



Ansible

CMXXXX

Lecture XX - IaC part 3

Dom Routley



Objectives

- Know what Ansible is
- Know why we are using Ansible over the alternatives
- Have looked at an example of using Ansible to configure an Apache2 web server

IaC part 3

What is Ansible





Ansible background

- Owned by RedHat
- Agentless operation (no need for program on target machines)
- Built of modules
- Modules are idempotent
- Built in Python

Why are we using
Ansible over the
alternatives





Why Ansible

Ansible is one of the simplest system provisioning programs that is currently available.

Unlike many of its competitors it is agentless and is therefore perfect for the initial provisioning of short lived cloud instances.

Examples and
explanations





Config file

Settings in Ansible are adjustable via a config file. The default config should be sufficient for most users, but there may be reasons you would want to change it.

Changes can be made and used in a user defined configuration file which will be processed in the following order:

- `ANSIBLE_CONFIG` (an environment variable)
- `ansible.cfg` (in the current directory)
- `.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg` (This is the stock config file)




Config file

This is a basic user created config file.

Ansible looks in the section headed [defaults] for most of its configuration.

This sets the location that Ansible will look to fill its inventory of hosts. By convention this is called inventory or hosts

```
1  [defaults]
2  inventory = hosts
3  remote_port = 22
4  host_key_checking = False
```



hint: you should not need a more complex config file than this for your coursework



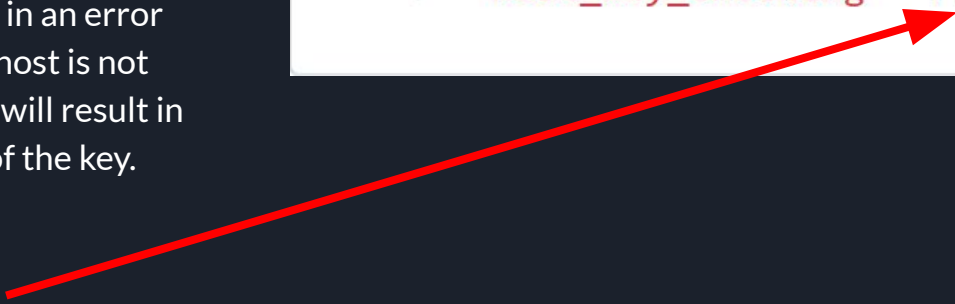

Config file

This just confirms that we will be using the SSH port 22.

If a host is reinstalled and has a different key in 'known_hosts', this will result in an error message until corrected. If a host is not initially in 'known_hosts' this will result in prompting for confirmation of the key.

To prevent this you can set hosts_key_checking to False

```
1  [defaults]
2  inventory = hosts
3  remote_port = 22
4  host_key_checking = False
```





Hosts file

The hosts file contains a list of all of the “targetable” endpoints for Ansible to talk to.

These endpoints can be described or grouped under titles with square brackets so that Ansible can hit multiple hosts at once with the same configuration



```
1  ---
2  [servers]
3  192.0.2.52
```

Hosts file - children tags

Host file titles allow you to group hosts and refer to multiple titles at once without having to write down the address multiple times.

This is done by using the :children suffix to the title, the titles of the “children” groups can then be listed

```
1 ---
2 [servers:children]
3 webServers
4 dbServers
5 cicdServers
6
7 [webServers:children]
8 europeWebCluster
9 asiaWebCluster
10 americaWebCluster
11
12 [europeWebCluster]
13 192.0.2.11
14 192.0.2.12
15
```



A red arrow points from the `[servers:children]` tag on line 2 to the `[webServers:children]` tag on line 7. Another red arrow points from the `[webServers:children]` tag to the `[europeWebCluster]` tag on line 12. The `[europeWebCluster]` tag is circled in red, and the IP addresses `192.0.2.11` and `192.0.2.12` on lines 13 and 14 are also circled in red.

```
15
16 [asiaWebCluster]
17 192.0.2.13
18 192.0.2.14
19
20 [americaWebCluster]
21 192.0.2.15
22 192.0.2.16
23
24 [dbServers]
25 192.0.2.21
26
27 [cicdServers]
28 192.0.2.31
```



Hosts file - children tags

Children groups do not have to replace simply listing addresses.

They can also be used in conjunction with them.

In this example the “servers” group refers to four addresses, but you can also just refer to the “webServers” group if wished

```
1  ---
2  [servers]
3  192.0.2.13
4  192.0.2.14
5
6  [servers:children]
7  webServers
8
9  [webServers]
10 192.0.2.11
11 192.0.2.12
```



Hosts file - children tags - Why?

For example, you could have two web servers and a backend database server. 90% of the changes you want to make are made to all three, but you don't want to install Apache on your database server, or SQL on your web servers.

You could write three different playbooks, one that called all of the servers to do the 90% setup, and one each for the database and webserving specific tasks.

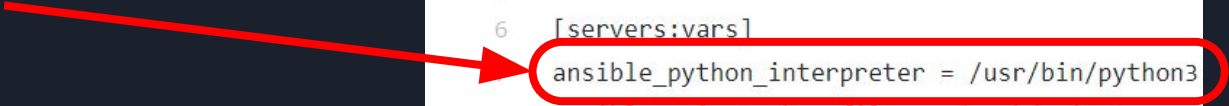


Hosts file - host variables

A hosts file can also contain variables that are specified against specific server groups.

Here the interpreter that Ansible will use on the remote machines is being set to Python3, by default the interpreter is set to Python2.7

```
1  ---
2  [servers]
3  192.0.2.11
4  192.0.2.12
5
6  [servers:vars]
7  ansible_python_interpreter = /usr/bin/python3
8  ansible_private_key_file = ~/.ssh/privateKey
```






Hosts file - host variables

You can also set the private key file that Ansible will use to SSH into the target machine/s.

This is useful if you have multiple machines that are going to be provisioned with the same playbook, but have different SSH keys

```
1  ---
2  [servers]
3  192.0.2.11
4  192.0.2.12
5
6  [servers:vars]
7  ansible_python_interpreter = /usr/bin/python3
   ansible_private_key_file = ~/.ssh/privateKey
```



Definition (playbook) files

Playbooks are the files that Ansible uses to define what it should do.

They are generally made up of two sections.

1. The settings/variables preamble
2. Tasks

```
1
2
3  - name: Setting up my server
4    hosts: servers
5    become: yes
6    remote_user: ubuntu
7
8  tasks:
9
10   - name: Update system packages
11     raw: sudo apt update -y
12
13   - name: Install Apache2
14     apt:
15       name: apache2
16       state: latest
17
18   - name: Make sure Apache is running
19     systemd:
20       name: apache2
21       state: started
```

Preamble/settings/header

This line tells Ansible what group of hosts to target with this playbook

This line tells Ansible that it should use admin privileges (sudo) when executing commands

This line tells Ansible what user to login to the target servers with

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6
7  tasks:
8    - name: Update system packages
9      raw: sudo apt update -y
10
11   - name: Install Apache2
12     apt:
13       name: apache2
14       state: latest
15
16   - name: Make sure Apache is running
17     systemd:
18       name: apache2
19       state: started
```



Modules

In Ansible, Modules are pre-written chunks of code that can be executed by Ansible on a target machine.

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6
7    tasks:
8      - name: Update system packages
9        raw: sudo apt update -y
10
11      - name: Install Apache2
12        apt:
13          name: apache2
14          state: latest
15
16      - name: Make sure Apache is running
17        systemd:
18          name: apache2
19          state: started
```



Modules

An example of a module being used is the apt module in this code.

All modules have specific parameters that you can set to ensure that they do what you want them to do.

Here we are telling the apt module to make sure that apache2 is at the latest version.

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6
7    tasks:
8      - name: Update system packages
9        raw: sudo apt update -y
10
11      - name: Install Apache2
12        apt:
13          name: apache2
14          state: latest
15
16      - name: Make sure Apache is running
17        systemd:
18          name: apache2
19          state: started
```



Modules

One of the most simple modules is the raw module.

This module allow you to inject commands directly into the target machines terminal/runtime environment.

This module is not recommended, but is useful if you have to do something that cannot be achieved with a module.

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6
7    tasks:
8      - name: Update system packages
9        raw: sudo apt update -y
10
11     - name: Install Apache2
12       apt:
13         name: apache2
14         state: latest
15
16     - name: Make sure Apache is running
17       systemd:
18         name: apache2
19         state: started
```



A bit more about modules

There are literally thousands of modules available for Ansible that can do everything from copying files to a remote machine to retrieving tokens from AWS.

You can also build your own modules, most of the modules listed on Ansibles' module index are community built and supported however some core modules are built and maintained by the Ansible team.

All modules can be found at this link:

https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html

Input variables





Input variables

Input variables in Ansible are different than standard internal variables. (Although you can set internal variables with input variables).

Input variable names are not predetermined like the internal variables (like `ansible_private_key_file`) and so can be named whatever you wish.

Input variables can then be used to set things in Ansibles runtime environment that are not included in static files.



Declaring variables in the playbook file

Variables can be simply declared in the playbook file under the vars heading.

Here the internal variable defining what http_port should be used by Ansible is being set to 80.

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6    vars:
7      - http_port: 80
8
```



Using vars_prompt

The vars_prompt section will get Ansible to prompt the user at runtime for an input.

In this example the input is then stored in the variable http_port, the prompt given to the user can be any string of characters.

```
1  ---
2  - name: Setting up my server
3    hosts: servers
4    become: yes
5    remote_user: ubuntu
6    vars_prompt:
7      - name: "http_port"
8        prompt: "HTTP Port number"
9
```



Command line arguments

Command line arguments can also be passed to Ansible that can set variables.

In this example the internal variable `http_port` is being set to 80

Multiple variables can be set at once this way, you simply need to include the key value pairs of `variable_name=variable` in the speech marks

```
10:01 $ ansible-playbook main.yml --extra-vars "http_port=80"
```



Localhost

Worth noting is that Ansible can be run against your local machine (the one running Ansible) using the host “localhost”.

```
1  ---
2  - hosts: localhost
3    vars_prompt:
4      - name: "dev_ip"
5        prompt: "Development server IP"
6    tasks:
7
8      - name: Add development server IP to hosts
9        add_host:
10          name: "{{ dev_ip }}"
11          group: development
12
13
14     - name: Webserver setup
15       hosts: webserver
16       become: yes
17       remote_user: ubuntu
18
19     tasks:
20
21       - name: Update system packages
22         raw: sudo apt update -y
```



Input variables

You can also merge multiple ways of inputting variables into Ansible.

In this example the custom variable `dev_ip` is being prompted for in the first play of this playbook.

It is worth noting that `vars_prompt` will not run if the custom variable has already been filled by some other means (say, by command line arguments)

```
1  ---
2  - hosts: localhost
3    vars_prompt:
4      - name: "dev_ip"
5        prompt: "Development server IP"
6    tasks:
7
8      - name: Add development server IP to hosts
9        add_host:
10          name: "{{ dev_ip }}"
11          group: development
12
13
14 - name: Webserver setup
15   hosts: webservers
16   become: yes
17   remote_user: ubuntu
18
19   tasks:
20
21     - name: Update system packages
22       raw: sudo apt update -y
```



Input variables

The singular task of the first play in this playbook is to use a module called `add_host` that will add a host to Ansibles internal host memory.

This will NOT add the host to the hosts file.

```
1  ---
2  - hosts: localhost
3    vars_prompt:
4      - name: "dev_ip"
5        prompt: "Development server IP"
6    tasks:
7
8      - name: Add development server IP to hosts
9        add_host:
10          name: "{{ dev_ip }}"
11          group: development
12
13
14  - name: Webserver setup
15    hosts: webservers
16    become: yes
17    remote_user: ubuntu
18
19    tasks:
20
21      - name: Update system packages
22        raw: sudo apt update -y
```



Input variables

In this example we assume that the webserver host group includes as a child the development group.

The development IP may not be static, and therefore we can set the IP when we run Ansible and not have to edit any files.

This also shows an example of multiple plays in one playbook, the plays will run sequentially so a lower one can rely on settings and changes made higher up.

```
1  ---
2  - hosts: localhost
3    vars_prompt:
4      - name: "dev_ip"
5        prompt: "Development server IP"
6    tasks:
7
8      - name: Add development server IP to hosts
9        add_host:
10          name: "{{ dev_ip }}"
11          group: development
12
13
14  - name: Webserver setup
15    hosts: webserver
16    become: yes
17    remote_user: ubuntu
18
19    tasks:
20
21      - name: Update system packages
22        raw: sudo apt update -y
```