ANSIBLE

PLAYBOOKS

❖ WORKING WITH PLAYBOOKS:

- Playbooks are the files where ansible code is written. playbooks are written in YAML format.
- Playbooks are one of the core features of ansible and tell ansible what to execute.
- Playbooks are regularly used to **automate IT infrastructure** such as operating systems, networks, security systems, and code repositories like GitHub.
- Playbooks are designed to be human-readable and are developed in a basic text language.
- Ansible uses YAML syntax for expressing ansible playbooks because it is very easy for humans to understand, read and write a than other common data formats like XML or JSON.
- Each playbook is an aggregation of one or more plays in it. Playbooks are structured using plays.
- There can be more than one play inside a playbook.

YAML SYNTAX:

- It provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.
- We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

YAML BASICS

- For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.
- There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with --- and end with .... This is part of the YAML format and indicates the start and end of a document.

➢ EXAMPLE OF AN ANSIBLE PLAYBOOK:

CREATING AN USER ACCOUNT:

Using AD-HOC command:

```
$ansible -m user -a "name=jai uid=1010 state=present" webservers --become
-K
```

Using Playbook:

```
$mkdir playbooks

$cd playbooks

$vi user_account.yml

          ---
          - hosts: webservers
            become: true
            become_user: root
            tasks:
                - name: User Account Creation
                  user:
                     name: jai
                     uid: 1020
                     state: present
          ...
```

CHECK-SYNTAX ERROR:

- Playbooks are expressed in YAML format with a minimum of syntax.
- You can find mistakes in your playbooks with **syntax-check.**

```
$ansible-playbook --syntax-check <playbook_name>

$ansible-playbook –syntax-check user_accout.yml
```

LINTER TOOL:

- A linter is a tool designed to catch data errors before processing a file. One linter specifically designed for Ansible playbooks is Ansible Lint, a readily available Python command-line tool that helps content creators to write, standardize, and package high-quality Ansible content.
- Traditionally, to check for basic syntax errors in an Ansible playbook, you would run the playbook with **--syntax-check.** However, the **--syntax-check** flag is not as comprehensive or in-depth as the ansible-lint tool.

→ **To check syntax error:**

```
$ansible-lint <playbook_name>

$ansible-lint user_account.yml
```

DRY-RUN:

- When ansible-playbook is executed with **--check** it will not make any changes on remote systems.
- Instead, any module instrumented to support 'check mode' will report what changes they would have made rather than making them.

```
$ansible-playbook -c user_account.yml -K
```

RUN ANSIBLE PLAYBOOKS:

- When we are running a playbook, Ansible executes each task in order, one at a time, for all the hosts that we selected. This default behavior could be adjusted according to different use cases using strategies.
- If a task fails, Ansible stops the execution of the playbook to this specific host but continues to others that succeeded. During execution, Ansible displays some information about connection status, task names, execution status, and if any changes have been performed.
- At the end, Ansible provides a summary of the playbook's failures/success.

$ansible-playbook <playbook_name>

```
$ansible-playbook user_account.yml
```

TASKS:

- Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task.
- It is important to understand that, within a play, all hosts are going to get the same task directives. It is the purpose of a play to map a selection of hosts to tasks.
- When running the playbook, which **runs top to bottom**, hosts with failed tasks are taken out of the rotation for the entire playbook. If things fail, simply correct the playbook file and rerun.
- The goal of each task is to execute a module, with very specific arguments.

Example:

```
tasks:
  - name: User Account Creation
    user:
      name: jai
      uid: 1021
      state: present
```

PLAY:

- The play is the element that ties tasks to the servers where they'll run.
- Ansible Playbooks are lists of tasks that automatically execute for your specified inventory or groups of hosts.
- One or more Ansible tasks can be combined to make a play an ordered grouping of tasks mapped to specific hosts and tasks are executed in the order in which they are written.

MANAGE FILES AND FILE PROPERTIES:

- File module is mainly used to deal with files, directories, and symlinks. This module is used to manage properties and set or remove attributes of files, directories, and symlinks.
- For windows systems, Ansible provides a similar module named **win_file**.

CREATE A SIMPLE FILE:

```
- hosts: webservers
  become: true
  become_user: root
  tasks:
      - name: Create a New File
        file:
          path: /opt/testfile
          state: touch
          mode: u=rw,g=r,o=x  ### 0641

$ansible-playbook  file.yml -K
$ls /opt   [From Node Machine]
```

LINEINFILE MODULE: It will manage lines in text files.

```
- hosts: webservers

  become: true

  become_user: root

  tasks:

      - name: Create a New File

        file:

          path: /opt/testfile

          state: touch

          mode: u=rw,g=r,o=x
```

```
- name: line insert at top of file
  lineinfile:
    path: /opt/testFile.txt
    line: 'Added Line 1'
    insertbefore: BOF


    insertafter: EOF
    line: 'Added as last line'
```

CREATING A DIRECTORY:

```
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Create a New Directory
      file:
        path: /opt/cloud
        state: directory
        mode: 0700
```

COPY MODULE:

- The copy module executes a simple copy on the file or directory on the local or on the remote machine.
- To copy files from a local source to a local destination
- To copy files from a remote source to a remote destination (remote_src)
- To copy files from a local source to a remote destination
- You can use the fetch module to copy files from the remote source to local on the other hand.

$vim copyfile.yml

```
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: copy “script.sh” file into destination of /opt
      copy:
        src: script.sh
        dest: /opt/
      tags:
        - simple_copy
```

NOTE: Create a script.sh file in the current location.

HOW TO DISABLE FORCE COPY OF ANSIBLE COPY:

```
copy:
  src: script.sh
  dest: /opt/
  force: no
```

OVERWRITE AND BACKUP THE ORIGINAL FILE:

```
copy:
  src: script.sh
  dest: /opt/
  backup: yes
```

INSTALLING APACHE HTTPD:

```
$vi apache.yml
     ---
     - hosts: webservers
       become: true
       become_user: root
       tasks:
         - name: Installation of HTTPD
           yum:
             name: httpd
             state: present

         - name: Creation of index.html page
           copy:
             content: "WELCOME TO DEVOPS"
             dest: /var/www/html/index.html

         - name: Starting HTTPD Service
           service:
             name: httpd
             state: started
             enabled: true
     ...
$ansible-playbook --syntax-check apache.yaml
$ansible-playbook apache.yml
```

MULTI-PLAYS EXAMPLE:

INSTALLATION OF HTTPD & FIREWALLD:

```
$vi multiplay.yml
---
- hosts: webservers
  become: true
  become_user: root
  tasks:
    - name: Installation of HTTPD & Firewalld Packages
      yum:
        name:
         - httpd
         - firewalld

    - name: Creating index.html
      copy:
        content: "WELCOME TO DEVOPS\n"
        dest: /var/www/html/index.html

    - name: Firewalld Start & Enabled
      service:
        name: firewalld
        state: started
        enabled: true
    - name: Firewalld permits access to httpd service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes
    - name: Httpd Start & Enabled
      service:
        name: httpd
        state: started
        enabled: true

- name: Test Intranet Web Server Connectivity
  hosts: webservers
  become: no
  tasks:
    - name: connect to Intranet Web Server
      uri:
        url: http://Node1
        return_content: yes
        status_code: 200
```

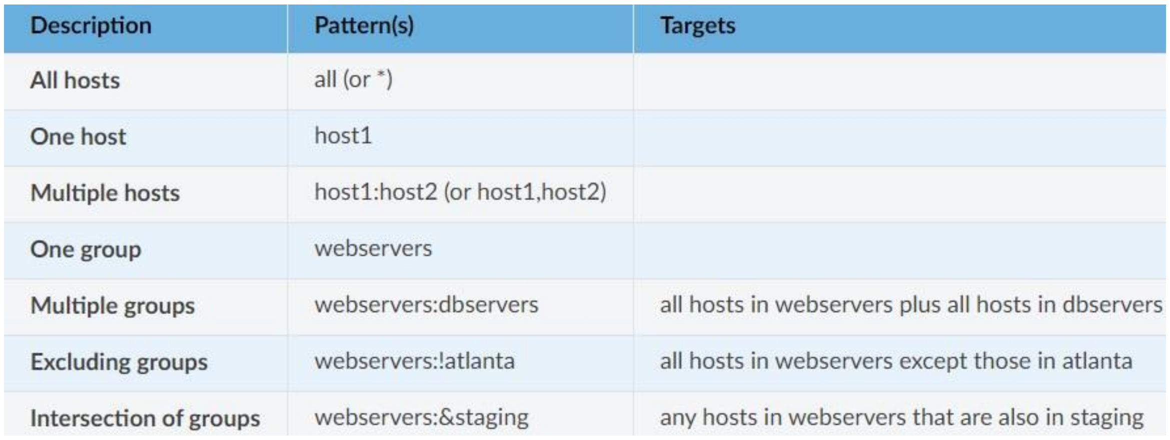
```
$ansible-playbook --syntax-check multiplay.yml --step
$ansible-playbook multiplay.yml -K --step
$curl  http://Node1          [To check output]
```

STEP MODE:

- To execute a playbook interactively, use **--step**.
- With this option, Ansible stops on each task and asks if it should execute that task.
- Answer **"y"** to execute the task, answer **"n"** to skip the task, and answer **"c"** to exit step mode, executing all remaining tasks without asking.

➢ PATTERNS: TARGETING HOSTS AND GROUPS:

- When you execute Ansible through an ad hoc command or a playbook, you must choose which managed nodes or groups you want to execute against.
- Patterns let you run commands and playbooks against specific hosts and/or groups in your inventory.
- An Ansible pattern can refer to a single host, an IP address, an inventory group, a set of groups, or all hosts in your inventory.
- Patterns are highly flexible - you can exclude or require subsets of hosts, use wildcards or regular expressions, and more.
- Bellow table lists common patterns for targeting inventory hosts and groups.



Description	Pattern(s)	Targets
All hosts	all (or *)	
One host	host1	
Multiple hosts	host1:host2 (or host1,host2)	
One group	webservers	
Multiple groups	webservers:dbservers	all hosts in webservers plus all hosts in dbservers
Excluding groups	webservers:!atlanta	all hosts in webservers except those in atlanta
Intersection of groups	webservers:&staging	any hosts in webservers that are also in staging

LIMIT ARGUMENT:

- We can use the **--limit** flag to limit the Playbook's execution to specific host

```
$ansible-playbook apache.yml --limit Node1
```