Introduction

©2018 Pearson, Inc.

About You

- Expected to have knowledge and experience about Ansible Fundamentals
- May have attended Ansible EX294
- Do NOT expect an Ansible course for newbies

©2018 Pearson, Inc.

Relation to My Other Ansible Courses

- Live courses
  - Ansible in 4 Hours is an introduction level course
  - Ansible RHCE EX294 is a deep-going 16 hour course that prepares for the RHCE EX294 exam
  - Ansible Tower is an introduction to managing Ansible with Tower
- This course focus is on features not discussed in Ansible in 4 Hours and RHCE EX294
- Recorded courses
  - Ansible Fundamentals is obsolete
  - Ansible Hands-on is the introduction level Ansible course
  - Ansible Certification is obsolete
  - A new advanced Ansible course is upcoming

©2018 Pearson, Inc.

Agenda (subject to Change)

- Getting Started Lab
- Exploring Task Control
- Optimizing Task Control
- Optimizing Ansible for Speed
- Using Filters and Plugins
- Managing Containers
- Managing Windows
- Managing Network Devices
- Managing Cloud
- Ansible Tower

©2018 Pearson, Inc.

Poll Question 1

- Which of the following statements best applies to you
  a) I am new to Ansible
  b) I have just learned the basics and now want to get more in-depth knowledge about Ansible
  c) I have been working with Ansible for some time already
  d) I consider myself very experienced with Ansible

©2018 Pearson, Inc.

Poll Question 2

- Did you attend any of my Ansible courses?
  a) Ansible in 4 Hours
  b) RHCE EX294
  c) Ansible Tower
  d) no

©2018 Pearson, Inc.

Poll Question 3

Which of the following topics are most important for you (choose 3)?

- Reviewing Ansible basics (overlap with Ansible in 4 Hours)
- Exploring Task Control Examples (overlap with RHCE EX294)
- Exploring Advanced Ansible Options
- Working with Ansible Tower
- Managing Network Devices with Ansible
- Managing Windows with Ansible
- Managing Cloud with Ansible
- Managing Docker with Ansible

©2018 Pearson, Inc.

About this Course

©2018 Pearson, Inc.

Course Files

- The demo files used in this course are available on github
- Use **git clone https://github.com/sandervanvugt/ansible-advanced** to download

©2018 Pearson, Inc.

Lab Environment

©2018 Pearson, Inc.

Lab Environment

- ansible-control: CentOS system that is set up for managing all nodes but not itself
- ansible1: CentOS system that has been set up for management through Ansible
- ansible2: CentOS system that has been set up for management through Ansible
- ansible3: CentOS 7 system that has been set up for management through Ansible
- windows: basic installation of Windows Server 2016 standard, not set up for management yet

©2018 Pearson, Inc.

Getting Started Lab

©2018 Pearson, Inc.

Lab - part 1

1. Create an Ansible configuration that sets up hosts Ansible1 and Ansible2 for automatic installation. Create custom facts for both hosts and use variable inclusion to realize this. To configure ansible1, use a host group with the name "file", to configure ansible2, use a host group with the name "lamp"
2. Create a file with the name custom.fact that defines custom facts. In this file, define two sections. The section package contains the following:
   smb_package = smb
   ftp_package=ftp
   db_package=mariadb
   web_package=http
   The section service contains service variables for the packages mentioned above. Use the name smb_service etc. and set the variable to the appropriate name of the service
3. Create a playbook with the name copy_facts.yml that copies these facts to all managed hosts. Define a variable with the name "remote_dir" and a variable with the name "fact_file" and use these. Use the file and copy modules.
4. Run the playbook and verify it worked

©2018 Pearson, Inc.

Lab - part 2

5. Create a variable inclusion file with the name ./vars/allvars.yml and set the following variables
   web_root: /var/www/html
   ftp_root: /var/ftp
6. Create a tasks directory in the project folder. In this directory, create two YAML files, one that installs, starts, and enables the LAMP services; and one that installs, starts, and enables the file services
7. Create the main playbook that will set up the lamp servers and the file servers with the packages they need, using inclusions to the previously-defined tasks file. Also, ensure that it opens the firewalld firewall to allow access to these servers. Finally, the web service should be provided with an index.html file that shows "managed by Ansible" on the first line
8. Run the playbook
9. Use ad hoc commands to verify the services have been started

©2018 Pearson, Inc.

Lab Solution

1. Create the inventory file (see lab-inventory)
2. see custom.fact
3. see lab-copy-facts.yml
4. ansible-playbook -i lab-inventory lab-copy-facts.yml; ansible -i lab-inventory all -m setup -a 'filter=ansible_local*'
5. see lab-vars/allvars.yml
6. see lab-tasks/lamp.yml and file.yml
7. see lab-playbook.yml
8. ansible-playbook lab-playbook.yml
9. ansible lamp -a 'systemctl status mariadb'; ansible file -a 'systemctl status vsftpd'

©2018 Pearson, Inc.

Files and Directories Best Practices

©2018 Pearson, Inc.

Organizing Ansible Contents

- Even simple projects should have their own directory structure
- Within that directory structure, you'll have an ansible.cfg, inventory as well as playbooks
- If the project grows bigger, variable files as well as includes may be used
- Roles can be used to standardize and easily re-use specific parts of Ansible
- Consider a role a complete project dedicated to a specific task that is going to be included from the main playbook

©2018 Pearson, Inc.

Directory Layout Best Practices

- Ansible Documentation describes best practices (https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html)
- Some highlights:
  - On top in the directory, use site.yml as the master playbook
  - From site.yml, call specific playbooks, such as webservers.yml etc.
  - Use different inventory files to differentiate between production and staging
  - Use group_vars/ and host_vars/ to set host related variables
  - Use roles to standardize common tasks

©2018 Pearson, Inc.

Managing Task Control

©2018 Pearson, Inc.

Limitations to Handlers

- If a task triggers a handler, but a task after that task fails on the current host, task execution will be stopped on that host and the handler will never run
- Use **force_handlers** to overwrite that behavior
  - Use the --force-handlers command line option
  - Set force_handlers = True as a default in ansible.cfg
  - Use force_handlers: True in a play
- Use **ignore_errors** to force the playbook to continue running

©2018 Pearson, Inc.

Combining Loops and Conditionals

- Ansible facts may present a dictionary with multiple values
- In that case, you can iterate through each value until a specific condition is met and use the result in a when statement

```
...
- name: install vsftpd if sufficient space on /var/ftp
  package:
    name: vsftpd
    state: latest
  with_items: "{{ ansible_mounts }}"
  when: item.mount == "/var/ftp" and item.size.available > 100000000
```


©2018 Pearson, Inc.

Understanding Parallelism

- Running tasks in parallel will make Ansible faster
- Ansible can run tasks in parallel on all hosts
- By default, tasks can run on 5 hosts at once
  - Set **forks = nn** in /etc/ansible/ansible.cfg to increase this number
  - Alternatively, use the **--forks** option with the **ansible-playbook** or **ansible** command
- Use the **serial** keyword in a playbook to reduce the number of parallel tasks to a value that is lower than what is specified with the **forks** option

©2018 Pearson, Inc.

Understanding Asynchronous Tasks

- Normally, Ansible waits for completion of tasks before starting the next task
- Use the **async** keyword in a task to run the task in the background
  - **async: 3600** tells Ansible to give the task an hour to complete, note that this will be the maximum amount of time permitted for the job to run
  - **poll: 10** indicates that Ansible will poll every 10 seconds to see if the command has completed
- Using **async** allows the next task to be started so it will make playbooks more efficient
  - Recommended for backup jobs, yum updates, large file downloads, etc.

©2018 Pearson, Inc.

Using **wait_for**

- The **wait_for** module can be used in a task to check if a certain condition was met
- Using this module may be useful to verify successful restart of servers, etc.
- Use **poll: 0** in a task to tell Ansible not to wait for completion of this task, but to move on to the next task. Add ignore_errors as well, to prevent an error condition arising and have this task fail

©2018 Pearson, Inc.

Optimizing Task Execution

©2018 Pearson, Inc.

Managing Privilege Escalation

- Common parameters are used for privilege escalation
  - become
  - become_user
  - become_method
  - become_flags
- These parameters can be used at different levels
  - In ansible.cfg
  - Using command line options with **ansible** ans **ansible-playbook**
  - In play headers
  - In tasks
  - In blocks
  - Using connection variables

©2018 Pearson, Inc.

Privilege Escalation in Variables

- Use a **vars:** section in your playbook to work with privilege escalation
- Alternatively, include them in host or group variables

©2018 Pearson, Inc.

Understanding Escalation Priorities

- Privilege escalation settings are applied in the following order
  - through variables
  - in tasks
  - in blocks
  - in plays
  - on command line
  - in ansible.cfg

©2018 Pearson, Inc.

Privilege Escalation Guidelines

- Don't use privilege escalation if you don't need it
- Always keep your playbook as simple as possible

©2018 Pearson, Inc.

Controling Host Execution Order

- Ansible 2.4 and later runs plays on hosts in the order they occur in inventory
- Use the **order** directive in the play header to change that
  - **order: [reverse_]sorted** processes hosts in alfabethical order
  - **order: shuffle** applies random order
  - **order: [reverse_]inventory** applies inventory order
- Note that because Ansible runs tasks in parallel, the output observed might not reflect the order specified

©2018 Pearson, Inc.

Using Tags

- A tag is a label that can be set to an item (play, block, task, role)
- Using tags allows you to run specific tasks only
  - Add the **--tags** option to run specific tasks, or **--skip-tags** to skip specific tasks
  - Use **ansible-playbook --list-tags** to show a list of all tags in a playbook

©2018 Pearson, Inc.

Optimizing Ansible for Speed

©2018 Pearson, Inc.

Speed Optimization Solutions

- Keep control node close
- Disable fact gathering
- Increase parallism
- Avoid loops with Package Manager Modules
- Efficiently copy files
- Use templates
- Optimize SSH connections
- Profiling Playbooks

©2018 Pearson, Inc.

Keeping Control Node Close

- Ansible generates a lot of network traffic. Keeping the control node physically close helps
- Also, make sure to install updates and use the latest Ansible software

©2018 Pearson, Inc.

Fact Gathering

- Use fact gathering only when you need it
- Without fact gathering, use the magic variables **inventory_hostname** and **inventory_hostname_short** instead of **ansible_hostname**
- Use the **setup** module where fact gathering is necessary

©2018 Pearson, Inc.

Parallelism

- By default, **forks** is set to 5, which means that each task is executed on 5 hosts in parallel before moving forward to the next host
- Consider setting **forks** to 100 for better performance

©2018 Pearson, Inc.

Avoid Loops

- Loops are great for keys that don't take a multi-value argument
- Package management modules do take multi-value arguments, which is much better for performance

©2018 Pearson, Inc.

Effectively Copy Files

- The **copy** module can take a long time when dealing with large amounts of files
- Consider using the **synchronize** module, as it works more efficiently

©2018 Pearson, Inc.

Using Templates

- When changing file contents, different options exist
- Use **lineinfile** for just a few simple changes
- Use **template** when multiple changes need to be applied

©2018 Pearson, Inc.

Optimizing SSH

- Ansible uses SSH to manage hosts
- Establishing a new SSH session takes a lot of time
- To make SSH connections more efficient, three SSH features can be used
  - **ControlMaster** allows multiple simultaneous SSH sessions with a remote host using one network connection
  - **ControlPersist** keeps a connection open for xx seconds
  - **pipelining** allows more commands to use a simultaneous SSH connection
- Use an **[ssh_connection]** section in the Ansible config file to specify how to use these options

```
[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=120s
pipelining = true
```


©2018 Pearson, Inc.

Using Callback plug-ins

- Callback plugins extend Ansible by modifying how it respons to events
- Use **ansible-doc -t callback -l** for a complete list
- Use **ansible-doc -t callback <plugin>** for more information on how to use a specific plugin
- Specific plugins can be enabled in ansible.cfg, using **callback_whitelist**
  - **callback_whitelist=timer, profile_tasks, cgroup_perf_recap**

©2018 Pearson, Inc.

Understanding Specific Callback Plugins

- **cgroup_perf_recap** shows performance statistics after running a playbook
  - This requires Cgroups support and needs **sudo cgcreate -a user:user -t user:user -g cpuacct,memory,pids:ansible_profile**
  - Does NOT work in RHEL 8.2!
- **timer** shows the duration of playbook execution
- **profile_tasks** shows start time of each tasks, as well as how much time spent
- **profile_roles** does the same for roles

©2018 Pearson, Inc.

Using Filters and Plugins

©2018 Pearson, Inc.

Understanding Filters

- Filters are used to modify or process the value from a variable before using it in playbook or template
- Some filters are provided by the Jinja2 language
- Other filters are included with Ansible as plug-ins
- Filters are common when dealing with different variable types in Ansible

©2018 Pearson, Inc.

Understanding Ansible Variable Types

- String: sequence of characters - the default variable type in Ansible
- Numbers: numeric value, treated as Integer or Float. When placing a number in quotes it is treated as a string
- Booleans: true/false values
- Dates: calendar dates
- Null: undifined variable type
- List or Arrays: a sorted collection of values
- Dictionary or Hash: a collection of key/value pairs

©2018 Pearson, Inc.

Processing Variables with Filters

- Filters allow variables to be processed before using them in Ansible
- Filters do not change the value of a variable, they just change how the variable is being used
- Use **{{ varname | filter }}** to identify the filter to be used

©2018 Pearson, Inc.

Exploring Common Filters

- **mandatory** fails a play if a variable does not have a value: **{{ my_var | mandatory }}**
- **default** will set a variable without value to a default value. Add the optional True to also set the variable if it has an empty string or a boolean false as its value: **{{ my_var | default(myvalue, True) }}**
- **int** will convert a variable to an integer
- **float** converts a variable to a float
- **+ - / *** perform basic calculations: **{{ (vgsize | int) - 1 }}**
- **random** extracts a random element from a list: **{{ [a, b, c] random }}**
- **sort** will sort a list: **{{ [4, 8, 2, 6] | sort }}**

©2018 Pearson, Inc.

Managing Containers

©2018 Pearson, Inc.

Managing Containers with Ansible

- Ansible offers different modules to work with Docker containers, use **ansible-doc -l | grep docker** for an overview
- Notice that RHEL 8 and related don't support Docker anymore, but use Podman instead

©2018 Pearson, Inc.

Managing Containers: High-level Overview

- Add repository to install Docker
- Install and start the Docker daemon
- Install Python pip module to support Docker
- Create a container
- Generate a list of running containers

©2018 Pearson, Inc.

Managing Network Devices

©2018 Pearson, Inc.

Managing Network Devices

- To manage network devices, you need modules
- Modules exist for many network devices, but not all, see https://www.ansible.com/integrations/networks for an overview
- Network devices need to be identified in inventory
- Using dynamic inventory is suggested
- Connecting to network devices works on SSH in most cases, sometimes HTTPS is an option
- For more information: see “Network Automation” in docs.ansible.com

©2018 Pearson, Inc.

How Network Automation is Different

- Modules are executed on the control node
- If a backup of configuration files is defined, this backup is also stored on the control node
- Multiple communication protocols are available:
  - **ansible_connection: local** -> provider specific protocol
  - **ansible_connection: network_cli** -> CLI over SSH
  - **ansible_connextion: netconf** -> XML over SSH
  - **ansible_connection: httpapi** -> API over HTTP/HTTPS

©2018 Pearson, Inc.

Specifying Network Connection Params

- To connect to a network device, inventory variables are used
- Inventory variables should not be set in inventory anymore, but in **group_vars/groupname**:

```
ansible_connection: network_cli
ansible_network_os: ios
ansible_become: yes
ansible_become_method: enable
```

- Read the documentation of vendor specific modules to figure out which parameters to use

©2018 Pearson, Inc.

Running Ansible Commands on Network Devices

- Ad-hoc: **ansible all -i routers -c network_cli -u my_user -k -m ios_facts -a "gather_subset=all" -e ansible_network_os=ios**
  - The **-c** option specifies how to connect to a device (see previous slide), other options are more or less obvious
- Playbook: see network_facts.yaml. Run it using **ansible-playbook -i routers -u ansible -k -e ansible_network_os=ios network_facts.yaml**
- And see **network_config.yaml** for an example on how to change network parameters

©2018 Pearson, Inc.

Understanding Network Device Modules

- Modules to manage network devices are developed in a consistent way:
  - Name starts with the platform
  - Next, there is the specific function, like **facts**, **ping** and many more
  - The number of modules for a platform heavily depends on support by the platform vendor: many for Cisco IOS, not as many for other platforms

©2018 Pearson, Inc.

Building Inventory

- Specific hardware types should be included in an inventory group
- For each inventory group, specific connection parameters can be specified
- Using inventory variables for network devices is common, but group_vars/ is also supported

```
[routers]
router01 ansible_host=10.1.0.101 ansible_network_os=ios ansible_user=cisco_user
router02 ansible_host=10.1.0.102 ansible_network_os=ios ansible_user=cisco_user
```


©2018 Pearson, Inc.

Taking the Next Steps

- For managing network devices, different modules are available for specific platforms
- To automate setting up common tasks, multiple roles are provided as well. See **https://galaxy.ansible.com** for an overview of roles

©2018 Pearson, Inc.

Managing Windows

©2018 Pearson, Inc.

Setting up a Windows Host

- Supported Windows versions
  - Windows 7, 8.1, and 10
  - Windows Server 2008, 2012, and 2016
- Required Software
  - Powershell 3.0 or later
  - .NET 4.0 or later
- A WinRM Listener should be created and activated: https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html
- Note that some modules may have different requirements

©2018 Pearson, Inc.

Managing Windows - 1

- Install Windows 2016 server standard, ensure there is a Windows user with admin privileges (will be created automatically)
- login as Admin
- Open powershell: **winrm quickconfig**
- Set up WinRM, using the script that is provided on docs.ansible.com/ansible/2.5/user_guide/windows_setup.html in WinRM Setup
- In Windows control panel, create Ansible user, set password and make this user an Administrator

©2018 Pearson, Inc.

Managing Windows - 2

- On Ansible control, create windows project directory with an ansible.cfg and an inventory
- Set up /etc/hosts for host name resolution to the windows box - notice that Windows firewall disallows ping incoming
- On ansible control: **sudo pip3 install pywinrm**
  - If Python3 is used, **export PYTHONPATH=$PYTHONPATH:/usr/local/bin/python3.6/site-packages/winrm** (and put this in .bashrc)
- On ansible control: **ansible win -i inventory -m win_ping**
- **ansible-playbook playbook.yml**
- Verify on Windows by showing the users in the control panel

©2018 Pearson, Inc.

Managing AWS

©2018 Pearson, Inc.

Understanding the Process

- Cloud instances need to be discovered, using dynamic inventory
- Specific Ansible modules exist for working with instances in different public cloud environments
  - Hundreds of modules exist that relate to the different cloud platforms

©2018 Pearson, Inc.

Step 1: Defining Access Credentials

- Log in to AWS console, and in your account setting generate an access key

aws | Services | Resource Groups | A F J van Vugt | Global | Support

Search IAM

Dashboard
Groups
Users
Roles
Policies
Identity providers
Account settings
Credential report

Encryption keys

Your Security Credentials

Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) users, use the IAM Console.

To learn more about the types of AWS credentials and how they're used, see AWS Security Credentials in AWS General Reference.

+ Password
+ Multi-factor authentication (MFA)
− Access keys (access key ID and secret access key)

You use access keys to sign programmatic requests to AWS services. To learn how to sign requests using your access keys, see the signing documentation. For your protection, store your access keys securely and do not share them. In addition, AWS recommends that you rotate your access keys every 90 days.

Note: You can have a maximum of two access keys (active or inactive) at a time.

Created	Deleted	Access Key ID	Last Used	Last Used Region	Last Used Service	Status	Actions
Aug 22nd 2018		[redacted]	N/A	N/A	N/A	Active	Make Inactive | Delete

Create New Access Key

Important Change - Managing Your AWS Secret Access Keys

As described in a previous announcement, you cannot retrieve the existing secret access keys for your AWS root account, though you can still create a new root access key at any time. As a best practice, we recommend creating an IAM user that has access keys rather than relying on root access keys.

+ CloudFront key pairs
+ X.509 certificate
+ Account identifiers

©2018 Pearson, Inc.

Step 1: Defining Access Credentials - continued

- Specify the AWS access keys as variables in the .bash_profile file in your home directory

```
export AWS_ACCESS_KEY_ID=123
export AWS_SECRET_ACCESS_KEY=abc
```

- At this point you can start using dynamic inventory as well as Ansible modules

©2018 Pearson, Inc.

Step 2: Installing Python Boto

- To access AWS, you need to install the Python Boto library
  - **pip install boto**
- Next, verify connectivity to EC2 using a python shell

```
[ansible@ansiblecontrol ~]$ python
Python 2.7.5 (default, Jul 13 2018, 13:06:57)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import boto.ec2
>>> conn = boto.ec2.connect_to_region("us-west-2")
>>> statuses = conn.get_all_instance_status()
>>> statuses
[InstanceStatus:i-07256f36804be4229]
>>>
```


©2018 Pearson, Inc.

Step 3: Using Dynamic Inventory

- At this point, you can download the Inventory files
  - **wget https://raw/githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.py**
  - **wget https://raw/githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini**
- Put them in the directory "inventory" and add the execute permission to the ec2.py file
- Run the inventory, using the command **./ec2.py --list**. Output will be presented in Json format

©2018 Pearson, Inc.

Understanding Inventory Caching

- Gathering Inventory information takes time, which is why an inventory cache is created in ~/.ansible/tmp
- To force a cache refresh, use **./ec2.py --refresh-cache**
- Make sure to do this after adding or destroying instances

©2018 Pearson, Inc.

Managing Instances in EC2

- EC2 instances come with a default user name, and as host keys are not known in advance for new instances, it's useful to add 2 custom parameters to ansible.cfg to manage Instances in EC2
  - **remote_user = yourinstanceuser**
  - **host_key_checking = False**
- Next, find image (AMI) ID's available for your region
  - **aws ec2 describe-images --region us-west-2**
- Then, use the ec2 module to deploy instances

©2018 Pearson, Inc.

Demo

- Sample playbook: ec2/ubuntu.yaml
  - image: the AMI image
    - Finding images: login to EC2 Management Console, select Images >AMIs, from the drop down list, select Public images, use a filter to filter on Ubuntu. Select source: aws-marketplace to restrict the number of results.
- Alternatively: from the aws-client software, use **aws ec2 describe-images --region us-west-2 --filters Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64***
    - Note that some software requires a subscription. if that is the case, a URL will be displayed. Follow the link and subscribe before you continue
  - region: your region
  - instance_type: look for free tier!
  - key_name: create the key first in EC2 and use it
  - group: the security group in EC2
  - instance_tags: optional tags to be used for grouping in EC2

©2018 Pearson, Inc.

Managing EC2 Instances

- After setting up Ansible EC2 connections, many modules are available to do many things
  - List instances: use **ec2_instance_facts**
  - Terminate instances: use **ansible localhost -m ec2 -a 'instance_id=i-1234 state=absent'**
  - Manage keys: **ec2_key**
  - Manage Security Groups: **ec2_group**
  - Find the latest ami: **ec2_ami_find**

©2018 Pearson, Inc.

Working with Tower

©2018 Pearson, Inc.

Understanding Tower

- Tower is used to bring Ansible to the enterprise
- Web-based management
- API access
- Users and credentials management, connecting to external sources
- Scheduled jobs
- Multi-playbook workflows
- Audit trails of all that has happened
- Based on the AWX open source project

©2018 Pearson, Inc.

Installing Tower

- Use a dedicated VM
  - 4GB RAM (more if possible)
  - 2 CPU's
  - At least 10GB in /var
  - Red Hat / CentOS / Ubuntu are supported
- On CentOS: add the EPEL repository
- Add the local host name to /etc/resolv.conf
- Download and extract evaluation version
- Set passwords in inventory file
- Run **./setup.sh**
- Start a browser, connect to https://tower.example.com and log in with admin credentials

©2018 Pearson, Inc.

Preparing the Environment

- Request a demo license and import it
- Check organizations, you'll have one default organization containing users, teams, projects and inventories
  - Self-support license doesn't allow creation of additional organizations
- From here, work with the following items
  - Credentials: how to connect to managed hosts
  - Inventory: which hosts to connect to
  - Project: where to get the playbooks from
  - Job Template: what to run
  - Job: an attempted run of a playbook

©2018 Pearson, Inc.

Understanding the Process

- Create credentials: this is the alternative to ansible.cfg, where you specify as which user to connect and how to escalate permissions. Copy the ansible user private key here and make sure the user specified here exists on all managed hosts
- Create an inventory: this is where you add all hosts that should be managed
- Create a project: this is where you connect to the source of your playbooks. (GitHub is preferred)
- Create a template: this is where you specify the job that you want to run
- Run the job

©2018 Pearson, Inc.

Understanding Projects

- A project is a collection of playbooks
- Projects can connect to Github, or read playbooks from a local directory on the Ansible tower server
- Review how the demo project connects to a Github repository
- Before you can use a project, it must be synced. Click Get latest SCM revision to do that
  - After a successful sync, you'll see the last updated timestamp
  - New projects will sync automatically after creation

©2018 Pearson, Inc.

Creating a Job Template

- A job template connects the playbook from a project with the settings required to launch it, so the job template runs the playbook
- Check the demo template, where you can see the Yaml file that comes from the project, as well as the demo credential that allows to connect to managed servers
- Click the Rocket icon to launch the job in this template
- Observe in the Jobs view if it worked out well

©2018 Pearson, Inc.

Where to go Next

©2018 Pearson, Inc.

Further Learning

- Live courses
  - Ansible in 4 Hours is an introduction level course
  - Ansible RHCE EX294 is a deep-going 16 hour course that prepares for the RHCE EX294 exam
  - Ansible Tower is an introduction to managing Ansible with Tower
- Recorded courses
  - Ansible Fundamentals is obsolete
  - Ansible Hands-on is the introduction level Ansible course
  - Ansible Certification is obsolete
  - A new advanced Ansible course is upcoming

©2018 Pearson, Inc.