Ansible Playbooks

- Playbooks are the orchestration language.
- It’s the playbook where we define what we want ansible to do.
- Playbooks are a way to congregate ordered processes and manage configuration needed to build out a remote system.
- Playbooks make configuration management easy and gives us the ability to deploy to a multi-machine setup.

Ansible Playbook

Single YAML file which consists of plays

A play is a set of activities (tasks) to run on hosts (or a group of hosts)

Task is an action to be performed on the hosts

- Execute a command
- Run a script
- Install a package
- Shutdown/Restart

```yaml
---
- name: Play 1
  hosts: localhost
  tasks:
    - name: Execute 'date' command
      command: date

    - name: Execute a script on server
      command: test_script.sh

---
- name: Play 2
  hosts: localhost
  tasks:
    - name: Install web server
      yum:
        name: httpd
        state: present

    - name: Start web server
      service:
        name: httpd
        state: started
```

Ansible Modules

- System Module
- Command Module
- Script Module
- File Module
- Database Module
- Cloud Module
- Windows Module

Validating a Playbook

- Syntax validation with --syntax-check.
- Running --syntax-check on the playbook will verify that it can be ingested by ansible.
- The option can either be before or after the playbook

Example -- ansible-playbook --syntax-check filename.yml

Validating a Playbook

- We can use the -C option to perform a dry run of the playbook execution. This causes Ansible to report what changes would have occurred if the playbook were executed but does not make any actual changes to managed hosts.

- **Example –**
  ansible-playbook –C filename.yml

Outline

- Variables in Ansible
- Host and group variables
- Conditions in Ansible
- Loops in Ansible
- Ansible vaults
- Roles in Ansible

Variables in Ansible

- Ansible supports variables that you can use to store values for reuse throughout an Ansible project.
- This simplifies the creation and maintenance of a project and reduce the number of errors.
- Variables provide a convenient way to manage dynamic values.
- Examples of values that variables might contain:

Users to create, modify or delete.
Software to install or uninstall.
Services to stop, start, or restart.
Files to create, modify, or remove.

Variables in Ansible

- Variables can be defined in multiple ways.
- One common method is to place a variable in a vars block at the beginning of a play:

```
- hosts: all
  vars:
    user_name: saurabh
    user_state: present
```

- It is also possible to define play variables in external files.
- Use vars_files at the start of the play to load variables from a list of files into the play

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

Referencing variable

- After declaring variables, you can use them in tasks.
- Reference a variable (replace it with its value) by placing the variable name in double braces: {{ variable_name }}
- Ansible substitutes the variable with its value when it runs the task

Referencing variable

```yaml
- name: Example play
  hosts: all
  vars:
    user_name: joe
  tasks:
    # This line will read: Creates the user joe
    - name: Creates the user {{ user_name }}
      user:
        - name: "{{ user_name }}"
          state: present
```

Host Variables and Group Variables

- Host variables apply to a specific host.
- Group variables apply to all hosts in a host group or in a group of host groups.
- Host variables take precedence over group variables, but variables defined inside a play take precedence over both.
- Host variables and group variables can be defined

  In the inventory itself.

  In host_vars and group_vars directories in the same directory as the inventory.

  In host_vars and group_vars directories in the same directory as the playbook.

Conditions in Ansible

- The when statement is used to run a task conditionally.
- If the condition is met, the task runs. If the condition is not met, the task is skipped.

Conditions in Ansible

- Equal (value is a string) ansible_machine == "x86_64“
- Equal (value is numeric) max_memory == 512
- Less than min_memory < 128
- Greater than min_memory > 256
- Less than or equal to min_memory <= 256
- Greater than or equal to min_memory >= 512
- Not equal to min_memory != 512
- Variable exists min_memory is define

Conditions in Ansible

• Variable does not exist	min_memory is not defined
• Boolean variable is true. The values of 1, True, or yes evaluate to true.	memory_available
• Boolean variable is false. The values of 0, False, or no evaluate to false.	not memory_available

Ansible Vaults

- Ansible Playbooks sometimes need access to sensitive data like passwords, API keys, other secrets.
- These secrets are often passed to Ansible through variables.
- It is a security risk to store these secrets in plain text files.
- Ansible Vault provides a way to encrypt and decrypt files used by playbooks.
- The ansible-vault command is used to manage these file

Ansible Vaults

- You will need to provide a password to use or manipulate an Ansible Vault encrypted file.
- Create a new encrypted file using the command
  ansible-vault create filename
- View an encrypted file using the command
  ansible-vault view filename
- Edit an encrypted file using the command
  ansible-vault edit filename

Encrypt an existing file

- Encrypt an existing file using the command
  ansible-vault encrypt filename

- Save the encrypted file to a new name using the option
  --output=new_filename

- Decrypt a file with –
  ansible-vault decrypt filename

Roles

- An Ansible role has a defined directory structure with seven main standard directories.
- You must include at least one of these directories in each role. You can omit any directories the role does not use.
- For example: (see figure)

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
    webservers/
        tasks/
        defaults/
        meta/
```

Roles

By default Ansible will look in each directory within a role for a main.yml file for relevant content --

- tasks/main.yml - the main list of tasks that the role executes.
- handlers/main.yml - handlers, which may be used within or outside this role.
- library/my_module.py - modules, which may be used within this role.

Roles

- defaults/main.yml - default variables for the role. These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.
- vars/main.yml - other variables for the role (see Using Variables for more information).
- files/main.yml - files that the role deploys.
- templates/main.yml - templates that the role deploys.
- meta/main.yml - metadata for the role, including role dependencies.

Roles

Storing and finding roles:

By default, Ansible looks for roles in two locations:

- In a directory called roles/, relative to the playbook file
- In /etc/ansible/roles

Roles

You can use roles in three ways:

1. At the play level with the roles option: This is the classic way of using roles in a play.

2. At the tasks level with include_role: You can reuse roles dynamically anywhere in the tasks section of a play using include_role.

3. At the tasks level with import_role: You can reuse roles statically anywhere in the tasks section of a play using import_role.