CHECK POINT™

Automation with Ansible

Task Execution with Ansible

Khalid Al-Shawwaf | Solutions Architect

March 2023

YOU DESERVE THE BEST SECURITY

Agenda

- Host Groups
- Ad-hoc Commands
- YAML Syntax
- Working with Playbooks

1

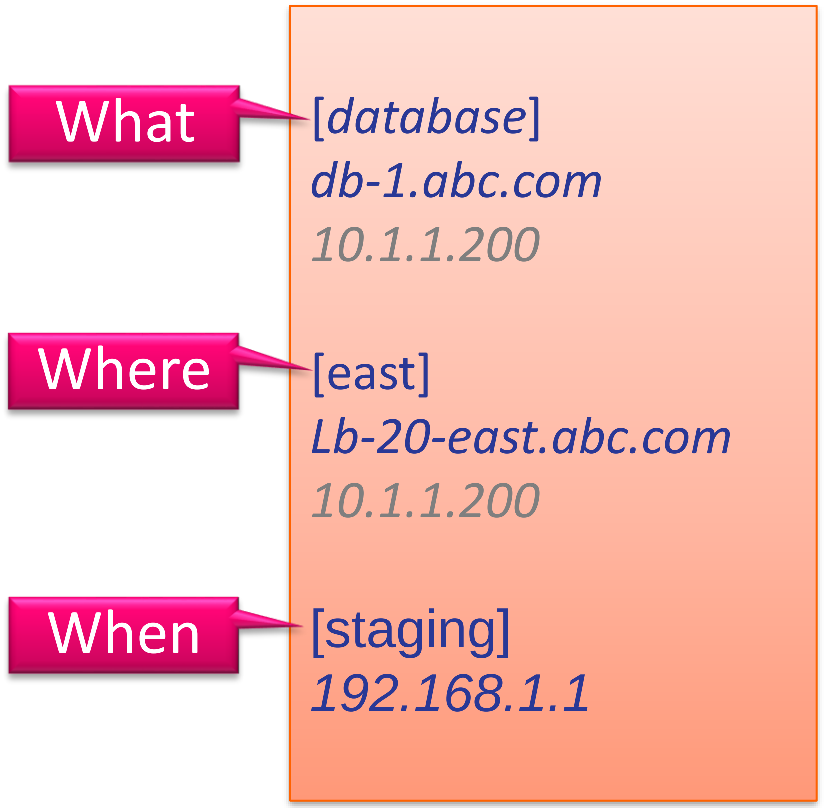
HOST GROUPS

Host Groups

- Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as **Inventory**.
- Once your inventory is defined, you use patterns to select the hosts or groups you want Ansible to run against.
- Groups are defined -*when using INI format*- using brackets “**[group_name]**”.
- There are two default groups: **all** and **ungrouped**.
- The all group contains every host. The ungrouped group contains all hosts that don’t have another group aside from all.
- Every host will always belong to at least 2 groups ((*all and ungrouped*) or (*all and some other group*)).
- You can put each host in more than one group.

Host Groups

- For example a production webserver in a datacenter in the east region might be included in groups called [**production**] and [**east_us**] and [**webservers**].
- You can create groups that track:
- What - An application, stack or microservice.
  - For example, ***database*** servers, *web* servers.
- Where - A datacenter or *region*, to talk to local DNS, storage.
  - For example, **east**, west.
- When - The development *stage*, to avoid testing on production resources.
  - For example, *production*, *test*, ***staging***.



Host Groups

- More advanced features are available for advanced inventory configuration including ranges and variables.

```
[cp_mgmt]
db-[1-5].abc.com
10.1.1.200

[staging]
192.168.1.1 http_port=80

[cp_mgmt:vars]
ansible_httpapi_use_ssl=True
ansible_httpapi_validate_certs=False
ansible_user=admin
ansible_password=vpn123
```

Include a range of hosts from db-1.abc.com to db-5.abe.com

Defining the variable http_port for a single host

Group variable. Assigning variables for all hosts in the database group

2

AD-HOC COMMANDS

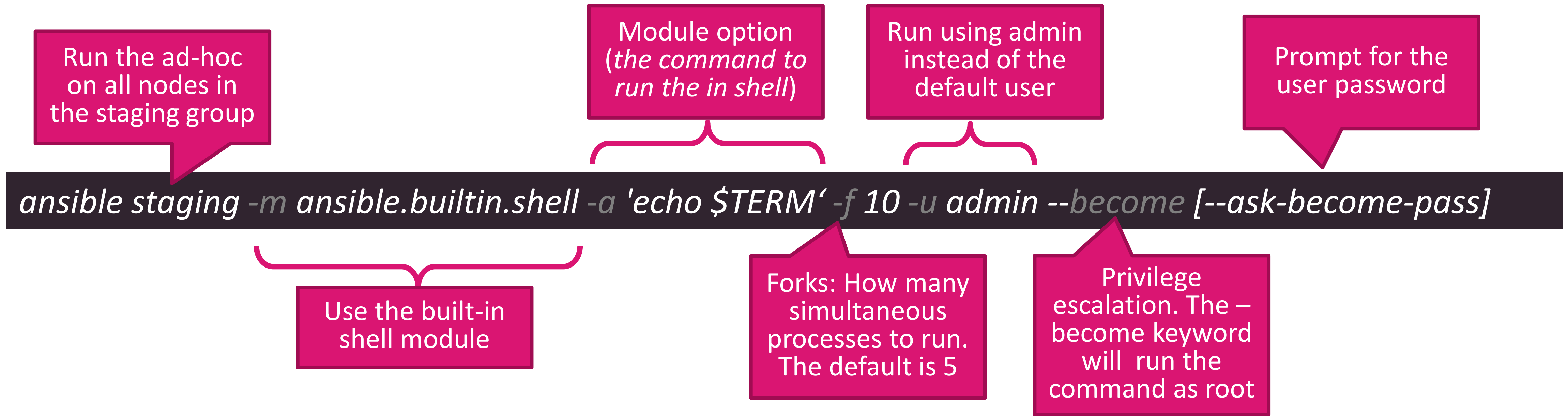
Ad-hoc commands

- An Ansible ad-hoc command uses the /usr/bin/ansible command-line tool to automate a single task on one or more managed nodes.
- Ad-hoc commands are **quick** and **easy**, but they are **not reusable**.
- Ad-hoc commands are great for tasks you repeat rarely.
- For example, if you want to power off all the machines in your lab, you could execute a quick one-liner in Ansible without writing a playbook.
- You can use any Ansible module in an ad-hoc task.
- An ad-hoc command looks like this:

```
ansible [pattern] -m [module] -a "[module options]"
```

Ad-hoc Commands

- The example below shows the command to reboot all the server in the staging group. This group must be defined in the inventory.



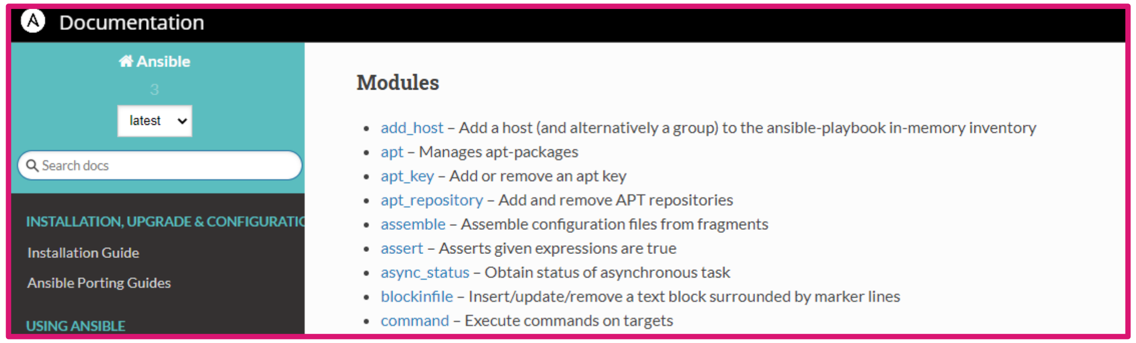
```
ansible staging -m ansible.builtin.shell -a 'echo $TERM‘ -f 10 -u admin --become [--ask-become-pass]
```

Run the ad-hoc on all nodes in the staging group
Module option (*the command to run the in shell*)
Run using admin instead of the default user
Prompt for the user password
Use the built-in shell module
Forks: How many simultaneous processes to run. The default is 5
Privilege escalation. The –become keyword will run the command as root

https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html

Ad-hoc Commands

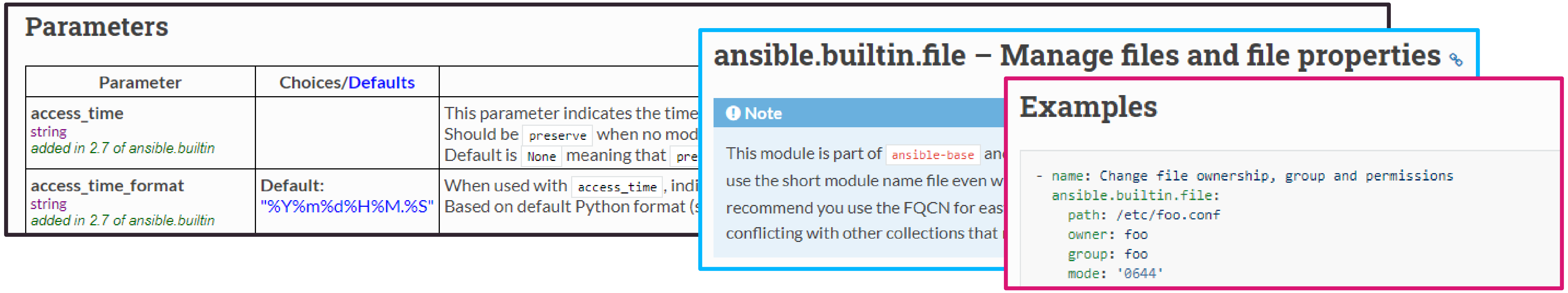
- The documentation site (https://docs.ansible.com) contains all the built-on modules and plugins.



Modules

- add_host – Add a host (and alternatively a group) to the ansible-playbook in-memory inventory
- apt – Manages apt-packages
- apt_key – Add or remove an apt key
- apt_repository – Add and remove APT repositories
- assemble – Assemble configuration files from fragments
- assert – Asserts given expressions are true
- async_status – Obtain status of asynchronous task
- blockinfile – Insert/update/remove a text block surrounded by marker lines
- command – Execute commands on targets

- You can find the description, parameters and examples for each module.



Parameters

Parameter	Choices/Defaults	
access_time string *added in 2.7 of ansible.builtin*		This parameter indicates the time Should be `preserve` when no mod Default is `None` meaning that `pre`
access_time_format string *added in 2.7 of ansible.builtin*	Default: "%Y%m%d%H%M.%S"	When used with `access_time`, indi Based on default Python format (s

ansible.builtin.file – Manage files and file properties

Note

This module is part of `ansible-base` an
use the short module name file even w
recommend you use the FQCN for eas
conflicting with other collections that

Examples

```
- name: Change file ownership, group and permissions
  ansible.builtin.file:
    path: /etc/foo.conf
    owner: foo
    group: foo
    mode: '0644'
```

3

YAML SYNTAX

YAML Syntax

- YAML is a data serialization language (standard format to transfer data). Other languages exist such as XML and JSON.
- YAML is human readable and intuitive that depends on line separation and indentations.

YAML

```
---
- name: playbook name
  tasks:
    - name: task to have network
      check_point.mgmt.cp_mgmt_network:
        name: "network name"
        subnet: “10.1.1.0"
        mask_length: 24
        auto_publish_session: true
…
```

JSON

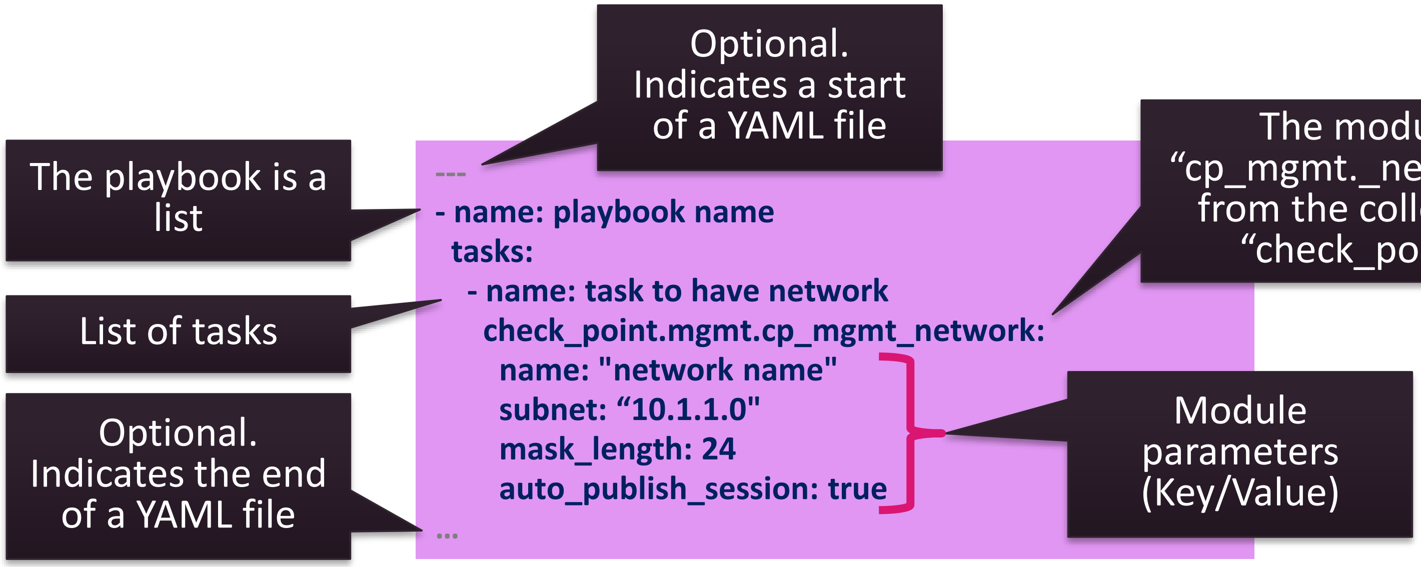
```
 {
    "name": "playbook name",
    "tasks": [
      {
        "name": "task to have network",
        "check_point.mgmt.cp_mgmt_network":
{
          "name": "network name",
          "subnet": "“10.1.1.0\"",
          "mask_length": 24,
          "auto_publish_session": true
        }
      }
    ]
  }
```

XML

```
<name>playbook name</name>
  <tasks>
    <name>task to have network</name>
    <check_point.mgmt.cp_mgmt_network>
      <name>network name</name>
      <subnet>“10.1.1.0"</subnet>
      <mask_length>24</mask_length>
      <auto_publish_session>true</auto_publish_session>
    </check_point.mgmt.cp_mgmt_network>
  </tasks>
```

YAML Syntax

- 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”.
- All YAML files can optionally begin with **---** and end with **...**
- This is part of the YAML format and indicates the **start** and **end** of a document.
- All members of a list are lines beginning at the same indentation level starting with a "- " (a **dash** and a **space**):



```
---
- name: playbook name
  tasks:
    - name: task to have network
      check_point.mgmt.cp_mgmt_network:
        name: "network name"
        subnet: “10.1.1.0"
        mask_length: 24
        auto_publish_session: true
...
```

Optional. Indicates a start of a YAML file
The playbook is a list
List of tasks
Optional. Indicates the end of a YAML file
The module “cp_mgmt._netwrork” from the collection “check_point”
Module parameters (Key/Value)

4

WORKING WITH PLAYBOOKS

Ansible Playbooks

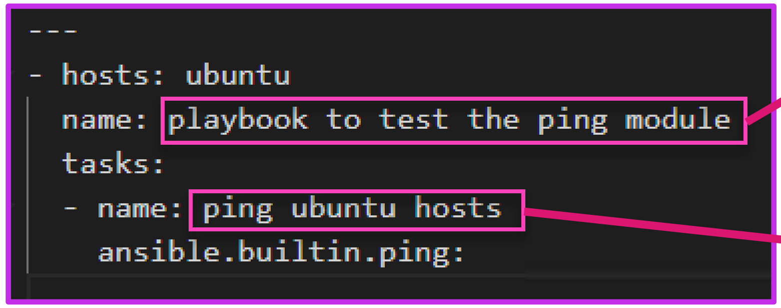
- Ansible Playbooks offer a **repeatable**, **re-usable**, **simple** configuration management and **multi-machine** deployment system, one that is well **suited to deploying complex applications**.
- Playbooks can include variables as well as tasks. Playbooks are **written in YAML** and are easy to read, write, share and understand.
- A playbook is composed of one or more ‘plays’ in an ordered list.
- Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.
- Playbooks can:
  - Declare configurations.
  - Orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order.
  - Launch tasks synchronously or asynchronously.

Ansible Playbooks

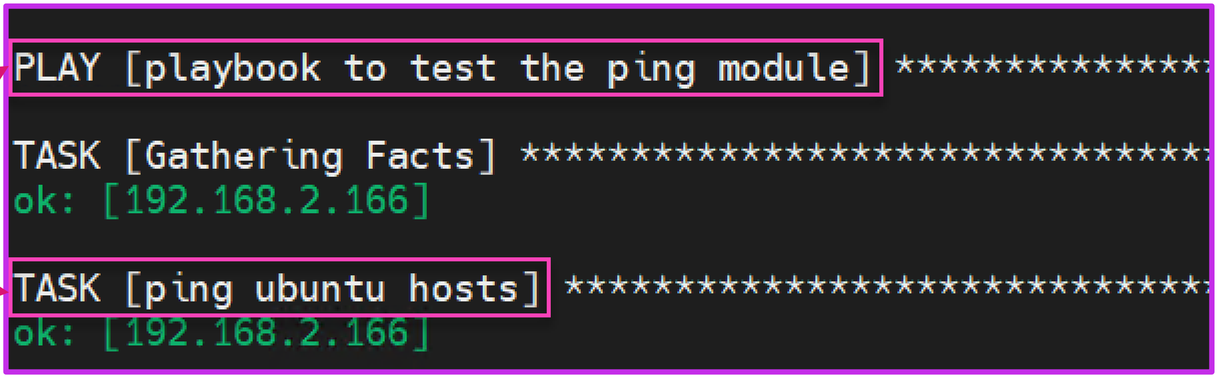
- A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom.
- At a minimum, each play defines two things:
  - The managed nodes to target.
  - At least one task to execute.
- By default, Ansible executes each task **in order**, one at a time, against all machines matched by the host pattern. When a task has executed on all target machines, Ansible moves on to the next task.
  - You can use strategies to change this default behavior.
  - Will be discussed in a later section.

Ansible Playbooks

- When you run a playbook, Ansible returns information about connections, the **name** lines of all your plays and tasks.



```
---
- hosts: ubuntu
  name: playbook to test the ping module
  tasks:
  - name: ping ubuntu hosts
    ansible.builtin.ping:
```

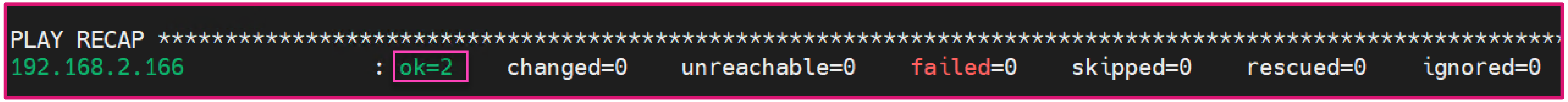


```
PLAY [playbook to test the ping module] **************

TASK [Gathering Facts] *******************************
ok: [192.168.2.166]

TASK [ping ubuntu hosts] *****************************
ok: [192.168.2.166]
```

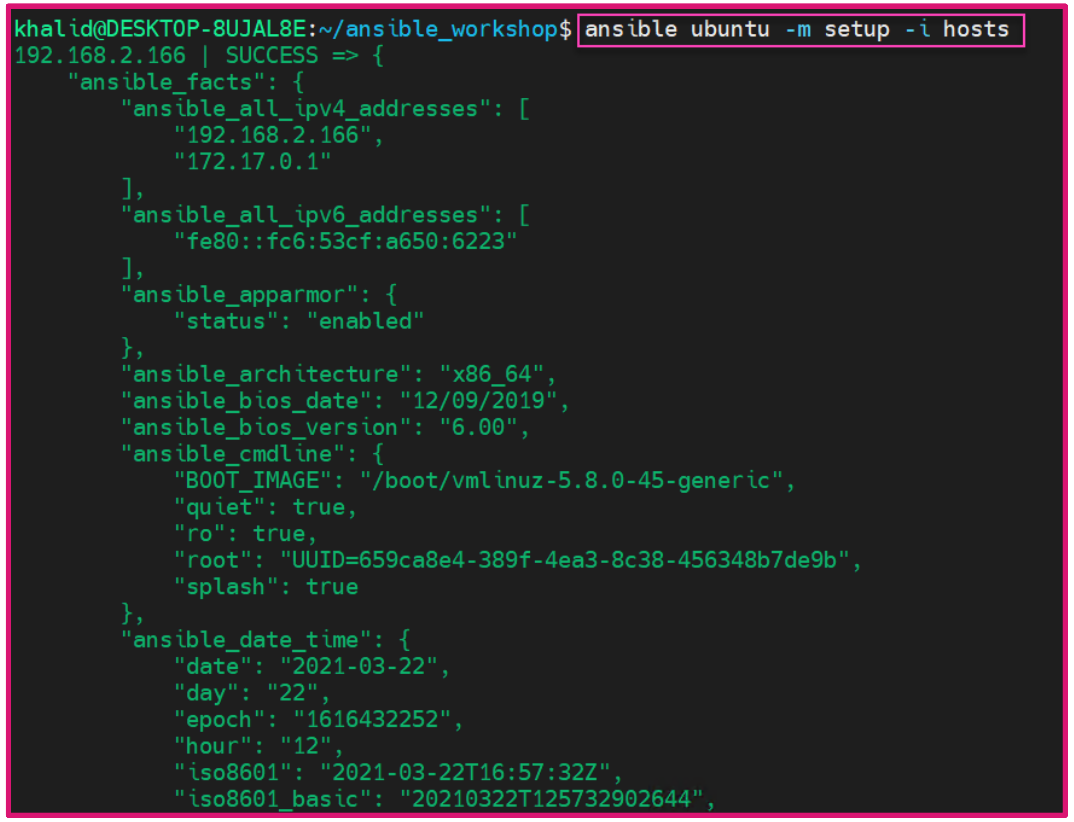
- At the bottom of the playbook execution, Ansible provides a summary of the nodes that were targeted and how they performed. General failures and fatal “unreachable” communication attempts are kept separate in the counts.



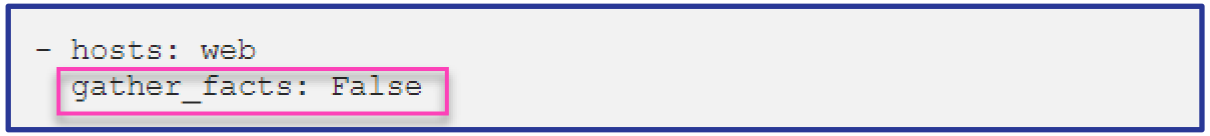
```
PLAY RECAP *******************************************************************************************
192.168.2.166              : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ansible Playbooks

- Ansible collects almost all the information about the remote hosts as it runs a playbook.
- The task of collecting this remote system information is called as **Gathering Facts**.
- This information can be obtained manually using Ansible ad-hoc command and a specialized module named setup. In fact, ansible playbooks call this setup module by default to perform Gathering Facts task.
- Facts can be used directly in the play book.
- To speed up our play execution we can specify the keyword **gathering_facts** to false in playbook.



```
khalid@DESKTOP-8UJAL8E:~/ansible_workshop$ ansible ubuntu -m setup -i hosts
192.168.2.166 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.2.166",
            "172.17.0.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::fc6:53cf:a650:6223"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/09/2019",
        "ansible_bios_version": "6.00",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-5.8.0-45-generic",
            "quiet": true,
            "ro": true,
            "root": "UUID=659ca8e4-389f-4ea3-8c38-456348b7de9b",
            "splash": true
        },
        "ansible_date_time": {
            "date": "2021-03-22",
            "day": "22",
            "epoch": "1616432252",
            "hour": "12",
            "iso8601": "2021-03-22T16:57:32Z",
            "iso8601_basic": "20210322T125732902644",
```



```
- hosts: web
  gather_facts: False
```

Ansible playbook

First Play in the playbook

First Play tasks

Second Play in the playbook

Use a keyword to define remote user

Second Play tasks

```
- name: update web servers
  hosts: centos
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: '{{apache_state}}'
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: update db servers
  hosts: all: !ubuntu
  remote_user: root
  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

Run Play against the Ubuntu group

The state is a **variable**. The variable can be defined in the inventory, playbook or can be provided at runtime with –*extra-vars* or *-e*:

ansible-playbook webservers.yml --extra-vars "apache_state=latest"

Apply to all hosts except the Ubuntu group

Use the yum module

States can be defined differently based on the module. Common terms include:

present, absent, started, latest etc..

Summary

- Define groups.
- Understand YAML.
- Write and run playbooks.

CHECK POINT™

thank you

YOU DESERVE THE BEST SECURITY