Ansible Playbook Process:

1. **Create Playbook**
    - Create an empty playbook

2. **Outline each Task**
    - Name each task required

3. **Build each Task**
    - Build each task required according to documentation

4. **Check Playbook**
    - ansible-playbook -C <playbook> -K

Critical Exam Tips:

- Always number your playbooks for speed, ex. **one, two three** (can't be numbers)
- Always use **^string.*** when using regexp flag to make sure entire line gets replaced
- Always use **aliases** in **~/.bashrc** for ansible-playbook and ansible-doc (ap, and ad)
- Always use **3 slashes in baseurl** for FTP and HTTP
    - ftp:///controller/(BaseOS/AppStream) OR http:///controller/(BaseOS/AppStream)
- Always use **block/rescue/always** for error handling
- Use --- and import_playbook with nothing else when using multiple playbooks in one file

Critical Ad-Hoc Flags:

Description	Flag
Ask Become Pass	-K
Ask Pass	-k

Become	-b
Module Name	-m
Arguments	-a
Check syntax	-c
Inventory file	-I
Remote user	-u
Extra vars	-e

Basics

Remote user must exist on remote host

Remote user must be in wheel group on remote host

Remote user must be able to do password-less sudo on remote host

SSH Public Key must be copied into ~/.ssh/authorized_keys file on remote host under remote user home directory

Critical Commands:

Description	Command
Display all Ansible keywords	ansible-doc --type keyword --list
Get help using a Module	ansible-doc <module>
Display all installed Ansible modules	ansible-doc -l
Display parameters for a Module	ansible-doc -s <module>
Test Reachability of Hosts – Ping	ansible all -m ping

Run Ansible Ad-Hod commands	ansible -m <module> -a <commandarguments>
Which config file is in use	ansible --version
Which hosts are in inventory	ansible all --list-hosts
Modules related to gathering facts	ansible-doc -l | grep fact
List all gathering facts variables	ansible -m setup <hostname> > facts.yml
Install Containerized AAP	ansible-playbook -i inventory-growth ansible.containerized_installer.install -K
Verify playbook syntax	ansible-lint <playbook/inventory/ansible.cfg>

Documentation

All of these can be found under Ansible Core documentation - This is all you need

Documentation

Building Ansible inventories

Using Ansible command line tools

Using Ansible playbooks

Protecting sensitive data with Ansible vault

Using Ansible modules and plugins

Using Ansible collections

Ansible tips and tricks

Reference & Appendices

Frequently Asked Questions

Ansible Configuration Settings

Playbook Keywords

Special Variables

Controlling how Ansible behaves: precedence rules

YAML Syntax

Hyphen or No-Hyphen:

Vars is a **dictionary of key/value pairs**. Each key is just one property, not an "item in a sequence," so **no hyphen**

- **Playbooks, roles, tasks, handlers -> lists** (start with -)
- **Vars and module args -> mappings** (just key: value)

Playbooks will still run with vars missing hyphens, this is just good practice

Playbook & Config Structure:

Refer to documentation on structure of both files

Playbooks

name

hosts

become / become_user

vars / vars_files

tasks

module

Config

[defaults]

inventory = inventory

remote_user = awx

ask_pass = false

private_key_file = /path/to/private_key

[privilege_escalation]

become = true

become_user = root

become_method = sudo

become_ask_pass = false

In project directory

mkdir collections group_vars host_vars vars roles

Ansible Cheat Sheet:

Ansible when:

Check if host is in a group

```
- name: Install nmap
  dnf:
    name: nmap
    state: latest
  when: inventory_hostname in groups['dev']
```

Check if host is not in a group

```
- name: Install nmap
  dnf:
    name: nmap
    state: latest
  when: inventory_hostname not in groups['dev]'
```

Check multiple conditions

```
- name: Install nmap
  dnf:
    name: nmap
    state: latest
  when: "'dev' in group_names or 'prod' in group_names"
```

Check if a fact doesn't exist

```
- name:
  fail:
    msg: "value is not in variable"
  when: ansible_devices.sdb is not defined
```

Check if a string equals something

```
- name: Do something only on RHEL
  command: echo "This is RHEL"
  when: ansible_distribution == "RedHat"
```

Check if boolean is true

```
- name: Only run when debug_mode is true
  debug:
    msg: "Debug mode enabled"
  when: debug_mode | bool
```

Check for value in a list

```
- name: Only run for specific datacenter
  debug:
    msg: "This DC is allowed"
  when: datacenter in ['dc1', 'dc2']
```

Check that a number starts with 1

```
 - name: Ensure users with UID starting with 1 are present
   user:
```

```
    state: present
    name: "{{ item.username }}"
    shell: /bin/bash
  when:
    - item.uid | string is regex('^1')
  loop: "{{ users }}"
```

Check and

```
- name: Only run for specific datacenter
  debug:
    msg: "This is true"
  when: something==true and something2==true
```

Check or

```
- name: Only run for specific datacenter
  debug:
    msg: "This is true"
  when: something==true or something2==true
```

Variables

Inventory/group_vars/host_vars

a) Inventory variables

- Set directly in the inventory file:

```
[web]
web1 ansible_host=192.168.1.10 username=julie
```

b) Group variables (group_vars/) -> Only defines variables for a specific group

- Define variables for a whole group of hosts.
- File structure:

group_vars/prod.yml

group_vars/dev.yml

- Example group_vars/prod.yml:

username: julie

- Automatically applied to hosts in that group.

c) Host variables (host_vars/) -> Only defines variables for a specific host

- Variables specific to one host.

- File structure:

host_vars/web1.yml

- Example:

username: julie

timezone: UTC

Playbook/Task level

a) vars: in a playbook

- Example:

```
- hosts: prod
  vars:
    username: julie
  tasks:
    - debug:
        msg: "User is {{ username }}"
```

b) vars: at task level

- Example:

```
- debug:
    msg: "User is {{ username }}"
  vars:
    username: julie
```

External files (vars_files:)

- Include a YAML file for variables:

- hosts: prod

  vars_files:

    - vars/prod.yml

  tasks:

    - debug:

        msg: "User is {{ username }}"

Extra vars (CLI)

- Passed at runtime, highest precedence:

ansible-playbook playbook.yml -e "username=julie"

Facts

- Gathered automatically or set via set_fact:

- set_fact:

    username: julie

- Can be host-specific and dynamic.

Precedence Summary (RHCE-level)

From lowest → highest:

1. Inventory/group_vars/host_vars

2. Playbook vars:

3. vars_files:

4. set_fact

5. Extra vars (-e)

Extra vars always override everything else.

Critical Information:

Always use Debug module!!!

List all gathered facts

ansible <hostname> -m setup > facts

Use group vars for group-specific variables

Create a custom fact

1. Create a file in /etc/ansible/facts.d on the managed host
2. Name the file ending in .fact
3. Put the following content in file

[fact_name]

custom_key=some_value

4. Verify the custom fact

ansible all -m setup -a "filter=ansible_local"

This will set the facts every time ansible runs

If you want it to run once just use the **set_fact** flags

```
- name: Example playbook
  hosts: all
  tasks:

    - name: Set a custom fact
      set_fact:
        my_custom_fact: "hello world"

    - name: Show custom fact
      debug:
        msg: "My fact is {{ my_custom_fact }}"
```

Using vars_files

```
  vars_files:
    - vars/user_list.yml
```

You must create a vars directory

Use stat module to check if something exists

stat -> register in a variable, use ansible-doc stat

variable.stat.exists and not variable.stat.exists

Use uri module for testing web service reachability

uri and curl -I both return headers

Calling Facts (These are Equivalent)

```
    - debug:
        msg: "{{ ansible_facts['distribution_release'] }}" -> Remove ansible_ from key name
    - debug:
        msg: "{{ ansible_distribution_release }}" -> Use whole key name
```

Calling a Nested Fact

```
    - debug:
        msg: "{{ ansible_date_time.date }}" -> Calls a key within a Dictionary
```

Create user on Remote Hosts

ansible all -u awx -k -b -K -m user -a "name=remote_user"

Set a password on user

ansible all -u awx -k -b -K -m shell -a "echo <password> | passwd --stdin awx"

Create SSH keys, and Copy SSH keys

for i in vm1, vm2; do ssh-copy-id $i; done or sh-copy-id -i ~/.ssh/id_rsa.pub awx@remote_host

For keywords consult:

ansible-doc --type keyword –list

Generate a Template ansible.cfg file (found in /etc/ansible/ansible.cfg)

ansible-config init --disabled -t all > ansible.cfg

Generate an Inventory file in yaml format (creates a yaml formatted inventory)

ansible-inventory --list --yaml > inventory

Enables privilege escalation, with sudo, no password prompt – Bad practice in production, good for lab

ansible all -k -b -K -m copy -a "content='awx ALL=(ALL) NOPASSWD: ALL' dest=/etc/sudoers.d/awx"

Update sudoers.d so Controller so user can escalate privileges without being prompted for a password

echo "awx ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers.d/awx

Copy everything stored on ISO to repo server starting with A or B (AppStream & BaseOS Repos)

sudo cp -R /mnt/[AB]* /reposerver

Use command module to restore context after making changes using SELinux modules

command: restorecon –Rv /localdirectory

Fail if a command didn't successfully run:

```
- name: ok
  command: chronyc tracking || echo "Time could not be synchronized"
```

Install ansible.posix (firewalld) module

ansible-galaxy collection install ansible.posix

Default directory cron module will create files in if cron_file isn't used

/var/spool/cron/

Use Jinja2 template to populate files

{% for host in groups['all'] %}

"This is each systems FQDN: {{ hostvars[host]['ansible_fqdn'] }}"

{% endfor %}

Use 'ansible all –m setup > facts' to find names of dictionaries and keys to use in your template files

- **Avoid non-idempotent behavior**
- **Don't use ansible-navigator, use ansible-playbook**
- **Take 3 breaks during exam**
- **Read the "essential information" section very carefully**
- **It doesn't matter *how* you complete a task, as long as it's completed successfully**

Playbook Commands:

Description	Command
Check syntax of a Playbook	ansible-playbook --syntax-check
Complete a playbook dry run	ansible-playbook -C
Run a Playbook against a single host (limit)	ansible-playbook <playbook> --limit <host>
Run a Playbook against a list	ansible-playbook -i localhost, server1, server2 <playbook>.yaml
Run a Playbook against a list of hosts	ansible-playbook -i /etc/ansible/hosts <playbook>.yaml
Run a Playbook to update packages on hosts	ansible-playbook all -m command -a "yum update -y"
Which tasks will the playbook run	ansible-playbook <playbook>.yaml --list-tasks
Run a playbook using encrypted password	ansible-playbook <playbook> --ask-vault-pass
Start running a playbook at a specific task	ansible-playbook --step <playbook>.yaml
Start playbook execution as a specific task	ansible-playbook --start-at-task="task name" <playbook>.yaml

Ansible Vault:

Description	Command
Create an encrypted password file	ansible-vault create/edit/view <vault.yml>
Change password on encrypted file	ansible-vault rekey <vault.yml>
Create ansible vault with a vault ID	ansible-vault create <vault_id>@<vault.yml>
Create directory for encrypted vault and unencrypted vault secrets	mkdir secrets vars # vault goes in vars
Prompt ansible-playbook for vault password	--ask-vault-pass
Point ansible-playbook to unencrypted file containing vault secret	--vault-password-file
Include the vault file with vars_files	vars_files
Use ask vault pass or vault pass file	--vault-password-file OR --ask-vault-pass, not both
Running a playbook with multiple different vaults	ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt site.yml

Important Information:

Description	Command
Ansible Verbosity	ansible-playbook -v -i <HOST> <playbook>.yaml <-displays output data ansible-playbook -vv -i <HOST> <playbook>.yaml <-displays output & input

	data ansible-playbook -vvv -i <HOST> <playbook>.yaml <-information about connections ansible-playbook -vvvv -i <HOST> <playbook>.yaml <-information about plugins
Ansible Priorities	Priority in which the config files are processed: 1) ANSIBLE_CONFIG (environment variable) 2) ./ansible.cfg (in the current directory) 3) ~/.ansible.cfg 4) /etc/ansible/ansible.cfg
Commonly Modified Settings in 'ansible.cfg	inventory= Location of Ansible inventory file remote user= Remote user account used to establish connections to managed hosts become= Enables/disables privilege escalation for operations on managed hosts (NOT BY DEFAULT!) become_method= Defines privilege escalation method become_user= User account for escalating privileges become_ask_pass= Defines whether privilege escalation prompts for password
Ansible Files (Defaults, if you don't create your own project dirs)	/etc/ansible/hosts # Inventory (hosts) file /etc/ansible/ansible.cfg # Configuration file /etc/ansible/<myplaybook.yml> # Playbook file
Ansible Forks (Used on devices that don't have a python stack, so processing must be completed on the control node, The default value for forks is 5, and should be increased if managed nodes are	You can manage this setting by using the forks parameter in ansible.cfg or with -f option at command line It prevents the control node from being overloaded Because processing in most environments is done on the managed hosts, the maximum setting of five forks just slows down the working of Ansible, and it is a good idea to increase this maximum to something significantly

linux)	higher. If only Linux hosts are managed, there is no reason to keep the maximum number of simultaneous tasks much lower than 100
Serial Task Execution (By default, Ansible runs task by task. It will run Task 1 on all hosts, then Task 2 on all hosts. If you need it to run only a few hosts at a time, set serial: #)	For instance, the serial: 3 keyword is used in the header of a play, all tasks are executed on three hosts, and after completely running all tasks on three hosts, the next group of three hosts is handled Useful for making sure that not all systems are down at the same time Ex. task: Install updates & reboot With serial: 3 -> All hosts won't be down at the same time, it will run on 3 hosts, then move onto the next 3