4. Using playbooks

4. Using playbooks

Ansible playbooks are YAML files that use a special keywords to inform Ansible what to do. Ansible works its way through a list of commands and arguments.
Playbooks can be more closely described as a model of a system than rather than to be described as a programming language or script. Each playbook has one or more 'plays'. A 'play' maps a group of hosts to some roles, this is called tasks. A task is a call to an Ansible module, or a custom module.

4.1 Compose a playbook

A playbook is written in YAML, because of this you need to follow all of the same rules and standards a YAML file does. For example: you might find that the whitespace sensitivity of YAML takes some getting used to, especially because an indent is 2 spaces.
In a playbook there can be a quite a lot of plays that can affect your system to do different things. The first thing you have to do in a playbook is tell Ansible where this playbook should run. This has to be done by specifying a host group. The line where host is defined is a list of one or more groups or hosts. Your playbook should look like this:

```
---
-  hosts: all
```

Ansible now knows where to run, you now can tell what you want to run. This has to be done by adding a tasks section. Inside this section you specify the several tasks Ansible has to run. For example, you can use the task ping to make sure your hosts are connected to each other:

```
 ---
- hosts: all
  tasks:
    -  ping:
```

4.1.1 Tasks list

Each play contains a list of tasks. The tasks are executed one at a time against all machines, that match the defined host group. A playbook runs from top to bottom, when a task failed the hosts, where the task failed, are taken out of the playbook. If it fails, simply correct your playbook and run it again.
The goal of a task is to execute a module, with declared arguments. A playbook should be idempotent, this means that running the playbook multiple times would provide the same output/effect it should have when it runs for the first time. The command and shell modules will return the same output every time, this is because the same command is used for example, chmod.

A task should always have a name to specify it, the name is included in the output from running the playbook. A basic task looks like this:

```
tasks:
 - name: run apache
   service:
     name: httpd
     state: started
```

if you want to execute a command or shell line it will look like this:

```
tasks:
  - name: enable selinux
    command: /sbin/setenforce 1
```

or this:

```
tasks:
  - name: run this command and ignore the result
    shell: /usr/bin/somecommand || /bin/true
```

4.1.2 Handlers

As mentioned before, a playbook should be idempotent and can relay when they made changes. A playbook recognizes this and can respond to these changes. These notifications are triggered at the end of each block of tasks.
For example, multiple resources indicate that nginx should restart because of a change in a config file, but nginx will only restart once to avoid unnecessary restarts. Here is an example of it:
tasks:

```
- name: Install nginx
    package:
     name: nginx
     state: present
    notify:
     Start nginx
```

The thing listed in the notify section is called a handler. Handlers are a list of tasks, it is not very different from the regular tasks. These handlers are only run once, regardless how many tasks notify a particular handler. The handlers are defined in a section, it would look like this:

```
handlers:
 - name: Start nginx
    service:
     name: nginx
     state: started
```

If you want to use variables in handlers, you can’t use them in names. This is because Ansible may not have a value available for a handler name. Handlers can also “listen” to generic topics and tasks can notify those topics:

```
handlers:
    - name: restart memcached
      service:
        name: memcached
        state: restarted
      listen: "restart web services"
    - name: restart apache
      service:
        name: apache
        state: restarted
      listen: "restart web services"
 tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

This use makes it easier to trigger multiple handlers. It also makes it easier to share handlers among playbooks and roles. Roles are described in a later chapter.