RHCE (EX294) Crash Course

Sander van Vugt

Pearson

Course Agenda

- Day 1:
  - Fundamentals quick overview
    - What is ansible
    - Setup
    - Ad-hoc commands and playbooks
    - Modules and Content Collections
  - Variables
- Day 2:
  - More variables
  - Task control
  - File Management

Course Agenda

- Day 3
    - Roles
    - Advanced Usage scenarios

June 2023 notes

- The contents of this class slide deck is based on the upcoming "RHCE (EX294) RHEL 9 Complete Video Course"
- This course is expected to be available late August 2023
- To provide students with all course contents, this slide deck contains almost all slides from the recorded course
- Notice that not all topics will be covered in this live training

Pearson

Poll Question 1

- How would you rate your experience with Ansible (choose one)
  - None
  - Beginner
  - Intermediate
  - Advanced

Pearson

Poll Question 2

- Do you have RHCSA level knowledge / experience
  - yes
  - no

Pearson

Poll Question 3

- Where are you from?
  - Africa
  - India
  - Asia (other)
  - Europe
  - USA/Canada
  - South and Central America
  - Australia / Pacific

Pearson

Lesson 1: Understanding the Ansible Product Offering

1.1 Understanding Ansible

What is Ansible

- Ansible is software for configuration management
- It uses modules in YAML files to describe the current state of managed systems
- Using Ansible commands, the desired state is compared to the current state of managed systems
- If a difference occurs, Ansible will update the current state with the desired state

How it Works

- Ansible is designed to be simple
- It is developed on top of Python, and uses manifest files written in YAML
- Ansible uses Secure Shell (SSH) to reach out to the managed systems and apply the desired state
- It uses no agents
- You can use Ansible to manage a wide range of different platforms

Lesson 1: Understanding the Ansible Product Offering

1.2 Understanding Community Ansible Solutions

Understanding Community Ansible

- Ansible core is open-source software, provided through https://github.com/ansible/ansible
- Ansible core provides essential modules only
- Additional modules are provided through collections on https://galaxy.ansible.com
- Ansible AWX is open-source software, provided through https://github.com/ansible/awx that offers the following:
    - A web-based user interface
    - A REST API
    - A task engine
- **ansible-navigator** is a text-based user interface for Ansible

Pearson

Lesson 1: Understanding the Ansible Product Offering

1.3 Understanding the Red Hat Product Offering

Understanding Red Hat Ansible

- The **ansible-core** package is offered as an independent package in RHEL 9, and using it does not require Ansible Automation Platform (AAP)
- It offers limited support
- AAP bundles different open source components
    - **ansible-core**
    - Ansible Automation Controller, which is based on AWX
    - Supported Ansible Content Collections through Automation Hub, which is available at https://console.redhat.com
    - Automation content navigator (**ansible-navigator**) which is introduced as an alternative for **ansible-playbook** and other older commands
    - Automation Execution Environments, which contain runtime environments with specific modules and other content, used by **ansible-navigator**

Automation Execution Environments

- An Automation Execution Environment is a container image that contains Ansible Core, Ansible Content Collections and all dependencies needed to run a playbook
- Execution Environments were introduced to offer a complete solution where it is no longer required to install dependencies separately
- While using **ansible-navigator**, an execution environment is used to start the playbook
- The **ansible-builder** tool can be used to build custom execution environments
- To use execution environments, a container engine must be available
- To access content, you'll have to log into the appropriate container registries: **podman login registry.redhat.io**

ansible-navigator and Execution Environments

- When using **ansible-navigator**, a default execution environment is provided
- Additional execution environments can be loaded from navigator
- Within an execution environment, specific content collections can be provided

Lesson 2: Installing Ansible

2.1 Understanding What to Install

What to Prepare

- Install 3 VM's, using RHEL (recommended) or CentOS Stream
  - control.example.local (Server with GUI)
  - ansible1.example.local (minimal install)
  - ansible2.example.local (minimal install)
- The latest version of RHEL can be obtained for free from https://developers.redhat.com
- The machines require Internet access
- Clone git repository: **git clone https://github.com/sandervanvugt/rhce9**

Understanding the Required Lab

- The *control node* is where you install Ansible
- For RHCE 9, you'll install on top of RHEL 9 (preferred) or CentOS Stream
- Make sure a fixed IP address is set up, as well as a hostname
- The *managed nodes* are managed by Ansible
- Managed nodes can be anything: servers running RHEL, but also other Linux distributions, Windows, Network Devices, and much more
- For RHCE 9, you'll need 2 managed nodes pre-installed with RHEL 9 or CentOS Stream
- Ensure host name lookup is configured, /etc/hosts is good enough

Lesson 2: Installing Ansible

2.2 Understanding Required Components

Requirements for Using Ansible

To use Ansible, the following is needed

- Host name resolution to address managed hosts by name
- Python on all nodes
- SSH running on the managed servers
- A dedicated user account with sudo privileges
- Optional: SSH keys to make logging in easier
- An inventory to identify managed nodes on the control node
- Optional: an ansible.cfg to specify default parameters

Lesson 2: Installing Ansible

2.3 Installing Ansible on the Control Node

Demo: Setting up the Control Node

1. On the control node: use **sudo subscription-manager repos --list** to verify the name of the latest available repository
2. Use **sudo subscription-manager repos --enable=rhel-9-for-x86_64-appstream.rpms** to enable the repository
3. **sudo subscription-manager repos --enable ansible-automation-platform-2.2-for-rhel-9-x86_64-rpms**
4. Use **sudo dnf install ansible-core** to install the Ansible software
5. Use **sudo dnf install ansible-navigator** to install Ansible Navigator
6. **ansible --version to verify**

Pearson

Lesson 2: Installing Ansible

2.4 Using Ansible to Set up Managed Nodes

Demo: Setting up Managed Nodes

- **cat >> inventory <<EOF**
  ansible1
  ansible2
  EOF
- **ansible -i inventory all -u student -k -b -K -m user -a "name=ansible"**
- **ansible -i inventory all -u student -k -b -K -m shell -a "echo password | passwd --stdin ansible"**
- **ssh-keygen** (for user student, as well as for user ansible)
- **for i in ansible1 ansible2; do ssh-copy-id $i; done** (user student & ansible)
- **ansible -i inventory all -u ansible -b -m command -a "ls -l /root" -K**
- **ansible -i inventory all -u student -k -b -K -m shell -a 'echo "ansible ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/ansible'**

Pearson

Lesson 2: Installing Ansible

2.5 Configuring Inventory

Managing Static Inventory

- In a minimal form, a static inventory is a list of host names and/or IP addresses that can be managed by Ansible
- Hosts can be grouped in inventory to make it easy to address multiple hosts at once
- A host can be a member of multiple groups
- Nested groups are also available
- It is common to work with project-based inventory files
- Variables can be set from the inventory file - but this is deprecated practice
- Ranges can be used:
  - server[1:20] matches server1 up to server20
  - 192.168.[4:5].[0:255] matches two full class C subnets

Pearson

Inventory File Locations

- /etc/ansible/hosts is the default inventory
- Alternative inventory location can be specified through the ansible.cfg configuration file
- Or use the **-i inventory** option to specify the location of the inventory file to use
- It is common practice to put the inventory file in the current project directory

Static Inventory Example

```
[webservers]
web[1:9].example.com
web12.example.com

[fileservers]
file1.example.com
file2.example.com

[servers:children]
webservers
fileservers
```

Testing Inventory

- **ansible -i inventory all --list-hosts**
- **ansible -i inventory file --list-hosts**
- **ansible-navigator inventory -i inventory -m stdout --list**

Pearson

Lesson 2: Installing Ansible

2.6 Creating ansible.cfg

Understanding ansible.cfg

- Different connection settings can be specified in ansible.cfg
- If the file /etc/ansible/ansible.cfg exists, it will be used
- If an ansible.cfg file exists in the current project directory, that will be used and all settings in /etc/ansible/ansible.cfg are ignored
- Use **ansible --version** to see which ansible.cfg will be used
- If a playbook contains settings, they will override the settings from ansible.cfg
- Settings that are provided as command line arguments have the highest priority

Understanding ansible.cfg

- Settings in ansible.cfg are organized in two (or more) sections
  - `[defaults]` sets default settings
  - `[privilege_escalation]` specifies how Ansible runs commands on managed hosts
- The following settings are used:
  - `inventory` specifies the path to the inventory file
  - `remote_user` is the name of the user that logs in on the remote hosts
  - `ask_pass` specifies whether or not to prompt for a password
  - `become` indicates if you want to automatically switch to the become_user
  - `become_method` sets how to become the other user
  - `become_user` specifies the target remote user
  - `become_ask_pass` sets if a password should be asked for when escalating

Connecting to the Remote Hosts

- The default protocol to connect to the remote host is SSH
  - Key-based authentication is the common approach, but password-based authentication is possible as well
  - Use **ssh-keygen** to generate a public/private SSH key pair, and next use **ssh-copy-id** to copy the public key over to the managed hosts
- Other connection methods are available, to manage Windows for instance, use `ansible_connection: winrm` and set `ansible_port: 5986`

Pearson

Escalating Privileges

- **sudo** is the default mechanism for privilege escalation
- Password-less escalation is OK on the RHCE exam, but insecure and not recommended in real world
- To set up password-less sudo, create a drop-in file in /etc/sudoers.d/ with the following contents:
  - `ansible ALL=(ALL) NOPASSWD: ALL`
- To securely escalate privileges using passwords, add the following to /etc/sudoers:
  - `Defaults timestamp_type=global,timestamp_timeout=60`

Understanding Localhost Connections

- Ansible has an implicit localhost entry to run Ansible commands on the localhost
- When connecting to localhost, the default **become** settings are not used, but the account that ran the Ansible command is used
- Ensure this account has been configured with the appropriate **sudo** privileges

Lesson 2: Installing Ansible

2.7 **ansible-navigator** Settings

ansible-navigator initial setup

- After installing **ansible-navigator**, it needs additional setup
- To provide this setup, you'll need to login to a container registry
- This is because **ansible-navigator** uses an execution environment, which is based on a container image

Demo: **ansible-navigator** initial setup

- **sudo dnf install ansible-navigator**
- **ansible-navigator --version**
- **podman login registry.redhat.io**
- **podman pull registry.redhat.io/ansible-automation-platform-22/ee-supported-rhel8:latest**
- **ansible-navigator images**
- **ansible-navigator inventory -i inventory file --list**

Managing Settings for **ansible-navigator**

- Settings for **ansible-navigator** can be specified in a configuration file
- Define **~/.ansible-navigator.yml** for generic settings
- If an **ansible-navigator.yml** file is found in the current project directory, this will have higher priority
- Recommended settings
  - `pull.policy: missing` will only contact the container registry if no container image has been pulled yet
  - `playbook-artifact.enable: false` required when a playbook prompts for a password

ansible-navigator.yml Example

```
ansible-navigator:
  execution-environment:
    image: ee-supported-rhel8:latest
    pull:
      policy: missing
  playbook-artifact:
    enable: false
```

Pearson

Lesson 3: Using Ad Hoc Commands

3.1 Exploring Modules and Content Collections

Understanding Modules and Content Collections

- In Ansible 2.9 and earlier, thousands of modules were provided
- Since Ansible 2.10, only essential modules are provided in the ansible-core package
- Additional modules are provided through content collections
- By organizing modules in content collections, responsibility for module development can be delegated to specific projects and companies
- Additional content collections can be installed from different sources:
  - Ansible Galaxy at https://galaxy.ansible.com
  - Ansible Automation Platform
  - Directly from tar archives

Pearson

ansible-navigator and Modules

- **ansible-navigator** gets modules from content collections using its container-based execution environments
- Using an execution environment means that you don't have to set up an Ansible control host, but just run required content from the execution environment

Using **ansible-navigator** Execution Environments

- To access RHEL-provided execution environments, first use a valid Red Hat account to log in to registry.redhat.io: **podman login registry.redhat.io**
- Next, use **ansible-navigator images** to download the execution environment container image; this will show the ee-supported-rhel8 image (or a later version if that is available)
- Press Esc to exit the image list

Understanding Module Names

- Modules that are provided through collections have a Fully Qualified Collection Name (FQCN):
  - `ansible.builtin.service`
  - `ansible.posix.firewalld`
- Using an FQCN is recommended, as duplicate names may occur between different collections
- If no duplicate names occur, you may use short module names instead
- Using short module names is compatible to Ansible 2.9 and earlier and allows you to run old playbooks without making any modification

Lesson 3: Using Ad Hoc Commands

3.2 Learning How to Use Modules

Getting Help about Modules

- **ansible-doc** provides documentation about all aspects of Ansible
- Different types of items are documented, see **ansible-doc --help** for an overview
- If no type is specified, the module is used as the default type
- To get a list of available modules, use **ansible-doc -l**
- To show usage information about a specific item, use **ansible-doc [-t itemtype] item**

Pearson

Getting Help with **ansible-navigator**

- **ansible-navigator** can also provide help about items
- **ansible-navigator doc -m stdout ansible.core.ping** provides help about the ping module
- Use **ansible-navigator --pp never** to tell navigator to never look for a newer container image
- You may find using **ansible-doc** faster

Demo: Using **ansible-doc**

- **ansible-doc --help**
- **ansible-doc -t keyword -l**
- **ansible-doc -t keyword delegate_to**
- **ansible-doc -l**
- **ansible-doc user**
- **ansible-doc ansible.posix.firewalld**

Lesson 3: Using Ad Hoc Commands

3.3 Installing Content Collections

Understanding Collections

- Regardless of whether you're using just Ansible Core, or **ansible-navigator** in AAP, you'll always have access to one content collection: `ansible.builtin`
- To get access to more collections, you may want to define and use a custom execution environment (which is not a part of the EX294 objectives)
- Collections are provided through Ansible Galaxy or Ansible Automation Platform

Installing Collections

- Use **ansible-galaxy collection install my.collection -p collections** to install new collections
- While installing collections, use the **-p collections** option to tell the execution environment in which directory the collection is available
- Without the **-p path** option, the collection is installed in the default `collections_path`, which is ~/.ansible/collections:/usr/share/ansible/collections
- This default `collections_path` works with the **ansible** and **ansible-playbook** commands, but is not available from within the **ansible-navigator** execution environment
- Using the **-p collections** option, ensures that the collection will always be found, as this directory will always be searched first, even if not specified in the `collections_path`

Installing Collections from Different Locations

Collections can be installed from different locations:

- From Galaxy: Use **ansible-galaxy collection install my.collection -p collections**
- From a tar ball: Use **ansible-galaxy collection install /my/collection.tar.gz -p collections**
- From a URL: Use **ansible-galaxy collection install https://my.example.local/my.collection.tar.gz -p collections**
- From Git: Use **ansible-galaxy collection install git@git.example.local:mygitaccount/mycollecion.git -p collections**

Demo: Using Collections

- **ansible-galaxy collection install community.crypto -p collections**
- **ansible-galaxy collection list**
- **ansible-navigator**
  - **:collections**
- Edit ansible.cfg to include the following:
  - `collections_path=./collections`
- **ansible-galaxy collection list**
- Change the ansible.cfg collection path to the following
  - `collections_path=./collections:~/.ansible/collections:/usr/share/ansible/collections`
- **ansible-galaxy collection list**

Lesson 3: Using Ad Hoc Commands

3.4 Using requirements.yml

Pearson

Understanding collections/requirements.yml

- A requirements.yml file can be provided in the current project directory to list all collections that are needed in the project
- It lists all required collections, and installs them using **ansible-galaxy collections install -r collections/requirements.yml -p collections**
- Don't forget the **-p collections** option, or else the collection won't be found by **ansible-navigator!**

Pearson

Example collections/requirements.yml

```
---
collections:
  - name: community.aws
  - name: ansible.posix
    version: 1.2.1
  - name: /tmp/my-collection.tar.gz
  - name: http://www.example.local/my-collection.tar.gz
  - name: git+https://github.com/ansible-
collections/community.general.git
    version: main
```

Lesson 3: Using Ad Hoc Commands

3.5 Using Content Collections in **ansible-navigator**

Using Collections in Navigator

- The **ee-supported-rhel8** default execution environment comes with a set of common collections
- From **ansible-navigator**, use **:collections** to show collections that are currently available
- To install collections and make them available in **ansible-navigator**, use **ansible-galaxy collection install -p collections** as described before

Lesson 3: Using Ad Hoc Commands

3.6 Exploring Essential Modules

Exploring Essential Modules

- `ansible.builtin.ping`: verifies host availability
  - **ansible all -m ping**
- `ansible.builtin.service`: checks if a service is currently running
  - **ansible all -m service -a "name=httpd state=started"**
- `ansible.builtin.command`: runs any command, but not through a shell
  - **ansible all -m command -a "/sbin/reboot -t now"**
- `ansible.builtin.shell`: runs arbitrary commands through a shell
  - **ansible all -m shell -a set**
- `ansible.builtin.raw`: runs a command on a remote host without a need for Python
- `ansible.builtin.copy`: copies a file to the managed host
  - **ansible all -m copy -a 'content="hello world" dest=/etc/motd'**

Lesson 3: Using Ad Hoc Commands

3.8 Using docs.ansible.com

Using docs.ansible.com

- Ansible documentation is provided on docs.ansible.com
- It may be a bit hard to find what you need, as the documentation is focused on Automation Platform
- The best resource for documentation about Ansible Core is under the Core button on this website

Lesson 3: Using Ad Hoc Commands

3.7 Idempotency

Understanding Idempotency

- Idempotency in Ansible means that no matter the current state of the managed node, running an Ansible module should always give the same result
- The result is that the desired state as expressed by the module should be implemented
- If the current state already matches the desired state, nothing should happen
- Most important: if the current state already matches the desired state, the Ansible module should not generate an error message
- In Ansible you should always configure idempotent solutions
- Some modules - including `command` however are not idempotent

Demo: Exploring Idempotency

- **ansible ansible1 -m command -a "useradd lisa"**
- **ansible ansible1 -m command -a "useradd lisa"**
- **ansible ansible1 -m user -a "name=linda"**
- **ansible ansible1 -m user -a "name=linda"**

Lesson 4: Getting Started with Playbooks

4.1 Using YAML to Write Playbooks

Pearson

Why Use Playbooks

- Ad-hoc commands can be used to run one or a few tasks
- Ad-hoc commands are convenient to test, or when a complete managed infrastructure hasn't been set up yet
- Ansible playbooks are used to run multiple tasks against managed hosts in a scripted way
- In playbooks, one or multiple plays are started
  - Each play runs one or more tasks
  - In these tasks, different modules are used to perform the actual work
- Playbooks are written in YAML and have the .yml or .yaml extension
- Note that in Ansible, it is common to use the .yml extension, not the .yaml extension

Understanding YAML

- YAML is Yet Another Markup Language according to some
- According to others it stands for YAML Ain't Markup Language
- Anyway, it's an easy-to-read format to structure tasks/items that need to be created
- In YAML files, items are using indentation with white spaces to indicate the structure of data
- Data elements at the same level should have the same indentation
- Child items are indented more than the parent items
- There is no strict requirement about the number of spaces that should be used, but 2 is common

Writing Your First Playbook

```
---
- name: deploy vsftpd
  hosts: ansible2
  tasks:
  - name: install vsftpd
    yum: name=vsftpd
  - name: enable vsftpd
    service: name=vsftpd enabled=true
  - name: create readme file
    copy:
      content: "free downloads for everybody"
      dest: /var/ftp/pub/README
...
```

Pearson

Lesson 4: Getting Started with Playbooks

4.2 Running Playbooks

Pearson

Running Your First Playbook

- Use **ansible-playbook vsftpd.yml** to run the playbook
- Notice that a successful run requires the *inventory* and *become* parameters to be set correctly, and also requires access to an inventory file
- The output of the **ansible-playbook** command will show what exactly has happened
- Playbooks should be idempotent, which means that running the same playbook again should lead to the same result
- Notice there is no easy way to undo changes made by a playbook

Using **ansible-navigator**

- **ansible-navigator run -m stdout --pp never vsftpd.yml** will run the playbook using navigator
  - **-m stdout** will write output to STDOUT instead of using interactive mode
  - **--pp never** will not check for newer container images
- **ansible-navigator run --pp never vsftpd.yml** will run the playbook in interactive mode
  - Use the indicated numbers to get more details about each step
  - Use Esc to get out of any interface
  - Set `policy: missing` in .ansible-navigator.yml to avoid using the **--pp never** option

Pearson

Lesson 4: Getting Started with Playbooks

4.3 Verifying Playbook Syntax

Verifying Syntax

- **ansible-playbook --syntax-check vsftpd.yml** will perform a syntax check
- Use **-v[vvv]** to increase output verbosity
  - **-v** will show task results
  - **-vv** will show task results and task configuration
  - **-vvv** also shows information about connections to managed hosts
  - **-vvvv** adds information about plug-ins, users used to run scripts, and names of scripts that are executed
- Use **ansible-navigator run [--pp never] -m stdout vsftpd.yml --syntax-check** to perform a syntax check with navigator

Lesson 4: Getting Started with Playbooks

4.4 Using Multiple-Play Playbooks

Understanding Plays

- A playbook can contain multiple plays
- A play is a series of tasks that are executed against selected hosts from the inventory, using specific credentials
- Using multiple plays allows running tasks on different hosts, using different credentials from the same playbook
- Within a play definition, escalation parameters can be defined:
  - `remote_user`: the name of the remote user
  - `become`: to enable or disable privilege escalation
  - `become_method`: to allow using an alternative escalation solution
  - `become_user`: the target user used for privilege escalation
- When these parameters are used at a more specific level, they will overwrite the more generic setting

Lesson 5: Working with Variables

5.1 Understanding Variables

Understanding Variables

- A variable is a label that is assigned to a specific value to make it easy to refer to that value throughout the playbook
- Variables can be defined by administrators at different levels
- A fact is a special type of variable, that refers to a current state of an Ansible-managed system
- Variables are particularly useful when dealing with managed hosts where specifics are different
    - Set a variable `web_service` on Ubuntu and Red Hat
    - Refer to the variable `web_service` instead of the specific service name

Defining Variables

- In Ansible, different types of variables can be used and defined
- Variables can be defined in playbooks
- Alternatively, include files can be used
- Variables can be specified as command line arguments
- The output of a command or task can be used in a variable using `register`
- `vars_prompt` can be used to ask for input and store that as a variable
- **ansible-vault** is used to encrypt sensitive values
- Facts are discovered host properties stored as variables
- Host variables are host properties that don't have to be discovered
- System variables are a part of the Ansible system and cannot be changed

Pearson

Variable Precedence

- Variables can be set with different types of scope
  - Global scope: this is when a variable is set from inventory or the command line
  - Play scope: this is applied when it is set from a play
  - Host scope: this is applied when set in inventory or using a host variable inclusion file
- When the same variable is set at different levels, the most specific level gets precedence
- When a variable is set from the command line, it has highest precedence
  - **ansible-playbook site.yml -e "web_package=apache"**

Lesson 5: Working with Variables

5.2 Using Variables in Playbooks

Defining Playbook Variables

- Variables can be defined in a vars section in the beginning of a play

```
- hosts: all
  vars:
    web_package: httpd
```

- Alternatively, variables can be defined in a variable file, which will be included from the playbook

```
- hosts: all
  vars_files:
    - vars/users.yml
```

Pearson

Using Variables

- After defining the variables, they can be used later in the playbook
- Refer to a variable as `{{ web_package }}`
- In conditional statements (discussed later), no curly braces are needed to refer to variable values
  - `when: "'not found' in command_result.err"`

  -
    ```
    {% if ansible_facts['devices']['sdb'] is defined %}
        Secondary disk size: {{
    ansible_facts['devices']['sdb']['size'] }}
    ```
- If the variable is the first element, using quotes is mandatory: `"{{ web_package }}"`

Lesson 5: Working with Variables

5.3 Including Variables

Includes

- While writing playbooks, it is good practice not to include site specific data in the playbooks
- Playbooks that define variables within the playbook are less portable
- To make variables more flexible, they should be included in the play header using `vars_files`
- See **ansible-doc -t keyword vars_files** for a short description
- The variables file itself contains variable definitions as `key: value`

Pearson

Lesson 5: Working with Variables

5.4 Managing Host Variables

Using Host Variables

- Host variables are variables that are specific to a host only
- They are defined in a YAML file that has the name of the inventory hostname and are stored in the host_vars directory in the current project directory
- To apply variables to host groups, a file with the inventory name of the host group should be defined in the group_vars directory in the current project directory
- Host variables defined this way will be picked up by the hosts automatically
- Host variables can also be set in the inventory, but this is now deprecated

Lesson 5: Working with Variables

5.5 Using System Variables

System Variables

Some variables are built in and cannot be used for anything else

- `hostvars`: a dictionary that contains all variables that apply to a specific host
- `inventory_hostname`: inventory name of the current host
- `inventory_hostname_short`: short host inventory name
- `groups`: all hosts in inventory, and groups these hosts belong to
- `group_names`: list of groups the current host is a part of
- `ansible_check_mode`: boolean that indicates if play is in check mode
- `ansible_play_batch`: active hosts in the current play
- `ansible_play_hosts`: same as ansible_play_batch
- `ansible_version`: current Ansible version

Pearson

Lesson 5: Working with Variables

5.6 Using Register to Set Variables

Understanding register

- `register` is used to record the result of a command or task and store it in a variable
- Using `register` is convenient to work with variables that have a dynamic value

Lesson 5: Working with Variables

5.7 Using Vault to Manage Sensitive Values

Dealing with Sensitive Data

- Some modules require sensitive data to be processed
- This may include webkeys, passwords, and more
- To process sensitive data in a secure way, Ansible Vault can be used
- Ansible Vault is used to encrypt and decrypt files
- To manage this process, the **ansible-vault** command is used

Demo: Using Vault

- To create an encrypted file, use **ansible-vault create sensitive_values**
- To view a vault encrypted file, use **ansible-vault view sensitive_values**
- To edit, use **ansible-vault edit sensitive_values**
- Use **ansible-vault encrypt sensitive_values** to encrypt an existing file, and use **ansible-vault decrypt sensitive_values** to decrypt it
- To change a password on an existing file, use **ansible-vault rekey sensitive_values**

Using Playbooks with Vault

- To use a vault-encrypted file, include the vault file using vars_files:
- To run a playbook that uses one vault encrypted file, use **--ask-vault-pass**
- To run a playbook that accesses Vault encrypted files, you need to use the **--vault-id key@prompt** option to be prompted for a password
  - **ansible_playbook --vault-id username@prompt --vault-id pwhash@prompt ...**
- Alternatively, you can store the password as a single-line string in a password file, and access that using the **--vault-password-file=vault-file** option

Managing Vault Files

- When setting up projects with Vault encrypted files, it makes sense to use separate files to store encrypted and non-encrypted variables
- To store host or host-group related variable files, you can use the following structure:

Managing Vault Files

- When setting up projects with Vault encrypted files, it makes sense to use separate files to store encrypted and non-encrypted variables
- To store host or host-group related variable files, you can use the following structure:

```
|-group_vars
| |--dbservers
|     |- vars
|     |- vault
```

- This replaces the solution that was discussed earlier, where all variables are stored in a file with the name of the host or host group

demo: using Vault

- ansible-vault create secret.yml
  - username: bob
  - pwhash: password
- vim create-user.yml
- ansible-playbook --ask-vault-pass create-users.yml
- echo password > vault-pass; chmod 600 vault-pass
- ansible-playbook --vault-password-file=vault-pass create-users.yml

Lesson 5: Working with Variables

Lesson 5 Lab: Using Ansible Vault

Lesson 5 Lab: Using Ansible Vault

- Create a playbook that creates a user. The user as well as the password should come from a variable.
- Define both in separate include files.
- Make sure the password is encrypted with vault, whereas the username is not encrypted with vault.
- Verify it works.

Lesson 6: Working with Facts

6.1 Understanding Facts

Understanding Facts

- Ansible Facts are variables that are automatically set and discovered by Ansible on managed hosts
- Facts contain information about hosts that can be used in conditionals
  - Available memory
  - IP address
  - Distribution and distribution family
  - and much more
- Using facts allows you to perform actions only if certain conditions are true

Managing Fact Gathering

- By default, all playbooks perform fact gathering before running the actual plays
- Use `gather_facts: no` in the play header to disable
- Even if fact gathering is disabled, it can be enabled again by running the `setup` module in a task
- You can run fact gathering in an ad hoc command using the `setup` module
- To show facts, use the debug module to print the value of the `ansible_facts` variable
- Notice that in facts, a hierarchical relation is shown where you can use the dotted format to refer to a specific fact

demo

- ansible -m setup all
- ansible-playbook facts.yml
- ansible-playbook ipfact.yml

Lesson 6: Working with Facts

6.2 Using Multi-tier Variables

Understanding Fact Organization

- In Ansible 2.4 and before, Ansible facts were stored as individual variables, such as `ansible_hostname` and `ansible_interfaces`, containing different sub-variables stored in a dictionary
- To refer to the next-tier variables, dots were used as a separator: `ansible_date_time.date`
- In Ansible 2.5 and later, all facts are stored in one dictionary variable with the name `ansible_facts`, and referring to specific facts happens in a different way: `ansible_facts['hostname']`

```
ansible_date_time.date
ansible_facts.date_time.date
ansible_facts['date_time']['date']
```

Lesson 6: Working with Facts

6.3 Understanding Dictionaries and Arrays

Understanding Array and Dictionary

- Multi-valued variables can be used in playbooks and can be seen in output, such as Ansible facts
- When using a multi-valued variable, it can be written as an array (list), or as a dictionary (hash)
- Each of these has their own specific use cases
    - Dictionaries are used in Ansible facts
    - Arrays are common for multi-valued variables, and easily support loops
    - Loops are not supported on dictionaries

Dictionaries versus Arrays

- Dictionaries and arrays in Ansible are based on Python dictionaries and arrays
- An array (list) is an ordered list of values, where each value can be addressed individually

```
List = ["one", "two", "three"]
print(List[0])
```

- A dictionary (hash) is an unordered collection of values, which is stored as a key-value pair

```
Dict = {1: 'one', 2: 'two', 3: 'three'}
print(Dict)
```

- Notice that a dictionary can be included in a list, and a list can be a part of a dictionary

Understanding Dictionary (Hash)

- Dictionaries can be written in two ways:

Understanding Dictionary (Hash)

- Dictionaries can be written in two ways:

```
users:
  linda:
    username: linda
    shell: /bin/bash
  lisa:
    username: lisa
    shell: /bin/sh
```

- Or as:

```
users:
  linda:{ username: 'linda', shell: '/bin/bash' }
  lisa: { username: 'lisa', shell: '/bin/bash' }
```

Pearson

Using Dictionary

- To address items in a dictionary, you can use two notations:
  - variable_name['key'], as in `users['linda']['shell']`
  - variable_name.key, as in `users.linda.shell`
- Dictionaries are used in facts, use arrays in conditionals

Using Array (List)

- Arrays provide a list of items, where each item can be addressed separately

```
users:
- username: linda
  shell: /bin/bash
- username: lisa
  shell: /bin/sh
```

- Individual items in the array can be adressed, using the `{{ var_name[0] }}` notation
- Use arrays for looping, not dictionaries
- To access all variables, you can use `with_items` or `loop` (covered in the next lesson)

Pearson

Recognizing Arrays and Dictionaries

- In output like facts, a list is always written between `[ ]`
- Dictionaries are written between `{ }`
- And if the output is just showing `" "`, it is a string
- Check `ansible_mounts` in the facts, which actually presents a list of dictionaries

demo

- arrays/vars/users-dictionary
- arrays/vars/users-list

Lesson 6: Working with Facts

6.4 Defining Custom Facts

Understanding Custom Facts

- Host (inventory) variables exist on the control machine and the host itself doesn't know about them
- Custom facts allow administrators to dynamically generate variables which are on the host stored as facts
- Custom facts are stored in an ini or json file in the /etc/ansible/facts.d directory on the managed host
    - The name of these files must end in .fact
    - Custom facts must have a [label] to help identify the variables

Using Custom Facts

- Custom facts are stored in the `ansible_facts.ansible_local` variable
- Use **ansible hostname –m setup –a "filter=ansible_local"** to display local facts
- Notice how fact filename and label are used in the fact
- To refer to custom facts, use `ansible_facts.ansible_local`, not `ansible_facts.local`

Demo

- ansible-playbook newlocalfacts.yml
- ansible hostname -m setup -a "filter=ansible_local"

Lesson 6: Working with Facts

6.5 Understanding Variable Precedence

Understanding Variable Scope

- Variables set in a playbook only exist in the specific *play* where they are defined
- Variables can get a *playbook* scope by defining them as follows:
  - Through host or host group
  - In inventory
  - By vars plugins
  - Using modules like `set_fact` and `include_vars`
- Inventory and host [group] variables have a host scope. This also goes for the `hostvars[]` dictionary
- Use **-e "var=name"** to define a variable with a playbook scope that overrides all other variables

Understanding Variable Precedence

- More specific will always win, Ansible will give precedence to variables that were defined more recently, more actively, or with more explicit scope
- Below is a partial list of precendence, from lowest to highest
  - role defaults
  - inventory variables
  - host facts
  - play variables
  - included play variables
  - task variables
  - variables provided on the command line

Lesson 6: Working with Facts

Lesson 6 Lab: Working with Facts

Lesson 6 Lab: Working with Facts

- Create a playbook that sets a local fact type=production on host ansible1
- Use an ad-hoc command with a filter to check that the fact was created successfully

Lesson 7: Using Task Control

7.1 Understanding Conditionals

Conditionals Overview

- `loop`: allows you to loop over a list of items instead of calling the same task repeatedly
- `when`: performs conditional task execution based on the value of specific variables
- `handlers`: used to perform a task only if triggered by another task that has changed something

Lesson 7: Using Task Control

7.2 Writing Loops

Understanding Loops

- The `loop` keyword allows you to iterate through a simple list of items
- Before Ansible 2.5, the `with_` keyword was used instead

```
- name: start some services
  service:
    name: "{{ item }}"
    state: started
  loop:
    - vsftpd
    - httpd
```

Pearson

Using Variables to Define a Loop

- The list that `loop` is using can be defined by a variable:

```
vars:
  my_services:
    - httpd
    - vsftpd
tasks:
  - name: start some services
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ my_services }}"
```

Pearson

Using Dictionaries in Loops

- Each item in a loop can be a hash/dictionary with multiple keys in each hash/dictionary

```
- name: create users using a loop
  hosts: all
  tasks:
  - name: create users
    user:
      name: "{{ item.name }}"
      state: present
      groups: "{{ item.groups }}"
    loop:
      - name: anna
        groups: wheel
      - name: linda
        groups: users
```

Pearson

loop versus with_

- The `loop` keyword is the current keyword
- In previous versions of Ansible, the `with_*` keywords were used for the same purpose
- Using `with_X` often is easier, but using plugins and filters offers more options
  - `with_items`: equivalent to the `loop` keyword
  - `with_file`: the `item` contains a file, which content is used to loop through
  - `with_sequence`: generates a list of values based on a numeric sequence
- Look up "Migrating from with_X to loop" in the Ansible documentation for instructions on how to migrate
- Notice that the ansible team has changed their mind, and there are no plans to remove with_X support in the near future

demo

- Run loopservices.yml
- Notice the deprecation warning, this is because of the – in the inventory group name
- Run loopusers.yml
- Run loop-over-variable.yml

Pearson

Lesson 7: Using Task Control

7.3 Using when

Using Conditionals

- `when` statements are used to run a task conditionally
- A condition can be used to run a task only if specific conditions are true
- Playbook variables, registered variables, and facts can be used in conditions and make sure that tasks only run if specific conditions are true
- For instance, check if a task has run successfully, a certain amount of memory is available, a file exists, etc.
- When using conditions, it is important to address the right variable type

Example Conditionals

`ansible_machine == "x86_64"`	Variable contains string value
`ansible_distribution_version == "8"`	Variable contains string value
`ansible_memfree_mb == 1024`	Variable value is equal to integer
`ansible_memfree_mb < 256`	Variable value is smaller than integer
`ansible_memfree_mb > 256`	Variable value is bigger than integer
`ansible_memfree_mb <= 256`	Variable value is smaller than or equal to integer
`ansible_memfree_mb != 512`	Variable value is not integer
`my_variable is defined`	Variable exists (nice for facts)
`my_variable is not defined`	Variable does not exist
`my_variable`	Variable is Boolean true
`ansible_distribution in supported_distros`	Variable contains another variable

Understanding 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 (yes/no, y/n, on/off also supported)
- Dates: calendar dates
- Null: undefined variable type
- List or Arrays: a sorted collection of values
- Dictionary or Hash: a collection of key/value pairs

Using Filters to Enforce Variable Types

- While working with variables in a when statement, the variable type may be interpreted wrongly
- To ensure that a variable is treated as a specific type, filters can be used
  - int (integer) `when vgsize | int > 5`
  - bool (boolean) `when runme | bool`
- Using a filter does not change the variable type, it only changes the way it is interpreted

Demo

- distro.yml
- The key issue in this playbook is on the last line, "ansible_distribution" is a fact, and "supported_distros" is a variable that is set in the playbook
- quicktest.yml

Lesson 7: Using Task Control

7.4 Using When and Register to Check Multiple Conditions

Testing Multiple Conditions

- `when` can be used to test multiple conditions as well
- Use `and` or `or` and group the conditions with parentheses

```
when: ansible_distribution == "CentOS" or \
ansible_distribution == "RedHat"
when:  ansible_machine == "x86_64" and \
ansible_distribution == "CentOS"
```

- The `when` keyword also supports a list and when using a list, all of the conditions must be true
- Complex conditional statements can group conditions using parentheses

Demo

- when_multiple
  - 1st run: put 512". between quotes – it will fail, talk about the string error message
  - 2nd run: set 512 to 1512
  - talk about the alternative notation using “and”
- when_multiple_complex
- ifsize.yml # notice that teh ansible_mount fact is a *list* of *dictionaries*

Using Register Conditionally

- The `register` keyword is used to store the results of a command or tasks
- Next, `when` can be used to run a task only if a specific result was found

Tip! Use `debug` to explore the variable that was created by register

```
 - name: show register on random module
    user:
      name: "{{ username }}"
    register: user
  - name: show register results
    debug:
      var: user
```

demo

- register.yml
- register_command.yml

Lesson 7: Using Task Control

7.5 Conditional Task Execution with Handlers

Understanding Handlers

- Handlers run only if the triggering task has changed something
- By using handlers, you can avoid unnecessary task execution
- In order to run the handler, a `notify` statement is used from the main task
- Handlers typically are used to restart services or reboot hosts

```
 - name: copy index.html
   copy:
     src: /tmp/index.html
     dest: /var/www/html/index.html
   notify:
     - restart_web
handlers:
      - name: restart_web
        service:
          name: httpd
          state: restarted
```

Using Handlers

- Handlers are executed after running all tasks in a play
- Use `meta: flush_handlers` to run handlers now
- Handlers will only run if a task has *changed* something
- If one of the next tasks in the play fails, the handler will not run, but this may be overwritten using `force_handlers: True`
- One task may trigger more than one handler

demo

- handlers.yml

Using ansible.builtin.meta

- Handlers are executed at the end of the play
- To change this behavior, the `ansible.builtin.meta` module can be used
- This module specifies options to influence the Ansible internal execution order
  - `flush_handlers`: will run all notified handlers now
  - `refresh_inventory`: refreshes inventory at the moment it is called
  - `clear_facts`: removes all facts
  - `end_host`: ends playbook execution for this host

Lesson 7: Using Task Control

7.6 Using Blocks

Understanding Blocks

- A block is a logical group of tasks
- It can be used to control how tasks are executed
- For instance, one block can be enabled using a single `when`
- Notice that items cannot be used on blocks

Using Blocks in Error Handling

- Blocks can be used in error condition handling
  - Use **block** to define the main tasks to run
  - Use **rescue** to define tasks that run if tasks defined in the **block** fail
  - Use **always** to define tasks that will always run

```
- name: using blocks
  hosts: all
  tasks:
  - name: intended to be successful
    block:
    - name: remove a file
      shell:
        cmd: rm /var/www/html/index.html
    rescue:
    - name: create a file
      shell:
        cmd: touch /tmp/rescuefile
```

Demo

- blocks.yml
- blocks2.yml

Lesson 7: Using Task Control

7.7 Managing Task Failure

Understanding Failure Handling

- Ansible looks at the exit status of a task to determine whether it has failed
- When any task fails, Ansible aborts the rest of the play on that host and continues with the next host
- Different solutions can be used to change that behavior
- Use `ignore_errors` in a task or play to ignore failures
- Use `force_handlers` to force a handler that has been triggered to run, even if (another) task fails
  - Notice that if `ignore_errors: yes` and `force_handlers: no` both have been set, the handlers will run after failing tasks

Pearson

Defining Failure States

- As Ansible only looks at the exit status of a failed task, it may think a task was successful where that is not the case
- To be more specific, use `failed_when` to specify what to look for in command output to recognize a failure

```
- name: run a script
    command: echo hello world
    ignore_errors: yes
    register: command_result
    failed_when: "'world' in command_result.stdout"
```

Demo

- Failure.yml/failure.yml

Using the fail Module

- The `failed_when` keyword can be used in a task to identify when a task has failed
- The `fail` module can be used to print a message that informs why a task has failed
- To use `failed_when` or `fail`, the result of the command must be registered, and the registered variable output must be analyzed
- When using the `fail` module, the failing task must have `ignore_errors` set to yes

demo

- listing725.yaml

Lesson 7: Using Task Control

7.8 Managing Changed Status

Understanding Changed

- Ansible looks at the exit status of commands it runs
- Idempotent modules can make a difference between "changed" and "no change required"
- Non-idempotent modules like `command` and `shell` cannot do that, and only work with an exit status of 0 or 1, which is next processed by Ansible to determine module success or failure
- Because of this, a non-idempotent module may falsely report changed, when really no change has happened

Pearson

Managing Changed Status

- Managing the changed status may be important, as handlers trigger on the changed status
- The result of a command can be registered, and the registered variable can be scanned for specific text to determine that a change has occurred
- This allows Ansible to report a changed status, where it normally would not, thus allowing handlers to be triggered
- Using `changed_when` is common in two cases:
  - to allow handlers to run when a change would not normally trigger
  - to disable commands that run successfully to report a changed status

Pearson

Demo

- Using changed_when is usable in two cases:
  - to allow handlers to run when a change would not normally trigger
  - to disable commands that run successful to report a changed status
- Run changed.yml
- Run again, but now disable the changed_when

Lesson 7: Using Task Control

7.9 Including and Importing Files

Understanding Inclusion

- If playbooks grow larger, it is common to use modularity by using includes and imports
- Includes and imports can happen for roles, playbooks, as well as tasks
- An *include* is a dynamic process; Ansible processes the contents of the included files at the moment that this include is reached
- An *import* is a static process; Ansible preprocesses the imported file contents before the actual play is started
    - Playbook imports must be defined at the beginning of the playbook, using `import_playbook`

Including Task Files

- A task file is a flat list of tasks
- Use `import_tasks` to statically import a task file in the playbook, it will be included at the location where it is imported
- Use `include_tasks` to dynamically include a task file
- Dynamically including tasks means that some features are not available
  - **ansible-playbook --list-tasks** will not show the tasks
  - **ansible-playbook --start-at-task** doesn't work
  - You cannot trigger a handler in an imported task file from the main task file

Best Practice: store task files in a dedicated directory to make management easier

Demo

- includes.yml
- imports.yaml: notice that its weird as the include is not dynamic but the import is!

Lesson 7: Using Task Control

Lesson 7 Lab: Running Tasks Conditionally

Lesson 7 Lab: Running Tasks Conditionally

- Write a playbook that writes "you have a second disk" if a second disk was found on a node, and "you have no second disk" in the case that no second disk was found.

Lesson 8: Managing Files

8.1 Modifying Files

File Management Modules

Different Modules are available for managing files

- `ansible.builtin.lineinfile`: ensures that a line is in a file, useful for changing a single line in a file
- `ansible.builtin.blockinfile`: manipulates multi-line blocks of text in files
- `ansible.builtin.file`: sets attributes to files, and can also create and remove files, symbolic links and more
- `ansible.builtin.stat`: used to request file statistics. Useful when combined with register

demo

- file.yml
  - state: touch doesn't use a source file
  - selinux context is set on file, not on policy level
- addnewhost.yml
- addcustomfacts.yml

Lesson 8: Managing Files

8.2 Copying Files To and From Managed Hosts

File Copying Modules

Different Modules are available for managing files

- `ansible.builtin.copy`: copies a file from a local machine to a location on a managed host
- `ansible.builtin.fetch`: used to fetch a file from a remote machine and store it on the management node
- `ansible.posix.synchronize`: synchronizes files **rsync** style. Only works if the Linux **rsync** utility is available on managed hosts
- `ansible.posix.patch`: applies patches to files

demo

- copy.yml

Lesson 8: Managing Files

8.3 Using Jinja2 Templates

Understanding Jinja2 Templates

- `lineinfile` and `blockinfile` can be used to apply simple modifications to files
- For more advanced modifications, use Jinja2 templates
- Jinja2 is a templating engine for Python
- In a Jinja2 template, variables can be used that are defined in the playbook
- The `ansible.builtin.template` module renders the Jinja2 template to a final configuration file

Best practice: use Jinja2 templates if you need to generate files with different variables-based content on managed hosts. For simple modifications, use lineinfile and blockinfile

Marking Managed Files

- To prevent administrators from overwriting files that are managed by Ansible, set the `ansible_managed` string
  - First, in ansible.cfg set `ansible_managed = # Ansible managed`
  - On top of the Jinja2 template, include the following line: `# {{ ansible_managed }}`
- In the `ansible_managed` string, different variables can be used:

```
ansible_managed = {file} modified by Ansible on %d-%m-%Y
by {uid}
```

demo

- add **ansible_managed = {file} modified by Ansible on %d-%m-%Y by {uid}**
- vsftpd-template.yml and vsftpd.j2.

Lesson 8: Managing Files

8.4 Applying Conditionals in Jinja2 Templates

Using for Statements

- Jinja2 templates can loop over the value of a variable using a for statement

```
{% for user in users %}
    {{ user }}
{% endfor %}

{% for host in groups['webservers'] %}
    {{ host }}
{% endfor %}
```

demo

- hostsfile.yml
- Explain that **hostvars** is one of the magic variables
- Explain that this stuff can be found in the FAQ

Lesson 8: Managing Files

8.5 Managing SELinux File Context

Using Modules to Manage SELinux

- `ansible.builtin.file`: sets attributes to files, including SELinux context, and can also create and remove files, symbolic links and more
- `community.general.sefcontext`: manages SELinux file context in the SELinux Policy (but not on files)
- `ansible.builtin.command`: required to run **restorecon** after `sefcontext`
- Notice that `file` sets SELinux context directly on the file (like the **chcon** command), and not in the policy. DO NOT USE!
- Also consider using the RHEL system role for managing SELinux

Pearson

demo

- selinux.yml

Lesson 8: Managing Files

Lesson 8 Lab: Applying Conditionals in Templates

Pearson

Lesson 8 Lab: Applying Conditionals in Templates

- Create a playbook that sets the hostnames of the managed hosts to the names that are used in the inventory.
- Reboot the managed hosts after setting the hostnames.

Lesson 9: Using Ansible Roles

9.1 Understanding Roles

Understanding Roles

Roles are about reusable code

- Many Ansible projects exist, and often the same things need to be done
- Roles provide standardized solutions
- Community roles are provided through galaxy.ansible.com
- Custom roles can be created and distributed through an organizations local Git repositories
- Roles are included in tasks, using a `roles:` section in the playbook play header

Lesson 9: Using Ansible Roles

9.2 Using Ansible Galaxy to Get Roles

Getting Roles from External Sources

Roles can be provided in different ways

- Through Ansible Galaxy, either as roles or as a part of a content collection
- As tar balls
- From RPM packages
- Through Ansible Content Collections from Red Hat automation hub at https://console.redhat.com

Using Galaxy Roles

- Community roles are provided through Ansible Galaxy
- From there, find the role that you would like to use and fetch it, using **ansible-galaxy role install**
- While installing roles, the `roles_path` setting is used
- The default `roles_path` will use roles in the following order of precedence
  - A roles directory in the current project directory
  - The ~/.ansible/roles directory
  - /etc/ansible/roles
  - /usr/share/ansible/roles
- Optionally, use **ansible-galaxy role install -p mypath** to install in an alternative path

Demo

- Show galaxy website, download a role using **-p roles** and show the structure
- Do NOT include it in a playbook yet
- **ansible-galaxy role list**
- add **roles_path = roles** to [defaults] in ansible.cfg
- **ansible-galaxy role list**
- Change **roles_path** to **roles:~/.ansible/roles:/etc/ansible/roles:/usr/share/ansible/roles**
- **ansible-galaxy role list**

Pearson

Using the **ansible-galaxy** Command

- The **ansible-galaxy** command has a few useful options
- **ansible-galaxy [role] search 'docker' --author geerlingguy --platforms EL** will search for roles containing the keyword docker, written for usage on RHEL and related by author geerlingguy
- **ansible-galaxy [role] info geerlingguy.docker** shows information about the role
- **ansible-galaxy [role list]** will list roles
- **ansible-galaxy [role] remove geerlingguy.docker** will remove the role

Pearson

Using a Requirements File

- If specific roles are needed in a project, they can be specified in the `roles/requirements.yml` file in the project directory
- By using a `requirements.yml` file, you'll have one location in the project to manage which roles are needed
- In a requirements.yml file, different sources can be used
  - Git:
    ```
    https://github.com/mygit/myrole
    scm: git
    ```
  - files: `file:///tmp/myrole.tar`
  - web: `https://www.example.local/myrole.tar`
- If a role is hosted in Git, the `scm: git` attribute is required

Pearson

Sample Requirements File

```
- src: https://github.com/myaccount/myrole.role
  scm: git
  version: "2.0"
- src: file:///tmp/myrole.tar
  name: mytarrole
- src: https://example.local/myrole.tar
  name: mywebrole
```

Demo

- Use **ansible-galaxy role install -r roles/requirements -p roles** to install roles specified in the requirements file to the project roles directory
- Without the **-p roles** option, the roles are installed to the first directory in the roles_path, which is ~/.ansible/roles
- **ansible-navigator** cannot find roles in that directory, which is why the **-p roles** option is needed
- Alternatively, redefine **roles_path = roles** in the ansible.cfg

Lesson 9: Using Ansible Roles

9.3 Understanding Roles in Ansible Content Collections

Roles and Ansible Content Collections

- Roles may be distributed through Ansible Content Collections
- This is not commonly done
- When roles are provided through collections, you'll find them using **ansible-navigator collections**
- Select **redhat.insights** to see some roles (use **:21** to address numbers higher than 9)

Pearson

Lesson 9: Using Ansible Roles

9.4 Writing Playbooks that use Roles

Pearson

Using Roles in Playbooks

- Roles can be used in different ways
  - Using a `roles:` section in the play header
  - Using the `import_role:` or `include_role:` module in a task
- Roles listed in the `roles:` section are executed before the tasks in the play
- Use `pre_tasks:` to trigger tasks to run before the roles
- Use `post_tasks:` to force tasks to run after the roles and `tasks:`
- Notice that you don't have to use a `tasks:` section in the play, it will also work if you just have a `roles:` section

Best practice: if roles should be executed after some tasks, define the tasks and use `import_role` where needed

include_role versus import_role

- `include_role` dynamically includes the role when it is referred to in a task
- `import_role` is processed when the playbook is parsed
- While using `import_role` the role handlers and variables are exposed to all tasks in the play, even if the role has not run yet

Best practice: in most cases, using `include_role` is better than using `import_role`

Variables in Roles

- Roles are often pre-configured with standard variables
  - Variables in the defaults directory in the role provide default variables that are intended to be changed in plays
  - Variables in the vars directory in the role are used for internal purposes in the role and they are not intended to be overwritten in the playbook
- Site-specific information should be set through playbook variables, which will be picked up by the role

Best practice: Don't define local variables or vault encrypted variables in the role, they should always be defined locally in the playbook

Defining Variables are Role Parameters

- Variables can be set as a role parameter while calling the role from the playbook
- This is useful if the same role is used multiple times, with different values for the same variable

```
---
- hosts: webservers
  roles:
    - role: myrole
      message: hello
    - role: myrole
      message: bye
```

Pearson

Role Variable Precedence

- Role variables in the vars directory are not supposed to be overwritten, but will be overwritten by facts, registered variables, and variables loaded with `include_vars`
- Role variables in the default directory are overwritten by any other variable definition
- If a variable is declared as a role parameter, it has the highest precedence

Demo

- ansible-galaxy role install geerlingguy.nginx
- ansible-galaxy role list
- ansible-playbook nginx-role

Lesson 9: Using Ansible Roles

9.5 Writing Custom Roles

Creating Custom Roles

- To create a custom role, just make sure to create the required directory structure somewhere in the `roles_path`
- Consider using **ansible-galaxy role init myrole** to create the default directory structure for the myrole role
- Next, complete the main.yml files to provide role content

Demo

- ansible-galaxy role init motd
- explore defaults, check the motd.yml playbook

Lesson 9: Using Ansible Roles

9.6 Using RHEL System Roles

Understanding RHEL System Roles

- RHEL system roles are provided as an easy interface to manage specific parameters between different RHEL versions
- Use **dnf install -y rhel-system-roles** to install them
- RHEL system roles come with sample playbooks in /usr/share/doc/rhel-system-roles/ if installed from the RPM package

Exam tip! If you need to use RHEL system roles on the exam, copy a sample playbook from /usr/share/doc/rhel-system-roles/ to your project directory and modify it instead of creating a new one!

Installing RHEL System Roles

- For Ansible Automation Platform customers, the RHEL system roles are available as a content collection: `redhat.rhel_system_roles`
- Alternatively, they are available in rhel-system-roles.rpm for Ansible Core users

Tip! Best install rhel-system-roles using **dnf install rhel-system-roles**. This installs them to /usr/share/ansible/roles and /usr/share/ansible/collections, which ensures easy access from **ansible-navigator** as well as the **ansible-playbook** utility and also provides useful examples in /usr/share/doc

Pearson

Lesson 9: Ansible Roles

9.7 Configuring Ansible Roles and Collection Sources

Configuring Role and Collection Sources

- The **ansible-galaxy** command fetches collections from https://galaxy.ansible.com
- To configure alternative sources and manage source priority, modify the ansible.cfg file
- To access content collections from automation hub, you need to get an authentication token
- Generate this token from https://console.redhat.com/ansible/automation-hub/token
- As an alternative for specifying a token, you may include a `username` and `password` as well

Example ansible.cfg File

```
[galaxy]
server_list = automation_hub, galaxy

[galaxy_server.automation.hub]
url=https://console.redhat.com/CHECK_YOUR_SETTINGS/
auth_url=https://sso.redhat.com/auth/realms/ \
     redhat-external/protocol/openid-connect/token
token=ekT..3e

[galaxy_server.galaxy]
url=https://galaxy.ansible.com
```

Using the Token as an Environment Variable

- To avoid having the token listed in the ansible.cfg, you may want to set it as an environment variable
- Use **export ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN='ekT.. 3e'**
- Next, the `token` line can be removed from ansible.cfg

Lesson 9: Using Ansible Roles

9.8 Using the TimeSync RHEL System Role

Lesson 9: Using Ansible Roles

9.9 Using the SELinux RHEL System Role

Lesson 9: Using Ansible Roles

Lesson 9 Lab: Using Roles

Lesson 9 Lab: Using Roles

- Create a requirements file to install the geerlingguy.nginx and the geerlingguy.docker roles.
- Configure your roles path to store newly installed roles to the roles directory in the current project directory.
- Use the RHEL system role to configure time synchronization against the pool.ntp.org time servers.
- Create a custom role that generates an /etc/issue file. This file should print the message "Today is DATE, welcome to SYSTEM", where DATE and SYSTEM are using the current date and the Ansible hostname of the current system.

Lesson 9: Using Ansible Roles

Lesson 9 Lab Solution: Using Roles

Lesson 10: Troubleshooting Ansible

10.1 Ansible and Logging

Troubleshooting without Logs

- Relevant Ansible processing information is often written to STDOUT while using the **ansible-playbook** command
- When using **ansible-navigator**, add the option **-m stdout**
- In the output, start by reading the PLAY RECAP section, and if anything is wrong, scroll up to read relevant output
- To increase verbosity of the output of Ansible commands, use one to four **-v** options while running the command: **ansible-playbook -vvv myplay.yml**

Understanding Logging

- Ansible commands don't produce any logging; all output is written to STDOUT
- If you want to write output to a log file, use the `log_path:` setting to specify the name of the log directory to be used
  - Do not log to /var/log, as it requires **sudo** for write access
- **ansible-navigator** creates playbook artifact files in the directory that contains the playbook. These artifact files have detailed information containing all relevant information about the playbook execution
- The best way to use these artifact files is by accessing the logs from **ansible-navigator** in interactive mode

Managing **ansible-navigator** Artifacts

- Artifact files are generated for every playbook run
- This will fill up the project directory if the playbook was run multiple times
- Also, artifact files contain all information that was used, which may contain sensitive information
- To skip artifact creation, configure ansible-navigator.yml to contain the following:

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

Lesson 10: Troubleshooting Ansible

10.2 Using the debug Module

Using the debug Module

- The `debug` module is useful for analyzing variables
- While using this module, you can use the `verbosity` argument to specify when it should run:
    - Include `verbosity: 2` if you only want to run the `debug` module when the command was started with the **-vv** options

Pearson

demo

- ansible-playbook verbose_debug.yml
- ansible-playbook verbose_debug.yml -vv

Lesson 10: Troubleshooting Ansible

10.3 Checking Playbooks for Issues

Checking Playbooks for Issues

- To check a playbook for real errors, use the **--syntax-check** option with either the **ansible-playbook** or the **ansible-navigator** command
- Also consider avoiding errors by applying some best practices

Best Practices

- Use consistency in whatever you do
- Make sure every task and play has a name that describes what the play is doing
- Use consistent indentation
- In case of doubt, use the `debug` module to check for variable values
- Keep it simple
- Always use the most specific Ansible solution
- Avoid non-idempotent modules

Pearson

ansible-lint

- **ansible-lint** is an optional and not currently supported command
- Also, **ansible-lint** is not integrated in the default execution environment
- Use it to check playbooks to verify that best practices have been applied
- Notice that **ansible-lint** may give warnings about issues that aren't really an issue
- For instance: **ansible-lint** will complain if you don't use FQCN while referring to modules; this may be something you deliberately do differently

```
[ansible@control rhce8-live]$ ansible-lint knowthehost.yml
WARNING  Overriding detected file kind 'yaml' with
'playbook' for given positional argument: knowthehost.yml
WARNING  Listing 11 violation(s) that are fatal
fqcn-builtins: Use FQCN for builtin actions.
```

Lesson 10: Troubleshooting Ansible

10.4 Using Check Mode

Using Check Mode

- Use the **--check** option while running a playbook to perform check mode; this will show what would happen when running the playbook without actually changing anything
  - Modules in the playbook must support check mode
  - Check mode doesn't always work well in conditionals
- Set `check_mode: yes` within a task to always run that specific task in check mode
  - This is useful for checking individual tasks
  - When setting `check_mode: no` for a task, this task will *never* run in check mode and give you normal behavior (as if running without **--check**)

Pearson

Using Check Mode on Templates

- Add **--diff** to an Ansible playbook run to see differences that would be made by template files on a managed hosts
  - **ansible-playbook --check --diff myplaybook.yml**

```
$ ansible-playbook --check --diff vsftpd-template.yml
...
TASK [use template to copy FTP config]
************************************************************
--- before
+++ after: /home/ansible/.ansible/tmp/ansible-local-
49556z3m7a5wh/tmp72ew0his/vsftpd.j2
@@ -0,0 +1,11 @@
+# this file is owned by ansible
+anonymous_enable=True
+local_enable=True
```

Pearson

Lesson 10: Troubleshooting Ansible

10.5 Using Modules for Troubleshooting and Testing

Understanding Modules for Checking

- `uri`: checks content that is returned from a specific URL
- `script`: runs a script from the control node on the managed hosts
- `stat`: checks the status of files; use it to register a variable and then tests to determine if a file exists
- `assert`: this module will fail with an error if a specific condition is not met

Using **stat**

- The `stat` module can be used to check on file status
- It returns a dictionary that contains a stat field which can have multiple values
    - `atime`: last access time of the file
    - `isdir`: true if file is a directory
    - `exists`: true if file exists
    - `size`: size in bytes
    - and many more

demo

- bashversion.yml
- assertstat.yml

Lesson 10: Troubleshooting Ansible

10.6 Troubleshooting Connectivity Issues

Understanding Connection Issues

- Connection issues include the following
  - Issues setting up the physical connection
  - Issues running tasks as the target user

```
[ansible@control lesson9]$ ansible-playbook --check --diff vsftpd-
template.yml

TASK [Gathering Facts]
***************************************************************************
**
ok: [ansible1.example.com]
ok: [ansible2.example.com]
fatal: [ansible3]: UNREACHABLE! => {"changed": false, "msg": "Failed to
connect to the host via ssh: ssh: connect to host ansible3 port 22: No route
to host", "unreachable": true}
```

Analyzing Authentication Issues

- Confirm the `remote_user` setting and existence of remote user on the managed host
- Confirm host key setup
- Verify `become` and `become_user`
- Check that Linux **sudo** is configured correctly

Connecting to Managed Hosts

- When a host is available at different IP addresses / names, you can use `ansible_host` in inventory to specify how to connect
- This ensures that the connection is made in a persistent way, using the right interface
- `web.example.com ansible_host=192.168.4.100`

Using Ad-hoc Commands to Test Connectivity

- The `ping` module was developed to test connectivity
- Use the **--become** option to run with administrative privileges
  - **ansible ansible1 -m ping**
  - **ansible ansible1 -m ping --become**
- Use the `command` module to test different items
  - **ansible ansible1 -m command -a 'df'**
  - **ansible ansible1 -m command -a 'free -m'**

Pearson

Lesson 10: Troubleshooting Ansible

Lesson 10 Lab: Troubleshooting Playbooks

Lesson 10 Lab: Troubleshooting Playbooks

- Write a playbook that checks if network interface ens34 exists on ansible2.

Pearson

Lesson 11: Managing Software with Ansible

11.1 Managing Packages

Understanding Software Management Tasks

- To manage software on RHEL systems, different tasks need to be managed
- Systems need to be subscribed
- Repositories and software channels need to be configured
- Software packages need to be installed, updated, and removed

Managing Software Packages

- Different modules are available for managing software packages
  - ansible.builtin.dnf: used on recent Red Hat and family and backwards compatible with the ansible.builtin.yum module
  - ansible.builtin.yum: used on older Red Hat and family
  - ansible.builtin.apt: used to manage packages on Ubuntu and family
  - ansible.builtin.package: a generic module that can install and remove packages on different Linux distributions
  - ansible.windows.win_package: manages packages on Windows
- While managing packages, the recommendation to use the most specific module applies
- Also realize that package names are not always the same between different distributions

Managing Package Groups

- The Linux **dnf** command supports working with package groups
- Use **dnf group list** to show a list of available groups
- To install a package group, put a @ in front of the group name:

```
- name: install virtualization software
  ansible.builtin.dnf:
    name: '@Virtualization Host'
    state: present
```

Pearson

Managing Package Modules

- The **dnf** command also supports working with package modules
- Use **dnf module list** for a list of available modules
- To install a module,put a @ in front of its name, optionally followed by the module stream version and profile

```
- name: install the php module
  ansible.builtin.dnf
    name: 'php:7.3/minimal'
    state: latest
```

Gathering Package Facts

- Facts about packages are not gathered by the ansible.builtin.setup module
- To gather facts about packages, use the ansible.builtin.package_facts module
- When gathered, package facts are written to the ansible_facts['packages'] variable
- As packages are stored in an array, while addressing the package you need to address the correct array index value, which is [0] in many cases

Pearson

Demo

- packagefacts.yml

Lesson 11: Managing Software with Ansible

11.2 Managing Repositories and Repository Access

Accessing Repositories

- To access repositories, the ansible.builtin.yum_repository module is used
- This module creates a repository file in the /etc/yum.repos.d/ directory
- If the module argument gpgcheck: yes is used, the ansible.builtin.rpm_key module must be used to install the GPG key

Demo

- exercise122-server
- exercise122-client

Lesson 11: Managing Software with Ansible

11.3 Managing Subscriptions

Managing RHEL Subscriptions

- To work with RHEL, you need to use your subscription
- Free RHEL subscriptions are available through https://developers.redhat.com
- To register a RHEL server from the Linux command line, you would use the following:
  - **subscription-manager register --username=yourname --password=password**
  - **subscription-manager attach**
  - **subscription-manager repos list**
  - **subscription-manager repos --enable "repo name"**

Using Modules to Manage Subscriptions

- The redhat_subscription module enables you to perform subscription and registration in one task
- The rhsm_repository module is used to enable subscription manager repositories

demo

- subscription.yml

Lesson 11: Managing Software with Ansible

Lesson 11 Lab: Managing Repositories

Lesson 11 Lab: Managing Repositories

- Set up a repository on control. This repository should offer multiple packages, including the nmap package.
- Provide a package list using variables.
- Configure ansible1 and ansible2 to use the repository that is provided through this repository.
- Install the nmap package from this repository.

Pearson

Lesson 12: Managing Users

12.1 Understanding User-related Tasks

Understanding User Management

- Most of the user oriented tasks can be performed with the ansible.builtin.user module
- If additional groups need to be created, use ansible.builtin.group before running the user module
- The user module can also take care of less common tasks, like generation of ssh keys using the generate_ssh_key: boolean

Creating Users

- The ansible.builtin.user module contains all that is needed to manage basic user properties
- If a user needs to be a member of additional (secondary) groups, ensure these groups exist before using the user module
- While adding users to secondary groups, use the append: true option
- Ansible can also be used to manage SSH keys and create sudo configuration
- For setting passwords, you'll have to ensure that encrypted passwords are generated before providing them to the user module

Pearson

User Management Modules Overview

- ansible.builtin.user: manages core user properties
- ansible.builtin.group: creates and manages groups
- ansible.builtin.known_hosts: updates the /etc/ssh/ssh_known_hosts file with the host key of a managed host
- ansible.builtin.authorized_key: manages authorized keys for user accounts on managed hosts
- ansible.builtin.lineinfile: used to configure sudo access by adding a line to a configuration file

Lesson 12: Managing Users

12.2 Managing SSH Keys

Managing SSH Keys

- The `ansible.builtin.user` module can use the `generate_ssh_key` argument to generate an SSH key for a user on a managed host
- The `ansible.builtin.known_host` module copies host keys from managed hosts
  - This ensures that users are not prompted to verify the remote host SSH key fingerprint before connecting to it
- The `ansible.builtin.authorized_keys` module can be used to copy a control host user public key to the corresponding user account on a managed host
  - To use it, the public key must be in a public location, where it is readable: if it is in a hidden directory in the user home directory it cannot be used

demo

- userkeys.yml

Lesson 12: Managing Users

12.3 Managing Encrypted Passwords

Understanding Password Encryption

- On Linux, the hash of the encrypted user password is stored in /etc/shadow
- This hash looks like $6$23879879879/$927395798793847t982798789237
- It consists of 3 parts:
  - The hashing algorithm that is used
  - The random salt that was used to encrypt the password
  - The encrypted hash of the user password
- To create an encrypted password, a random salt is used to ensure that two users that have identical passwords would not have identical entries in /etc/shadow
- This salt and the unencrypted password are combined and encrypted, which generates the encrypted hash that is stored in /etc/shadow

Creating Encrypted Passwords in Ansible

- The ansible.builtin.user module does not generate encrypted passwords
- To generate an encrypted password, an external utility must be used to generate an encrypted string
- Next, the encrypted string can be used in a variable to set the password
- For enhanced security, store the password hash in a vault encrypted file
- One method is to use the password_hash filter to generate an encrypted password
- Another option would be to use the shell module to run **passwd --stdout** and capture the result of that command in a variable using register

Demo

- Check Ansible Documentation FAQ: How do I generate encrypted passwwords for the user module
- **ansible localhost –m debug –a "msg={{ 'password' | password_hash('sha512', 'mypassword') }}"**
- **ansible-playbook userpw.yml**
- **ssh anna@ansible2**
- **shell: echo {{ password }} | passwd --stdin {{ user }} register**

Lesson 12: Managing Users

12.4 Managing Sudo Privileges

Managing Sudo

- Ansible doesn't offer a specific module for managing **sudo** privileges
- Use generic modules instead:
  - ansible.builtin.lineinfile can be used to manage the /etc/sudoers.d/whatever file
  - ansible.builtin.template can be used with a Jinja2 template to generate this file

Lesson 12: Managing Users

Lesson 12 Lab: Managing Users

Lesson 12 Lab: Managing Users

- Create a playbook that creates new users interactively
- The playbook should read the users from the "usernames" variable which may be set in the playbook
- The playbook should prompt for a password, without the text that the user enters being visible
- Ensure that the password is stored as an encrypted password

Lesson 13: Managing Processes and Tasks

13.1 Managing Services and Targets

Managing Services and Targets

- ansible.builtin.service provides a generic interface used to manage the state of services in different service management systems
- ansible.builtin.systemd manages the state of services in systemd, as well as additional systemd specific properties
- There are no modules to manage the default target, use ansible.builtin.command instead
- ansible.builtin.reboot will reboot a managed host. It can use a reboot_timeout and a test_command to verify that the host is available again

demo

- settargetandreboot.yml

Lesson 13: Managing Processes and Tasks

13.2 Scheduling Processes

Scheduling Processes

- ansible.posix.at is used to run a one-time job at a future time
- ansible.builtin.cron is used to run repeating jobs through the Linux cron daemon

demo

- setup-crontab.yml; delete-crontab.yml
  - show /etc/cron.d on managed host
  - delete-crontab will fail, it requires user=ansible
- setup at-task.yml

Lesson 13: Managing Processes and Tasks

Lesson 13 Lab: Managing the Default Target

Lesson 13 Lab: Managing the Default Target

- Set the default boot state of the managed servers to multi-user.target.
- Reboot your server after doing so.
- Configure your playbook such that it will show the message "successfully rebooted" once it is available again.

Lesson 14: Managing Storage

14.1 Managing Partitions and Mounts

Solutions for Managing Storage

- `ansible.posix.mount` is used to mount existing filesystems
- `community.general.parted` is used to manage partitions
- `community.general.lvg` manages volume groups
- `community.general.lvol` manages logical volumes
- `community.general.filesystem` can be used to create filesystems on the new devices
- Notice that the community.general content collection is unsupported. Use the `redhat.rhel_system_roles.storage` role if you want to use a supported solution

Pearson

demo

- create_partition.yml
- get_diskname.yml

Lesson 14: Managing Storage

14.2 Managing Logical Volumes

Managing Logical Volumes

- Currently, the community.general modules are not supported
- To manage storage devices in a supported way, either use ansible.builtin.command, or redhat.rhel_system_roles.storage (which both are not ideal)

Exam Tip! "Not supported" is not an issue on the exam. You just have to create a working solution and it's OK doing that with unsupported modules

Using redhat.rhel_system_roles.storage

- The storage role only supports the following
  - unpartitioned devices, where it creates a file system on the whole disk device (which is a bad idea)
  - LVM on whole devices

Creating LVM with the Storage Role

- To use the RHEL system role to create LVM, the storage_pools variable must be defined
- Within storage_pools you define the LVM environment, using the following variables
  - name: the name of the VG
  - disks: the complete disk to use
  - volumes: the LVM logical volumes and their properties

Tip! If installed using the rhel-system-roles.rpm package, you'll find a README.md with an example in /usr/share/doc/rhel-system-roles/storage.

Pearson

Lesson 14: Managing Storage

14.3 Developing Advanced Playbooks

Developing Advanced Playbooks

To develop advanced Ansible (exam lab) solutions, it's recommeded to use a phased approach

- First, set up a generic framework that shows how you are going to take care of any conditional task execution. In this phase don't use specific modules but put as much as you can in the debug module and focus on structure.
- Next, work out the conditionals and facts you may want to check. Still no details about specific modules but check the facts using the debug module.
- Finally, work out module specifics and produce the working solution. In this phase, consider using tags to allow you to test specific parts only.

Lesson 14: Managing Storage

Lesson 14 Lab: Managing Storage

Lesson 14 Lab: Managing Storage

- Tasks in this playbook should only be executed on hosts where a second hard disk is installed.
- If no second hard disk exists, the playbook should print "second hard disk not present" and stop executing tasks on that host.
- Configure the device with one partition including all available disk space
- Create an LVM volume group with the name vgfiles.
- If the volume group is bigger than 5 GB, create an LVM logical volume with the name lvfiles and a size of 5 GB. Note that you must check the LVM Volume Group size and not the /dev/sdb size, because in theory you could have multiple block devices in a volume group.
- If the volume group is equal to or smaller than 5 GB, create an LVM logical volume with the name lvfiles and a size of 3 GB.
- Format the volume with the XFS filesystem.
- Mount it on the /files directory.

Lesson 15: Managing Networking

15.1 Using Roles for Network Management

Using the Network System Role

- The `redhat.rhel_system_roles.network` system role allows for the configuration of network related settings
- After installing the `rhel-system-roles.rpm` package, many examples are provided in the /usr/share/doc/rhel-system-roles directory
- Based on these examples, many different network interface types can be configured
- To configure the role, the `network_provider` and the `network_connections` variables must be set
  - `network_provider` should be set to nm on RHEL 7 and later
  - `network_connection` defines the network connection and its properties

demo

- show contents of networkvars.yml
- Explain zone: external which is the firewalld zone to use
- Compare these variables to the defaults in /usr/share/ansible/roles
- show setupnw.yml

Pearson

Lesson 15: Managing Networking

15.2 Managing Network Settings with Facts and Modules

Using Network Related Modules

- `ansible.posix.firewalld` allows you to create rules for firewalld
- `ansible.builtin.hostname` allows for setting the hostname
- These modules can be used with ansible facts as stored in `ansible_facts['interfaces']`

demo

- **ansible all -m setup -a 'gather_subset=network filter=ansible_interfaces'** will show current interfaces
- **ansible all -m setup -a 'gather_subset=network filter=ansible_ens33'** will show info for that specific interface only
- **ansible-playbook setup_nic.yml**

Lesson 15: Managing Networking

Lesson 15 Lab: Managing Networking

Lesson 15 Lab: Managing Networking

- This lab requires ansible2.example.com to be configured with a second network interface.
- Use ansible fact filters to find the name of the second network interface on ansible2.example.com.
- Configure a playbook that sets up the IPv6 address fc00::202/64 on the second interface on ansible2.example.com.

Lesson 16: Sample Exam

16.1 Before you Start: Essential Exam Tips

Essential Exam Tips

- All Ansible documentation is available on the exam. Make sure you practice finding the appropriate documentation; it sometimes is well hidden.
- You don't have to use **ansible-navigator**. All assignments can also be done through **ansible-playbook** and other older tools.
- According to Red Hat exam philosophy, it doesn't matter how you do it, as long as you're doing the right thing.
- Avoid non-idempotent behavior as much as you can.
- You think better behind your coffee machine. During the exam you may have 3 breaks, use them!
- Read the "essential information" section very carefully; it has essential information.

Essential Exam Tips

Make sure you are very comfortable with the following:

- Using roles, collections, and requirements files
- Using conditionals
- Using block and rescue

All tasks expect you to create a working solution. If needed, install packages, create users, start services, and more to ensure the playbook solution will work.

Lesson 16: Sample Exam

16.2 Setting up the Environment

Task 1: Setting up the Environment

To work through the assignments in this exam, you need the following:

- 3 virtual machines running either RHEL 9 or CentOS Stream:
  - A control host
  - A node1 host
  - A node2 host
- Add a second disk to node1, not node2.
- Install with GUI on the control host and use the minimal installation pattern on the other nodes.
- If running RHEL 9, make sure your control host is (manually) registered using subscription manager.
- Ensure that the httpd package is copied to the control host while installing (or after installation).

Task 1: Setting up the Environment

Make sure your virtual machines meet the following hardware requirements:

- control
  - 20GB disk space
  - 2GB RAM
  - 2 vCPUs
- node1 and node2
  - 10 GB disk space
  - 1GB RAM
  - 1 vCPU
  - A second 10 GB disk on node2

Lesson 16: Sample Exam

16.3 Configuring the Control Node

Task 2: Configuring the Control Node

- Install the Ansible software on the control node.
- Create a user ansible on all nodes, and ensure that:
  - User ansible has sudo privileges on all nodes
  - Use ansible can remote login using SSH keys

Task 2: Configuring the Control Node

Create an inventory file that meets the following requirements:

- A host group with the name dev is created, and node1 is a member of this group
- A host group with the name prod is created, and node2 is a member of this group
- A host group with the name servers is created, and has the groups prod and dev as its members

Task 2: Configuring the Control Node

In control host user ansible homedirectory, create an ansible.cfg that meets the following requirements:

- It refers to the inventory file in the current directory.
- privilege escalation is defined. The mechanism is sudo, no passwords should be asked.
- The default location for collections is the directory collections in the user ansible home directory.
- The default location for roles is the directory roles in the user ansible home directory.

Pearson

Lesson 16: Sample Exam

16.4 Setting up a Repository Server

Task 3: Setting up a Repository Server

- Configure the control node to host the contents of the RHEL 9 installation disk as repositories.
- Ensure the repository content is copied to /reposerver/BaseOS and /reposerver/AppStream.
- Provide the contents of these two directories as symbolic links in the Apache server documentroot (/var/www/html).
- Ensure that the Apache server is installed and started automatically, and is available through the firewall.

Lesson 16: Sample Exam

16.5 Setting up Repository Clients

Task 4: Setting up Repository Clients

- Configure all nodes as repository clients to the repository server that was configured in the previous task

Lesson 16: Sample Exam

16.6 Installing Collections

Task 5: Installing Collections

- Use a requirements.txt file to install the following collections:
  - community.general
  - ansible.posix

```
- src: https://github.com/myaccount/myrole.role
  scm: git
  version: "2.0"
- src: file:///tmp/myrole.tar
  name: mytarrole
- src: https://example.local/myrole.tar
  name: mywebrole
- community.general # no SRC if it comes from ansible
galaxy!
```

Task 5: Installing Collections

- Use a requirements.yml file to install the following collections:
  - community.general
  - ansible.posix

Lesson 16: Sample Exam

16.13 Generating a Report

Task 12: Generating a Report

- Use the file report.txt from the course git repository at https://github.com/sandervanvugt/rhce9.
- Copy this file to /tmp/HOSTNAME-report.txt on the managed host, where HOSTNAME is replaced with the name of that host
- Modify the contents of this file such that it lists the hardware for each of the managed nodes; make sure to use the existing file
- If a disk /dev/sdb was found, the line SECOND_DISK should read SECOND_DISK=present, if it was not found it should read SECOND_DISK=absent

Lesson 16: Sample Exam

16.8 Creating a Vault Encrypted File

Task 7: Creating a Vault Encrypted File

- Encrypt the file task7file.yml with ansible vault, using the password "mypassword"
- Store this password in the file vaultpass.txt in the current project directory.
- Ensure that the decrypted contents of the file can be shown using a vault password file

Task 7: Creating a Vault Encrypted File

- Store this password in the file vaultpass.txt in the current project directory.
- Ensure that the decrypted contents of the file can be shown using a vault password file.

Lesson 16: Sample Exam

16.9 Creating Users

Task 8: Creating Users

- Use the vault encrypted password file from the previous task to include variables in your playbook
- Create users that are a member of the group profs on node1, and users that are a member of the group students on node2.
- Set these groups as the secondary groups for the users.
- Use the vault encrypted password file that you created in the previous task to set SHA512 encrypted passwords for these users.

Lesson 16: Sample Exam

16.10 Creating a Role

Task 9: Creating a Role

- Create a custom role with the name motdrole that generates an /etc/motd file.
- This file should show the following line after logging in:
  - Welcome on SERVERNAME. This file was generated on DATE
- Replace SERVERNAME with the host on which the file is created, and DATE with the current date.
- Create a playbook with the name makemotd that uses this role.

Pearson

Lesson 16: Sample Exam

16.12 Creating a Logical Volume

Task 11: Creating a Logical Volume

- Write the playbook createlv.yml that creates a logical volume with the name lvdata, meeting the following requirements:
  - If a managed host doesn't contain a second hard disk, the playbook prints "no second harddisk available" and exits
  - If a second hard disk does exist, the playbook uses all of its disk space to create a volume group with the name vgdata
  - The playbook tries to create the logical volume with a size of 30 GB. If that doesn't work, it should print the message "insufficient disk space for large LV" and create the logical volume with a size of 1 GB.
  - Format the logical volume with the XFS filesystem

Lesson 16: Sample Exam

16.11 Creating a Cron Job

Task 10: Creating a Cron Job

- Create a Cron job that runs as user marcha and runs the command **touch /tmp/marchafile** on node1 every 2 minutes.

Lesson 16: Sample Exam

16.7 Generating an /etc/hosts File

Task 6: Generating an /etc/hosts File

- Use an automated solution to create the contents of the /etc/hosts file on all managed hosts based on information that was found from the Inventory.

Task 6: Generating an /etc/hosts File

- Use an automated solution to create the contents of the /etc/hosts file on all managed hosts based on information that was found from the Inventory.

```
{% for host in groups['all'] %}
{{
hostvars[host]['ansible_facts']['default_ipv4']['address']
}} {{ hostvars[host]['ansible_facts']['fqdn'] }} {{
hostvars[host]['ansible_facts']['hostname'] }}
{% endfor %}
```