Contents

Ansible Playbooks and Ad Hoc Commands

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. The tasks themselves, because of the way in which they are written, document the steps needed to deploy your application or infrastructure.

Formatting an Ansible Playbook

To help you understand the format of a playbook, review this ad hoc command from a previous chapter:

```
[student@workstation ~]$ ansible -m user -a "name=newbie uid=4000
state=present" \
> servera.lab.example.com
```

This can be rewritten as a single task play and saved in a playbook. The resulting playbook appears as follows:

```
---
- name: Configure important user consistently
  hosts: servera.lab.example.com
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
```

```
        uid: 4000
        state: present
```

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.

You can also add blank lines for readability.

Only the space character can be used for indentation; tab characters are not allowed.

If you use the **vi** text editor, you can apply some settings which might make it easier to edit your playbooks. For example, you can add the following line to your `$HOME/.vimrc` file, and when **vi** detects that you are editing a YAML file, it performs a 2-space indentation when you press the **Tab** key and autoindents subsequent lines.

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

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. An item in a YAML list starts with a single dash followed by a space. For example, a YAML list might appear as follows:

```
- apple
- orange
- grape
```

In , the line after `---` begins with a dash and starts the first (and only) play in the 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 original example play has three keys, `name`, `hosts`, and `tasks`, because these keys all have the same indentation.

The first line of the example play starts with a dash and a space (indicating the play is the first item of a list), and then the first key, the `name` attribute. The `name` key associates an arbitrary string with the play as a label. This identifies what the play is for. 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.

```
- name: Configure important user consistently
```

The second key in the play is a `hosts` attribute, which specifies the hosts against which the play's tasks are run. Like the argument for the **ansible** command, the `hosts` attribute takes a host pattern as a value, such as the names of managed hosts or groups in the inventory.

```
  hosts: servera.lab.example.com
```

Finally, the last key in the play is the `tasks` attribute, whose value specifies a list of tasks to run for this play. This example has a single task, which runs the `user` module with specific arguments (to ensure user `newbie` exists and has UID 4000).

```
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        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`).

The following is another example of a `tasks` attribute with multiple tasks, using the `service` module to ensure that several network services are enabled to start at boot:

```
  tasks:
    - name: web server is enabled
      service:
        name: httpd
        enabled: true

    - name: NTP server is enabled
      service:
```

```
        name: chronyd
        enabled: true

    - name: Postfix is enabled
      service:
        name: postfix
        enabled: true
```

The order in which the plays and tasks are listed in a playbook is important, because Ansible runs them in the same order.

The playbooks you have seen so far are basic examples, and you will see more sophisticated examples of what you can do with plays and tasks as this course continues.

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:

```
[student@workstation ~]$ ansible-playbook site.yml
```

When you run the playbook, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.

The following example shows the contents of a simple playbook, and then the result of running it.

```
[student@workstation playdemo]$ cat webserver.yml
---
- name: play to setup web server
  hosts: servera.lab.example.com
  tasks:
  - name: latest httpd version installed
    yum:
      name: httpd
      state: latest
...output omitted...
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server]
**********************************************

TASK [Gathering Facts]
*****************************************************
ok: [servera.lab.example.com]

TASK [latest httpd version installed]
**************************************
changed: [servera.lab.example.com]

PLAY RECAP
*********************************************************************
```

```
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

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. This is covered later in the course.) For playbooks with multiple plays and tasks, setting `name` attributes makes it easier to monitor the progress of a playbook's execution.

You should also see that the `latest httpd version installed` task is `changed` for `servera.lab.example.com`. This means that the task changed something on that host to ensure its specification was met. In this case, it means that the httpd package probably was not installed or was not the latest version.

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. For example, assume that the playbook from the previous example is run again:

```
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server]
*********************************************

TASK [Gathering Facts]
*********************************************************
ok: [servera.lab.example.com]

TASK [latest httpd version installed]
***************************************
ok: [servera.lab.example.com]

PLAY RECAP
*********************************************************************
servera.lab.example.com    : ok=2    changed=0    unreachable=0    failed=0
```

This time, all tasks passed with status `ok` and no changes were reported.

Increasing Output 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.

Option	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

Syntax Verification

Prior to executing a playbook, it is good practice to perform a verification to ensure that the syntax of its contents is correct. The **ansible-playbook** command offers a `--syntax-check` option that you can use to verify the syntax of a playbook. The following example shows the successful syntax verification of a playbook.

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml

playbook: webserver.yml
```

When syntax verification fails, a syntax error is reported. The output also includes the approximate location of the syntax issue in the playbook. The following example shows the failed syntax verification of a playbook where the space separator is missing after the `name` attribute for the play.

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to have been in ...output omitted... line 3, column 8, but
may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

- name:play to setup web server
  hosts: servera.lab.example.com
       ^ here
```

Executing a Dry Run

You 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.

The following example shows the dry run of a playbook containing a single task for ensuring that the latest version of httpd package is installed on a managed host. Note that the dry run reports that the task would effect a change on the managed host.

```
[student@workstation ~]$ ansible-playbook -C webserver.yml

PLAY [play to setup web server]
*********************************************

TASK [Gathering Facts]
*****************************************************
ok: [servera.lab.example.com]
```

```
TASK [latest httpd version installed]
*****************************************
changed: [servera.lab.example.com]

PLAY RECAP
*********************************************************************
servera.lab.example.com    : ok=2    changed=1    unreachable=0    failed=0
```

`ansible-playbook`(1) man page

Source: https://rha.ole.redhat.com/rha/app/courses/rh294-8.0/pages/ch03