Welcome to the Linux Academy Red Hat Certified Engineer (RHEL 8 RHCE) course.

This course is designed to prepare you to sit and pass the Red Hat Certified Engineer exam (EX294).

As of the creation of this course, there are two versions of the Red Hat Certified Engineer exam - one for Red Hat Enterprise Linux 7 (EX300) and one for Red Hat Enterprise Linux 8 (EX294). This course was created based on the objectives of the RHEL 8 version of the exam which is EX294.

This diagram will be used as a reference point throughout the course and can be used as a study guide as you prepare for the exam.

This course was split up, Rob creating all of the labs, and me teaching the lessons. We thank you for taking this course and look forward to working through the material with you!

Matthew Pearson
Rob Marti

Linux Academy Training Architects
Exam Format

- It is a hands-on exam that requires you to perform real-world tasks.
- You will have four hours to complete the exam.
- You will be given multiple systems and must install and configure Ansible in order to perform system administration tasks.
- Your work will be evaluated by running the playbooks created in the exam against fresh systems.
- Internet access will not be provided and candidates are not allowed to bring physical or electronic documentation or notes.
- Exam results are usually reported within 3 days.

The exam objectives can be viewed here:
Topics in this section include:

Understanding and using the basic tools for an operating system is essential to administering that system. In this section, we will review these tools and show examples of how to use these on a Red Hat Enterprise Linux 8 host.

- Log into a Remote Server via SSH
- Create Files and Directories
- Input/Output Redirection
- View and Analyze Text
- Archive Files and Directories
- Escalate Privileges
- File and Directory Permissions
- System Documentation
• Log into remote server: ssh user_name@host
• Log out of host: exit
Basic Red Hat Certified Administrator Skills

Understand and Use Essential Tools

- **Create directory**: `mkdir <directory_name>`
- **Create file**: `touch file` or `vi file`
- **Remove directory**: `rm -r directory` or `rmdir` (for empty directory)
- **Remove file**: `rm file`

Back
Understand and Use Essential Tools

- Standard output (STDOUT): >, >>
- Standard input (STDIN): <, <<
- Standard error (STDOUT): 2>, 2>>
- Pipes: |
Understand and Use Essential Tools

- View with text editor: `vi file`
- Print file contents to STDOUT: `cat`
- Print lines matching a pattern: `grep`
- Compression options: gzip, bzip, xz, etc.
- Create archive:
  `tar -cvzf archive_name file1 file2`
- Extract archive:
  `tar -xvzf archive.gz`
Understand and Use Essential Tools

- **Become the root user:** `sudo -i` or `sudo su -`
- **Run command as root user:** `sudo command`
Understand and Use Essential Tools

- **Change ownership**: `chown user:group file`
- **Change permissions (numeric)**: `chmod 764 file`
- **Change permission (symbolic)**: `chmod u+rw file`
- **Special permissions**: `setuid(4 or u+s), setgid(2 or g+s), sticky bit(1 or a+t)`

Back
Basic Red Hat Certified Administrator Skills

Understand and Use Essential Tools

- **man pages**: `man command` or `man section command`
- **info**: `info command`
- `/usr/share/doc`
- **apropos**: `apropos command` or `man -k command`

Back
Basic Red Hat Certified Administrator Skills

Operate Running Systems

Shut down and Reboot Systems

- systemctl poweroff
- systemctl reboot
- systemctl --help | man systemctl

Interrupt Boot Process and Change the root Password

1. **Edit** the kernel boot parameters by pressing `e`
2. **Go to** the end of the “linux” line by pressing `Ctrl+e`, removing `ro crash` and `add rd.break enforcing=0`
3. **Start** the system by pressing `Ctrl+x`
4. **Remount** the root of the system:
   ```
   mount -o remount,rw /sysroot
   ```
5. **Switch** to `/sysroot`:
   ```
   chroot /sysroot
   ```
6. **Reset** the root password:
   ```
   passwd
   ```
7. **Enable** SELinux relabeling:
   ```
   touch ~/.autorelabel
   ```
8. **Exit** the shell:
   ```
   exit
   ```
Start, Stop, and Check Network Service Status

- Check the status of a service:
  ```bash
  systemctl status service_name.service
  ```
- Start a service:
  ```bash
  systemctl start service_name.service
  ```
- Stop a service:
  ```bash
  systemctl status service_name.service
  ```
- Gain more information:
  ```bash
  journalctl -xe
  ```

View Processes and Resource Utilization

- Print a list of active process:
  ```bash
  ps -ef
  ```
- View real-time list of processes and resource utilization:
  ```bash
  top
  ```
- Terminate a running process:
  ```bash
  kill -15, kill -9, kill -l
  ```

Copy Files Between Remote Systems

- Secure Copy:
  ```bash
  scp file_name user_name@server:/path/to/dir
  ```
- Secure FTP:
  ```bash
  sftp user_name@server
  ```
List Storage Devices:

- `df`
- `lsblk`
- `blkid`
- `fdisk -l`

Create a Partition:

1. Use `fdisk` to manipulate partition table:
   
   ```
   fdisk /dev/device_name
   ```
2. Use `p` to print the partition table and `o` to create a DOS (MBR) partition table.
3. Use `n` to create a new partition.
4. Set it as a primary partition using `p` and accept the defaults for partition number, first sector, and last sector.
5. List partition types using `l`. Change the partition type to Linux LVM (8e) using `t`.
6. Write the table to disk using `w`.
Basic Red Hat Certified Administrator Skills

Configure Local Storage

Create an LVM Logical Volume:
1. Create physical volume:
   `pvcreate /dev/device_name`
2. List physical volumes:
   `pvs`
3. Create volume group:
   `vgcreate vol_group /dev/device_name`
4. List volume groups:
   `vgs`
5. Create logical volume:
   `lvcreate -L 1G -n new_lv vol_group`
6. List logical volumes:
   `lvs`

Delete a Logical Volume, Volume Group, and Physical Volume:
1. Delete logical volume:
   `lvremove vol_group/new_lv`
2. Delete volume group:
   `vgremove vol_group`
3. Delete physical volume:
   `pvremove /dev/device_name`
Creating and Mounting File Systems

1. Create a file system on a logical volume:
   `mkfs.ext4 /path/to/lv`

2. Mount a file system:
   `mount /path/to/lv /path/to/dir`

3. List mounted file systems: `df -h`

4. Add mount information to `/etc/fstab` for boot persistence:
   
   ```
   UUID=UUID_NUMBER /mount/point
   fs_type defaults 0 0
   ```

Extend Logical Volumes

1. Increase underlying logical volume:
   `lvextend -L +500M /dev/vol_grp/log_vol`

2. Unmount filesystem: `umount /path/to/mount`

3. Run file system check:
   `e2fsck -f /dev/vol_grp/log_vol`

4. Resize file system:
   `resize2fs /dev/vol_grp/log_vol`

5. List mounted file systems: `df -h`
Basic Red Hat Certified Administrator Skills

Mount a Network File Systems
1. Install required packages:
   yum install nfs-utils
2. Start required services:
   systemctl start rpcbind
3. Show file system exports on the client:
   showmount -e SERVER_IP
4. Mount a network file system:
   mount -t nfs
   SERVER_IP:/server/dir /client/dir

Create Collaborative Directories with set-GID
1. Create directory:
   mkdir /new/dir
2. Configure set-GID on directory:
   chmod g+s /new/dir
3. Create file in new directory:
   touch /new/dir/newFile

Working with Virtual Data Optimizer (VDO)
1. Install required packages:
   yum install vdo
2. Create a vdo volume:
   vdo create
   --name=vdo_vol --device=/dev/devName
   --vdoLogicalSize=vol_size
3. View information on vdo volumes:
   vdostats --hu

Back to Main
Start, Stop, and Enable Services

- View yum repositories:
  `ls -a /etc/yum.repos.d`
- Install packages:
  `yum install packageName`
- Start a service:
  `systemctl start name.service`
- Enable a service:
  `systemctl enable name.service`
- Stop a service:
  `systemctl stop name.service`

Schedule Tasks Using `at` and `cron`

- View the crontab:
  `cat /etc/crontab`
- Add a task to a user crontab:
  `crontab -e`
- List tasks in a user's crontab:
  `crontab -l`
- Schedule a task using the `at` command:
  `at time`
- List the scheduled jobs:
  `atq`
- Delete a job:
  `atrm job_num`
Configure Systems to Boot into Specific Target Automatically

- Check the current configuration:
  
  ```sh
  systemctl get-default
  ```
- Change the current target:
  
  ```sh
  systemctl isolate name.target
  ```
- Set the default configuration:
  
  ```sh
  systemctl set-default name.target
  ```
- Change target to rescue mode:
  
  ```sh
  systemctl rescue
  ```

Configure Time Service Client

1. Install required packages:
   
   ```sh
   yum install chrony
   ```
2. Start and enable the chronyd service:
   
   ```sh
   systemctl start chronyd && systemctl enable chronyd
   ```
3. Add the NTP server (server SERVER_IP) address to:
   
   ```sh
   /etc/chrony: vi /etc/chrony
   ```
4. Restart the chronyd service:
   
   ```sh
   systemctl restart chronyd
   ```
Create, Delete, and Modify Local Users

- View user information:
  id username
  groups username
  /etc/passwd
  /etc/shadow
  /etc/group

- Create a user: useradd username
- Modify a user: usermod -d -aG -l -U
- Delete a user: userdel username

Change Passwords

- Change a password: passwd username
- View password expiry information:
  chage -l username
- Set password expiration by max days:
  chage -M days username
- Set password expiration by date:
  chage -E YYYY-MM-DD username
Create, Delete, and Modify Groups

- View group information:
  
  id username
groups username
/etc/passwd
/etc/shadow
/etc/group

- Create a group: groupadd groupname
- Add user to a group: usermod -g -aG
- Modify a group: groupmod -n -g
- Delete a group: groupdel groupname

Configure Superuser Access

- View the sudoers file: vi /etc/sudoers
- Edit the sudoers file: visudo
- Grant a user sudo access by adding the following line to /etc/sudoers:
  
  username    ALL=(ALL)    ALL

- Grant members of a group sudo access by adding the following line to /etc/sudoers:
  
  %groupname   ALL=(ALL)    ALL
Configure Firewall Settings

- Install firewall: `yum install firewalld`
- Start and enable firewalld:
  
  ```
  systemctl start firewalld &&
  systemctl enable firewalld
  ```
- View firewall-cmd options:
  
  ```
  firewall-cmd --help | man firewall-cmd
  ```
- List zones: `firewall-cmd --get-zones`  
  ```
  --get-default-zone
  ```
- List everything added for or enabled in a zone:
  
  ```
  firewall-cmd --list-all --zone=public
  ```
- Add a service for a zone:
  
  ```
  firewall-cmd --add-service=service
  ```
  ```
  --permanent
  ```
- Add a port for a zone:
  
  ```
  firewall-cmd --add-port=port/protocol
  ```
  ```
  --permanent
  ```
- Reload firewall rules: `firewall-cmd --reload`

Configure Key Based Authentication for SSH

- Generate public and private key pair: `ssh-keygen`
- Copy a public key to a remote server:
  
  ```
  ssh-copy-id username@remote_host
  ```
- Default public/private key location:
  
  ```
  /home/username/.ssh/
  ```
Working with SELinux

- View SELinux modes: getenforce
- Set mode to permissive or enforcing:
  ```bash
  setenforce 0 | 1
  ```
- List booleans: getsebool -a
- Turn booleans on or off:
  ```bash
  setsebool boolean on | off
  ```
  (P for permanent)
- List SELinux contexts: semanage fcontext -l
- View context on files and process:
  ```bash
  ls -Z | ps -axZ
  ```
- Change SELinux context:
  ```bash
  semanage fcontext -a -t context_type '/directory(/.*)'?
  ```
- Restore default contexts:
  ```bash
  restorecon -R /directory
  ```
- View SELinux policy violations:
  ```bash
  sealert -a /var/log/audit/audit.log
  ```
Inventories are what Ansible uses to locate and run against multiple hosts.

- Default location of the hosts file: /etc/ansible/hosts
- The default location of the hosts file can be set in /etc/ansible/ansible.cfg.
- It can be specified using the `-i` option when running ansible.
- The file can contain individual hosts, groups of hosts, groups of groups, and host and group level variables.
- It can also contain variables that determine how you connect to a host.

INI-based inventory file:

```
[webservers]
web01.example.com
web02.example.com

[dbservers]
db[01:04].example.com
```
YAML-based inventory file:

```yml
all:
  hosts:
    mail.example.com
  children:
    webservers:
      hosts:
        web01.example.com
        web02.example.com
    dbservers:
      hosts:
        db[01:04].example.com
```
Understand Core Components of Ansible

**Understanding Modules**

- Modules are essentially tools for particular tasks.
- Modules can take, and usually do take, parameters.
- Modules return JSON.
- Run modules from the command line or within a playbook.
- Ansible ships with a significant amount of modules by default.
- Custom modules can be written.
Understanding Variables in Ansible

- Variables names should only contain letters, numbers, and underscores.
- Variables should always start with a letter.
- There are three main scopes for variables:
  - Global
  - Host
  - Play

- They are typically used for configuration values and various parameters.
- Variables can store the return value of executed commands.
- Variables may also be dictionaries.
- Ansible provides a number of predefined variables.
Example of Host Level Variables:

INI format:

```
[webservers]
host1 http_port=80 maxRequestsPerChild=500
host2 http_port=305 maxRequestsPerChild=600
```

YAML format:

```
webservers:
  host1:
    http_port: 80
    maxRequestsPerChild: 500
  host2:
    http_port: 305
    maxRequestsPerChild: 600
```
Understand Core Components of Ansible

Understanding Ansible Facts

- Facts provide certain information about a given target host.
- Facts are automatically discovered by Ansible when it reaches out to a host.
- Facts can be disabled.
- Facts can be cached for use in playbook executions.

```
[cloud_user@mspearson4c ansible]$ ansible --inventory inv.ini mspearson2c -m setup
mspearson2c | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.31.101.166",
            "192.168.122.1"
        ],
        "ansible_all_ipv6_addresses": [
            "2600:1f18:502:2f01:a37b:b7b5:61a6:3659",
            "fe80::835:c7ff:fe1d:f35e"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "10/16/2017",
        "ansible_bios_version": "1.0",
        "ansible_cmdline": {
            "BOOT_IMAGE": 
            "(hd0,msdos2)/boot/vmlinuz-4.18.0-80.7.2.el8_0.x86_64",
            "console": "tty0",
        }
    }
```
Understanding Plays and Playbooks

- The goal of a play is to map a group of hosts to some well-defined roles.
- A play can consist of one or more tasks which make calls to Ansible modules.
- A playbook is a series of plays.

Example of an Ansible Playbook:

```yaml
---
- hosts: webbservers
  become: yes
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: write our custom apache config file
      template:
        src: /srv/httpd.j2
        dest: /etc/httpd/conf/httpd.conf
    - name: ensure that apache is started
      service:
        name: httpd
        state: started
    - hosts: dbbservers
      become: yes
      tasks:
        - name: ensure postgresql is at the latest version
          yum:
            name: postgresql
            state: latest
        - name: ensure that postgresql is started
          service:
            name: postgresql
            state: started
```
Understand Core Components of Ansible

The Ansible Configuration File

- Possible locations of Ansible configuration files (in order processed):
  - ANSIBLE_CONFIG (environment variable)
  - ansible.cfg (in the current directory)
  - ~/.ansible.cfg (in the home directory)
  - /etc/ansible/ansible.cfg
- A configuration file will not automatically load if it is in a world-writable directory.
- Configuration can be set in environment variables.

Common Ansible Configurations

- The ansible-config command can be used to view configurations:
  - list - Prints all configuration options
  - dump - Dumps configuration
  - view - View the configuration file
- Commonly used settings:
  - inventory - Specifies the default inventory file
  - roles_path - Sets paths to search in for roles
  - forks - Specifies the amount of hosts configured by Ansible at the same time (Parallelism):
  - ansible_managed - Text inserted into templates which indicate that file is managed by Ansible and changes will be overwritten.
Install and Configure an Ansible Control Node

Install Ansible Using YUM

- sudo subscription-manager repos --enable ansible-2.8-for-rhel-8-x86_64-rpms
  (if needed)

- sudo yum install ansible

Install Ansible from Source

1. sudo yum install git
2. git clone --single-branch
   --branch stable-2.8
   https://github.com/ansible/ansible.git
3. cd ansible/
4. source ./hacking/env-setup
5. pip2.7 install --user -r
   ./requirements.txt

Test the installation:
ansible 127.0.0.1 -m ping
Install and Configure an Ansible Control Node

Create a Static Host Inventory File

A Inventory Files

Ansible Inventory

An inventory is a list of hosts that Ansible manages.

- Inventory files may contain hosts, patterns, groups and variables.
- Multiple inventory files may be specified using a directory.
- Inventory files may be specified in INI or YAML format.

Inventory Locations:

Default:
/etc/ansible/hosts

Specified by CLI:
ansible -i <filename>

Can be set in:
ansible.cfg

Back

Next
## Example Inventory Files

**INI-based inventory file:**

```ini
[web servers]
web01.example.com
web02.example.com

[web servers:vars]
http_port=8080

[db servers]
db[01:99].example.com
```

### Inventory Variables Best Practices

1. Variables should be stored in YAML files located relative to the inventory file.
2. Host and group variables should be stored in the `host_vars` and `group_vars` directories respectively (directories must be created).
3. Variable files should be named after the host or group for which they contain variables (files may end in `.yml` or `.yaml`).
Example

Inventory Files

---

all:
  hosts:
    mail.example.com
    ansible_port: 5556
    ansible_port: 192.168.0.20
  children:
    webservers:
      hosts:
        web01.example.com
        web02.example.com
      vars:
        http_port: 8080
    dbbservers:
      hosts:
        db[01:99].example.com
Create a Static Host Inventory File

**Groups of**

**INI-based inventory file:**

```
[east]
host1
host2

[west]
host3
host4

[usa:children]
east
west
```

**YAML-based inventory file:**

```
all:
  children:
    usa:
      children:
        east:
          hosts:
            host1:
            host2:
        west:
          hosts:
            host3:
            host4:
```
Default ansible.cfg

# config file for ansible -- https://ansible.com/
# ---------------------------
# nearly all parameters can be overridden in
# ansible-playbook
# or with command line flags. ansible will read
ANSIBLE_CONFIG,
# ansible.cfg in the current working directory,
./ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg,
whichever it
# finds first
[defaults]
# some basic default values...
#inventory = /etc/ansible/hosts
#library = /usr/share/my_modules/
#module_utils = /usr/share/my_module_utils/
#remote_tmp = ~/ansible/tmp
#local_tmp = ~/ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml
#forks = 5
#poll_interval = 15
#sudo_user = root
#ask_sudo_pass = True
....

Order of preference for ansible.cfg:
- ANSIBLE_CONFIG (environment variable)
- ansible.cfg (in the current directory)
- ~/ansible.cfg (in the home directory)
- /etc/ansible/ansible.cfg
Configure Ansible Managed Nodes
Create and Distribute SSH Keys to Manage Nodes and Configure Privilege Escalation

1. Generate SSH Keys
   # ssh-keygen

2. Distribute SSH Keys
   # ssh-copy-id

3. Escalate Privileges
   # visudo
   user_name ALL=(ALL) NOPASSWD: ALL

4. Control Node
   Managed Node 1
   Managed Node 2
Ansible Ad Hoc Command

Syntax:
ansible host -i inventory_file -m module -a “arguments”

- They are used to execute quick one liners.
- They are useful for non-routine tasks.
- Execute them using the ansible command (ansible-playbook is used to execute playbooks).
- Arguments require double quotes and are space delimited.
- Commands are executed as the user running Ansible.
- Use the -b option to execute commands as the root user.
- The -a option may be used without the -m option to run shell commands.
Configure Ansible Managed Nodes

Validate a Working Configuration Using Ad Hoc Ansible Commands

Common Uses

- File transfer
- Package management
- User and group management
- Managing services
- Fact gathering
- General system information
- Software deployment from Git
- Playbook creation testing
Shell Scripts

- The first line must include `#!/bin/bash`.
- Comments can be added by using the `#` symbol.
- Execute permission needs to be added to the script.
- Execute the script using the absolute path or `./script.sh` (if the script is in your current directory).

Simple echo Script:

```bash
#!/bin/bash
# hello world script

echo "Hello world!!"
```

A for Loop:

```bash
#!/bin/bash

for i in {1..5} do
    echo "Hello $i times!"
done
```
A case Statement

#!/bin/bash

echo -n "Enter the name of a state: "

read STATE

echo -n "The capital city of $STATE is "

case $STATE in
    Georgia)
        echo "Atlanta"
    ;;
    Virginia)
        echo "Richmond"
    ;;
    Texas)
        echo "Austin"
    ;;
    Maine)
        echo "Augusta"
    ;;
    *)
        echo "not in the database"
    ;;
esac
# Shell Script with Ad Hoc Ansible Commands

```bash
#!/bin/bash

# Create the user matt
ansible msppearson3c.mylabserver.com -i inv -b -m user -a "name=matt"

# Create the demo directory in matt’s home directory
ansible msppearson3c.mylabserver.com -i inv -b -m file -a "path=/home/matt/demo state=directory owner=matt group=matt mode=0755"

# Copy testFile to matt’s home directory
ansible msppearson3c.mylabserver.com -i inv -b -m copy -a "src=/home/cloud_user/ansible/testFile dest=/home/matt/testFile mode=0644 owner=matt group=matt"

# Install httpd to the webservers group, then start and enable the httpd service
ansible webservers -i inv -b -m yum -a "name=httpd state=latest"
ansible webservers -i inv -b -m service -a "name=httpd state=started enabled=yes"
```
Common Modules

- Ping
  - Validates a server is running and reachable
  - No required parameters
- Setup
  - Gather Ansible facts
  - No required parameters
- Yum
  - Manage packages with the YUM package manager
  - Common parameters (not required):
    - name and state
- Service
  - Control services on remote hosts
  - Common parameters:
    - name (required), state, and enabled
- User
  - Manage user accounts and attributes
  - Common parameters:
    - name (required), state, group, and groups
Common Modules (cont.)

- Copy
  - Copy files to a remote host
  - Common parameters: src, dest (required), owner, group, and mode
- File
  - Manage files and directories
  - Common parameters: path (required), state, owner, group, and mode
- Git
  - Interact with git repositories
  - Common parameters: repo (required), dest (required), and clone
Register the Results of Running a Command

- Use the `register` keyword to store the results of running a command as a variable.
- Variables can be referenced by other tasks in the playbook.
- Registered variables are only valid on the host for the current playbook run.
- Return values differ from module to module.

Use `register` in a playbook

```yaml
---
- hosts: hostname
  tasks:
    - name: create a file
      file:
        path: /tmp/testFile
        state: touch
      register: variable
    - name: display debug message
      debug: msg="Register output is {{ variable }}"
```
Create Ansible Plays and Playbooks

Use Conditionals to Control Play Execution

Handlers

- Handlers take action when called.
- Handlers are called when a change is made.
- Handlers are called using the notify keyword.
- More than one handler can be defined for a playbook or play.
- Multiple handlers can be specified in the notify section.
- Handlers can have multiple tasks.
- Regardless of how many tasks notify a handler, it will only run once.
Example of a Handler in a Playbook

---
- hosts: msperson2c
  become: yes
  tasks:
    - name: update httpd.conf
      replace:
        path: /etc/httpd/conf/httpd.conf
        regexp: ‘^ServerAdmin.*$’
        replace: ‘ServerAdmin cloud_user@localhost’
        backup: yes
      notify: “restart web server”
    handlers:
      - name: “restart apache”
        service:
          name: httpd
          state: restarted
        listen: “restart web server”
Create Ansible Plays and Playbooks

Use Conditionals to Control Play Execution

The **when** Statement

- Allows a task to run or be skipped if certain conditions are met.
- Parentheses can be used to group conditions.
- Multiple conditions can be specified as a list.
- Mathematical operation comparisons can be used.

---

- hosts: webservers
  become: yes
  tasks:
    - name: copy file
      copy:
        src: /home/cloud_user/index.html
        dest: /var/www/html/index.html
        when: ansible_hostname == "mspearson3c"
Create Ansible Plays and Playbooks

Use Conditionals to Control Play Execution

Loops

- May be performed using the `loop` or `with_<lookup>` keywords
- Standard loop usage
  - Iterate over a simple list
  - Iterate over a list of hashes
  - Iterate over a dictionary
- When statements are processed separately for each item in a loop

Example of a loop in a Playbook

```yaml
---
- hosts: webservers
  become: yes
  tasks:
    - name: create a list of users
      user:
        name: "{{ item }}"
        state: present
        groups: wheel
      loop:
        - violet
        - graham
        - bethany
```
Error Handling

- Ignore errors by using the `ignore_errors` keyword.
- Force previously notified handler to run using the `force_handlers` keyword.
- Define failure conditions using the `failed_when` keyword.
- Override the “changed” status result using the `changed_when` keyword.
- Abort an entire play if any task fails using the `any_errors_fatal` keyword.
- Implement a block in order to logically group tasks and provide error handling using the following keywords:
  - `block`
  - `rescue`
  - `always`
Using ignore_errors keyword:

```yaml
---
- hosts: labservers
  tasks:
  - name: copy remote files
    fetch:
      src: /tmp/errorFile
      dest: /tmp
    ignore_errors: yes
```

Using a block to handle errors:

```yaml
---
- hosts: labservers
  tasks:
  - name: copy remote files
    block:
      - fetch:
        src: /tmp/blockFile
        dest: /tmp
    rescue:
      - debug:
        msg: "The file doesn't exist on {{ ansible_hostname }}."
    always:
      - debug:
        msg: "Playbook finished!"
```
Example Playbook

---
- hosts: webservers
  become: yes
  tasks:
    - name: install apache
      yum:
        name: httpd
        state: latest
    - name: create user and add them to the apache group
      user:
        name: "{{ item }}"
        groups: apache
        loop:
          - will
          - myles
    - name: create index.html
      template:
        src: /home/cloud_user/ansible/templates/index.html
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: 0644
    - name: start and enable httpd
      service:
        name: httpd
        state: started
        enabled: yes
Example Playbook (cont.):

```yaml
- hosts: dbbservers
  become: yes
  tasks:
    - name: install postgresql
      yum:
        name: postgresql-server
        state: latest
    - name: initialize db cluster
      command: /usr/bin/postgresql-setup --initdb
      user:
        name: "{{ item }}"
        groups: postgres
      loop:
        - corey
        - aaron
    - name: start and enable postgres
      service:
        name: postgresql
        state: started
        enabled: yes
```
The **yum Module** - Use the yum package manager to install, upgrade, downgrade, remove, and list packages and groups.

**Examples of the yum module:**

- name: install a package
  yum:
    name: package_name
    state: latest

- name: Install a list of packages
  yum:
    name:
      - package_name
      - package_name
    state: latest

- name: Install rpm from a remote repo
  yum:
    name: http://website.com/path/to.rpm
    state: present

- name: Install rpm from a local file
  yum:
    name: /path/to/file.rpm
    state: present

- name: Remove a package
  yum:
    name: package_name
    state: absent
The **yum_repository module** - Add or remove a yum repository.

Examples of the yum_repository module:

- name: Add a repository
  yum_repository:
    name: repo_name
    description: Description of repo
    baseurl: https://website.com/full/path/of/base/url
    gpgcheck: no

- name: Remove a repository from a repo file
  yum_repository:
    name: repo_name
    file: repo_file_name (without the ".repo" extension)
    state: absent
The **service module** - This controls services on a remote host. The supported init systems are BSD init, OpenRC, SysV, Solaris SMF, systemd, and upstart.

**Example of the service module:**

```yaml
- name: Show options for service module
  service:
    name: service_name
    state: started|stopped|restarted|reloaded
    enabled: yes|no
    args: additional arguments provided on the command line
```

The **systemd module** - This controls systemd services on a remote host.

**Example of the systemd module:**

```yaml
- name: Show options for systemd module
  service:
    name: service_name
    state: started|stopped|restarted|reloaded
    enabled: yes|no
    daemon_reload: yes|no
    force: yes|no
```
The `firewalld` module - Allows for the addition or deletion of running or permanent firewall rules by services or ports (TCP or UDP)

Example of the `firewalld` module:

```yaml
- name: add firewall rules by service
  firewalld:
    zone: public|dmz|internal|external|trusted|etc.
    service: service_name
    permanent: yes|no
    immediate: yes|no
    state: enabled|disabled|present|absent

- name: add firewall rules by port(s)
  firewalld:
    zone: public|dmz|internal|external|trusted|etc.
    port: 8080/tcp|170-179/udp
    permanent: yes|no
    immediate: yes|no
    state: enabled|disabled|present|absent

- name: add firewall rules using a Rich Rule
  firewalld:
    zone: public|dmz|internal|external|trusted|etc.
    rich_rule: rule family=ipv4 forward-port
    port=443 protocol=tcp to-port=8443
    permanent: yes|no
    immediate: yes|no
    state: enabled|disabled|present|absent
```
Create Partitions

The `parted` module - Uses the parted command line tool in order to configure block device partitions.

Example of the `parted` module:

```
- name: Show options for parted module
  parted:
    device: /dev/sdc|/dev/nvme0n1
    number: 1
    state: present|absent|info
    part_end: 1GiB|100%
    label: msdos|gpt
    flags: [ lvm ]
```
Create Logical Volumes

The `lvg` module - Create, remove, and resize volume groups

Example of the `lvg` module:

```
- name: show options for lvg module
  lvg:
    force: no|yes
    pesize: 8
    pvs: /dev/nvme1n1p1,/dev/nvme2n1p1
    vg: vg_name
    state: present|absent
```

The `lvvol` module - Create, remove, and resize logical volumes

Example of the `lvvol` module:

```
- name: show options for lvvol module
  lvvol:
    vg: vg_name
    lv: lv_name
    size: 512m|1g|100%FREE
    state: present|absent
    shrink: yes|no
    resizefs: no|yes
    force: no|yes
    opts: free form options passed to the lvcreate command
```
Manage File Content

The file module - Manage files and file properties.

Example of the file module:

- name: create a file
  file:
    path: /path/to/file
    state: touch

The copy module - Copy files to remote locations.

Example of the copy module:

- name: add content to specific file
  copy:
    content: file_content
    dest: /path/to/file

The lineinfile module - Manage lines in text files.

Example of the lineinfile module:

- name: replace a line
  lineinfile:
    path: /path/to/file
    regexp: 'regular_expression'
    line: line to insert/replace in the file
Create Files and and Replace Strings (cont.)

The replace module - Replace all instances of a particular string within a file.

Example of the replace module:

```
- name: replace strings within a file
  replace:
    path: /path/to/file
    regexp: regular_expression
    replace: 'string that replaces regexp matches'
```

The template module - Template a file out to a remote server.

Example of the template module:

```
- name: show template options
  template:
    src: /path/to/template.j2
    dest: /path/to/dest
    owner: owner_name
    group: group_name
    mode: file_permissions
```
Create Files and and Replace Strings (cont.)

Example of a template file:

```yaml
Hostname = {{ ansible_hostname }}
Operating System = {{ ansible_distribution }} {{ ansible_distribution_version }}
IPV4 Address = {{ ansible_default_ipv4.address }}
IPV6 Address = {{ ansible_default_ipv6.address }}
Interfaces = {{ ansible_interfaces|join(' ', ') }}
Block Devices = {{ ansible_devices|join(' ', ') }}
```
Create a Filesystem

The `filesystem module` - Create a filesystem.

Example of the filesystem module:

```
- name: options for filesystem module
  filesystem:
    ftype: fs_type
    dev: /path/to/device
    force: no|yes
    resizefs: no|yes
    opts: options to pass to the mkfs command
```

Mount a Filesystem

The `mount module` - Control and configure mount points.

Example of the mount module:

```
- name: option for the mount module
  mount:
    path: /path/to/mount/point
    src: /path/to/device
    ftype: fs_type
    state: mounted|absent|present|unmounted
    opts: mount options
    backup: no|yes
```
Create an Archive

The `archive` module - Creates a compressed archive on one or more files or directories.

Example of the `archive` module:

```yaml
- name: show options for archive module
  archive:
    path:
      - /path/to/file
      - /path/to/file
      - /path/to/dir
      - /globbed/path/using/*
    exclude_path:
      - /file/to/exclude
      - /dir/to/exclude
      - /glob/to/exclude/ex*
    format: gz|bz2|tar|xz|zip
    dest: /name/of/archive.tgz
```
Unpack an Archive

The `unarchive` module - Copy (optional) and unpack an archive.

Example of the `unarchive` module:

```yaml
- name: show options for unarchive module
  unarchive:
    src: /path/to/file.zip|www.website.com/path/to/file.zip
    dest: /path/to/unpack/in
    remote_src: no|yes
```
Schedule Tasks using `cron` and `at`

The `cron` module - Manage `cron.d` and crontab entries.

Example of the `cron` module:

```yaml
- name: show options for the cron module
cron:
  name: “job_name”
special_time: reboot|daily|weekly|etc.
  minute: 0-59|*|*/2
  hour: 0-23|*|*/2
  day: 1-31|*|*/2
  month: 1-12|*|*/2
  weekday: 0-6 for Sunday to Saturday|
  user: user_name
cron_file: file_name
state: present|absent
job: command
```

____________________________________

Back   Next
**Schedule Tasks using** `cron` **and** `at` **(cont.)**

**The at module** - Schedule the execution of a command or script with the `at` command.

---

**Example of the at module:**

```yaml
- name: show options for the at module
  at:
    command: command_to_run
    script_file: /path/to/script.sh
    count: count of units in the future to execute the command or script
    unit: minutes|hours|days|weeks
    state: present|absent
```

---

**Back** | **Next**
The `selinux module` - Change the policy and state of SELinux.

Example of the `selinux module`:

- name: show options for selinux module
  selinux:
    configfile: /path/to/config
    policy: targeted
    state: enforcing|permissive|disabled

The `seboolean module` - Toggle SELinux booleans.

Example of the `seboolean module`:

- name: show options for seboolean module
  seboolean:
    name: boolean_name
    state: no|yes
    persistent: no|yes
The `sefcontext` module - Manage SELinux file context mappings definitions.

Example of the `sefcontext` module:

- name: show options for sefcontext module
  sefcontext:
    ftype: a|d|etc.
    reload: yes|no
    target: '/path/to/dir(/.*)?'
    setype: selinux_type
    state: present|absent
- name: apply new SELinux context to the filesystem
  command: restorecon -irv /path/to/dir
Manage Users

The **user module** - Manage user accounts and user attributes.

```yaml
- name: show options for user module
  user:
    name: user_name
    shell: /bin/bash|/bin/zsh|etc.
    home: /path/to/home/dir
    comment: user_description
    uid: set the uid of the user
    group: set the user’s primary group
    groups: list of groups to add the user to
    append: no|yes
    state: present|absent
```

Manage Groups

The **group module** - Add or remove groups.

```yaml
- name: show options for the group module
  group:
    name: group_name
    gid: set the gid for the group
    system: no|yes
    state: present|absent
```
Understanding Variables

- Can contain letters, numbers, and underscores
- Must begin with a letter
- Can be stored as dictionaries, which map keys to values
  - Dictionary variables can be referenced using bracket notation or dot notation:
    - Example: `dictionary_name['field1']` or `dictionary.field1`
- Variables stored as a list (array) may be accessed by putting the element number in brackets:
  - Example (first element of an array):
    ```
    {{ array_name[0] }}
    ```
- Variables may be defined or set in the following locations:
  - Inventories (also in `host_vars` and `group_vars` directories)
  - Playbooks (i.e., `vars`, `vars_files`, and `vars_prompt`)
  - Roles (i.e., set in `roles/role_name/vars/main.yml`)
  - The command line (i.e., `-e` or `--extra-vars`)
Understanding Variables (cont.)

- Referenced defined variables using the Jinja2 templating system.
  - example: This is my {{ variable }}
    - Note: YAML requires values starting with a variable to be quoted.
- Transform variable values using Jinja2 filters (i.e. join, capitalize, etc.)
- Ansible stores information about remote hosts in variables known as Ansible facts.
- Ansible provides special, reserved variables known as magic variables.
  - Examples: hostvars, groups, group_names, and inventory_hostname
- Custom facts (AKA local facts) can be added to the remote system by the user:
  - Defined in files that end with .fact
  - Local fact files are stored in /etc/ansible/facts.d
  - Fact file directory can be changed using the fact_path keyword
  - Local facts can be viewed by running the following: ansible <hostname> -m setup -a "filter=ansible_local"
**Understanding Templates**

- Templates are files that contain both static values and dynamic values, through the use of variables.
- Ansible processes templates using jinja2.
- Templates are designated with the .j2 extension.
- Templates are often used for configuration file management.
- Templates have access to the same variables as the plays that call them.

**The template module** - Process a template and push it out to a remote server.

**Example of the template module:**

```yaml
- name: show template options
  template:
    src: /path/to/template.j2
    dest: /path/to/dest
    owner: owner_name
    group: group_name
    mode: file_permissions
    validate: validation_command %s
    backup: no|yes
```
httpd.conf Template

# This is the main Apache HTTP server configuration file.
#
# {{ ansible_managed }}

ServerRoot "/etc/httpd"

Listen {{ http_port }}

Include conf.modules.d/*.conf

User apache
Group apache

ServerAdmin {{ admin }}@{{ ansible_hostname }}

<Directory />
    AllowOverride none
    Require all denied
</Directory>

DocumentRoot "{{ content_dir }}"
...

Back Next
Welcome to {{ ansible_hostname }}!

- The ipv4 address is {{ ansible_default_ipv4['address'] }}

- The current memory usage is {{ ansible_memory_mb['real']['used'] }}mb out of {{ ansible_memory_mb['real']['total'] }}mb

- The {{ ansible_devices| first }} block device has the following partitions:
  -{{ ansible_devices['nvme0n1']['partitions']|join('
 -') }}
Playbook Example

```yaml
---
- hosts: webservers
  become: yes
  vars:
    content_dir: /webcontent
    http_port: 8080
    admin: cloud_user
  tasks:
    - name: push config template
      template:
        src: /home/cloud_user/ansible/templates/httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
        backup: yes
        notify: "restart apache"
    - name: push index.html template
      template:
        src: /home/cloud_user/ansible/templates/index.html.j2
        dest: /webcontent/index.html
  handlers:
    - name: restart web servers
      service:
        name: httpd
        state: restarted
        notify: "restart apache"
```
Roles Overview

- The default location for roles is `/etc/ansible/roles`.
- Roles provide a way to automatically load certain vars_files, tasks, and handlers based on a known file structure.
- Roles expect a particular directory structure.
- Directories not being used may be excluded.
- Each directory in use most contain a `main.yml` with relevant content.

```
[cloud_user@mspearson1c roles]$ tree -A common/
common/
    ├── defaults
    │    └── main.yml
    ├── files
    └── handlers
        ├── main.yml
        └── meta
            └── main.yml
    └── tasks
        └── main.yml
    └── templates
    └── tests
        └── inventory
            └── test.yml
    └── vars
        └── main.yml
```
Roles Directories

- **tasks** - This directory contains the main list of tasks to be executed by this role.
- **handlers** - This directory contains handlers, which may be used by this role or even anywhere outside this role.
- **defaults** - This directory contains default variables for the role and is meant to provide a value to a variable if no other value is given. (low precedence)
- **vars** - This directory contains variables used within the role. (high precedence)
- **files** - This directory contains files which can be deployed via this role.
- **templates** - This directory contains templates which can be deployed via this role.
- **meta** - This directory defines some of the meta data for this role.
Red Hat Enterprise Linux (RHEL) System Roles

- Provided by the RHEL Extras repository.
- Requires the `rhel-system-roles` package to be installed.
- Can be used by Ansible Engine and Ansible Tower to manage RHEL systems.
- Needs to be installed on the Ansible control node(s) which can then be used to manage and configure the client nodes.
- Supported system roles include: kdump, network, selinux, timesync, and postfix (tech preview)
- The documentation can be found in the following location:
  `/usr/share/doc/rhel-system-roles-<version>/SUBSYSTEM/`
- The Ansible roles may be found in the following location:
  `/usr/share/ansible/roles/rhel-system-roles.SUBSYSTEM/`
Role Variables (timesync role):

# List of NTP servers
timesync_ntp_servers:
- hostname: foo.example.com  # Hostname or address of the server
  minpoll: 4  # Minimum polling interval (default 6)
  maxpoll: 8  # Maximum polling interval (default 10)
  iburst: yes  # Flag enabling fast initial synchronization
  pool: no  # (default no)
  # Flag indicating that each resolved address
  # of the hostname is a separate NTP server
  # (default no)

# List of PTP domains
timesync_ptp_domains:
- number: 0  # PTP domain number
  interfaces: [ eth0 ]  # List of interfaces in the domain
  delay: 0.000010  # Assumed maximum network delay to the
  # grandmaster in seconds
  # (default 100 microsecond)
  transport: UDPv4  # Network transport: UDPv4, UDPv6, L2
  # (default UDPv4)
  udp_ttl: 1  # TTL for UDPv4 and UDPv6 transports
  # (default 1)
Create and Work with Roles

Example Playbook (\texttt{timesync role}):

```yaml
- hosts: targets
  vars:
    timesync_ntp_servers:
      - hostname: foo.example.com
        iburst: yes
      - hostname: bar.example.com
        iburst: yes
      - hostname: baz.example.com
        iburst: yes
  roles:
    - rhel-system-roles.timesync
```
Create the Directory Structure for Roles:
ansible-galaxy init role_name

Create tasks:

---
# tasks file for apache
- name: create webcontent directory
task: file:
  name: {{ apache_content_dir }}
  state: directory
  mode: 0755
- name: set secontext on webcontent directory
task: secontext:
  target: {{ apache_content_dir }}(/.*)?
  setype: http_sys_content_t
  state: present
- name: run restorecon on webcontent
task: command: restorecon -irv {{ apache_content_dir }}
- name: install apache
  yum:
    name: httpd
    state: latest
- name: deploy httpd.conf template
task: template:
  src: httpd.conf.j2
  dest: /etc/httpd/conf/httpd.conf
  backup: yes
  notify: "restart apache"
- name: deploy index.html template
task: template:
  src: index.html.j2
  dest: {{ apache_content_dir }}/index.html
  backup: yes
- name: start and enable httpd service
task: service:
  name: httpd
  enabled: yes
  state: started
---
Create and Work with Roles

Creating and Using Roles

Create variables:

```
---
# defaults file for apache
apache_content_dir: /webcontent
apache_http_port: 8080
apache_admin: cloud_user
```

Create templates (index.html.j2):

```
Welcome to {{ ansible_hostname }}!

- The ipv4 address is {{ ansible_default_ipv4['address'] }}
- The current memory usage is {{ ansible_memory_mb['real']['used'] }}mb out of {{ ansible_memory_mb['real']['total'] }}mb
- The {{ ansible_devices|first }} block device has the following partitions:
  -{{ ansible_devices['nvme0n1']['partitions']|join('
 - ') }}
```
Create templates (httpd.conf.j2):

```bash
# This is the main Apache HTTP server configuration file.
#
# {{ ansible_managed }}

ServerRoot "/etc/httpd"

Listen {{ apache_http_port }}

Include conf.modules.d/*.[ch]

User apache
Group apache

ServerAdmin {{ apache_admin }}@{{ ansible_hostname }}

<Directory />
    AllowOverride none
    Require all denied
</Directory>

DocumentRoot "{{ apache_content_dir }}"
```

...
Create and Work with Roles

Creating and Using Roles

Create handlers:

```yaml
---
# handlers file for apache
- name: restart web servers
  service:
    name: httpd
    state: restarted
    listen: "restart apache"
```

Create playbook (examples):

```yaml
---
- hosts: webservers
  become: yes
  roles:
    - role_name
    - role: role_name
      vars:
        var_name: value

---
- hosts: webservers
  become: yes
  tasks:
    - include_role:
        name: role_name
        vars:
          var_name: value
```

Back  Next
Understanding Ansible Galaxy

Ansible Galaxy - A large public repository for downloading and sharing community developed roles.

The ansible-galaxy utility - Create and remove roles or install roles from Ansible Galaxy or a Git-based SCM (software configuration management).

Syntax: ansible-galaxy
[delete|import|info|init|install|
list|login|remove|search|setup] [--help]
[options] ...
Create and Work with Roles

Ansible Galaxy Commands

Create roles directory structure:
ansible-galaxy init role_name

Search Galaxy for roles:
ansible-galaxy search keyword --author authorname

Install a role:
(from galaxy) ansible-galaxy install role_name
(from a file) ansible-galaxy install -r file.yml

List installed roles:
ansible-galaxy list

Print information about installed roles:
ansible-galaxy info role_name

Remove a role:
ansible-galaxy remove role_name
Understanding Parallelism

- Parallelism is the word used for Ansible’s default ability to interact with multiple hosts at the same time.
- The parallel processes spawned by Ansible are known as forks.
- The default number of forks in Ansible is 5.
- The default number of forks can be changed by editing the forks parameter in ansible.cfg.
- The number of forks can be changed on a per command basis by using the -f flag.
- Ansible allows for rolling updates using the serial keyword.

Change default forks value in ansible.cfg:
```
forks = desired_number
```

Change forks value in the command line:
```
anible host -f desired_number -m module -a "arguments"
anible-playbook -f desired_number playbook.yml
```
Use the serial keyword to perform rolling update:

```yaml
- hosts: labservers
  serial:
    - 1
    - 2
    - 50%
  tasks:
    - name: create new file
      file:
        path: /tmp/serialFile
        state: touch
```
Understanding Ansible Vault

- Can be used to encrypt any structured data file used by Ansible:
  - Variable files in `group_vars` and `host_vars` directories
  - Variable files loaded by `include_vars` and `vars_files` in playbooks
  - Variable files passed on the command line using `-e @var_file.yml`
- Passwords on an encrypted file can be updated using the `ansible-vault rekey` command.
- Can be used to encrypt individual variables inside a YAML file using the `!vault` tag
- Uses vault IDs to support multiple vault passwords (labels are used to distinguish between the individual passwords):
  - `example: --vault-id label@source`
The Ansible Vault Command

- Create an encrypted file: `ansible-vault create file.yml`
- Create an encrypted file with a vault ID: `ansible-vault create --vault-id label@source file.yml`
- Edit an encrypted file: `ansible-vault edit file.yml`
- Edit an encrypted file with a vault ID: `ansible-vault edit --vault-id label@source file.yml`
- Rekey encrypted files: `ansible-vault rekey file1.yml file2.yml`
- Rekey encrypted files a vault ID: `ansible-vault rekey --vault-id label@source file1.yml file2.yml`
- Encrypt an existing file: `ansible-vault encrypt file.yml`
- Encrypt an existing file with a vault ID: `ansible-vault encrypt --vault-id label@source file.yml`
- Decrypt a file: `ansible-vault decrypt file.yml`
- View an encrypted file: `ansible-vault view file.yml`
- Encrypt a string to be used as a variable in a YAML file: `ansible-vault encrypt_string --ask-vault-pass 'string_value' --name 'secret_var'
- Output of encrypted variable:

```yaml
$ANSIBLE_VAULT:1.1;AES256
63373266439386152663331643832663539663339626236313639632313836623130383063461
3534323866539626132613561634376663993713931649a313564643963666434633962373133
38656632303264346383738363639616163373632353265613366386555383436363864653135
643366361323437330a64366531343630626661393635313753665663261383735346535363031
3038
Encryption successful
```
Providing Vault Passwords

- Use password stored in a text file: `ansible-playbook --vault-password-file /path/to/password/file playbook.yml`

- Use password stored in a text file with a vault ID: `ansible-playbook --vault-id label@password_file playbook.yml`

- Prompt for a password: `ansible-playbook --ask-vault-pass playbook.yml`

- Prompt for a password with a vault ID: `ansible-playbook --vault-id label@prompt playbook.yml`

- Use multiple passwords with vault ID: `ansible-playbook --vault-id label1@password_file --vault-id label2@prompt playbook.yml`
Using Ansible Vault in a Playbook

Create an encrypted file:
ansible-vault create file.yml

Encrypt an already existing file:
ansible-vault encrypt file.yml

Edit an encrypted file:
ansible-vault edit file.yml

Run playbook with encrypted variables file:
• Using password prompt:
  ansible-playbook --ask-vault-pass playbook.yml
• Using password file:
  ansible-playbook --vault-password-file /path/to/password/file playbook.yml

Rekey an encrypted file:
ansible-vault rekey file.yml

Decrypt an encrypted file:
ansible-vault decrypt file.yml
The `ansible-doc` Command - Documentation Tool for Plugins and Modules

Command syntax:
```
ansible-doc [-l|F|-s] [options] [-t <plugin type> ] [plugin]
```

List modules:
```
ansible-doc -l
```

Show documentation for a module:
```
ansible-doc module_name
```

Show documentation snippet of a module:
```
ansible-doc -s module_name
```
Example ansible-doc Command Output:

```yaml
# ansible-doc service
...

Controls services on remote hosts. Supported init systems include BSD init, OpenRC, SysV, Solaris SMF, systemd, upstart. For Windows targets, use the [win_service] module instead.

* This module is maintained by The Ansible Core Team
* note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):
- arguments
  Additional arguments provided on the command line.
  (Aliases: args)[Default: (null)]
  type: str
- enabled
  Whether the service should start on boot.
  *At least one of state and enabled are required.*
  [Default: (null)]
  type: bool
= name
  Name of the service.
  type: str

EXAMPLES:
- name: Start service httpd, if not started
  service:
    name: httpd
    state: started
...
```
Ansible Documentation:
https://docs.ansible.com/

Ansible Galaxy Documentation:
https://galaxy.ansible.com/docs/

Points of interest:
• Ansible Documentation
• Installation | Installation Guide
• User Guide
• Playbooks | Working With Playbooks
• Module Index
• Ansible Galaxy | Galaxy Documentation
Recommendations for Your Next Course

- Red Hat Certified Architect
  - Red Hat Certified Specialist in Virtualization (ex318)
  - Red Hat Certified Specialist in Ansible Automation (ex407)
  - Red Hat Certified Specialist in OpenShift Administration (ex280)
  - Red Hat Certified Specialist in Server Hardening (ex413)
  - Red Hat Certified Specialist in Security (ex415)
  - Red Hat Certified Specialist in Linux Diagnostics and Troubleshooting (ex342)

- Cloud platforms
  - AWS
  - Azure
  - Google Cloud

- DevOps
  - Chef
  - Puppet
  - Saltstack
  - Kubernetes

- Big Data
  - Elastic
  - Splunk
  - Hadoop
Preparing for the Exam

- Make sure you have completed all the lessons and hands-on labs.
- Make use of the instructor-provided flash cards and create your own.
- Use the interactive diagram as a study guide reference.
- Go back over any topic multiple times to help with retention (labs and lessons).
- Reach out to the Linux Academy community for any additional questions.
- Join a study group.
- Be sure to get some good sleep the night before the exam.
- Eat a modest meal before taking the exam and be sure that you are reasonably hydrated.
- Arrive at the testing center with plenty of time before the exam starts.

Signing up for the Exam

- Register for your desired format: classroom or individual.
- Follow the email instructions from Red Hat to schedule the location and time of your exam (for individuals).