Automation & Configuration Management Tools

Ansible

Contents

1. Automation
2. Ansible introduction
3. Diving into Ansible

Automation

... in IT

History and Problem

History:

- Sysadmins install, upgrade the software and change the configuration manually.
- A sysadmin can only manage up to 3 servers.

Problem:

- A very labor intensive and error-prone process.
- Human resource needed.
- Data centers grow quickly, virtualization is supported.
- Difficulty to manage infrastructure.

Solution!

IT automation is the use of software to create repeatable instructions and processes to replace or reduce human interaction with IT systems.

What IT automation can do?

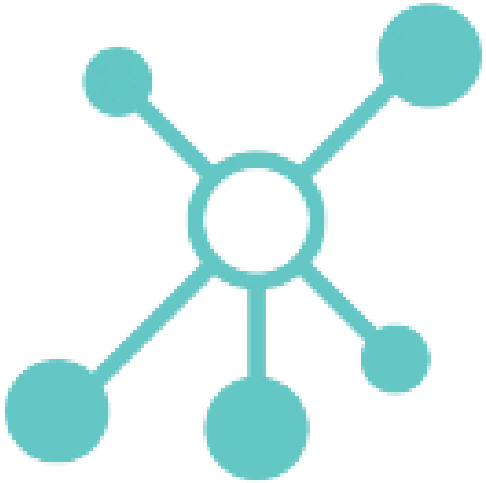
IT automation can help you get repetitive, manual processes out of the hands of your staff.

Features of IT automation:

- Provisioning
- Configuration management
- Orchestration
- Application deployment
- Security and compliance

Provisioning

- Building your infrastructure
- Play on Multiple Platforms: Bare metal, Virtualized
- Network
- Storage
- Public Cloud and Private Cloud



Configuration Management

Establishes and maintains consistency of the product performance by recording and updating detailed information.

Orchestration

Managing and servicing multiple app across multiple datacenters and infrastructure.

Application Deployment

- Effectively manage the entire application life cycle from development to production.
- Reduces opportunities for human error while improving efficiency and throughput.

Security and Compliance

- Define security and compliance policies, enforce them by building them as automated steps throughout your infrastructure.
- New compliance requirements are easily implemented consistently across your IT.

Why IT Automation?

- We have to do more with less!
- Consistently deliver stable predictable environment
- Increase number of deployments, reduce time between deployments.
- Deliver more secure environment
- Innovate faster

Automation Tools

Comparison

	Chef	Puppet	Ansible	SaltStack
Availability	Yes	Yes	Yes	Yes
Easy to Setup	Not very easy	Not very easy	Easy	Not very easy
Management	Not very easy	Not very easy	Easy	Easy
Database	PostgreSQL	PuppetDB	None	None
Configuration language	Ruby	Ruby	YAML	YAML
Transport	RabbitMQ	Mcollective	SSH	ZeroMQ

Ansible introduction

Ansible

- Original author(s): Michael DeHaan
- Initial release February 20, 2012; 9 years ago
- Developer(s): Ansible Community/Ansible Inc./Red Hat Inc.
- Repository: https://github.com/ansible/ansible
- Written in Python, PowerShell, Shell, Ruby, ...
- Operating system Linux, Unix-like, MacOS, Windows
- License Proprietary/GNU General Public License v3

What is Ansible?

- Ansible is an automation tool that allows you to create groups of machines, describe how these machines should be configured or what actions should be taken on them.
- Ansible issues all commands from a central location to perform these tasks.
- Ansible work on push mode over SSH connection.

Principles of Ansible

SIMPLE

Human readable

No special coding skills

Tasks executed in order

Get productive quickly

POWERFUL

App deployment

Configuration management

Workflow Orchestration

Orchestration the app lifecycle

AGENTLESS

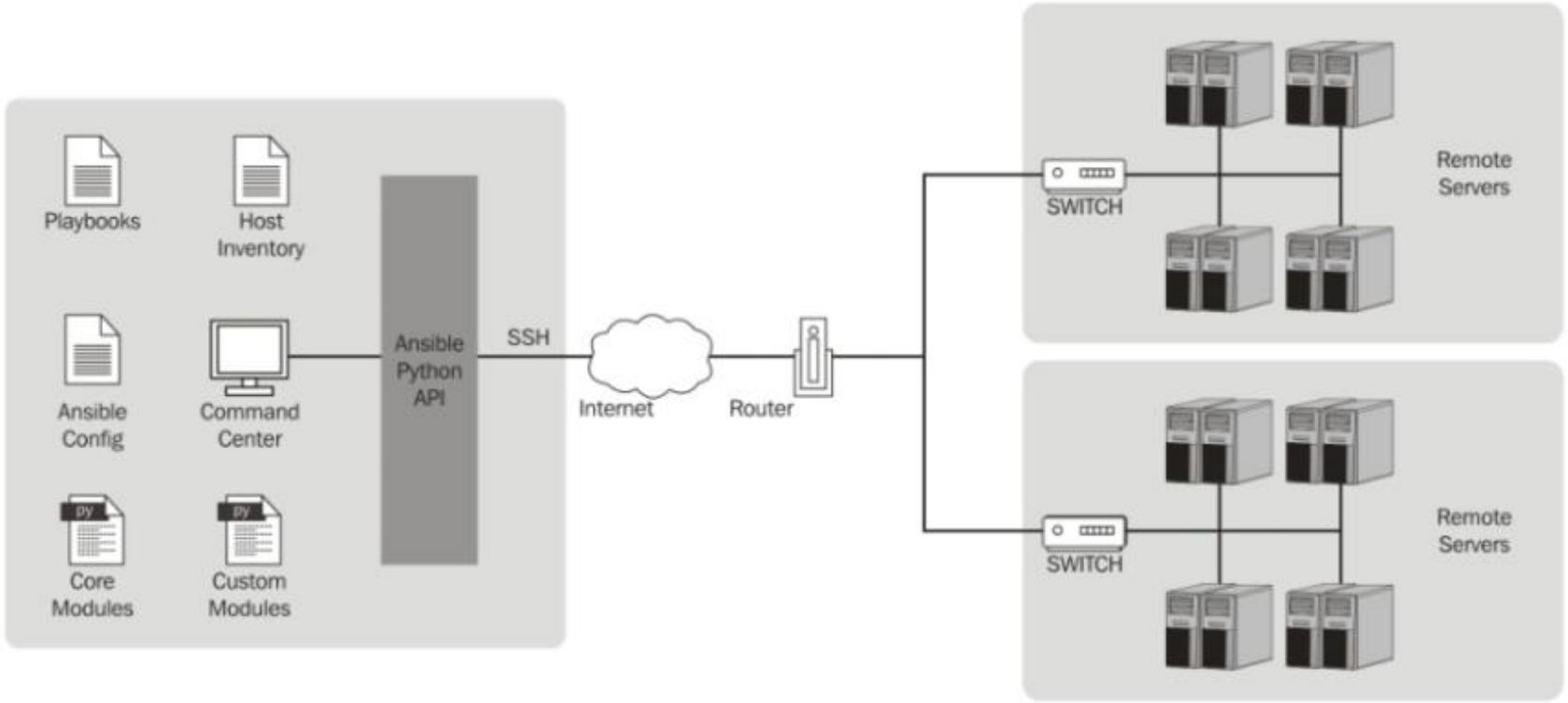
Agentless architecture

Uses OpenSSH & WinRM

No agents to exploit or update

More efficient & More secure

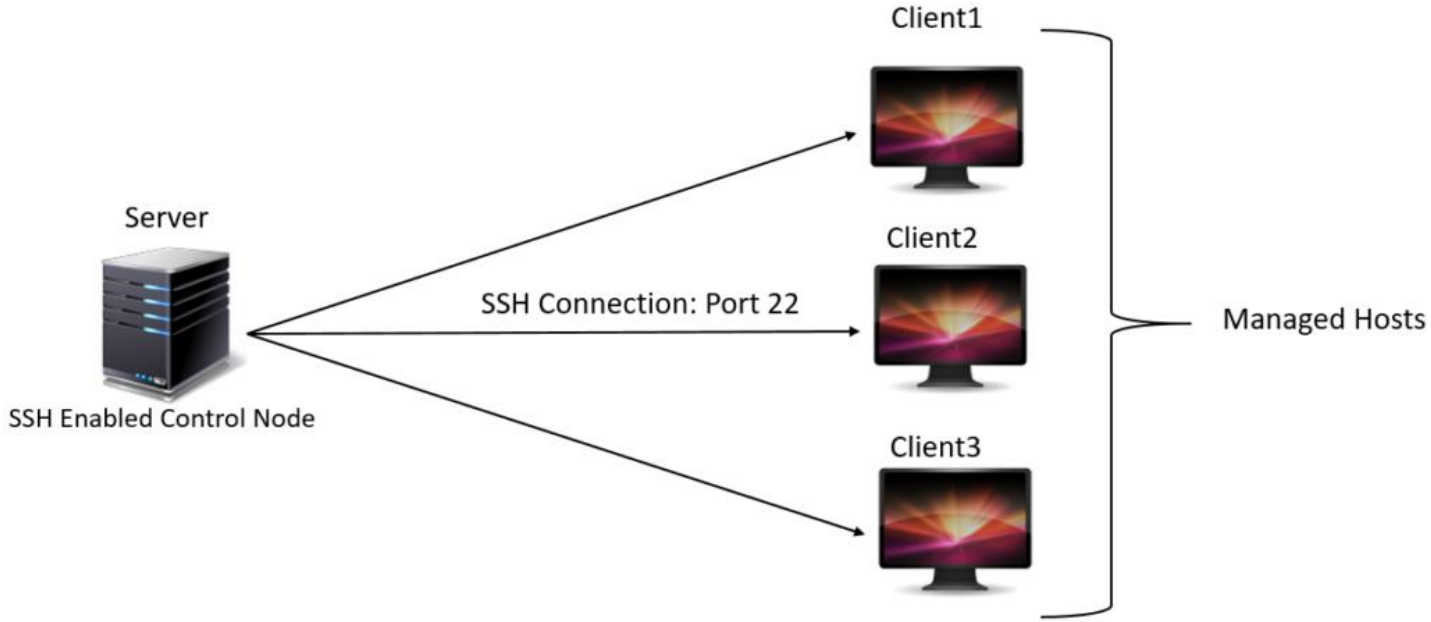
How Ansible Works?



Ansible Terminology

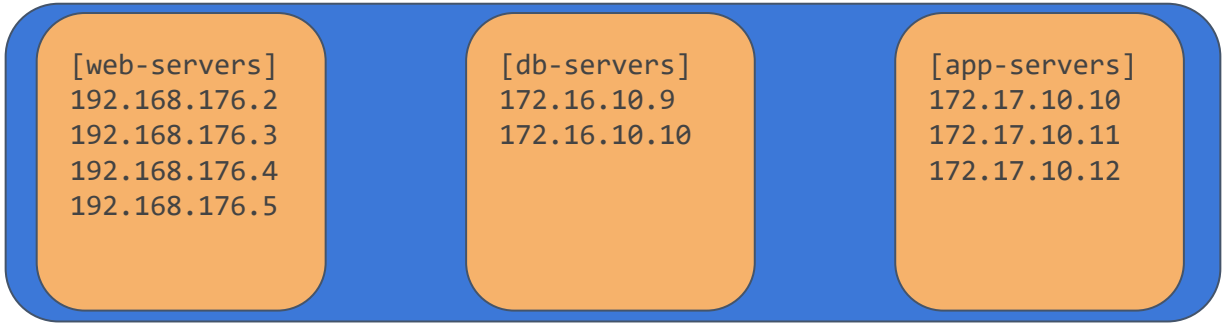
- Controller Machine
- Inventory
- Host and Group
- Playbook
- Task
- Module
- Role
- Facts and Variables

Controller Machine



Inventory

- Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible's inventory file.
- By default, inventory use a simple INI format and saved in the location `/etc/ansible/hosts`



Inventory

An Inventory file can contain:

- Hosts and Groups
- Host and Group Variables
- Group of Groups
- Inventory Parameter

Task

Tasks combine an action with a name and optionally some other keywords (like looping directives)

```
---
tasks:
- name: install nginx
  apt: name=nginx update_cache=yes

- name: copy nginx config file
  copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default
```

Playbook

An ansible playbook is built by defining tasks and run against an inventory which specifies the hosts to run those tasks.

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Module

- A module typically abstracts a system task, like dealing with packages or creating and changing files.
- Ansible has a multitude of built-in modules, but you can also create custom ones.
- Common modules: apt, copy, file, service, template.

Role

A pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of a provisioning.

```
apache-role/
    vars/
        Debian.yml
        RedHat.yml
    handlers/
        main.yml
    tasks/
        main.yml
        install-RedHat.yml
    defaults/
        main.yml
    meta/
        main.yml
```

Facts and Variables

- Facts are simply things that are automatically discovered about remote host by Ansible and they are inferred, rather than set.
- As opposed, Variables are names of value (integer, boolean, string, dictionaries, lists), that are declared things.
- Facts and variables can be used in templates and playbooks.

Demo

- Ping
- Gather Facts
- Setup/upgrade a package

Diving into Ansible

Ansible configuration

- Certain settings in Ansible are adjustable via a configuration file named: ansible.cfg
- Ansible will search in the following order and *use the first file found*, all others are ignored:
  - `ANSIBLE_CONFIG` (environment variable if set)
  - `ansible.cfg` (in the current directory)
  - `~/.ansible.cfg` (in the home directory)
  - `/etc/ansible/ansible.cfg`

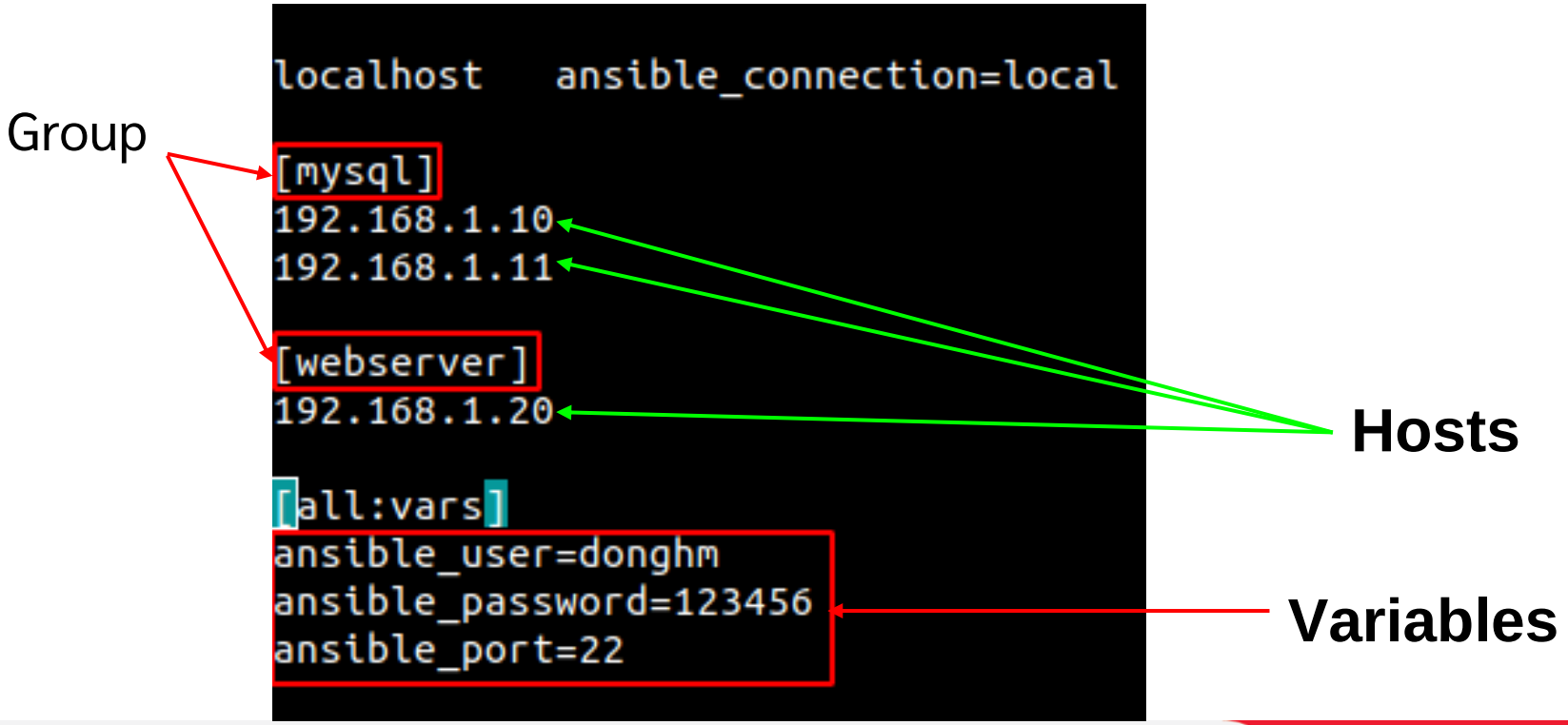
Ansible configuration (2)

- Ansible configuration file: using **INI** format

```
# cat /etc/ansible/ansible.cfg
[defaults]
host_key_checking = False
inventory = /etc/ansible/hosts
log_path = /var/log/
```

- For more configurations: https://docs.ansible.com/ansible/latest/reference_appendices/config.html#ansible-configuration-settings-locations

Ansible inventory



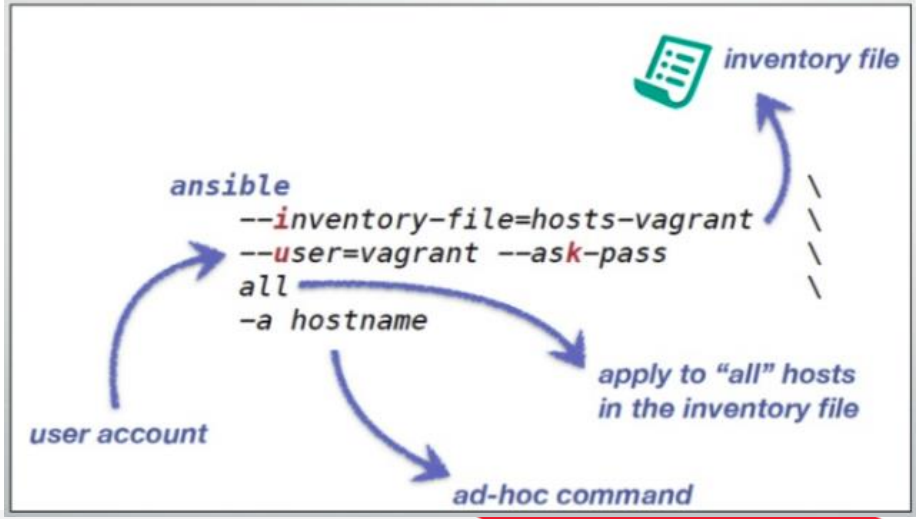
YAML format

```yaml
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

```yaml
...
  hosts:
    jumper:
      ansible_port: 5555
      ansible_host: 192.0.2.50
```

Ad-Hoc commands

An ad-hoc command is something that you might type in to do something quick, but don’t want to save for later or write a full playbook



Ad-Hoc commands: Example

Check ping to all hosts in inventory file:

```
$ ansible -i inventory all -m ping
```

File transfer: using module copy

```
$ ansible -i inventory webservers -m file \
-a "dest=/tmp/hello mode=755 owner=admin group=root"
```

Managing Packages: using module yum or apt

```
$ ansible -i inventory webservers -m yum -a "name=httpd state=present"
```

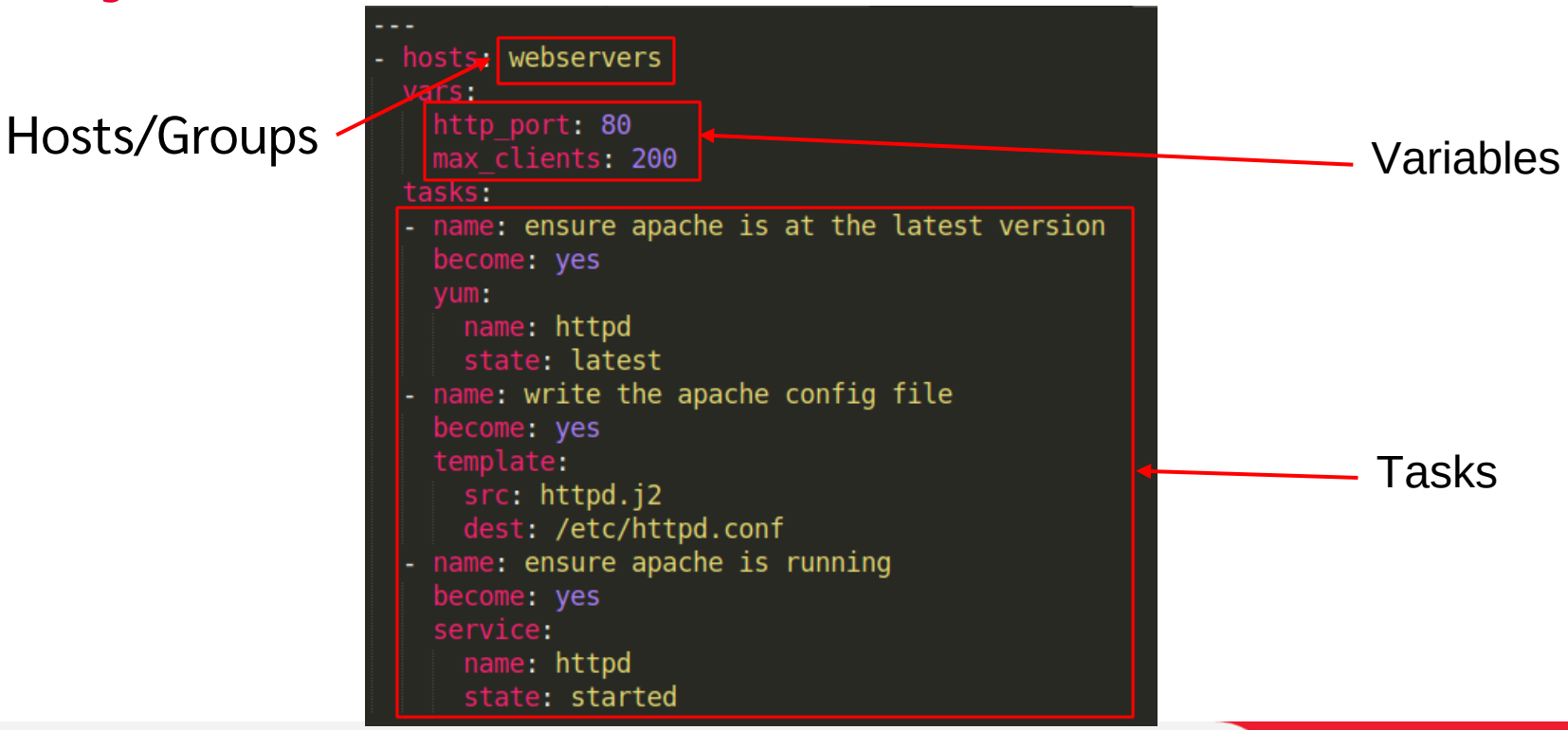
name can be specified the version: **httpd-2.4.6**

state can be **present** or **latest** or **absent**

Playbook

- Playbooks are Ansible's configuration, deployment and orchestration language.
- At basic level, playbooks contain the tasks to manage configurations of and deployments to remote machines
- At the advanced level:
  - Using to rolling updates services with zero downtime
  - Delegation: perform a task on one host with reference to other hosts
- Playbook must have at least one play.

Playbook structure



Playbook - Host and variables

- **Host**: that is a list of one or more groups or host patterns, separated by colons, to complete the steps tasks.
- **Variables**: used to pass into template and tasks

Playbook - Task

- Each play contain a list of task.
- Task are executed in order, one at a time, against all hosts matched by the host pattern before moving on to the next task.
- Hosts with failed tasks are taken out of the rotation for the entire playbook.

Playbook - execute

```
donghm@donghm:~/git/TIL/automation-course/lesson-2$ ansible-playbook -i inventory playbook.yml

PLAY [webservers] ***************************************************************

TASK [Gathering Facts] **********************************************************
ok: [10.240.201.238]
ok: [10.240.201.237]

TASK [ensure apache is at the latest version] ***********************************
changed: [10.240.201.237]
changed: [10.240.201.238]

TASK [ensure apache is running] *************************************************
changed: [10.240.201.237]
changed: [10.240.201.238]

PLAY RECAP **********************************************************************
10.240.201.237             : ok=3    changed=2    unreachable=0    failed=0
10.240.201.238             : ok=3    changed=2    unreachable=0    failed=0
```

Gathering Facts

- Gather all informations about the remote hosts: OS, hostname, interfaces (IP, MAC), ... to use in the playbook's tasks.
- Disable: **gather_facts: False** when using ansible caching for running play faster, in case have a lot of hosts or when we don't need to use gather_facts.

Variable

- Simple YAML format
- Can create dictionaries
- Can be defined at many level (default, role, playbook)
- Can tests conditionals on variables
- Using flexible with Jinja2

Valid variable name

- Variable names should be letters, numbers, and underscores.
- Variables should always start with a letter.
- Special character can be used: underscore
- Example about in-valid variables:
  `foo-port, foo port, foo.port, and 12`

Define and access variables example

bonding.yml

```
- hosts: localhost
  gather_facts: false
  vars:
    vlan_100: "bond0.100"
    vlan_666: "bond1.666"
  tasks:
  - name: Copying bond0 config
    template: src=ifcfg-bond0 dest=/etc/sysconfig/network-scripts/ifcfg-bond0
  - name: Copying bond0 config
    template: src=ifcfg-bond0 dest=/etc/sysconfig/network-scripts/ifcfg-bond1
  - name: Copying Vlan 100 config
    template: src="{{ vlan_100 }}" dest=/etc/sysconfig/network-scripts/ifcfg-"{{ vlan_100 }}"
  - name: Copying Vlan 666 config
    template: src="{{ vlan_666 }}" dest=/etc/sysconfig/network-scripts/ifcfg-"{{ vlan_666 }}"
```

Using facts - example

Get bonding speed of bond0 interface and check if speed is less than 20GE, then returned the failed status.

```
- hosts: localhost
  gather_facts: true
  tasks:
  - name: "Check Bond card speed"
    debug:
      msg: "OK! The speed of bond0 is {{ ansible_facts.bond0.speed }}"
    when: ansible_facts.bond0.speed is version('20000', '>=')
```

Practice 1

Host machine: Controller machine

Ubuntu virtual machine: Managed machine

- Setup Virtualbox
- Create Ubuntu virtual machine
- Setup Ansible on Controller machine
- Using Ansible to setup Docker on Ubuntu VM
- Using Ansible to deploy Wordpress (docker) on Ubuntu VM

Practice 2

Host machine: Controller machine

Ubuntu VM1: Managed machine (MariaDB)

Ubuntu VM2: Managed machine (Wordpress)

- Using Ansible to setup Docker on two VMs
- Using Ansible to deploy Wordpress (docker) on VM1
- Using Ansible to deploy MariaDB (docker) on VM2