Working with Ansible Playbook

YAML

- YAML is a human-readable data serialization standard that can be used in conjunction with all programming languages and is often used to write configuration files.
- The recursive YAML acronym stands for "YAML Ain't Markup Language," denoting it as flexible and data-oriented.

YAML Comments

- Comments can also be used to aid readability. In YAML, everything to the right of the number or hash symbol (#) is a comment. If there is content to the left of the comment, precede the number symbol with a space.

```
# this is yaml file
Ansible Automation # this way also comment can be used
```

YAML Strings

- Strings in YAML do not normally need to be put in quotation marks even if there are spaces contained in the string. You can enclose strings in either double quotes or single quotes.

```
This is YAML string
'This is YAML string'
"This is YAML string"
```

- There are two ways to write multiline strings. You can use the vertical bar (|) character to denote that newline characters within the string are to be preserved.

```
Multiple_line |
        Line 1
        Line 2
        Line 3
```

- You can also write multiline strings using the greater-than (>) character to indicate that newline characters are to be converted to spaces and that leading white spaces in the lines are to be removed. This method is often used to break long strings at space characters so that they can span multiple lines for better readability.

```
Multiple_line:  >
        Line 1
        Line 2
        Line 3
```

YAML Dictionaries

- Collections of key-value pairs written as an indented block, as follows:

```
name: svcrole
svcservice: httpd
svcport: 80
```

- Dictionaries can also be written in an inline block format enclosed in curly braces, as follows:

```
{name: svcrole, svcservice: httpd, svcport: 80}
```

YAML Lists

- Lists written with the normal single-dash syntax:

```
Hosts:
 -  Server1
 -  Server2
 - Server3
```

Obsolete key=value Playbook Shorthand

- Some playbooks might use an older shorthand method to define tasks by putting the key-value pairs for the module on the same line as the module name. For example, you might see this syntax:

```
tasks:
 - name: shorthand form
   service: name=httpd enabled=true state=started
```

- Normally you would write the same task as follows:

```
tasks:
 tasks:
  - name: normal form
    service:
     name: httpd
     enabled: true
     state: started
```

ANSIBLE PLAYBOOKS

- Ansible Playbook run using the **ansible-playbook** command.
- Ad hoc commands can run a single, simple task against a set of targeted hosts as a one-time command.
- The real power of Ansible, however, is in learning how to use playbooks to run multiple, complex tasks against a set of targeted hosts in an easily repeatable manner.
- A *play* is an ordered set of tasks run against hosts selected from your inventory.
- A *playbook* is a text file containing a list of one or more plays to run in a specific order.
- Plays allow you to change a lengthy, complex set of manual administrative tasks into an easily repeatable routine with predictable and successful outcomes.
- In a playbook, you can save the sequence of tasks in a play into a human-readable and immediately runnable form.

If you use the **vi or vim** text editor, you can apply some settings which might make it easier to edit your playbooks.
Create a file in user home location with name **.vimrc** add the following line.
When **vi** detects that you are editing a YAML file, it performs a 2-space indentation when you press the **Tab** key and auto indents subsequent lines.

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

Writing an Ansible Playbook

- A playbook is a text file written in YAML format, and is normally saved with the extension **yml**.
- The playbook uses indentation with space characters to indicate the structure of its data. YAML does not place strict requirements on how many spaces are used for the indentation, but there are two basic rules.
  - Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation.
  - Items that are children of another item must be indented more than their parents.
- A playbook begins with a line consisting of three dashes (**---**) as a start of document marker.

- It may end with three dots (**...**) as an end of document marker, although in practice this is often omitted. In between those markers, the playbook is defined as a list of plays.
- The play itself is a collection of key-value pairs. Keys in the same play should have the same indentation. The following example shows a YAML snippet with three keys. The first two keys have simple values. The third has a list of three items as a value.

```
---
name: just an example
hosts: webservers
tasks:
- first
- second
- third
```

- The **name** key is optional, but is recommended because it helps to document your playbook. This is especially useful when a playbook contains multiple plays.
- The second key in the play is a **hosts** attribute, which specifies the hosts against which the play's tasks are run.
- Finally, the last key in the play is the **tasks** attribute, whose value specifies a list of tasks to run for this play.

$ ansible wevserver -m user -a "name=newbie uid=4000 state=present"

```
---
- name: Creating Users
  hosts: all
  become: yes
    tasks:
    - name: Create an user with UID 2020
      user:
        name: anaya
        uid: 2020
        state: present
```

- The **tasks** attribute is the part of the play that actually lists, in order, the tasks to be run on the managed hosts. Each task in the list is itself a collection of key-value pairs.
- In this example, the only task in the play has two keys:
    - **name:** is an optional label documenting the purpose of the task. It is a good idea to name all your tasks to help document the purpose of each step of the automation process.

- **user:** is the module to run for this task. Its arguments are passed as a collection of key-value pairs, which are children of the module (name, uid, and state).

Debug Module:

- This module is used for printing message during execution and useful for debugging variable or expression.
- The debug module provides insight into what is happening in the play. This module can display the value for a certain variable at a certain point in the play. This feature is key to debugging tasks that use variables to communicate with each other
- **Debug modules has three parameters**

1. msg: to print message.

```
---
 - name:
   hosts: localhost
   tasks:
    - debug: msg="welcome to Ansible Class"
```

```
---
 - name:
   hosts: localhost
   tasks:
    - debug:
       msg:
              - " welcome to Ansible Playbook"
              - " welcome to Ansible Classroom"
```

2. **var:** use to print a variable value.

```
---
 - name: Examples of Debug Vars
   hosts: all
   tasks:
          - name: Display Variable
            debug:
                   var: inventory_hostname
```

3. Verbosity

- The default output provided by the **ansible-playbook** command does not provide detailed task execution information. The **ansible-playbook -v** command provides additional information, with up to four total levels.

Configuring the Output Verbosity of Playbook Execution	**Description**
-v	The task results are displayed.
-vv	Both task results and task configuration are displayed
-vvv	Includes information about connections to managed hosts
-vvvv	Adds extra verbosity options to The connection plug-ins, including users being used in the managed hosts to execute scripts, and what scripts have been executed.

```
---
 - name: Learning debug module  Verbosity
   hosts: localhost
   tasks:
     - name: Vebosity default
       debug:
         msg: "this is default verbosity, value is 0"
     - name: verbosity level 1
       debug:
          msg: " This is verbosity level 1"
          verbosity: 1
     - name: verbosity level 2
       debug:
          msg: " This is verbosity level 2"
          verbosity: 2
```

Running Playbooks

- The **ansible-playbook** command is used to run playbooks. The command is executed on the control node and the name of the playbook to be run is passed as an argument:
- *$ ansible-playbook playbook_name*

Note: That the value of the **name** key for each play and task is displayed when the playbook is run.
The **Gathering Facts** task is a special task that the setup module usually runs automatically at the start of a play. For playbooks with multiple plays and tasks, setting **name** attributes makes it easier to monitor the progress of a playbook's execution.

In general, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple times. If the targeted managed hosts are already in the correct state, no changes should be made.

Syntax check

- With the help of check, you can verify the playbook syntax.
- It is a good practice to use syntax check before executing your playbook to ensure that your syntax are correct.
- ***$ ansible-playbook --syntax-check playbook.yml***

Executing a Dry Run

- You can use the **-C** option to perform a *dry run* of the playbook execution.
- This will give you to see the output without affecting your managed host. Means your playbook will execute but does not make any changes in managed hosts.
- *$* ***ansible-playbook -C playbook.yml***

```
---
 - name: Installing httpd and enabling
   hosts: localhost
   gather_facts: false
   become: true
   tasks:                                  # Task 1
     - name: Installing Httpd
       yum:
         name: httpd
         state: present
     - name: Adding content                # Task 2
       copy:
        content: "Welcome to Ansible web page"
        dest: /var/www/html/index.html
     - name: Starting service              # Task 3
       service:
         name: httpd
         state: started
```

Verify: *curl://ipaddress*

Multiple Plays

- A playbook is a YAML file containing a list of one or more plays.
- Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory. Therefore, if a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts.
- This can be very useful when orchestrating a complex deployment which may involve different tasks on different hosts.
- You can write a playbook that runs one play against one set of hosts, and when that finishes runs another play against another set of hosts.
- Writing a playbook that contains multiple plays is very straightforward.
- Each play in the playbook is written as a top-level list item in the playbook.
- Each play is a list item containing the usual play keywords.

```
---
 - name: First Task
   hosts: localhost
   become: true
   tasks:
      - name: uninstalling httpd
        yum:
          name: httpd
          state: absent
 - name: Second Play
   hosts: 192.168.100.9
   tasks:
     - name: Copy file with owner and permissions
       copy:
         src: /etc/hosts
         dest: /tmp/
         owner: ansadmin
         group: ansadmin
         mode: '0600'
```

HAPPY LEARNING