Django based application files).
- **`/var/lib/awx/projects`** - projects root directory (in the subdirectories of this directory Tower will store project based files - for example **`git`** repository files.
- **`/var/lib/awx/job_status`** - job status outputs from playbooks will be stored here.

Common Troubleshooting Scenarios

Problems running playbooks

The default configuration confines playbooks to the **`/tmp`** directory and limits what the playbook can access locally on the Tower server. This can impact tasks which the playbook may delegate to the local system rather than the target host.

Check your license status and the number of unique hosts you have managed by the Tower server. If the license has expired, or too many hosts are registered, you will not be able to launch jobs.

Problems connecting to your host

If you encounter problems with connectivity errors while running playbooks, try the following:

- Verify that you can establish an **`ssh`** or **`winrm`** connection with the managed host. Ansible depends upon **SSH** (or **`winrm`** for Microsoft Windows systems) to access the servers you are managing.
- Review your inventory file. Check the hostnames and IP addresses.

Playbooks are not accessible in the Job Template drop-down

If your playbooks are not showing up in the Job Template list, review those options:

- Check the playbook YAML syntax and make sure that it can be parsed by Ansible.
- Make sure the permissions and ownership of the project path (**`/var/lib/awx/projects/`**) is correct, so that the **`awx`** system user can view the files.

Playbook stays in pending

When you are trying to run a Job and it stays in the **`Pending`** state, try the following:

- Ensure that Tower server has enough memory available and that the services governed by **`supervisord`** are running. Issue the **`supervisorctl status`** command.
- Ensure that the partition where **`/var/`** directory is located has more than 1 GB of space available. Jobs will not be completed when there is insufficient free space on the **`/var/`** partition.
- Restart the Tower infrastructure using **`ansible-tower-service restart`** command.

Error: provided hosts list is empty

If you encounter an error message **`Skipping: No Hosts Matched`** when you are trying to run a playbook through Tower, here are things you should check:

- Review and make sure that the hosts declaration line in your playbook has exactly the same name as your group/host in inventory. These are case sensitive.
- Make sure your group names have no spaces and modify them to use underscores or no spaces to ensure that the groups correctly recognized.
- If you have specified a limit in the Job Template, make sure that it is a valid limit and that it matches something in your inventory.

References

Further information is available in the *Administration Guide* for Ansible Tower at
https://docs.ansible.com/ansible-tower/3.1.1/html/administration/

Guided Exercise: Performing Basic Troubleshooting of Ansible Tower

In this exercise, you will identify the locations of Ansible Tower log files and use information in them for basic troubleshooting.

Outcomes
You should be able to:

- Start, stop, and restart Ansible Tower services.
- Use the log files to troubleshoot Ansible Tower.

Before you begin
Ensure that the **workstation** and **tower** virtual machines are started.

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

From **workstation**, run **lab admin-troubleshoot setup**, which prepares the **tower.lab.example.com** server for this exercise.

```
[student@workstation ~]$ lab admin-troubleshoot setup
```

Steps
1. On **workstation**, open the **Firefox** web browser and try to log in to Ansible Tower's web interface at *https://tower.lab.example.com/*.

   This should fail. This exercise investigates that failure.

   ```
   Server Error
   A server error has occurred.
   ```

2. On **workstation**, open a terminal and use **ssh** to log in to **tower.lab.example.com** as **root**.

   ```
   [student@workstation ~]$ ssh root@tower.lab.example.com
   Last login: Tue Mar 21 06:27:17 2017 from workstation.lab.example.com
   [root@tower ~]#
   ```

3. Ensure that services making up the main components of Tower are all running and that the server's firewall is not blocking communications.

   3.1. To eliminate potential connection errors caused by firewall rules, check the current **firewalld** configuration by using **firewall-cmd** command with the **--list-ports** option.

   ```
   [root@tower ~]# firewall-cmd --list-ports
   443/tcp 80/tcp
   ```

As you can see in the output of the command, port **80** and **443** are not blocked. This is not a surprise, since we got a response from the Tower web server earlier, even though the response was a **Server Error**.

3.2. Next, check the status of the services that make up the Ansible Tower infrastructure. Use the **ansible-tower-service** script to run **ansible-tower-service status**.

```
[root@tower ~]# ansible-tower-service status
Showing Tower Status
● postgresql-9.4.service - PostgreSQL 9.4 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-9.4.service; enabled;
 vendor preset: disabled)
   Active: inactive (dead) since Tue 2017-03-28 04:20:43 EDT; 24min ago
...output omitted...
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Tue 2017-03-28 04:20:49 EDT; 24min ago
...output omitted...
● supervisord.service - Process Monitoring and Control Daemon
   Loaded: loaded (/usr/lib/systemd/system/supervisord.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Tue 2017-03-28 04:20:55 EDT; 23min ago
  Process: 909 ExecStart=/usr/bin/supervisord -c /etc/supervisord.conf
 (code=exited, status=0/SUCCESS)
 Main PID: 1101 (code=exited, status=0/SUCCESS)
```

As you can see in the output of the command, one of the services required for Ansible Tower infrastructure is marked as **inactive (dead)**. The services directly started by **ansible-tower-service** for the Ansible Tower infrastructure to work properly are **postgresql-9.4**, **nginx**, **supervisord**, and **rabbitmq-server**.

3.3. Restart the Ansible Tower infrastructure using the **ansible-tower-service** admin utility script.

```
[root@tower ~]# ansible-tower-service restart
Restarting Tower
Redirecting to /bin/systemctl stop  postgresql-9.4.service
Stopping rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl stop  nginx.service
Redirecting to /bin/systemctl stop  supervisord.service
Redirecting to /bin/systemctl start  postgresql-9.4.service
Starting rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl start  nginx.service
Redirecting to /bin/systemctl start  supervisord.service
```

3.4. To confirm that the Ansible Tower infrastructure has started, go back to **workstation**, open Firefox browser and log in to the Ansible Tower web interface.

4. The next step is to determine why the PostgreSQL database was not running.

4.1. The **ansible-tower-service** script apparently controls PostgreSQL as a **postgresql-9.4.service** unit through **systemctl**, so we can investigate further using that tool:

```
[root@tower ~]# systemctl status postgresql-9.4 -l
```

```
● postgresql-9.4.service - PostgreSQL 9.4 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-9.4.service; enabled;
 vendor preset: disabled)
   Active: active (running) since Tue 2017-03-28 04:52:45 EDT; 2min ago
  Process: 28501 ExecStop=/usr/pgsql-9.4/bin/pg_ctl stop -D ${PGDATA} -s -m fast
 (code=exited, status=0/SUCCESS)
  Process: 29737 ExecStart=/usr/pgsql-9.4/bin/pg_ctl start -D ${PGDATA} -s -w -t
 300 (code=exited, status=0/SUCCESS)
  Process: 29732 ExecStartPre=/usr/pgsql-9.4/bin/postgresql94-check-db-dir
 ${PGDATA} (code=exited, status=0/SUCCESS)
 Main PID: 29741 (postgres)
   CGroup: /system.slice/postgresql-9.4.service
           ├─29741 /usr/pgsql-9.4/bin/postgres -D /var/lib/pgsql/9.4/data
           ├─29742 postgres: logger process
           ├─29744 postgres: checkpointer process
           ├─29745 postgres: writer process
           ├─29746 postgres: wal writer process
           ├─29747 postgres: autovacuum launcher process
           ├─29748 postgres: stats collector process
           ├─30579 postgres: awx awx 127.0.0.1(46134) idle
           ├─30582 postgres: awx awx 127.0.0.1(46146) idle
           ├─30583 postgres: awx awx 127.0.0.1(46148) idle
           └─30598 postgres: awx awx 127.0.0.1(46166) idle

Mar 28 04:52:44 tower.lab.example.com systemd[1]: Starting PostgreSQL 9.4
 database server...
Mar 28 04:52:44 tower.lab.example.com pg_ctl[29737]: < 2017-03-28 04:52:44.342
 EDT >LOG:  redirecting log output to logging collector process
Mar 28 04:52:44 tower.lab.example.com pg_ctl[29737]: < 2017-03-28 04:52:44.342
 EDT >HINT:  Future log output will appear in directory "pg_log".
Mar 28 04:52:45 tower.lab.example.com systemd[1]: Started PostgreSQL 9.4
 database server.
```

The PostgreSQL server appears to be logging to a **pg_log** directory. The **find** command shows us where that is:

```
[root@tower ~]# find / -name pg_log
/var/lib/pgsql/9.4/data/pg_log
```

4.2. Change to the **/var/lib/pgsql/9.4/data/pg_log** directory and look at the log files.

```
[root@tower ~]# cd /var/lib/pgsql/9.4/data/pg_log
[root@tower pg_log]# ls -l
total 600
-rw-------. 1 postgres postgres  85156 Mar 24 23:33 postgresql-Fri.log
-rw-------. 1 postgres postgres  19690 Mar 27 21:47 postgresql-Mon.log
-rw-------. 1 postgres postgres  74337 Mar 25 21:03 postgresql-Sat.log
-rw-------. 1 postgres postgres  90042 Mar 26 23:39 postgresql-Sun.log
-rw-------. 1 postgres postgres 112029 Mar 23 23:59 postgresql-Thu.log
-rw-------. 1 postgres postgres  68055 Mar 28 04:55 postgresql-Tue.log
-rw-------. 1 postgres postgres 151967 Mar 22 23:59 postgresql-Wed.log
```

4.3. Examine the log file that recorded messages near the time that PostgreSQL stopped. (In this example, this was at Tue 2017-03-28 04:20:43 EDT based on the output reported the first time we ran the **ansible-tower-service status** command and spotted the failure. The log file and time you should actually use may be different from the one

in this example, depending on what time and which day of the week it is when you do this exercise.)

```
[root@tower pg_log]# less postgresql-Tue.log
...output omitted...
< 2017-03-28 04:20:42.601 EDT >LOG:  received fast shutdown request
< 2017-03-28 04:20:42.601 EDT >LOG:  aborting any active transactions
< 2017-03-28 04:20:42.601 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.602 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.603 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.603 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.603 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.604 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.604 EDT >FATAL:  terminating connection due to
 administrator command
< 2017-03-28 04:20:42.605 EDT >LOG:  autovacuum launcher shutting down
< 2017-03-28 04:20:42.606 EDT >LOG:  shutting down
< 2017-03-28 04:20:42.744 EDT >LOG:  database system is shut down
...output omitted...
```

It looks like someone manually stopped the service! We'll stop our investigation of this particular issue here, for now....

5. Now that you've solved the mystery of the **Server Error**, the remainder of this exercise explores other log files and useful tools for troubleshooting Ansible Tower.

   The **supervisord** service is responsible for running a collection of programs that control the main logic of the Ansible Tower application. This includes the **awx-celeryd** worker queues that run jobs and the **awx-callback-receiver** processes that receive job events from running jobs.

   A number of the log files for the **supervisord** service are in the **/var/log/supervisor** directory on the Tower server. In that directory, display the end of the file **supervisord.log**.

```
[root@tower pg_log]# cd /var/log/supervisor
[root@tower supervisor]# tail -n 50 supervisord.log
...output omitted...
2017-03-29 05:14:07,247 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 20
2017-03-29 05:14:07,247 INFO stopped: awx-daphne (exit status 0)
2017-03-29 05:14:07,318 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 21
2017-03-29 05:14:07,318 INFO stopped: awx-celeryd-beat (exit status 0)
2017-03-29 05:14:08,280 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 22
2017-03-29 05:14:08,280 INFO stopped: awx-uwsgi (exit status 0)
2017-03-29 05:14:09,282 INFO waiting for awx-callback-receiver to die
2017-03-29 05:14:11,285 WARN killing 'awx-callback-receiver' (14267) with SIGKILL
2017-03-29 05:14:25,348 CRIT Supervisor running as root (no user in config file)
2017-03-29 05:14:25,349 WARN Included extra file "/etc/supervisord.d/tower.ini"
 during parsing
2017-03-29 05:14:25,349 INFO Increased RLIMIT_NOFILE limit to 4096
```

```
2017-03-29 05:14:25,396 INFO RPC interface 'supervisor' initialized
...output omitted...
2017-03-29 05:14:25,397 CRIT Server 'unix_http_server' running without any HTTP
 authentication checking
2017-03-29 05:14:25,403 INFO daemonizing the supervisord process
2017-03-29 05:14:25,409 INFO supervisord started with pid 16949
2017-03-29 05:14:26,422 INFO spawned: 'exit-event-listener' with pid 16950
2017-03-29 05:14:26,430 INFO spawned: 'awx-channels-worker' with pid 16951
2017-03-29 05:14:26,605 INFO spawned: 'awx-celeryd' with pid 16957
2017-03-29 05:14:27,812 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 2
2017-03-29 05:14:27,812 INFO success: awx-daphne entered RUNNING state, process has
 stayed up for > than 1 seconds (startsecs)
2017-03-29 05:14:27,813 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 3
2017-03-29 05:14:27,813 INFO success: awx-callback-receiver entered RUNNING state,
 process has stayed up for > than 1 seconds (startsecs)
2017-03-29 05:14:27,813 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 4
2017-03-29 05:14:27,813 INFO success: awx-celeryd-beat entered RUNNING state,
 process has stayed up for > than 1 seconds (startsecs)
2017-03-29 05:14:27,813 ERRO pool exit-event-listener event buffer overflowed,
 discarding event 5
2017-03-29 05:14:27,813 INFO success: awx-celeryd entered RUNNING state, process has
 stayed up for > than 1 seconds (startsecs)

...output omitted...
```

As you can see in the output of the command, some services governed by **supervisord** service had also been stopped. After executing the **ansible-tower-service restart** command to restart the stopped PostgreSQL server, they were also successfully spawned by **supervisord**, which allowed you to log in to the Ansible Tower web interface.

6. You can also check the status of the processes managed by **supervisord** by using the **supervisorctl** command.

```
[root@tower supervisor]# supervisorctl status
exit-event-listener                      RUNNING   pid 4111, uptime 0:42:55
tower-processes:awx-callback-receiver    RUNNING   pid 4116, uptime 0:42:55
tower-processes:awx-celeryd              RUNNING   pid 4118, uptime 0:42:55
tower-processes:awx-celeryd-beat         RUNNING   pid 4117, uptime 0:42:55
tower-processes:awx-channels-worker      RUNNING   pid 4112, uptime 0:42:55
tower-processes:awx-daphne               RUNNING   pid 4115, uptime 0:42:55
tower-processes:awx-fact-cache-receiver  RUNNING   pid 4114, uptime 0:42:55
tower-processes:awx-uwsgi                RUNNING   pid 4113, uptime 0:42:55
```

7. Other log files relevant to the Tower application are kept in the **/var/log/tower** directory. For example, if there is a problem with job execution using Ansible Tower, one file to examine is the **/var/log/tower/tower.log** file. This log contains useful information about the status of executed jobs and changes in Tower Inventories or Job Templates.

```
[root@tower supervisor]# less /var/log/tower/tower.log
...output omitted...
  File "/var/lib/awx/venv/tower/lib/python2.7/site-packages/psycopg2/__init__.py",
 line 164, in connect
    conn = _connect(dsn, connection_factory=connection_factory, async=async)
OperationalError: could not connect to server: Connection refused
        Is the server running on host "127.0.0.1" and accepting
        TCP/IP connections on port 5432?
```

```
2017-03-29 05:02:44,375 WARNING  awx.api.generics status 404 received by user admin
 attempting to access /api/v1/job_templates/55555/launch/ from 172.25.250.254
2017-03-29 05:05:59,920 WARNING  awx.api.generics status 404 received by user admin
 attempting to access /api/v1/job_templates/55555/launch/ from 172.25.250.254
2017-03-29 05:06:26,066 ERROR    awx.main.tasks Failed to update ProjectUpdate after
 5 retries.
2017-03-29 05:06:51,108 ERROR    awx.main.tasks Failed to update ProjectUpdate after
 5 retries.
2017-03-29 05:08:24,920 WARNING  awx.api.generics status 404 received by user admin
 attempting to access /api/v1/job_templates/55555/launch/ from 172.25.250.254
2017-03-29 05:08:27,523 ERROR    awx.main.scheduler Task
 awx.main.scheduler.partial.ProjectUpdateDict object at 0x6705710 appears
 orphaned... marking as failed
...output omitted...
```

The log messages here indicate that something is trying to launch a nonexistent Job Template.

8. Those attempts by **admin** to use the API to launch a nonexistent Job Template with **id** 55555 from 172.25.250.254 should also show up in the **nginx** web server's **access.log**.

   The logs for **nginx** are located in the **/var/log/nginx** directory. Look at the **access.log** file for events that happened at the same time as the **WARNING** in the **tower.log**:

```
[root@tower supervisor]# less /var/log/nginx/access.log
...output omitted...
172.25.250.254 - admin [29/Mar/2017:05:02:44 -0400] "POST /api/v1/
job_templates/55555/launch/ HTTP/1.1" 404 34 "-" "curl/7.47.1" "-"
...output omitted...
```

9. This concludes this exercise. Log out of the **root** account on the **tower** system.

```
[root@tower supervisor]# exit
```

Cleanup

From **workstation**, run the **lab admin-troubleshoot cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab admin-troubleshoot cleanup
```

Configuring TLS/SSL for Ansible Tower

Objectives

After completing this section, students should be able to:

- Replace the default TLS certificate used by Ansible Tower with a provided certificate appropriate for the server's name.

Nginx Web Server on Ansible Tower

The Ansible Tower web interface is provided by an Nginx web server running on the Tower server. When installed, Ansible Tower creates a self-signed TLS certificate and matching private key file which Nginx uses for HTTPS communication.

The main configuration files for Nginx are in the **`/etc/nginx`** directory, the most important of which is **`/etc/nginx/nginx.conf`**. Access logs for the Nginx web server hosting the Tower web interface are located in **`/var/log/nginx/access.log`** and error logs are in **`/var/log/nginx/error.log`**. Both log files are periodically rotated, and **`gzip`** compressed archives of older versions of those files may be found in the **`/var/log`** directory.

In general, no changes should be necessary to the **`/etc/nginx/nginx.conf`** configuration file. However, one scenario in which changes might be useful is if the server's default HTTPS configuration needs adjustment.

Default TLS Configuration

There are two reasons an administrator might need to know how to find the TLS configuration on Ansible Tower's TLS service. The first is to locate the TLS certificate and private key so that they can be replaced with versions signed by a Certificate Authority trusted by the browsers accessing Tower. The second would be if customization of the TLS configuration becomes necessary, particularly removing ciphers if vulnerabilities in their algorithms are found.

The TLS configuration for the Nginx web server is defined in the **`/etc/nginx/nginx.conf`** Nginx configuration file. The **`server`** block, which listens for **`ssl`** connections on port 443, contains the relevant configuration directives. In particular, this shows that the TLS certificate is **`/etc/tower/tower.cert`** and the matching private key is **`/etc/tower/tower.key`**:

```
    server {
                listen 443 default_server ssl;
        listen 127.0.0.1:80 default_server;
                listen [::1]:80 default_server;

        # If you have a domain name, this is where to add it
        server_name _;
        keepalive_timeout 65;

        ssl_certificate /etc/tower/tower.cert;
        ssl_certificate_key /etc/tower/tower.key;
        ssl_session_cache shared:SSL:50m;
        ssl_session_timeout 1d;
        ssl_session_tickets off;

        # intermediate configuration
```

```
        ssl_protocols TLSv1.2;
        ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-
AES128-SHA256:ECDHE-RSA-AES128-SHA256';
        ssl_prefer_server_ciphers on;
```

Replacing the TLS Certificate and Key

Most organizations will want to replace Ansible Tower's self-signed certificate with one signed by a TLS Certificate Authority (CA) that's trusted by the organization's web browsers. This might be a public CA, or an internal corporate CA.

Either way, a correct CA-signed TLS certificate in PEM format needs to be obtained for the server, along with a matching private key in PEM format. Assuming this has been done, they need to be used to replace the existing self-signed certificate and key as follows:

- Save the CA-signed TLS certificate in PEM format to **`/etc/tower/tower.cert`**.
- Save the matching private key in PEM format to **`/etc/tower/tower.key`**.
- Both files must be readable and writable only by user **awx** (the Tower user), and must also be owned by group **awx**:

```
[root@tower tower]# ls -l /etc/tower/tower.*
-rw-------. 1 awx awx 1281 Mar 31 23:32 tower.cert
-rw-------. 1 awx awx 1704 Mar 31 23:32 tower.key
```

- Restart Ansible Tower with the command **`ansible-tower-service restart`**.
- Test the connection from a browser that trusts the CA used to sign the Tower server's certificate. Review the certificate details presented by your browser and whether your browser considers the connection secure. The details on how to do this will vary depending on which web browser software is used.

Important

In the lab, you will use our classroom's FreeIPA server as the CA. This allows you to use **`ipa-getcert`** to request a TLS certificate which will be automatically renewed by the **`certmonger`** daemon when it expires.

However, Ansible Tower 3.1.1 labels all files in **`/etc/tower`** with the SELinux type **`etc_t`**. Both **`tower.cert`** and **`tower.key`** need to be labeled **`cert_t`** for the files to be managed correctly by the FreeIPA tools. Fortunately, Nginx can read TLS certificates labeled **`cert_t`** correctly.

To persistently set the SELinux type on these two files, you will need to make sure the **`semanage`** command is available (it is provided by the *policycoreutils-python* package). Then run the command **`semanage fcontext -a -t cert_t "/etc/tower/tower.(.*)"`** command to set the default context for those files in the system policy. Finally, run **`restorecon -FvvR /etc/tower/`** to correct the SELinux contexts in that directory based on the current policy settings.

References

Further information is available in the General Installation Notes section of the *Installation and Reference Guide* for Ansible Tower at

> https://docs.ansible.com/ansible-tower/

Guided Exercise: Configuring TLS/SSL for Ansible Tower

In this exercise, you will replace the existing TLS/SSL certificate with a valid one provided by the **utility** server.

Outcomes
You should be able to:

- Replace the default TLS/SSL certificate used by Ansible Tower with a provided certificate appropriate for the server's hostname.

Before you begin
- Ensure that the **workstation**, **tower** and **utility** virtual machines are started.
- Log in to **workstation** as **student** using **student** as the password.
- From **workstation**, run **lab admin-cert setup**, which verifies that Ansible Tower services are running and all the required resources are available.

```
[student@workstation ~]$ lab admin-cert setup
```

Steps
1. On **workstation**, open a terminal. In the following steps, you will change the Ansible Tower TLS/SSL certificate with a provided certificate appropriate for the server's hostname.
2. Use **SSH** to log in to the **tower** server as **root**.

```
[student@workstation ~]$ ssh root@tower
Last login: Tue Mar 21 06:27:17 2017 from workstation.lab.example.com
[root@tower ~]#
```

3. Ensure that the system policy will set the SELinux type to **cert_t** on **/etc/tower/tower.cert** and **/etc/tower/tower.key** in the system policy. Run **restorecon** on those files to make sure that SELinux type is set on those files. This is needed for **certmonger** to write to those files when requested by **ipa-getcert** in the next step.

```
[root@tower ~]# semanage fcontext -a -t cert_t "/etc/tower(/.*)?"
[root@tower ~]# restorecon -FvvR /etc/tower/
```

4. Remove the existing **/etc/tower/tower.cert** and **/etc/tower/tower.key** files.

```
[root@tower ~]# rm /etc/tower/tower.*
rm: remove regular file ‘/etc/tower/tower.cert’? y
rm: remove regular file ‘/etc/tower/tower.key’? y
```

5. Using the **ipa-getcert** command, get a certificate for **tower.lab.example.com** that is signed by the organization's FreeIPA-based CA on **utility.lab.example.com**.

```
[root@tower ~]# ipa-getcert request -f /etc/tower/tower.cert -k /etc/tower/tower.key
New signing request "20170328155831" added.
```

Note

Don't worry too much about why this works if you're not familiar with FreeIPA. The important part of this step is that you've somehow gotten a CA-signed TLS certificate and key for **tower.lab.example.com** that have been copied into the right locations on your Ansible Tower server.

6. Use the **ansible-tower-service** command to restart the Ansible Tower infrastructure and then exit the console session on the **tower** system.

```
[root@tower ~]# ansible-tower-service restart
Restarting Tower
Redirecting to /bin/systemctl stop  postgresql-9.4.service
Stopping rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl stop  nginx.service
Redirecting to /bin/systemctl stop  supervisord.service
Redirecting to /bin/systemctl start  postgresql-9.4.service
Starting rabbitmq-server (via systemctl):                  [  OK  ]
Redirecting to /bin/systemctl start  nginx.service
Redirecting to /bin/systemctl start  supervisord.service
[root@tower ~]# exit
```

7. Open a **Firefox** web browser and connect to the Ansible Tower web interface at *https://tower.lab.example.com*.

   The new SSL certificate is signed by trusted CA, that is why you do not receive any SSL certificate warnings. To review the new certificate details, click the padlock symbol in the browser URL bar. The new TLS/SSL certificate includes the **tower.lab.example.com** hostname and the **LAB.EXAMPLE.COM** organization.

Cleanup

From **workstation**, run the **lab admin-cert cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab admin-cert cleanup
```

Performing Command-Line Management with **tower-manage**

Objectives

After completing this section, students should be able to:

- Reset the admin password or import an existing inventory file using the **tower-manage** utility

Performing Management with **tower-manage**

Ansible Tower comes with the **tower-manage** command-line utility, which can be used to access detailed internal Tower information. The **tower-manage** command must be run as **root** or as the **awx** (Ansible Tower) user. This utility is most commonly used to reset the Tower's **admin** password and to import an existing static inventory file into the Tower server.

Changing the Tower Admin Password

The password for the built-in Ansible Tower **System Administrator** account, **admin**, is initially set when the Tower server is installed. The **tower-manage** command offers a way to change the administrator password from the command line. To do this, as the **root** or **awx** user on the Tower server, use the **changepassword** option:

```
[root@towerhost ~]# tower-manage changepassword admin
Changing password for user 'admin'
Password: new_password
Password (again): new_password
Password changed successfully for user 'admin'
```

After entering the new password twice, the password you have entered will be the **admin** password in the Ansible Tower web interface.

You can also create a new Ansible Tower superuser, with administrative privileges if needed. To create a new superuser you can use **tower-manage** with the **createsuperuser**.

```
[root@towerhost ~]# tower-manage createsuperuser
Username (leave blank to use 'root'): admin3
Email address: admin@demo.example.com
Password: new_password
Password (again): new_password
Superuser created successfully.
```

Inventory Import

tower-manage allows the administrator to import a static inventory from a file directly into the Tower server. The variables set in the **group_vars** or **host_vars** directories that are associated with the inventory file may also be imported with the inventory file.

To use this feature of the **tower-manage** command, there must be a destination inventory already set up in the Tower. The inventory file and associated inventory variables will be imported into this existing destination inventory.

Before importing your static inventory file and its host and group variables, the files should be organized in a directory structure like the following example:

```
[root@towerhost ~]# tree inventory/
inventory/
|-- group_vars
|   `-- mygroup
|-- host_vars
|   `-- myhost
`-- hosts
```

To import these hosts and variables into Tower, use the **tower-manage inventory_import** command, specifying the source directory with the **--source** option, and the name of the existing destination inventory in Tower with the **--inventory-name** option.

```
[root@towerhost ~]# tower-manage inventory_import --source=inventory/ \
> --inventory-name="My Tower Inventory"
```

If your inventory is simply a single flat file, the **--source** option can point directly at the inventory file itself rather than to a directory:

```
[root@towerhost ~]# tower-manage inventory_import --source=./my_inventory_file \
> --inventory-name="My Tower Inventory"
```

If the destination inventory in Tower is not empty, the imported data does not overwrite the existing data by default, but is combined with it. This default behavior adds any new variables from the imported inventory to any variable already in the imported inventory. You can overwrite any existing data by specifying the **--overwrite_vars** option.

```
[root@towerhost ~]# tower-manage inventory_import --source=inventory/ \
> --inventory-name="My Tower Inventory" \
> --overwrite
```

References

Further information is available in the "**tower-manage** Utility" section of the *Ansible Tower Administration Guide* at

https://docs.ansible.com/ansible-tower/3.1.1/html/administration/

Guided Exercise: Performing Command-Line Management with **tower-manage**

In this exercise, you will use the **tower-manage** utility to reset the **admin** password and import a static inventory file.

Outcomes
You should be able to:

- Use the **tower-manage** utility to reset the **admin** password.
- Use the **tower-manage** utility to import an existing inventory file.

Before you begin
Ensure that the **workstation** and **tower** virtual machines are started.

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

From **workstation**, run **lab admin-cli setup**, which verifies that Ansible Tower services are running and all the required resources are available.

```
[student@workstation ~]$ lab admin-cli setup
```

Steps

1. On **workstation**, open a terminal. In the following steps you will change the Ansible Tower **admin** user password and import existing static inventory file from the CLI.

2. To use the **tower-manage** command, log in to the **tower** server as **root** user using SSH.

```
[student@workstation ~]$ ssh root@tower
Last login: Tue Mar 21 06:27:17 2017 from workstation.lab.example.com
[root@tower ~]#
```

3. Create a new System Administrator user with the use of **tower-manage** command and **createsuperuser** subcommand.

```
[root@tower ~]# tower-manage createsuperuser
```

   3.1. Use **admin2** as the name of the new superuser.

```
Username (leave blank to use 'root'): admin2
```

   3.2. Leave the email address blank.

```
Username (leave blank to use 'root'): admin2
Email address:
```

   3.3. Type **redhat** as password.

```
Username (leave blank to use 'root'): admin2
Email address:
Password: redhat
```

3.4. Repeat previous step.

```
Username (leave blank to use 'root'): admin2
Email address:
Password:
Password (again): redhat
Superuser created successfully.
```

4. Using the **tower-manage** command and the **changepassword** subcommand, reset the **admin2** user password.

```
[root@tower ~]# tower-manage changepassword admin2
Changing password for user 'admin2'
Password:
```

4.1. Type **redhat2** as the new password twice.

```
Changing password for user 'admin2'
Password: redhat2
Password (again): redhat2
Password changed successfully for user 'admin2'
```

4.2. To confirm the change, on **workstation** open Firefox browser and log in to Ansible Tower as **admin2** user with the new **redhat2** password.

5. On the **tower** server, list the contents of the **root** user home directory, to ensure that the static inventory file for Ansible Tower is available.

```
[root@tower ~]# ls /root
anaconda-ks.cfg ansible-tower-setup-bundle-3.1.1-1.el7 original-ks.cfg example-
inventory
```

5.1. Using the **tower-manage** command and the **inventory_import** subcommand, import the **example-inventory** static inventory file containing an inventory. Use the existing **Exercise** name as the destination for the import. After the import has completed, exit the console session on the **tower** system.

```
[root@tower ~]# tower-manage inventory_import --source=/root/example-inventory
 --inventory-name="Exercise"
1.982 INFO     Updating inventory 1: Exercise
    2.073 INFO     Reading INI source: /root/example-inventory
    2.075 INFO     Loaded 3 groups, 7 hosts
    2.076 INFO     Inventory variables unmodified
    2.088 INFO     Group "dbsrv" added
    2.096 INFO     Group "filesrv" added
    2.103 INFO     Group "websrv" added
```

```
    2.113 INFO     Host "utility.lab.example.com" added
    2.115 INFO     Host "servera.lab.example.com" added
    2.118 INFO     Host "serverb.lab.example.com" added
    2.120 INFO     Host "serverc.lab.example.com" added
    2.123 INFO     Host "serverd.lab.example.com" added
    2.125 INFO     Host "servere.lab.example.com" added
    2.128 INFO     Host "serverf.lab.example.com" added
    2.145 INFO     Host "serverc.lab.example.com" added to group "dbsrv"
    2.145 INFO     Host "serverd.lab.example.com" added to group "dbsrv"
    2.154 INFO     Host "servere.lab.example.com" added to group "filesrv"
    2.154 INFO     Host "serverf.lab.example.com" added to group "filesrv"
    2.162 INFO     Host "servera.lab.example.com" added to group "websrv"
    2.162 INFO     Host "serverb.lab.example.com" added to group "websrv"
    2.261 INFO     Inventory import completed for "Exercise" (id=2) in 0.3s
[root@tower ~]# exit
```

5.2. In the Ansible Tower web interface, click the **INVENTORIES** quick navigation link to display the list of Inventories. You should see an Inventory named **`Exercise`** which was used in previous step for the import.

5.3. Click the **Exercise** link to view the details of the imported static Inventory. Look at the available **GROUPS** and **HOSTS** sections, you should see that the inventory is composed of multiple groups and hosts. This confirms that the static inventory has been successfully imported and is accessible.

5.4. Click the **Log Out** icon to log out of the Tower web interface.

Cleanup

From **`workstation`**, run the **`lab admin-cli cleanup`** script to clean up this exercise.

```
[student@workstation ~]$ lab admin-cli cleanup
```

Backing Up and Restoring an Ansible Tower Installation

Objectives

After completing this section, students should be able to:

- Backup and restore the Ansible Tower system.

Backing up Ansible Tower

The ability to manually back up and restore an Ansible Tower installation is integrated into Tower's installation software. You can then use other tools to automate the backup process and make sure that the backup files are stored in a safe and secure location off the Tower server.

The procedure uses the same **`setup.sh`** script and the **`inventory`** file that you used to install Tower.

Important

If you have deleted the original installation directory, you can still set up backups by unpacking the **`tar`** archive containing the installer for the same version of Tower that you're currently using. running.

You also need to edit The installer's **`inventory`** file to contain the current passwords for your Tower services (**`admin_password`**, **`pg_password`**, and **`rabbitmq_password`**). If you made any other edits to the **`inventory`** file before installing Tower, you must make those edits now as well.

The actual backup is started by running **`./setup.sh -b`** in the installation directory on the Tower server as **`root`**. It will create the backup as a **`tar`** archive in the installer's directory named **`tower-backup-`*`DATE`*`.tar.gz`**, where *DATE* is in **`date +%F-+%T`** format. It will also create a symlink, **`tower-backup-latest.tar.gz`**, pointing to the most recent backup archive in the directory.

The backup archive consists of the following files and directories:

- **`tower.db`** - PostgreSQL database dump file.
- **`./conf`** - configuration directory, contains files from **`/etc/tower/`** directory.
- **`./job_status`** - directory for job output files.
- **`./projects`** - directory for manual projects.
- **`./static`** - directory for web interface customization, for example custom logo.

As with every backup procedure, you should review the amount of disk space available, to ensure that there is enough free space to store the backup. Note that this procedure backs up the Tower's configuration, but not its logs or the programs installed by the Tower installer.

Warning

The manual backup procedure using **`setup.sh -b`** only creates the backup archive file. You are responsible for setting up a system which will periodically run the command to create the backups and store the backup archives in a safe place.

Backup Procedure

The following procedure creates a new backup of the running Ansible Tower's configuration.

1. As the **`root`** user, locate the Ansible Tower installation directory and change into that directory.

```
[root@towerhost ~]# find / -name ansible*
/root/ansible-tower-setup-bundle-3.1.1-1.el7
[root@towerhost ~]# cd /root/ansible-tower-setup-bundle-3.1.1-1.el7
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]#
```

2. As **`root`** user use the **`setup.sh`** script with **`-b`** option to initiate Ansible Tower configuration and database backup process.

```
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh -b
...output omitted...

RUNNING HANDLER [backup : Remove the backup tarball.] *************************
changed: [localhost] => {"changed": true, "path": "/var/backups/tower/tower-
backup-2017-03-29-08:34:43.tar.gz", "state": "absent"}

PLAY RECAP *********************************************************************
localhost                  : ok=24   changed=16   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-03-29-08:34:34.log
```

3. List the current directory to ensure that the backup archive has been created and is accessible.

```
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]# ls -l tower-*
-rw-r--r--.  1 root root 164317 03-29 08:34 tower-backup-2017-03-29-08:34:43.tar.gz
lrwxrwxrwx.  1 root root     84 03-29 08:35 tower-backup-latest.tar.gz -> /root/
ansible-tower-setup-bundle-3.1.1-1.el7/tower-backup-2017-03-29-08:34:43.tar.gz
```

Note

Notice the symbolic link **`tower-backup-latest.tar.gz`** pointing at the latest created backup archive. This link is by default used to recover the Ansible Tower infrastructure from backup.

Restoring Ansible Tower from Backup

The **setup.sh** script is used with the **-r** option to restore Ansible Tower from a backup archive. You will need the backup, the installer for the same version of Ansible Tower that was used to create the backup, and an **inventory** file for the installer.

Should you have multiple backup archives available, be sure that the **tower-backup-latest.tar.gz** symbolic link points to the exact backup file from which you want to restore. If you need to use an older backup file, delete the existing **tower-backup.latest.tar.gz** symbolic link and create a new link pointing to the correct backup archive.

Warning

When restoring a backup, be sure to use the same version of Ansible Tower that was used to create the backup.

Restore Procedure

The following is the procedure for restoring the Ansible Tower infrastructure from existing backup archive.

1. As the **root** user, locate the Ansible Tower installation directory and change into that directory.

```
[root@towerhost ~]# find / -name ansible*
/root/ansible-tower-setup-bundle-3.1.1-1.el7
[root@towerhost ~]# cd /root/ansible-tower-setup-bundle-3.1.1-1.el7
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]#
```

2. Ensure that the backup archive is in that directory.

```
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]# ls -l tower-*
-rw-r--r--.  1 root root 164317 03-29 08:34 tower-backup-2017-03-29-08:34:43.tar.gz
lrwxrwxrwx.  1 root root     84 03-29 08:35 tower-backup-latest.tar.gz -> /root/
ansible-tower-setup-bundle-3.1.1-1.el7/tower-backup-2017-03-29-08:34:43.tar.gz
```

Important

Remember that the symbolic link **tower-backup-latest.tar.gz** points to the latest backup archive. This link is used to recover the Ansible Tower infrastructure from backup. Should you have the need to restore from an older archive, you have to recreate that link by pointing to the correct archive.

3. As **root** user use **setup.sh -r** to initiate restoration of the Ansible Tower configuration and database.

Important

It is important not to forget the **`-r`** option to the **`setup.sh`** command. If you do, the Ansible Tower installer will start a new installation process from the beginning.

If that happens, wait. When the installation process finishes, restore the backup.

```
[root@towerhost ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh -r
...output omitted...
00ms", "Result": "success", "RootDirectoryStartOnly": "no", "RuntimeDirectoryMode":
 "0755", "SameProcessGroup": "no", "SecureBits": "0", "SendSIGHUP": "no",
 "SendSIGKILL": "yes", "Slice": "system.slice", "StandardError": "inherit",
 "StandardInput": "null", "StandardOutput": "journal", "StartLimitAction": "none",
 "StartLimitBurst": "5", "StartLimitInterval": "10000000", "StartupBlockIOWeight":
 "18446744073709551615", "StartupCPUShares": "18446744073709551615", "StatusErrno":
 "0", "StopWhenUnneeded": "no", "SubState": "dead", "SyslogLevelPrefix":
 "yes", "SyslogPriority": "30", "SystemCallErrorNumber": "0", "TTYReset": "no",
 "TTYVHangup": "no", "TTYVTDisallocate": "no", "TimeoutStartUSec": "1min 30s",
 "TimeoutStopUSec": "1min 30s", "TimerSlackNSec": "50000", "Transient": "no",
 "Type": "forking", "UMask": "0022", "UnitFilePreset": "disabled", "UnitFileState":
 "enabled", "WantedBy": "multi-user.target", "Wants": "system.slice",
 "WatchdogTimestampMonotonic": "0", "WatchdogUSec": "0"}, "warnings": []}

PLAY RECAP *********************************************************************
localhost                  : ok=26   changed=18   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-03-30-04:19:05.log
```

4. Log in to the Ansible Tower web interface and verify that the server has been restored from backup correctly.

References

Further information is available in the Backing Up and Restoring Tower section of the *Tower Administration Guide* at

> http://docs.ansible.com/ansible-tower

Guided Exercise: Backing Up and Restoring an Ansible Tower Installation

In this exercise, you will perform a backup of Ansible Tower database and configuration files.

Outcomes
You should be able to:

- Perform a full backup of the existing Ansible Tower installation.
- Restore the Ansible Tower configuration and database from existing backup.

Before you begin
Ensure that the **workstation** and **tower** virtual machines are started.

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

From **workstation**, run **lab admin-recovery setup**, which verifies that Ansible Tower services are running and all the required resources are available.

```
[student@workstation ~]$ lab admin-recovery setup
```

Steps

1. On **workstation**, open a terminal. In the following steps you will create backup of Ansible Tower from the CLI.

2. To perform the backup, log in to the **tower** server as **root** user using SSH.

```
[student@workstation ~]$ ssh root@tower
Last login: Tue Mar 21 06:27:17 2017 from workstation.lab.example.com
[root@tower ~]#
```

3. List the contents of the **root** user home directory, to ensure that the installation directory of Ansible Tower is available.

```
[root@tower ~]# ls /root
anaconda-ks.cfg  ansible-tower-setup-bundle-3.1.1-1.el7  original-ks.cfg
```

4. With the use of the **ansible-tower-service status** command, check that Ansible Tower services are running.

```
[root@tower ~]# ansible-tower-service status
(...output omitted...)
Redirecting to /bin/systemctl status  supervisord.service
● supervisord.service - Process Monitoring and Control Daemon
   Loaded: loaded (/usr/lib/systemd/system/supervisord.service; enabled; vendor
 preset: disabled)
   Active: active (running) since wto 2017-03-21 04:25:50 EDT; 2h 33min ago
  Process: 4145 ExecStart=/usr/bin/supervisord -c /etc/supervisord.conf
 (code=exited, status=0/SUCCESS)
(...output omitted...)
```

5. Change the current working directory to **/root/ansible-tower-setup-bundle-3.1.1-1.el7** directory.

```
[root@tower ~]# cd /root/ansible-tower-setup-bundle-3.1.1-1.el7
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]#
```

6. Create Ansible Tower configuration and database backup, using the **setup.sh** command with **-b** as option.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh -b
(...output omitted...)
TASK [backup : Download the database dump.] ************************************
changed: [localhost] => {"changed": true, "checksum":
 "7e243f152da5dd854db3a690fec2938becd58c76", "dest": "/root/ansible-
tower-setup-bundle-3.1.1-1.el7/tower-backup-2017-03-21-07:11:28.tar.gz",
 "md5sum": "806221f8e1e64cea9a0e3cc5107be7ac", "remote_checksum":
 "7e243f152da5dd854db3a690fec2938becd58c76", "remote_md5sum": null}

TASK [backup : Create symbolic link to the most recent backup] *****************
changed: [localhost -> localhost] => {"changed": true, "dest": "/root/
ansible-tower-setup-bundle-3.1.1-1.el7/tower-backup-latest.tar.gz",
 "gid": 0, "group": "root", "mode": "0777", "owner": "root", "secontext":
 "unconfined_u:object_r:admin_home_t:s0", "size": 84, "src": "/root/ansible-tower-
setup-bundle-3.1.1-1.el7/tower-backup-2017-03-21-07:11:28.tar.gz", "state": "link",
 "uid": 0}

RUNNING HANDLER [backup : Remove the backup directory.] ************************
changed: [localhost] => {"changed": true, "path": "/var/backups/
tower/2017-03-21-07:11:28/", "state": "absent"}

RUNNING HANDLER [backup : Remove the backup tarball.] **************************
changed: [localhost] => {"changed": true, "path": "/var/backups/tower/tower-
backup-2017-03-21-07:11:28.tar.gz", "state": "absent"}

PLAY RECAP *********************************************************************
localhost                  : ok=24   changed=16   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-03-21-07:11:21.log
```

7. Verify that the backup was created in the current working directory by issuing the **ls** command.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# ls
backup.yml  install.yml                         inventory_cluster  restore.yml
bundle      inventory                           licenses           roles
  tower-backup-2017-03-21-07:11:28.tar.gz
group_vars  inventory.1521.2017-03-20@11:31:45~  README.md          setup.sh
  tower-backup-latest.tar.gz
```

8. Change the Ansible Tower **admin** superuser password to **redhat2** using the **tower-manage** command.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# tower-manage changepassword
 admin
Changing password for user 'admin'
Password: redhat2
```

```
Password (again): redhat2
Password changed successfully for user 'admin'
```

9. On **workstation** open browser and log in to Ansible Tower as **admin** user using the new **redhat2** password. Then click the **Log Out** icon to log out of the Tower web interface.

10. Go back to **tower** server and restore backup from the file created in previous steps. Use the **setup.sh** command with **-r** as option. After the restore is complete, exit the console session on the **tower** system.

Important

It is important not to forget the **-r**, otherwise the Ansible Tower will start the installation process from the beginning. If that happens wait and when the installation process finishes simply restore the backup.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh -r
(...output ommitted...)
PLAY RECAP *********************************************************************
localhost                  : ok=26   changed=18   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-03-21-11:53:32.log
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# exit
```

11. Verify that the backup has been restored, by logging in to the Ansible Tower web interface as **admin** user with the old **redhat** password. Then click the **Log Out** icon to log out of the Tower web interface.

Cleanup

From **workstation**, run the **lab admin-recovery cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab admin-recovery cleanup
```

Launching Jobs with the Ansible Tower API and **tower-cli**

Objectives

After completing this section, students should be able to:

- Use the Tower API or the unsupported **tower-cli** utility to launch a job from an existing job template.

Ansible Tower's REST API

Ansible Tower provides a Representational State Transfer (REST) API. An API provides a mechanism which allows administrators and developers to control their Ansible Tower environment outside of the web interface. Custom scripts or external applications access the API using standard HTTP messages. This is useful when integrating Ansible Tower with other programs.

One of the benefits of the REST API is that any programming language, framework or system with support for HTTP protocol can use the API. This provides an easy way to automate repetitive tasks and integrate other enterprise IT systems with Ansible Tower.

Note

The API is in active development, and not all features of the web interface may be accessible through the API.

Using the REST API

In case you are not familiar with REST APIs, the way they work is relatively straightforward. A client sends a request to a server element located at a Uniform Resource Identifier (URI) and performs operations with standard HTTP methods, such as GET, POST, PUT, and DELETE. The REST architecture provides a stateless communication channel between the client and server. Each client request acts independently of any other request, and contains all necessary information to complete the request.

The following example request uses an HTTP GET method to retrieve a representation of the main entry point of the API. A graphical web browser or Linux command line tools could be used to issue the request manually. In this particular example, the **curl** command is used to make the request from the command line:

```
[user@demo ~]$ curl -X GET https://tower.lab.example.com/api/ -k
{"available_versions":{"v1":"/api/v1/"},"custom_logo":"","custom_login_info":"",
"description":"Ansible Tower REST API","current_version":"/api/v1/"}
```

The output of the API request is in JSON format, which is readily parsable by computer programs, but may be a little challenging for a human to read.

The Ansible Tower API is browsable. For example, if your Tower server is the host **tower.lab.example.com**, you can access the browsable API at *https://tower.lab.example.com/api/v1/*.

The following example shows how to do this using **curl**. The **json_pp** command comes from the *perl-JSON-PP* RPM package and "pretty-prints" the JSON output of the API for easier reading by a human.

```
[user@demo ~]$ curl -X GET https://tower.lab.example.com/api/v1/ -k -s | json_pp
{
   "settings" : "/api/v1/settings/",
   "groups" : "/api/v1/groups/",
   "project_updates" : "/api/v1/project_updates/",
   "workflow_jobs" : "/api/v1/workflow_jobs/",
   "unified_job_templates" : "/api/v1/unified_job_templates/",
   "job_templates" : "/api/v1/job_templates/",
   "hosts" : "/api/v1/hosts/",
   "me" : "/api/v1/me/",
   "system_jobs" : "/api/v1/system_jobs/",
   "workflow_job_templates" : "/api/v1/workflow_job_templates/",
   "organizations" : "/api/v1/organizations/",
   "inventory_scripts" : "/api/v1/inventory_scripts/",
   "inventory_updates" : "/api/v1/inventory_updates/",
   "workflow_job_template_nodes" : "/api/v1/workflow_job_template_nodes/",
   "activity_stream" : "/api/v1/activity_stream/",
   "credentials" : "/api/v1/credentials/",
   "schedules" : "/api/v1/schedules/",
   "workflow_job_nodes" : "/api/v1/workflow_job_nodes/",
   "system_job_templates" : "/api/v1/system_job_templates/",
   "teams" : "/api/v1/teams/",
   "inventory" : "/api/v1/inventories/",
   "config" : "/api/v1/config/",
   "job_events" : "/api/v1/job_events/",
   "inventory_sources" : "/api/v1/inventory_sources/",
   "jobs" : "/api/v1/jobs/",
   "dashboard" : "/api/v1/dashboard/",
   "notifications" : "/api/v1/notifications/",
   "ad_hoc_commands" : "/api/v1/ad_hoc_commands/",
   "labels" : "/api/v1/labels/",
   "unified_jobs" : "/api/v1/unified_jobs/",
   "users" : "/api/v1/users/",
   "projects" : "/api/v1/projects/",
   "ping" : "/api/v1/ping/",
   "notification_templates" : "/api/v1/notification_templates/",
   "authtoken" : "/api/v1/authtoken/",
   "roles" : "/api/v1/roles/"
}
```

This entry point provides a collection of links in the API environment. As you can see in the example, there are many links to choose from.

Let's look at one example of information accessible through the API. To examine what actions have been performed on the Ansible Tower server, you can use the **/api/v1/activity_stream/** URI. Make a GET request to that resource to retrieve the list of activity streams:

```
[user@demo ~]$ curl -X GET https://tower.lab.example.com/api/v1/activity_stream/ -k
{"detail":"Authentication credentials were not provided."}
```

As you can see in the output above, not all information generated by the API is publicly available. To access this resource, you must to log in with your user credentials.

The next example shows the output of the **activity_stream** resource when proper authentication information is provided:

```
[user@demo ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/activity_stream/ -k -s | json_pp
{
   "previous" : null,
   "count" : 112,
   "next" : "/api/v1/activity_stream/?page=2",
   "results" : [
      {
         "timestamp" : "2017-03-31T23:32:22.585297Z",
         "operation" : "create",
         "id" : 1,
         "object1" : "setting",
         "url" : "/api/v1/activity_stream/1/",
         "object_association" : "",
         "object2" : "",
         "changes" : {
            "id" : 1,
            "value" : "detailed",
            "key" : "PENDO_TRACKING_STATE"
         },
         "related" : {},
         "summary_fields" : {},
         "type" : "activity_stream"
      },
...output omitted...
```

Important

The output of the API may be *paginated*, as in the preceding example. Tower only returns a limited number of records for a particular request for performance reasons. The **`next`** value in the example gives the URI for the next page of results. If it is **`null`**, you are on the last page. Likewise, the value of **`previous`** is the URI for the previous page of results, and if it is **`null`** you are on the first page.

The output of the API is in JSON format, which can be difficult to read without running it through a parser like **`json_pp`**. You can also access the browsable REST API with a graphical web browser to get the same information in a more readable format. This example accesses the same API using the Firefox web browser:

```
GET /api/v1/activity_stream/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
X-API-Node: localhost
X-API-Time: 0.815s
```


```
{
    "count": 112,
    "next": "/api/v1/activity_stream/?page=2",
    "previous": null,
    "results": [
        {
            "id": 1,
            "type": "activity_stream",
            "url": "/api/v1/activity_stream/1/",
            "related": {},
            "summary_fields": {},
            "timestamp": "2017-03-29T07:48:56.991228Z",
            "operation": "create",
            "changes": {
                "key": "PENDO_TRACKING_STATE",
                "id": 1,
```


Figure 7.1: Activity Stream API output

Clicking on various links in the API allows you to explore related resources.

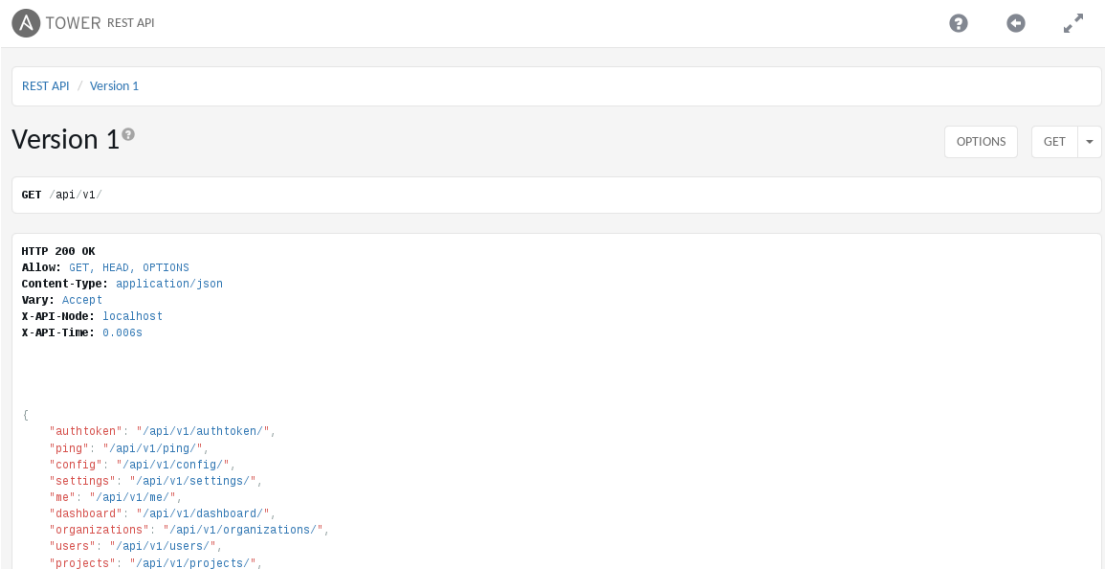

Figure 7.2: Browsable API

Click on the **?** icon next to the name for an API endpoint to get documentation on the access methods for that endpoint. The documentation also provides information on what data is returned when using those methods.

Important

You may find the documentation available to graphical browsers through the **?** icon to be very useful when trying to understand how to use the Tower API. Do not ignore this resource.

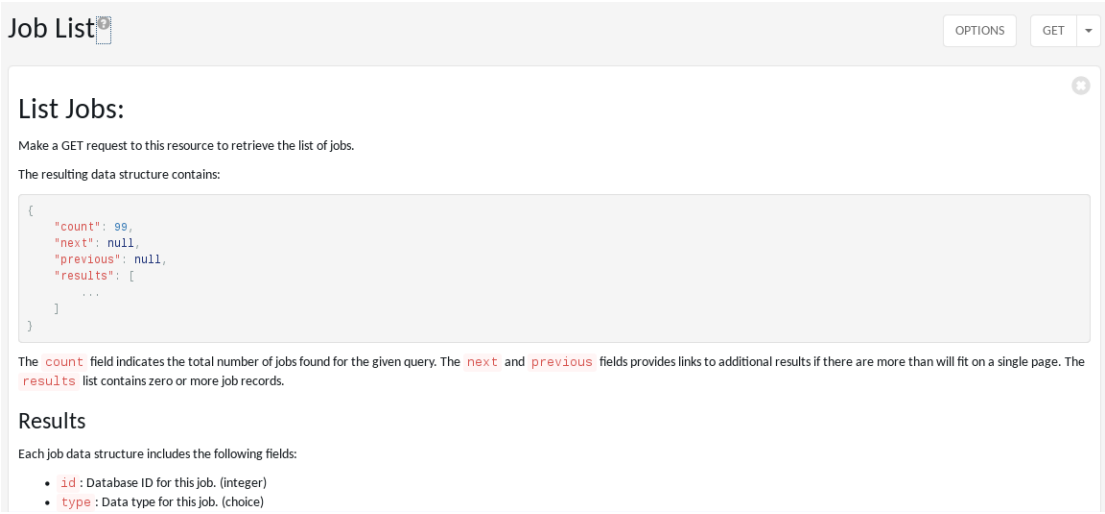

Figure 7.3: Documentation for API endpoints

You can also use PUT or POST methods on the specific API pages by providing JSON formatted text or files in the graphical interface.

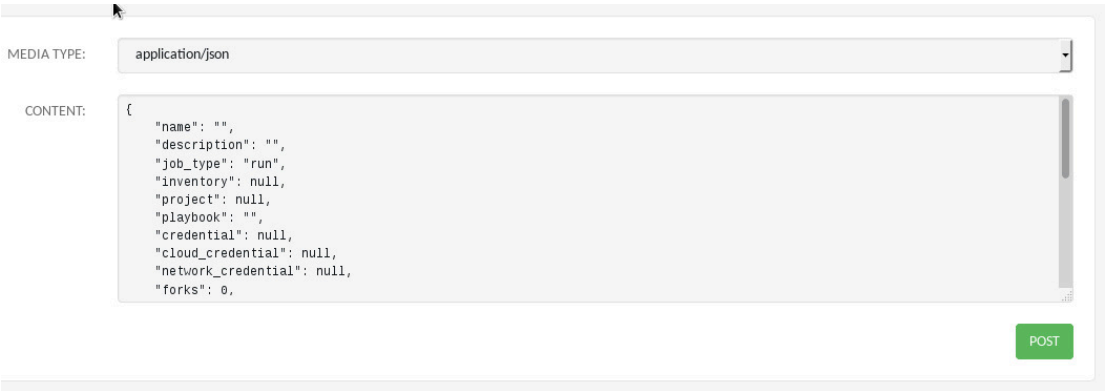

Figure 7.4: Example of POST method text field

Monitor Tower Status Using the API

You can use the Ansible Tower API to monitor the availability of the server hosting the Tower infrastructure. Basic information about the Tower server can be accessed by the publicly available **`/api/v1/ping/`** URI. This example accesses it using the **`curl`** command:

```
[user@demo ~]$ curl -X GET https://tower.lab.example.com/api/v1/ping/ -k -s | json_pp
{
   "active_node" : "localhost",
   "instances" : [
      {
         "node" : "localhost",
         "heartbeat" : "2017-04-11T14:13:28.827Z",
         "capacity" : 50
      }
   ],
   "ha" : false,
   "version" : "3.1.1"
}
```

Launching a Job Template Using the API

One common use of the API is to launch an existing Job Template. This example will quickly outline how to use the API to find and launch a Job Template that has already been configured in Tower, with the **`curl`** command.

First, you need to find the **`id`** number of your Job Template. If you know the name of the Job Template, you can use the API to search for it. For example, if the desired Job Template is named **Demo Job Template**, you can search for it with the following **`curl`** command:

```
[user@demo ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/?name="Demo Job Template" -k -s \
> | json_pp
{
   "results" : [
      {
         "verbosity" : 0,
         "description" : "",
         "type" : "job_template",
         "id" : 5,
         "project" : 4,
         "ask_variables_on_launch" : false,
...output omitted...
```

If you can't remember the exact name of your Job Template, but you know that it includes the word "Demo", you can search for all Job Templates containing the string "Demo" with the following command:

```
[user@demo ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/?name__contains="Demo" -k -s \
> | json_pp
```

Now that you know your Job Template **id**, use it to launch a new job, using its **launch** method. This is done in two steps. First, you may access it with the GET method to get information about any parameters or data that is needed to launch the job. Then, you must access it with the POST method to actually launch the job.

Here is a quick example. Assuming that your Job Template ID is **5**, you may use the following command to see if additional information is needed or can be provided when launching the job:

```
[user@demo ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/5/launch/ -k -s | json_pp
{
   "ask_skip_tags_on_launch" : false,
   "ask_credential_on_launch" : false,
   "ask_tags_on_launch" : false,
   "ask_limit_on_launch" : false,
   "can_start_without_user_input" : true,
   "credential_needed_to_start" : false,
   "defaults" : {
      "job_tags" : "",
      "inventory" : {
         "id" : 1,
         "name" : "Demo Inventory"
      },
      "skip_tags" : "",
      "credential" : {
         "name" : "Demo Credential",
         "id" : 1
      },
      "limit" : "",
      "extra_vars" : "",
      "job_type" : "run"
   },
   "inventory_needed_to_start" : false,
   "ask_job_type_on_launch" : false,
   "passwords_needed_to_start" : [],
   "ask_inventory_on_launch" : false,
   "variables_needed_to_start" : [],
   "ask_variables_on_launch" : false,
   "job_template_data" : {
      "name" : "Demo Job Template",
      "description" : "",
      "id" : 5
   },
   "survey_enabled" : false
}
```

Most of this information is discussed in more detail in the chapter of the *Ansible Tower API Guide* on launching job templates. But notice that the **id** and name of the Inventory and machine Credential for the job is listed, that it will be a run job, and that no extra information is needed to launch the job.

In that case, the job can be launched by accessing the URI with a POST method instead of a GET method as follows. The entire output from the API call is shown in the example. In particular, note the job **id**, the fact that it's in **status** pending since it's been launched but hasn't completed yet, and the other information about the job which is initially returned.

```
[user@demo ~]$ curl -X POST --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/5/launch/ -k -s | json_pp
{
   "start_at_task" : "",
   "ask_credential_on_launch" : false,
   "name" : "Demo Job Template",
   "description" : "",
   "status" : "pending",
   "ask_tags_on_launch" : false,
   "passwords_needed_to_start" : [],
   "ask_inventory_on_launch" : false,
   "related" : {
      "modified_by" : "/api/v1/users/1/",
      "stdout" : "/api/v1/jobs/16/stdout/",
      "credential" : "/api/v1/credentials/1/",
      "relaunch" : "/api/v1/jobs/16/relaunch/",
      "job_host_summaries" : "/api/v1/jobs/16/job_host_summaries/",
      "project" : "/api/v1/projects/4/",
      "notifications" : "/api/v1/jobs/16/notifications/",
      "activity_stream" : "/api/v1/jobs/16/activity_stream/",
      "unified_job_template" : "/api/v1/job_templates/5/",
      "start" : "/api/v1/jobs/16/start/",
      "job_events" : "/api/v1/jobs/16/job_events/",
      "inventory" : "/api/v1/inventories/1/",
      "created_by" : "/api/v1/users/1/",
      "cancel" : "/api/v1/jobs/16/cancel/",
      "labels" : "/api/v1/jobs/16/labels/",
      "job_template" : "/api/v1/job_templates/5/"
   },
   "modified" : "2017-04-18T16:24:55.116Z",
   "ask_skip_tags_on_launch" : false,
   "elapsed" : 0,
   "job_tags" : "",
   "timeout" : 0,
   "force_handlers" : false,
   "extra_vars" : "{}",
   "id" : 16,
   "limit" : "",
   "ask_limit_on_launch" : false,
   "job_env" : {},
   "network_credential" : null,
   "allow_simultaneous" : false,
   "finished" : null,
   "type" : "job",
   "result_traceback" : "",
   "failed" : false,
   "playbook" : "hello_world.yml",
   "scm_revision" : "",
   "job_explanation" : "",
   "job" : 16,
   "result_stdout" : "Waiting for results...",
   "created" : "2017-04-18T16:24:55.049Z",
   "job_template" : 5,
   "inventory" : 1,
   "skip_tags" : "",
   "url" : "/api/v1/jobs/16/",
   "summary_fields" : {
      "unified_job_template" : {
```

```
            "unified_job_type" : "job",
            "id" : 5,
            "description" : "",
            "name" : "Demo Job Template"
         },
         "modified_by" : {
            "username" : "admin",
            "last_name" : "",
            "first_name" : "",
            "id" : 1
         },
         "credential" : {
            "id" : 1,
            "name" : "Demo Credential",
            "cloud" : false,
            "description" : "",
            "kind" : "ssh"
         },
         "project" : {
            "scm_type" : "git",
            "id" : 4,
            "name" : "Demo Project",
            "description" : "",
            "status" : "successful"
         },
         "job_template" : {
            "description" : "",
            "name" : "Demo Job Template",
            "id" : 5
         },
         "labels" : {
            "results" : [],
            "count" : 0
         },
         "user_capabilities" : {
            "start" : true,
            "delete" : true
         },
         "inventory" : {
            "inventory_sources_with_failures" : 0,
            "has_inventory_sources" : false,
            "groups_with_active_failures" : 0,
            "hosts_with_active_failures" : 0,
            "has_active_failures" : false,
            "name" : "Demo Inventory",
            "description" : "",
            "total_inventory_sources" : 0,
            "id" : 1,
            "total_hosts" : 1,
            "total_groups" : 0
         },
         "created_by" : {
            "last_name" : "",
            "username" : "admin",
            "id" : 1,
            "first_name" : ""
         }
      },
      "ignored_fields" : {},
      "execution_node" : "",
      "unified_job_template" : 5,
      "started" : null,
      "job_args" : "",
      "cloud_credential" : null,
```

```
    "project" : 4,
    "ask_variables_on_launch" : false,
    "launch_type" : "manual",
    "credential" : 1,
    "artifacts" : {},
    "job_cwd" : "",
    "verbosity" : 0,
    "forks" : 0,
    "job_type" : "run",
    "ask_job_type_on_launch" : false
}
```

The JSON formatted output of this example shows the **id** of this job happens to be 16. That can be used to get updated status information on job execution, whether it has completed, and so on. For job 16, the URI **/api/v1/jobs/16/** is used, as indicated by the **url** field in the preceding output.

```
[user@demo ~]$ curl -X POST --user admin:redhat \
> https://tower.lab.example.com/api/v1/jobs/16/ -k -s | json_pp
```

In particular this reports job **status** (success or failure), at what time the job **finished**, the **result_stdout** of the Ansible playbook, which **playbook** was used, as well as the other data about inventory, credentials, and project from the job template, among many other things.

When needed, you can also pass extra variables when launching the Job Template as data in the POST request. This example uses **curl** to launch a job, with the **-d** option included to send extra variables as JSON data in the POST request:

```
[user@demo ~]$ curl -X POST --user admin:redhat \
> -d '{"extra_vars": "{\"version=1\": \"web\"}"}' \
> https://tower.lab.example.com/api/v1/job_templates/5/launch/ -k
```

Using tower-cli

A command line tool, **tower-cli**, has been developed by the open source community for use with Ansible Tower. It allows commands to be sent to a Tower server from the command line, using the Ansible Tower API.

Important

The **tower-cli** command is not supported by Red Hat and is not part of the Ansible Tower product.

It is developed by the open source community, and may not support all features of Ansible Tower or be completely in sync with the current release. It is being discussed in the context of this course because some customers have found it useful.

The **tower-cli** project is hosted at *https://github.com/ansible/tower-cli*.

Installation

tower-cli tool is available as a Python package on *PyPI* [https://pypi.python.org/pypi/ansible-tower-cli]. It is also available through *https://github.com/ansible/tower-cli* or as an RPM from the Fedora EPEL **yum** repository.

The easiest way to install may be through using **pip**:

```
[user@demo ~]$ pip install ansible-tower-cli
```

tower-cli can be used to edit its own configuration, or users can directly edit the configuration file. Usually, a user's configuration will be stored in the **.tower_cli.cfg** file in their home directory. The preferred way to configure **tower-cli** is by running it with its **config** option. The syntax is as follows:

```
[user@demo ~]$ tower-cli config key value
```

You can get a full list of configuration options by running **tower-cli config** with no arguments:

```
[user@demo ~]$ tower-cli config
# User options (set with `tower-cli config`; stored in ~/.tower_cli.cfg).
host: tower.lab.example.com
username: admin
password: redhat

# Defaults.
description_on: False
verbose: False
certificate:
format: human
color: True
verify_ssl: True
```

By default, you must set at least three configuration options: **host**, **username**, and **password**. These options correspond to the host name of your Ansible Tower server, your Tower username, and your password to authenticate to Tower.

```
[user@demo ~]$ tower-cli config host tower.lab.example.com
```

```
[user@demo ~]$ tower-cli config username Demo
```

```
[user@demo ~]$ tower-cli config password myDemoPassword
```

To check what versions of Ansible Tower and **tower-cli** you have installed, run the **tower-cli** command with the **version** option.

```
[user@demo ~]$ tower-cli version
Tower CLI 3.1.3
Ansible Tower 3.1.1
```

Other **tower-cli** commands usage follow the format:

```
[user@demo ~]$ tower-cli resource action ...
```

The *resource* is a type of object within Tower, such as **user**, **organization**, or **job_template**. The *action* is what you want to do in the Tower infrastructure. Most resources have the following actions: **get**, **list**, **create**, **modify**, **delete**.

This example shows how to use **tower-cli** to launch a job from the Job Template with **id** 33:

```
[user@demo ~]$ tower-cli job launch --job-template=33
Resource changed.
== ============ ======================== ======= =======
id job_template         created          status  elapsed
== ============ ======================== ======= =======
 4           33 2017-03-24T12:45:14.457Z pending
== ============ ======================== ======= =======
```

Another example is job monitoring, in this case, for the Job with **id** 4:

```
[user@demo ~]$ tower-cli job monitor 4
```

References

Further information is available in the *Ansible Tower API Guide* for Ansible Tower 3.1.1 at
https://docs.ansible.com/ansible-tower/3.1.1/html/towerapi/

Guided Exercise: Launching Jobs with the Ansible Tower API and **tower-cli**

In this exercise, you will use the Tower API and the unsupported **tower-cli** utility to launch a job from an existing job template.

Outcomes
You should be able to:

- Use the Tower API to launch a job from an existing template.
- Use the **tower-cli** utility to launch a job from existing template.

Important

The **tower-cli** utility is open source software not included with Ansible Tower. It is not supported by Red Hat. This exercise will explore that utility as one tool which uses the supported API, and because some customers have found it of value.

Before you begin
Ensure that the **workstation** and **tower** virtual machines are started.

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

From **workstation**, run **lab admin-api setup**, which verifies that Ansible Tower services are running and all the required resources are available.

```
[student@workstation ~]$ lab admin-api setup
```

Steps

1. In the first part of this exercise, you will directly use the REST API provided by Ansible Tower.

   On **workstation**, use the **Firefox** web browser to view the resources available from your Tower server's API.

   1.1. In Firefox, go to the URL **https://tower.lab.example.com/api/**. You should see your Ansible Tower server's browsable API.

   1.2. Click the **/api/v1/** link to access the browsable list of all the available resources accessible through the API.

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
X-API-Node: localhost
X-API-Time: 0.008s



{
```

```
    "authtoken: "/api/v1/authtoken/",
    "ping": "/api/v1/ping/",
    "config": "/api/v1/config/",
...output omitted...
```

1.3. From the list click the **/api/v1/ping/** link to access that URI. You should see a recent heartbeat time stamp. This URI could be used by an external program to verify that the Tower server is operating.

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
X-API-Node: localhost
X-API-Time: 0.008s


{
     "instances": [
         {
             "node": "localhost",
             "heartbeat": "2017-03-27T09:24:17.802Z",
...output omitted...
```

2. With Firefox, use the API to launch a job from the existing Job Template named **Demo Job Template**.

2.1. Click on the **Version 1** link at the top of the page to go back to the **/api/v1/** URI.

2.2. Click on the **/api/v1/job_templates/** link to access the list of available job templates.

2.3. From the list of **`results`**, find the Job Template containing **`"name": "Demo Job Template"`**. Find its **`url`** resource and click on the link to access the template. (The link will likely be something like **/api/v1/job_templates/5/**.)

2.4. On the resulting page, look for the Job Template's **`related`** resource named **`launch`**. Click on the link associated with that resource. (The link will be something like **/api/v1/job_templates/5/launch/**.)

This sends a GET request to the **`launch`** resource for that Job Template, providing information about what information is needed or may be provided to launch a job from the template. Note that the resource **`can_start_without_user_input`** is **`true`**, so you can immediately launch a job without adding information.

2.5. To monitor the execution of the job you are about to launch, open another Firefox tab and log in to the Ansible Tower web interface as the **`admin`** user with the password **`redhat`**. Click the **JOBS** link.

2.6. Go back to the **https://tower.lab.example.com/api/v1/job_templates/5/launch/** tab. Scroll down to the bottom of this page and click the green **POST** button to launch the job.

The page immediately refreshes with JSON information about the launched job, including the job's **`id`** number.

2.7. Quickly switch to the tab displaying the **JOBS** page in the Ansible Tower web interface.

The launched job should be at the top of the list of jobs that have been run. You can confirm it is the job you launched by matching its **ID** with the **`id`** in the JSON output for the job on the other tab.

3. The **`curl`** command can also use the API to launch jobs from existing Job Templates. Use **`curl`** to launch a job from the existing Job Template named **Demo Job Template**.

3.1. To make it easier to read the JSON output provided by **`curl`**, make sure the *perl-JSON-PP* RPM package is installed on **`workstation`**.

```
[student@workstation ~]$ sudo yum -y install perl-JSON-PP
[sudo] password for student:
...output omitted...
```

3.2. Use the API with a **`name`** filter to search for the Job Template named **Demo Job Template**. Determine what the Job Template's **`id`** is. (This should be the same as the one that you saw using Firefox earlier in this exercise.)

```
[student@workstation ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/?name="Demo Job Template" \
> -k -s | json_pp
```

3.3. Now that you have confirmed the **`id`** for **Demo Job Template**, use that number and the Job Template's **`launch`** resource to get information about how to launch the job.

```
[student@workstation ~]$ curl -X GET --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/5/launch/ -k -s \
> | json_pp
{
   "passwords_needed_to_start" : [],
   "ask_limit_on_launch" : false,
   "ask_inventory_on_launch" : false,
   "can_start_without_user_input" : true,
   "defaults" : {
...output omitted...
```

3.4. Again, since the job can be launched without user input, you can issue a POST request to the same URL without any other data in order to launch the job.

```
[student@workstation ~]$ curl -X POST --user admin:redhat \
> https://tower.lab.example.com/api/v1/job_templates/5/launch/ -k -s \
> | json_pp
{
   "unified_job_template" : 5,
   "network_credential" : null,
   "extra_vars" : "{}",
   "scm_revision" : "",
   "job" : 28,
   "modified" : "2017-05-24T16:34:49.086Z",
   "description" : "",
   "job_args" : "",
   "name" : "Demo Job Template",
```

```
...output omitted...
   "id" : 28,
   "verbosity" : 0,
   "timeout" : 0,
   "force_handlers" : false,
   "type" : "job",
   "playbook" : "hello_world.yml"
}
```

Note the **id** number for the launched job in the JSON output.

3.5. If you still have the tab open which displayed the **JOBS** page in the Ansible Tower web interface, go to it.

The job launched by **curl** should now be at the top of the list of jobs that have been run. Again, you can confirm that it is the job you just launched by comparing the **ID** in the web interface with the **id** in the JSON output from **curl**.

4. In the next few steps of this exercise, you will use the unsupported **tower-cli** command to launch jobs based on an existing Job Template.

On **workstation**, open a terminal. Use **su** to switch to the **support** user, using **redhat** as the password. We're using that separate user so that you can set up that user's **tower-cli** configuration cleanly.

```
[student@workstation ~]$ su - support
Password: redhat
[support@workstation ~]$
```

5. Create a new configuration file for the **tower-cli** utility.

5.1. Create a host configuration entry using the **tower-cli** command with the **config host** subcommand.

```
[support@workstation ~]$ tower-cli config host tower.lab.example.com
Configuration updated successfully.
```

5.2. Create a username configuration to authenticate to Ansible Tower. Use **tower-cli** command with the **config username** subcommand.

```
[support@workstation ~]$ tower-cli config username support
Configuration updated successfully.
```

5.3. Create a password configuration using the **tower-cli** command with the **config password** subcommand.

```
[support@workstation ~]$ tower-cli config password redhat
Configuration updated successfully.
```

5.4. View the newly created configuration file using the **cat** command.

```
[support@workstation ~]$ cat .tower_cli.cfg
[general]
```

```
host = tower.lab.example.com
username = support
password = redhat
```

6. Use the **tower-cli** command to launch the existing **Demo Job Template**, and to monitor the resulting job.

   6.1. List the available job templates using **tower-cli job_template list**. Find the **id** for **Demo Job Template**.

```
[support@workstation ~]$ tower-cli job_template list
== ================ ========= ======= ===============
id       name        inventory project    playbook
== ================ ========= ======= ===============
 5 Demo Job Template         1       4 hello_world.yml
== ================ ========= ======= ===============
```

   6.2. Launch a new job from **Demo Job Template** using the **tower-cli job launch** command, specifying the template's ID with the **--job-template=** option.

```
[support@workstation ~]$ tower-cli job launch --job-template=5
Resource changed.
== ============ ======================== ======= =======
id job_template         created          status  elapsed
== ============ ======================== ======= =======
 4            5 2017-03-24T12:45:14.457Z pending
== ============ ======================== ======= =======
```

   6.3. Monitor the status of the running job using the command, **tower-cli job monitor**, with the **id** of the job you launched in the previous step as an argument. (This will probably be different than the **id** shown in the sample output above.)

```
[support@workstation ~]$ tower-cli job monitor 4
Current status: pending...
...output omitted...
------Starting Standard Out Stream------

PLAY [Hello World Sample] ******************************************************

TASK [setup] *******************************************************************
ok: [localhost]

TASK [Hello Message] ***********************************************************
ok: [localhost] => {
    "msg": "Hello World!"
}

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0

------End of Standard Out Stream--------
Resource changed.
== ============ ======================== ========== =======
id job_template         created            status   elapsed
== ============ ======================== ========== =======
4            5 2017-03-29T09:37:03.361Z successful  20.778
== ============ ======================== ========== =======
```

7. This concludes this exercise. Exit the **support** user's session.

```
[support@workstation ~]$ exit
logout
[student@workstation ~]$
```

Cleanup

From **workstation**, run the **lab admin-api cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab admin-api cleanup
```

Quiz: Performing Maintenance and Routine Administration of Ansible Tower

Choose the correct answer(s) to the following questions:

1. Which of the following commands can be used to check the status of Ansible Tower services?

   a. **`firewall-cmd --list-services`**
   b. **`systemctl status ansible-tower`**
   c. **`ansible-tower-service status`**
   d. **`service tower status`**

2. Where is the default location of Ansible Tower TLS/SSL certificate defined?

   a. **`/etc/tower/tower.cert`**
   b. **`/etc/httpd/conf/ssl.conf`**
   c. **`/etc/nginx/nginx.conf`**
   d. **`/etc/nginx/conf.d/nginx.conf`**

3. Using the existing template named "Changing hostnames" with the ID of 91, which two commands will launch a new job? (Choose two.)

   a. **`tower-cli job launch --job_template=91`**
   b. **`tower-manage job launch --job-template=91`**
   c. **`tower-cli job launch --job-template=91`**
   d. **`tower-cli job monitor 91`**
   e. **`tower-cli job launch -J "Changing hostnames"`**

4. Which of the following resources exist in the Ansible Tower API? (Choose two.)

   a. **`https://tower.demo.com/api/v1/clean_jobs/`**
   b. **`https://tower.demo.com/api/v1/password_change/`**
   c. **`https://tower.demo.com/api/v1/ping/`**
   d. **`https://tower.demo.com/api/v1/job_templates/`**

5. What is the default Ansible Tower log directory location ?

   a. **`/etc/tower/logs/`**
   b. **`/var/log/ansible/`**
   c. **`/var/log/tower/`**
   d. **`/var/awx/log/tower/`**

Solution

Choose the correct answer(s) to the following questions:

1. Which of the following commands can be used to check the status of Ansible Tower services?

   a. `firewall-cmd --list-services`
   b. `systemctl status ansible-tower`
   c. `ansible-tower-service status`
   d. `service tower status`

2. Where is the default location of Ansible Tower TLS/SSL certificate defined?

   a. `/etc/tower/tower.cert`
   b. `/etc/httpd/conf/ssl.conf`
   c. `/etc/nginx/nginx.conf`
   d. `/etc/nginx/conf.d/nginx.conf`

3. Using the existing template named "Changing hostnames" with the ID of 91, which two commands will launch a new job? (Choose two.)

   a. `tower-cli job launch --job_template=91`
   b. `tower-manage job launch --job-template=91`
   c. `tower-cli job launch --job-template=91`
   d. `tower-cli job monitor 91`
   e. `tower-cli job launch -J "Changing hostnames"`

4. Which of the following resources exist in the Ansible Tower API? (Choose two.)

   a. `https://tower.demo.com/api/v1/clean_jobs/`
   b. `https://tower.demo.com/api/v1/password_change/`
   c. `https://tower.demo.com/api/v1/ping/`
   d. `https://tower.demo.com/api/v1/job_templates/`

5. What is the default Ansible Tower log directory location ?

   a. `/etc/tower/logs/`
   b. `/var/log/ansible/`
   c. `/var/log/tower/`
   d. `/var/awx/log/tower/`

Summary

In this chapter, you learned:

- How to start, stop, restart Ansible Tower infrastructure.
- Where to find the Ansible Tower configuration and log file.
- How to replace the self-signed certificate with a custom TLS/SSL certificate.
- How to change the **`admin`** password and import a static inventory file from the command line using **`tower-manage`**.
- How to create a backup and restore the Ansible Tower infrastructure.
- How to access and use the Ansible Tower REST API to launch jobs.
- How to launch a new job with the Ansible Tower REST API from the command line using **`tower-cli`** command.

CHAPTER 8

COMPREHENSIVE REVIEW: PROVISIONING AND MANAGING SYSTEMS USING ANSIBLE TOWER

Overview	
Goal	Demonstrate skills learned in this course by configuring an operating a new Organization in Ansible Tower using a provided specification, Ansible projects, and hosts to be provisioned and managed
Objectives	On an existing Ansible Tower server: • Create a new organization • Configure the organization with users, teams, inventories, projects, job templates, and credentials according to a specification • Successfully launch Ansible jobs to provision systems
Lab	• Lab: Restoring Ansible Tower from Backup • Lab: Adding Users and Teams • Lab: Creating a Custom Dynamic Inventory • Lab: Configuring Job Templates • Lab: Configuring Workflow Job Templates, Surveys, and Notifications • Lab: Testing the Prepared Environment

Lab: Restoring Ansible Tower from Backup

In this review, you will restore the Ansible Tower configuration from an existing backup.

Outcomes
You should be able to restore the Ansible Tower configuration from backup.

Before you begin
In this exercise, you will restore the configuration of your Ansible Tower server using a backup archive.

Important

Your work from earlier exercises in this course will be erased from Tower by this exercise. You will start this exercise with an empty Ansible Tower environment.

Set up your computers for this exercise by logging into **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab review-review setup
```

Instructions
On **tower.lab.example.com**, use the archive available at *http://materials.example.com/classroom/ansible/tower-backup-latest.tar.gz* to restore the Ansible Tower database from backup.

Evaluation
Log into your Tower as **admin** and verify that the configuration has been restored. One indication is that there should only be one Job Template, **Demo Job Template**, present after the restoration.

Solution

In this review, you will restore the Ansible Tower configuration from an existing backup.

Outcomes
You should be able to restore the Ansible Tower configuration from backup.

Before you begin
In this exercise, you will restore the configuration of your Ansible Tower server using a backup archive.

Important

Your work from earlier exercises in this course will be erased from Tower by this exercise. You will start this exercise with an empty Ansible Tower environment.

Set up your computers for this exercise by logging into **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab review-review setup
```

Instructions
On **tower.lab.example.com**, use the archive available at *http://materials.example.com/classroom/ansible/tower-backup-latest.tar.gz* to restore the Ansible Tower database from backup.

Steps
1. Restoring the Ansible Tower infrastructure from backup.

   1.1. Use **ssh** to log in as **root** on the **tower** server.

   ```
   [student@workstation ~]$ ssh root@tower
   Last login: Wed May  3 16:54:05 2017 from workstation.lab.example.com
   [root@tower ~]#
   ```

   1.2. Change directory to **/root/ansible-tower-setup-bundle-3.1.1-1.el7**.

   ```
   [root@tower ~]# cd /root/ansible-tower-setup-bundle-3.1.1-1.el7
   [root@tower ansible-tower-setup-bundle-3.1.1-1.el7]#
   ```

   1.3. Use the **wget** command to download the backup archive from *http://materials.example.com/classroom/ansible/tower-backup-latest.tar.gz*.

   ```
   [root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# wget http://
   materials.example.com/classroom/ansible/tower-backup-latest.tar.gz
   ...output omitted...
   Saving to: 'tower-backup-latest.tar.gz'

   100%[=====================================>] 44,787      --.-K/s   in 0s

   2017-05-17 07:01:46 (110 MB/s) - 'tower-backup-latest.tar.gz' saved
    [44787/44787]
   ```

1.4. Using the **setup.sh** command with the **-r** option, restore Ansible Tower from backup.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# ./setup.sh -r
...output omitted...
PLAY RECAP *********************************************************************
localhost                  : ok=26   changed=18   unreachable=0    failed=0

The setup process completed successfully.
Setup log saved to /var/log/tower/setup-2017-05-17-07:05:15.log
```

Evaluation

Log into your Tower as **admin** and verify that the configuration has been restored. One indication is that there should only be one Job Template, **Demo Job Template**, present after the restoration.

Lab: Adding Users and Teams

In this review, you will create new users and a new team.

Outcomes
You should be able to:

- Create a second Ansible Tower superuser.
- Create normal users.
- Create a new team.
- Add users to a team.

Before you begin
You must have completed the preceding exercise which restored a specific Ansible Tower backup on your server.

Instructions
Configure your Ansible Tower system at **`tower.lab.example.com`** based on the following specification:

- Add a second Tower **System Administrator** named **`admin2`**, which has **`redhat123`** as a password.
- Add a **Normal User** to the **`Default`** organization. Use the following information:

Field	Value
First Name	Anny
Last Name	Mage
Email	**`anny@lab.example.com`**
Username	**`anny`**
Password	**`redhat123`**

- Add a second **Normal User** to the **`Default`** organization. Use the following information:

Field	Value
First Name	Robert
Last Name	Farnham
Email	**`robert@lab.example.com`**
Username	**`robert`**
Password	**`redhat123`**

- Within the **`Default`** organization, create a Team called **`Devops`**. Give the team a description of **`Devops Team`**.
- Add **`anny`** as Member of the **`Devops`** Team. Add **`robert`** as Admin of the **`Devops`** Team.

Evaluation

Use the Tower interface to verify to your satisfaction that the users are correctly configured.

Note

There is a grading script that you will run at the end of the comprehensive review to evaluate all your work in this chapter.

Solution

In this review, you will create new users and a new team.

Outcomes
You should be able to:

- Create a second Ansible Tower superuser.
- Create normal users.
- Create a new team.
- Add users to a team.

Before you begin
You must have completed the preceding exercise which restored a specific Ansible Tower backup on your server.

Instructions
Configure your Ansible Tower system at **`tower.lab.example.com`** based on the following specification:

- Add a second Tower **System Administrator** named **`admin2`**, which has **`redhat123`** as a password.
- Add a **Normal User** to the **`Default`** organization. Use the following information:

Field	Value
First Name	Anny
Last Name	Mage
Email	**`anny@lab.example.com`**
Username	**`anny`**
Password	**`redhat123`**

- Add a second **Normal User** to the **`Default`** organization. Use the following information:

Field	Value
First Name	Robert
Last Name	Farnham
Email	**`robert@lab.example.com`**
Username	**`robert`**
Password	**`redhat123`**

- Within the **`Default`** organization, create a Team called **`Devops`**. Give the team a description of **`Devops Team`**.
- Add **`anny`** as Member of the **`Devops`** Team. Add **`robert`** as Admin of the **`Devops`** Team.

Steps

1. This task can be accomplished by using either the Ansible Tower web interface or the **`tower-manage`** command. This example solution uses the **`tower-manage`** command.

   1.1. On **`tower.lab.example.com`** server use **`tower-manage`** to create new superuser **`admin2`**.

```
[root@tower ansible-tower-setup-bundle-3.1.1-1.el7]# tower-manage \
>  createsuperuser
Username (leave blank to use 'root'): admin2
Email address: admin2@lab.example.com
Password: redhat123
Password (again): redhat123
Superuser created successfully.
```

   1.2. Log into the Tower web interface with the user **`admin2`** and the password **`redhat123`** to verify your changes.

   1.3. Log out from the Tower web interface.

2. Log into the Tower web interface as **`admin`** user, open the **SETTINGS** page by clicking the gear icon in the top right.

   2.1. Click the **USERS** link in the middle of the page to manage **`Users`**.

   2.2. Click the **+ADD** button in the middle right to add a new User.

   2.3. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Anny
LAST NAME	Mage
EMAIL	anny@lab.example.com
USERNAME	anny
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

   2.4. Click **SAVE** to create the new User.

   2.5. Scroll down the next screen and click the **+ADD** button in the middle right to add another User.

   2.6. On the next screen, fill in the details as follows:

Field	Value
FIRST NAME	Robert
LAST NAME	Farnham

Field	Value
EMAIL	robert@lab.example.com
USERNAME	robert
ORGANIZATION	Default
PASSWORD	redhat123
CONFIRM PASSWORD	redhat123
USER TYPE	Normal User

2.7. Click **SAVE** to create the new User.

3. To create the **Devops** Team:

3.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

3.2. Click **TEAMS** to manage Teams.

3.3. Click **+ADD** to add a new Team.

3.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Devops
DESCRIPTION	Devops Team
ORGANIZATION	Default

3.5. Click **SAVE** to create the new Team.

4. To add Users to **Devops** Team:

4.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

4.2. Click **TEAMS** to manage Teams.

4.3. Click the link for the **Devops** Team you created previously.

4.4. Click **USERS** to manage the Team's Users.

4.5. Click **+ADD** to add a new User to the Team.

4.6. In the first section on the screen, check the box next to **anny** and **robert** to select those Users. This adds **anny** and **robert** to the list of selected Users in the second section below.

4.7. On the second section, assign the Member role to **Anny Mage** and the Admin role to **Robert Farnham**.

4.8. Click **SAVE** button to save the changes.

4.9. You can verify on the next screen that **anny** is displayed in the list as a Member and **robert** as an Admin.

Evaluation

Use the Tower interface to verify to your satisfaction that the users are correctly configured.

Note

There is a grading script that you will run at the end of the comprehensive review to evaluate all your work in this chapter.

Lab: Creating a Custom Dynamic Inventory

In this review, you will create a Dynamic Inventory using a custom script.

Outcomes
You should be able to:

- Install custom inventory script.
- Create a Dynamic Inventory.

Before you begin
The previous exercises in this comprehensive review chapter must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with a custom dynamic inventory script, **`ldap-freeipa.py`**, based on the following specification:

- Add the **`ldap-freeipa.py`** custom inventory script to Tower. The Python script can be downloaded from *http://materials.example.com/classroom/ansible/ipa-setup/ldap-freeipa.py*.
- Create a new Inventory called **`Dynamic Inventory`**. Use **`Default`** organization, give the script a Description of **`Dynamic Inventory for IPA Server`**.
- Create a Group called **`Dynamic Group`** within the Inventory **`Dynamic Inventory`**. Give it a **Description** of **`Dynamic Group from IPA Server`**. As **Source** choose **`Custom Script`**.
- Update the Dynamic Inventory. When finished, review every group synchronized from the IPA Server, ensure that all groups contain hosts.
- Create a new Machine Credential, **`Devops`**. Use the following information:

Field	Value
Name	**`Devops`**
Description	Devops Credential
Organization	Default
Type	Machine
Username	student
Password	student
Privilege Escalation	Sudo
Privilege Escalation Username	root
Privilege Escalation Password	student

- Grant the **`Admin`** role on the **`Devops`** Credential to the **`Devops`** Team.

Evaluation
Use the Tower interface to verify to your satisfaction that the dynamic inventory script is correctly configured.

Note

There is a grading script that you will run at the end of the comprehensive review to evaluate all your work in this chapter.

Solution

In this review, you will create a Dynamic Inventory using a custom script.

Outcomes
You should be able to:

- Install custom inventory script.
- Create a Dynamic Inventory.

Before you begin
The previous exercises in this comprehensive review chapter must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with a custom dynamic inventory script, **`ldap-freeipa.py`**, based on the following specification:

- Add the **`ldap-freeipa.py`** custom inventory script to Tower. The Python script can be downloaded from *http://materials.example.com/classroom/ansible/ipa-setup/ldap-freeipa.py*.
- Create a new Inventory called **`Dynamic Inventory`**. Use **`Default`** organization, give the script a Description of **`Dynamic Inventory for IPA Server`**.
- Create a Group called **`Dynamic Group`** within the Inventory **`Dynamic Inventory`**. Give it a **Description** of **`Dynamic Group from IPA Server`**. As **Source** choose **`Custom Script`**.
- Update the Dynamic Inventory. When finished, review every group synchronized from the IPA Server, ensure that all groups contain hosts.
- Create a new Machine Credential, **`Devops`**. Use the following information:

Field	Value
Name	**`Devops`**
Description	Devops Credential
Organization	Default
Type	Machine
Username	student
Password	student
Privilege Escalation	Sudo
Privilege Escalation Username	root
Privilege Escalation Password	student

- Grant the **`Admin`** role on the **`Devops`** Credential to the **`Devops`** Team.

Steps
1. To add the **`ldap-freeipa.py`** custom inventory script to Ansible Tower:
   1.1. Log into the Ansible Tower as **`admin`** user.

1.2. Open the **SETTINGS** page by clicking the gear icon in the top right.

1.3. Click **INVENTORY SCRIPTS** to manage custom inventory scripts.

1.4. Click **+ADD** to add a custom inventory script.

1.5. On the next screen, fill in the details as follows:

Field	Value
NAME	ldap-freeipa.py
DESCRIPTION	Dynamic Inventory for IPA Server
ORGANIZATION	Default

1.6. Copy the contents of the **`ldap-freeipa.py`** script located at **`http://materials.example.com/classroom/ansible/ipa-setup/ldap-freeipa.py`** into the **CUSTOM SCRIPT** field.

1.7. Click **SAVE** to add the custom inventory script.

2. Click the **INVENTORIES** quick navigation link.

2.1. Click **+ADD** to create a new Inventory.

2.2. On the next screen, fill in the details as follows:

Field	Value
NAME	Dynamic Inventory
DESCRIPTION	Dynamic Inventory from IPA server
ORGANIZATION	Default

2.3. Click **SAVE** to create the Inventory. This will redirect you to the **`Dynamic Inventory`** details page.

3. Within the **`Dynamic Inventory`**:

3.1. Click **+ADD GROUP** to add a Group to the Inventory.

3.2. On the next screen, fill in the details as follows:

Field	Value
NAME	Dynamic Group
DESCRIPTION	Dynamic Group from IPA server
SOURCE	Custom Script
CUSTOM INVENTORY SCRIPT	ldap-freeipa.py

3.3. Click **SAVE** to create the Group.

4. Scroll down to the **GROUPS** section and click the **Start sync process** icon on the same line as the group, **`Dynamic Group`**.

5. When the update of the Inventory has completed, click on **Dynamic Group**.

6. Observe that **`Dynamic Group`** contains four Groups: **`development`**, **`ipaservers`**, **`production`**, and **`testing`**. Each of these groups contains one or more hosts.

7. To create new Machine Credentials:

   7.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   7.2. Click **CREDENTIALS** to manage Credentials.

   7.3. Click the **+ADD** button to add a new Credential.

   7.4. Create a new Credential, **Devops**, with the following information:

Field	**Value**
NAME	Devops
DESCRIPTION	Devops Credential
ORGANIZATION	Default
TYPE	Machine
USERNAME	student
PASSWORD	student
PRIVILEGE ESCALATION	Sudo
PRIVILEGE ESCALATION USERNAME	root
PRIVILEGE ESCALATION PASSWORD	student

   7.5. Leave the other fields untouched and click **SAVE** to create the new Credential.

8. To grant Admin role on Credential:

   8.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

   8.2. Click **CREDENTIALS** to manage Credentials.

   8.3. On the same line as the **Devops** Credential, click the pencil icon to edit the Credential.

   8.4. On the next page, click **PERMISSIONS** to manage the Credential's permissions.

   8.5. Click the **+ADD** button to add permissions.

   8.6. Click **TEAMS** to display the list of available Teams.

   8.7. In the first section, check the box next to the **Devops** Team. This causes the Team to display in the second section underneath the first one.

   8.8. In the second section below, select the **`Admin`** Role from the dropdown list.

   8.9. Click **SAVE** to finalize the role assignment. This redirects you to the list of permissions for the **Devops** Credential, which now shows that the Users, **anny** and **`robert`**, are assigned the **`Admin`** role on the **Devops** Credential.

Evaluation

Use the Tower interface to verify to your satisfaction that the dynamic inventory script is correctly configured.

Note

There is a grading script that you will run at the end of the comprehensive review to evaluate all your work in this chapter.

Lab: Configuring Job Templates

In this review, you will configure several Job Templates, as well as a supporting Project and SCM Credential.

Outcomes
You should be able to:

- Create an SCM credential.
- Create a Project.
- Create a Job Template.

Before you begin
The previous exercises in this comprehensive review chapter must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with three new Job Templates and supporting materials based on the following specification:

- Create a new SCM Credential that will be needed in order to download the Ansible materials for the Project that the new Job Templates will use, based on the following information:

Field	Value
NAME	student-git
DESCRIPTION	Student Git Credential
ORGANIZATION	Default
TYPE	Source Control
USERNAME	**`git`**
SCM PRIVATE KEY	Copy the contents of the **`/home/student/.ssh/lab_rsa`** private key file on **`workstation`** into this field

- Create a new Project based on the following information:

Field	Value
NAME	My Full-Stack Project
DESCRIPTION	Full Stack Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	*ssh://git.lab.example.com/home/git/full-stack-setup.git*
SCM CREDENTIAL	student-git

- Create a new Job Template called **`Setup Databases`** with the following configuration:

Field	Value
NAME	Setup Databases
DESCRIPTION	Setup all databases
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**`database-setup.yml`**
MACHINE CREDENTIAL	Devops

- Create a new Job Template called **`Setup Webservers`** with the following configuration:

Field	Value
NAME	Setup Webservers
DESCRIPTION	Setup all webservers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**`webserver-setup.yml`**
MACHINE CREDENTIAL	Devops

- Create a new Job Template called **`Setup Load Balancer`** with the following configuration:

Field	Value
NAME	Setup Load Balancer
DESCRIPTION	Setup all load balancers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**`lb-setup.yml`**
MACHINE CREDENTIAL	Devops

- The **`Devops`** Team should have **`Admin`** role on all three Job Templates (**`Setup Databases`**, **`Setup Webservers`**, and **`Setup Load Balancer`**) and on the Project (**`My Full-Stack Project`**).

Evaluation

Use the Ansible Tower interface to verify to your satisfaction that the project materials update and that your Job Templates are configured correctly.

Note

You will use these templates to launch jobs in an upcoming exercise, and there is a grading script at the comprehensive review that you will run to evaluate all your work in this chapter.

Solution

In this review, you will configure several Job Templates, as well as a supporting Project and SCM Credential.

Outcomes
You should be able to:

- Create an SCM credential.
- Create a Project.
- Create a Job Template.

Before you begin
The previous exercises in this comprehensive review chapter must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with three new Job Templates and supporting materials based on the following specification:

- Create a new SCM Credential that will be needed in order to download the Ansible materials for the Project that the new Job Templates will use, based on the following information:

Field	Value
NAME	student-git
DESCRIPTION	Student Git Credential
ORGANIZATION	Default
TYPE	Source Control
USERNAME	**`git`**
SCM PRIVATE KEY	Copy the contents of the **`/home/student/.ssh/lab_rsa`** private key file on **`workstation`** into this field

- Create a new Project based on the following information:

Field	Value
NAME	My Full-Stack Project
DESCRIPTION	Full Stack Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	*ssh://git.lab.example.com/home/git/full-stack-setup.git*
SCM CREDENTIAL	student-git

- Create a new Job Template called **`Setup Databases`** with the following configuration:

Field	Value
NAME	Setup Databases
DESCRIPTION	Setup all databases
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**database-setup.yml**
MACHINE CREDENTIAL	Devops

- Create a new Job Template called **Setup Webservers** with the following configuration:

Field	Value
NAME	Setup Webservers
DESCRIPTION	Setup all webservers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**webserver-setup.yml**
MACHINE CREDENTIAL	Devops

- Create a new Job Template called **Setup Load Balancer** with the following configuration:

Field	Value
NAME	Setup Load Balancer
DESCRIPTION	Setup all load balancers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	**lb-setup.yml**
MACHINE CREDENTIAL	Devops

- The **Devops** Team should have **Admin** role on all three Job Templates (**Setup Databases**, **Setup Webservers**, and **Setup Load Balancer**) and on the Project (**My Full-Stack Project**).

Steps

1. Steps to create the new SCM Credential:

   1.1. Open the **SETTINGS** page by clicking the gear icon in the Administration tool links.

   1.2. Click **CREDENTIALS** to manage Credentials.

   1.3. Click the **+ADD** button to add a new Credential.

1.4. On the next screen, fill in the details as follows:

Field	Value
NAME	student-git
DESCRIPTION	Student Git Credential
ORGANIZATION	Default
TYPE	Source Control
USERNAME	**`git`**
SCM PRIVATE KEY	Copy the contents of the **`/home/student/.ssh/lab_rsa`** private key file on **`workstation`** into this field

1.5. Leave the other fields untouched and click **SAVE** to create the new Credential.

2. To create the new Project:

2.1. Click the **PROJECTS** quick navigation link.

2.2. Click the **+ADD** button to add a new Project.

2.3. On the next screen, fill in the details as follows:

Field	Value
NAME	My Full-Stack Project
DESCRIPTION	Full Stack Project
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	ssh://git.lab.example.com/home/git/full-stack-setup.git
SCM CREDENTIAL	student-git

2.4. Click **SAVE** to create the new Project. This automatically triggers the SCM update of the Project. Ansible Tower uses the values provided in the **SCM URL** and **SCM CREDENTIAL** fields to pull down a local copy of that repository.

3. To verify the success of the **`My Full-Stack Project`** automatic SCM update:

3.1. Scroll down the page and wait a couple of seconds. In the list of Projects, there is a status icon left of the Project, **`My Full-Stack Project`**. This icon is white at the start, red with an exclamation mark when it fails, and green when it succeeds.

3.2. Click on the status icon to show the detailed status page of the SCM update job. As you can see in the **STANDARD OUT** window, the SCM update job runs like any other Ansible Playbook.

3.3. Verify that the **STATUS** of the job in the **RESULTS** section shows **`Successful`**.

4. To give **`Devops`** Team **`Admin`** role on the new Project:

4.1. Click the **PROJECTS** quick navigation link.

4.2. On the same line as the Project, **`My Full-Stack Project`**, click the pencil icon on the right to edit the Project.

4.3. On the next page, click **PERMISSIONS** to manage the Project's permissions.

4.4. Click the **+ADD** button on the right to add permissions.

4.5. Click **TEAMS** to display the list of available Teams.

4.6. In the first section, check the box next to the **`Devops`** Team. This causes the Team to display in the second section underneath the first one.

4.7. In the second section below, select the **`Admin`** role from the dropdown list.

4.8. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Project, **`My Full-Stack Project`**, which now shows that all members of the **`Devops`** Team are assigned the **`Admin`** role on the Project.

5. To create new Job Template called **`Setup Databases`**:

5.1. Click the **TEMPLATES** quick navigation link

5.2. Click the **+ADD** button to add a new Job Template.

5.3. From the drop-down list, select **Job Template**.

5.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Setup Databases
DESCRIPTION	Setup all databases
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	database-setup.yml
MACHINE CREDENTIAL	Devops

5.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

6. To grant the **`Devops`** Team **`Admin`** role on the **`Setup Databases`** Job Template:

6.1. Click **PERMISSIONS** to manage the Job Template's permissions.

6.2. Click the **+ADD** button on the right to add permissions.

6.3. Click **TEAMS** to display the list of available Teams.

6.4. In the first section, check the box next to **`Devops`** Team. This causes the Team to display in the second section underneath the first one.

6.5. In the second section below, select the **Admin** role from the drop-down list.

6.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Job Template, **Setup Databases**, which now shows that all members of the **Devops** Team are assigned the **Admin** role on the Job Template.

7. To create new Job Template called **Setup Webservers**:

7.1. Click the **TEMPLATES** quick navigation link

7.2. Click the **+ADD** button to add a new Job Template.

7.3. From the drop-down list, select **Job Template**.

7.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Setup Webservers
DESCRIPTION	Setup all webservers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	webserver-setup.yml
MACHINE CREDENTIAL	Devops

7.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

8. Grant the **Devops** Team **Admin** role on the **Setup Webservers** Job Template:

8.1. Click **PERMISSIONS** to manage the Job Template's permissions.

8.2. Click the **+ADD** button on the right to add permissions.

8.3. Click **TEAMS** to display the list of available Teams.

8.4. In the first section, check the box next to **Devops** Team. This causes the Team to display in the second section underneath the first one.

8.5. In the second section below, select the **Admin** role from the drop-down list.

8.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Job Template, **Setup Webservers**, which now shows that all members of the **Devops** Team are assigned the **Admin** role on the Job Template.

9. To create new Job Template called **Setup Load Balancer**:

9.1. Click the **TEMPLATES** quick navigation link

9.2. Click the **+ADD** button to add a new Job Template.

9.3. From the drop-down list, select **Job Template**.

9.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Setup Load Balancer
DESCRIPTION	Setup all loadbalancers
JOB TYPE	Run
INVENTORY	Dynamic Inventory
PROJECT	My Full-Stack Project
PLAYBOOK	lb-setup.yml
MACHINE CREDENTIAL	Devops

9.5. Leave the other fields untouched and click **SAVE** to create the new Job Template.

10. To grant the **Devops** Team **Admin** role on **Setup Load Balancer** Job Template:

10.1. Click **PERMISSIONS** to manage the Job Template's permissions.

10.2. Click the **+ADD** button on the right to add permissions.

10.3. Click **TEAMS** to display the list of available Teams.

10.4. In the first section, check the box next to **Devops** Team. This causes the Team to display in the second section underneath the first one.

10.5. In the second section below, select the **Admin** role from the drop-down list.

10.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Job Template, **Setup Load Balancer**, which now shows that all members of the **Devops** Team are assigned the **Admin** role on the Job Template.

Evaluation

Use the Ansible Tower interface to verify to your satisfaction that the project materials update and that your Job Templates are configured correctly.

Note

You will use these templates to launch jobs in an upcoming exercise, and there is a grading script at the comprehensive review that you will run to evaluate all your work in this chapter.

Lab: Configuring Workflow Job Templates, Surveys, and Notifications

In this review, you will create a Workflow Job Template which uses a survey and an email notification.

Outcomes
You should be able to:

- Create a Workflow Job Template.
- Add a Survey to an existing Workflow Job Template.
- Create and activate an e-mail Notification Template.

Before you begin
The previous exercises in this comprehensive review must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with a Workflow Job Template that uses your existing Job Templates, a new Survey, and a new Notification Template, based on the following specification:

- Create a Workflow Job Template in the **`Default`** organization called **`Full Stack Deployment`**. Its **Description** should be **`Deploy the Full Stack`**. Define an Extra Variable **`environment_name`** with the value **`"Development"`**.

  The Workflow Job Template should include the following steps:

  1. Sync the Inventory, **`Dynamic Inventory`**.
  2. Upon success of the previous step, sync the Project, **`My Full-Stack Project`**.
  3. Upon success of the previous step, launch a job using the Job Template, **`Setup Databases`**.
  4. Upon success of the previous step, launch a job using the Job Template, **`Setup Webservers`**.
  5. Upon success of the previous step, launch a job using the Job Template, **`Setup Load Balancer`**.

  The **`Devops`** Team should have the **`Admin`** role on the Workflow Job Template.

- Add a Survey to the **`Full Stack Deployment`** Workflow Job Template. Configure the Survey based on the following specification:

Field	Value
PROMPT	What environment are you deploying?
DESCRIPTION	The environment will be displayed at the bottom of the index page

Field	Value
ANSWER VARIABLE NAME	environment_name
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	Development
REQUIRED	Checked

- Create an **email** Notification Template named **Notify on Job Success and Failure**, based on the following specification:

Field	Value
NAME	Notify on Job Success and Failure
DESCRIPTION	Sends an email to notify the status of the Job
ORGANIZATION	Default
TYPE	Email
HOST	localhost
SENDER EMAIL	system@lab.example.com
RECIPIENT LIST	student@lab.example.com
PORT	25

- For the Workflow Job Template, **Full Stack Deployment**, activate the **SUCCESS** and the **FAILURE** Notifications using the Notification Template, **Notify on Job Success and Failure**.

Evaluation

Use the Ansible Tower interface to confirm to your satisfaction that everything is configured as specified.

Important

The next activity in this chapter will instruct you to launch the Workflow Job Template and otherwise test your work in this chapter.

Solution

In this review, you will create a Workflow Job Template which uses a survey and an email notification.

Outcomes
You should be able to:

- Create a Workflow Job Template.
- Add a Survey to an existing Workflow Job Template.
- Create and activate an e-mail Notification Template.

Before you begin
The previous exercises in this comprehensive review must be completed before starting this exercise.

Instructions
Configure your Ansible Tower server with a Workflow Job Template that uses your existing Job Templates, a new Survey, and a new Notification Template, based on the following specification:

- Create a Workflow Job Template in the **`Default`** organization called **`Full Stack Deployment`**. Its **Description** should be **`Deploy the Full Stack`**. Define an Extra Variable **`environment_name`** with the value **`"Development"`**.

  The Workflow Job Template should include the following steps:

  1. Sync the Inventory, **`Dynamic Inventory`**.
  2. Upon success of the previous step, sync the Project, **`My Full-Stack Project`**.
  3. Upon success of the previous step, launch a job using the Job Template, **`Setup Databases`**.
  4. Upon success of the previous step, launch a job using the Job Template, **`Setup Webservers`**.
  5. Upon success of the previous step, launch a job using the Job Template, **`Setup Load Balancer`**.

  The **`Devops`** Team should have the **`Admin`** role on the Workflow Job Template.

- Add a Survey to the **`Full Stack Deployment`** Workflow Job Template. Configure the Survey based on the following specification:

Field	Value
PROMPT	What environment are you deploying?
DESCRIPTION	The environment will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	environment_name
ANSWER TYPE	Text
MINIMUM LENGTH	1

Field	Value
MAXIMUM LENGTH	40
DEFAULT ANSWER	Development
REQUIRED	Checked

- Create an **email** Notification Template named **Notify on Job Success and Failure**, based on the following specification:

Field	Value
NAME	Notify on Job Success and Failure
DESCRIPTION	Sends an email to notify the status of the Job
ORGANIZATION	Default
TYPE	Email
HOST	localhost
SENDER EMAIL	system@lab.example.com
RECIPIENT LIST	student@lab.example.com
PORT	25

- For the Workflow Job Template, **Full Stack Deployment**, activate the **SUCCESS** and the **FAILURE** Notifications using the Notification Template, **Notify on Job Success and Failure**.

Steps

1. To create a Workflow Job Template called **Full Stack Deployment**:

   1.1. Click the **TEMPLATES** quick navigation link.

   1.2. Click the **+ADD** button to add a new Workflow Job Template.

   1.3. From the drop-down list, select **Workflow Job Template**.

   1.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Full Stack Deployment
DESCRIPTION	Deploy the Full Stack
ORGANIZATION	Default
EXTRA VARIABLES	environment_name: "Development"

   1.5. Click **SAVE** to create the new Workflow Job Template.

2. To configure the Workflow for the **Full Stack Deployment** Workflow Job Template:

   2.1. Click **WORKFLOW EDITOR** to open the Workflow Editor.

2.2. Click **START** to add the first action to be performed. This will display, in the right panel, a list of actions to be performed.

2.3. In the right panel, click **INVENTORY SYNC** to display the list of Inventories available.

2.4. Select **Dynamic Group** and click **SELECT**. This will link the **START** node with a blue line (always perform) to the node for the Dynamic Inventory Group, **Dynamic Group**, in the Workflow Editor window.

2.5. Move your mouse over the new node and click on the green **+** button to add an action after the Inventory Sync of **Dynamic Inventory**. This displays a list of actions to be performed in the right panel.

2.6. In the right panel, click **PROJECT SYNC** to display the list of Projects available.

2.7. Select **My Full-Stack Project** and click **SELECT**. In the Workflow editor window, this links the previous node to the node for the Project, **My Full-Stack Project**, with a green line, indicating that this progression will only be performed if the Inventory Sync step is successful.

2.8. Move your mouse over to the new node and click on the green **+** button to add an action after the Project Sync of **My Full-Stack Project**. This displays a list of actions to be performed in the right panel.

2.9. In the right panel, make sure you are in the **JOBS** section and select the Job Template, **Setup Databases**.

2.10. In the **TYPE** section below, select **On Success** and click **SELECT**. In the Workflow Editor window, this links the node for the Project, **My Full-Stack Project**, to the node for the Job Template, **Setup Databases**, with a green line, indicating that this progression will only be performed if the Project Sync step is successful.

2.11. Move your mouse over the new node and click on the green **+** button to add an action after the Job Template, **Setup Databases**.

2.12. In the right panel, make sure you are in the **JOBS** section and select the Job Template, **Setup Webservers**. In the **TYPE** section below, select **On Success** and click **SELECT**.

2.13. Move your mouse over the new node and click on the green **+** button to add an action after the Job Template, **Setup Webservers**.

2.14. In the right panel, make sure you are in the **JOBS** section and select the Job Template, **Setup Load Balancer**. In the **TYPE** section below, select **On Success** and click **SELECT**.

2.15. Click **SAVE** to save the Workflow Job Template.

3. To grant **Devops** Team **Admin** role on the Workflow Job Template:

3.1. Click **PERMISSIONS** to manage the Workflow Job Template's permissions.

3.2. Click the **+ADD** button on the right to add permissions.

3.3. Click **TEAMS** to display the list of available Teams.

3.4. In the first section, check the box next to **Devops** Team. This causes the Team to display in the second section underneath the first one.

3.5. In the second section below, select the **Admin** role from the drop-down list.

3.6. Click **SAVE** to make the role assignment. This redirects you to the list of permissions for the Workflow Job Template, **Full Stack Deployment**, which now shows that all members of the **Devops** Team are assigned the **Admin** role on the Workflow Job Template.

4. To add a Survey to the Workflow Job Template, **Full Stack Deployment**:

4.1. Click the **DETAILS** button to return to the Template details screen.

4.2. Click the **ADD SURVEY** button to add a Survey.

4.3. On the next screen, fill in the details as follows:

Field	Value
PROMPT	What environment are you deploying?
DESCRIPTION	The environment will be displayed at the bottom of the index page
ANSWER VARIABLE NAME	environment_name
ANSWER TYPE	Text
MINIMUM LENGTH	1
MAXIMUM LENGTH	40
DEFAULT ANSWER	Development
REQUIRED	Checked

4.4. Click **+ADD** to add the Survey Prompt to the Survey. This displays a preview of your Survey on the right.

Important

Before saving, make sure that the **ON/OFF** switch is set to **ON** at the top of the Survey editor window.

4.5. Click **SAVE** to add the Survey to the Job Template.

5. To create an **email** Notification Template called **Notify on Job Success and Failure**:

5.1. Open the **SETTINGS** page by clicking the gear icon in the top right.

5.2. Click **NOTIFICATIONS** to manage Notification Templates.

5.3. Click **+ADD** to add a Notification Template.

5.4. On the next screen, fill in the details as follows:

Field	Value
NAME	Notify on Job Success and Failure
DESCRIPTION	Sends an email to notify the status of the Job
ORGANIZATION	Default
TYPE	Email
HOST	localhost
SENDER EMAIL	system@lab.example.com
RECIPIENT LIST	student@lab.example.com
PORT	25

5.5. Leave all the other fields untouched and click **SAVE** to save the Notification Template. You are then redirected to the list of Notification Templates.

6. To activate both the **SUCCESS** and **FAILURE** Notifications for the **`Full Stack Deployment`** Workflow Job Template:

6.1. Click the **TEMPLATES** quick navigation link.

6.2. Click the link for the Workflow Job Template, **`Full Stack Deployment`**.

6.3. Click **NOTIFICATIONS** to manage notifications for the Workflow Job Template, **`Full Stack Deployment`**.

6.4. On the same line as the Notification Template, **`Notify on Job Success and Failure`**, set both **ON/OFF** switches for **SUCCESS** and **FAILURE** to ON.

Evaluation

Use the Ansible Tower interface to confirm to your satisfaction that everything is configured as specified.

Important

The next activity in this chapter will instruct you to launch the Workflow Job Template and otherwise test your work in this chapter.

Lab: Testing the Prepared Environment

In this review, you will test your work in this Comprehensive Review Chapter.

Outcomes
You should be able to:

- Launch the **`Full Stack Deployment`** workflow Job Template.
- Verify that the workflow sent an e-mail notification.
- Verify the end results of your work are correct.

Before you begin
All previous exercise in this comprehensive review should be completed before evaluating your work with this section.

Instructions
Manually test your work on this comprehensive review as follows:

- As user **`robert`** in the Ansible Tower interface, launch the **`Full Stack Deployment`** Workflow Job Template
- As the Linux user **`student`** on **`tower.lab.example.com`**, verify that the Workflow Job Template **`Full Stack Deployment`** sent you an email when it completed.
- Verify that the web servers have been installed and configured on **`servera.lab.example.com`**, **`serverd.lab.example.com`**, **`servere.lab.example.com`**, and **`serverf.lab.example.com`**
- Verify that the load balancer has been installed and configured on **`serverb.lab.example.com`**.

  If it is working correctly, it should direct sequential HTTP requests sent to *http://serverb.lab.example.com/* to different web servers. Every refresh of this page redirects the connection to a different web server.

  If the configuration is correct, it should redirect connections to the following web servers: **`servera`**, **`serverd`**, **`servere`** and **`serverf`**.

Evaluation
As the **`student`** user on **`workstation`**, run the **`lab review-review`** script with the **`grade`** argument, to confirm success on this exercises. Correct any reported failures and rerun the script until successful.

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

Solution

In this review, you will test your work in this Comprehensive Review Chapter.

Outcomes
You should be able to:

- Launch the **`Full Stack Deployment`** workflow Job Template.
- Verify that the workflow sent an e-mail notification.
- Verify the end results of your work are correct.

Before you begin
All previous exercise in this comprehensive review should be completed before evaluating your work with this section.

Instructions
Manually test your work on this comprehensive review as follows:

- As user **`robert`** in the Ansible Tower interface, launch the **`Full Stack Deployment`** Workflow Job Template
- As the Linux user **`student`** on **`tower.lab.example.com`**, verify that the Workflow Job Template **`Full Stack Deployment`** sent you an email when it completed.
- Verify that the web servers have been installed and configured on **`servera.lab.example.com`**, **`serverd.lab.example.com`**, **`servere.lab.example.com`**, and **`serverf.lab.example.com`**
- Verify that the load balancer has been installed and configured on **`serverb.lab.example.com`**.

  If it is working correctly, it should direct sequential HTTP requests sent to *http://serverb.lab.example.com/* to different web servers. Every refresh of this page redirects the connection to a different web server.

  If the configuration is correct, it should redirect connections to the following web servers: **`servera`**, **`serverd`**, **`servere`** and **`serverf`**.

Steps
1. To Launch the **`Full Stack Deployment`** Workflow as user **`robert`**:
   1.1. Log into the Ansible Tower web interface as user **`robert`** with **`redhat123`** as the password.
   1.2. Click the **TEMPLATES** section.
   1.3. On the same line as the Workflow Job Template, **`Full Stack Deployment`**, click the rocket icon on the right to launch the Workflow. This opens the Survey you just created and ask for your input.
   1.4. Type in **`Development`** in the text field and click **LAUNCH** to launch the Workflow. This redirects you to a detailed status page of the running Workflow.

1.5. Observe the running Jobs of the Workflow. You can click on the **DETAILS** link of a running or completed Job to see a more detailed live output of the Job.

2. To verify that the Workflow **`Full Stack Deployment`** triggered an email notification after completion:

2.1. Open a terminal and connect to the **`tower`** VM.

```
[student@workstation ~]$ ssh tower
Last login: Thu Apr 20 11:33:22 2017 from workstation.lab.example.com
[student@tower ~]$
```

2.2. View incoming messages to the local mailbox file of the **`student`** user using the **`tail`** command. You should see this type of successful Workflow Job completion notification email arrive:

```
[student@tower ~]$ tail -f /var/mail/student
[...output omitted...]
From system@tower.lab.example.com  Thu Apr 20 18:06:10 2017
Return-Path: <system@tower.lab.example.com>
X-Original-To: student@tower.lab.example.com
Delivered-To: student@tower.lab.example.com
Received: from tower.lab.example.com (localhost [IPv6:::1])
        by tower.lab.example.com (Postfix) with ESMTP id AAB30401FB3
        for <student@tower.lab.example.com>; Thu, 20 Apr 2017 18:06:10 -0400
 (EDT)
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit
Subject: Workflow Job #50 'Full Stack Deployment' succeeded on Ansible Tower:
 https://tower.lab.example.com/#/workflows/50
From: system@tower.lab.example.com
To: student@tower.lab.example.com
Date: Thu, 20 Apr 2017 22:06:10 -0000
Message-ID: <20170420220610.2290.67752@tower.lab.example.com>
[...output omitted...]
```

3. At this point, the web servers and the balancer are installed, configured, and functioning. Verify that the results are correct:

3.1. Open a web browser and go to **`http://servera.lab.example.com`**, **`http://serverd.lab.example.com`**, **`http://servere.lab.example.com`**, and **`http://serverf.lab.example.com`** in separate tabs. You should see this line at the bottom of each page:

```
Environment: Development
```

3.2. Open a web browser and go to **`http://serverb.lab.example.com`**. Every time you refresh the page, the load balancer redirects your request to one of the web servers verified in the previous step.

Evaluation

As the **student** user on **workstation**, run the **lab review-review** script with the **grade** argument, to confirm success on this exercises. Correct any reported failures and rerun the script until successful.

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