What is Ansible?

What is Ansible?

- Ansible is an *open-source* **configuration management**, **software provisioning**, and **application changes** and IT infrastructure operation very simple.
- Ansible is ***developed/written*** in **Python** and only requires the **Python libraries** to be present on the servers to be configured, the default on almost all **Linux distros** (Ubuntu, Amazon Linux2, Centos) already installed.
- Ansible is lightweight, relative ease of use and speed of configuration changes compared to other CM tools (Chef, Puppet)
- Ansible packages all commands in **YAML modules called playbooks.**

What is Ansible?

- Ansible will only be installed on **Control Node**.
- Python should be present on all managed nodes.
- Ansible is **agentless**. You do not need to have anything installed on the client's end and do not have to keep any agent running.
- It uses **OpenSSH** as transport protocol.
- Ansible scripts (commonly known as **playbooks**) are written in **yaml** and are easy to read.
- Ansible is very lightweight, easy to configure, and is not resource hungry because it doesn’t need an agent to run (**agentless**) unlike other automation tools, for example, **Puppet** which is agent based and is a bit complex to configure.

Why Ansible?

SIMPLE

Human readable automation

No special coding skills needed

Tasks executed in order

Get productive quickly

POWERFUL

App deployment

Configuration management

Workflow orchestration

Orchestrate the app lifecycle

AGENTLESS

Agentless architecture

Uses OpenSSH & WinRM

No agents to exploit or update

More efficient & more secure

How Ansible Works?

- Ansible works in **push** based model.
    - Control Node (perform some task on manage node)
- Ansible works by connecting from Control Node to Manage Node (remote servers) using through the **SSH keys** and pushing the **small programs out**.
- With the use of **modules**, the **playbooks** help the ansible clients in performing all the specific tasks.
- The particular functions can include the **service of restarting**, **installing packages**, **executing scripts**, and much more.

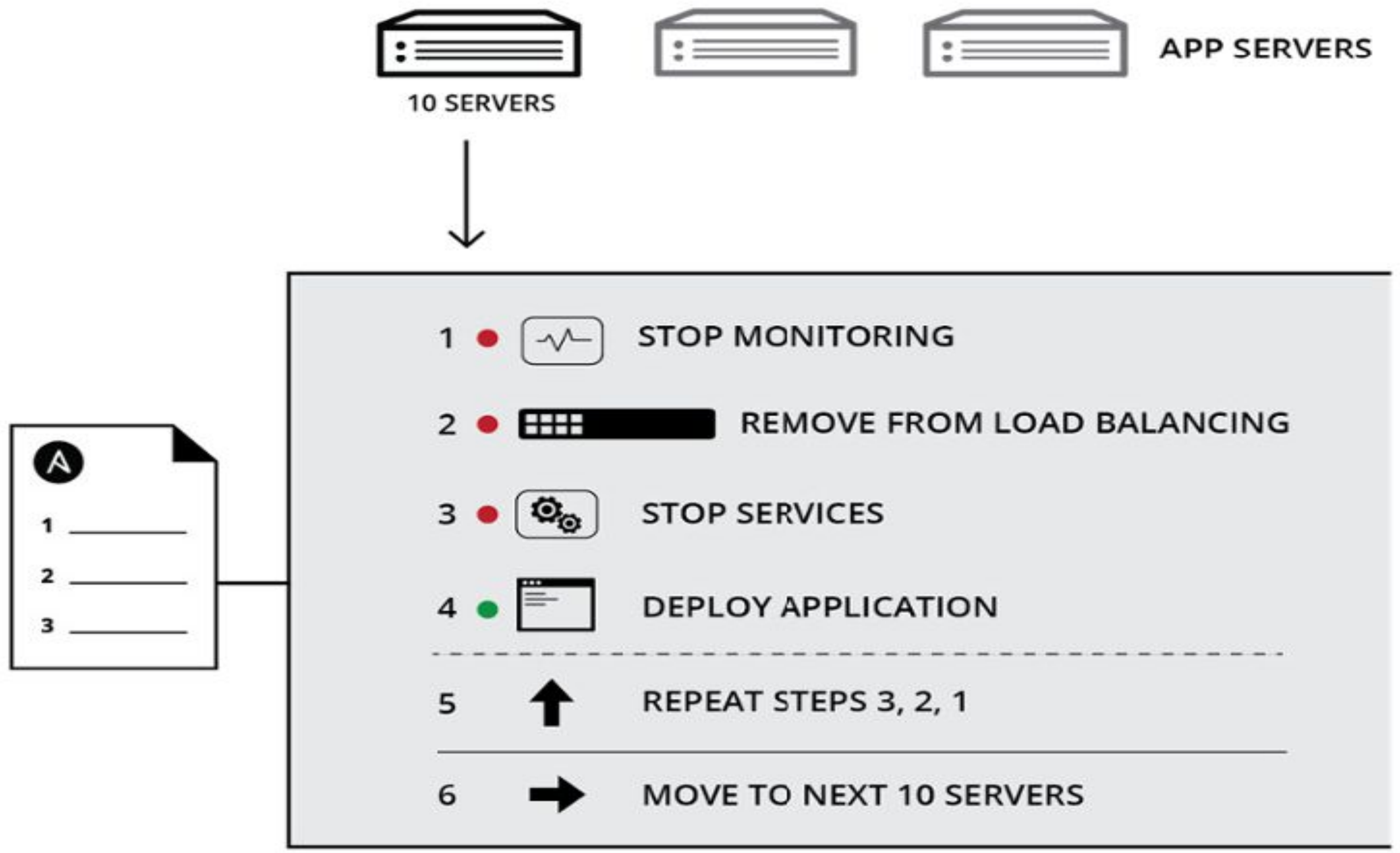
Ansible Nodes

Control Node

- The Ansible software only needs to be installed on the control node (or nodes) from which Ansible will be run.
- The control node should be a **Linux or UNIX system**. Microsoft Windows is not supported as a control node, although Windows systems can be managed hosts.

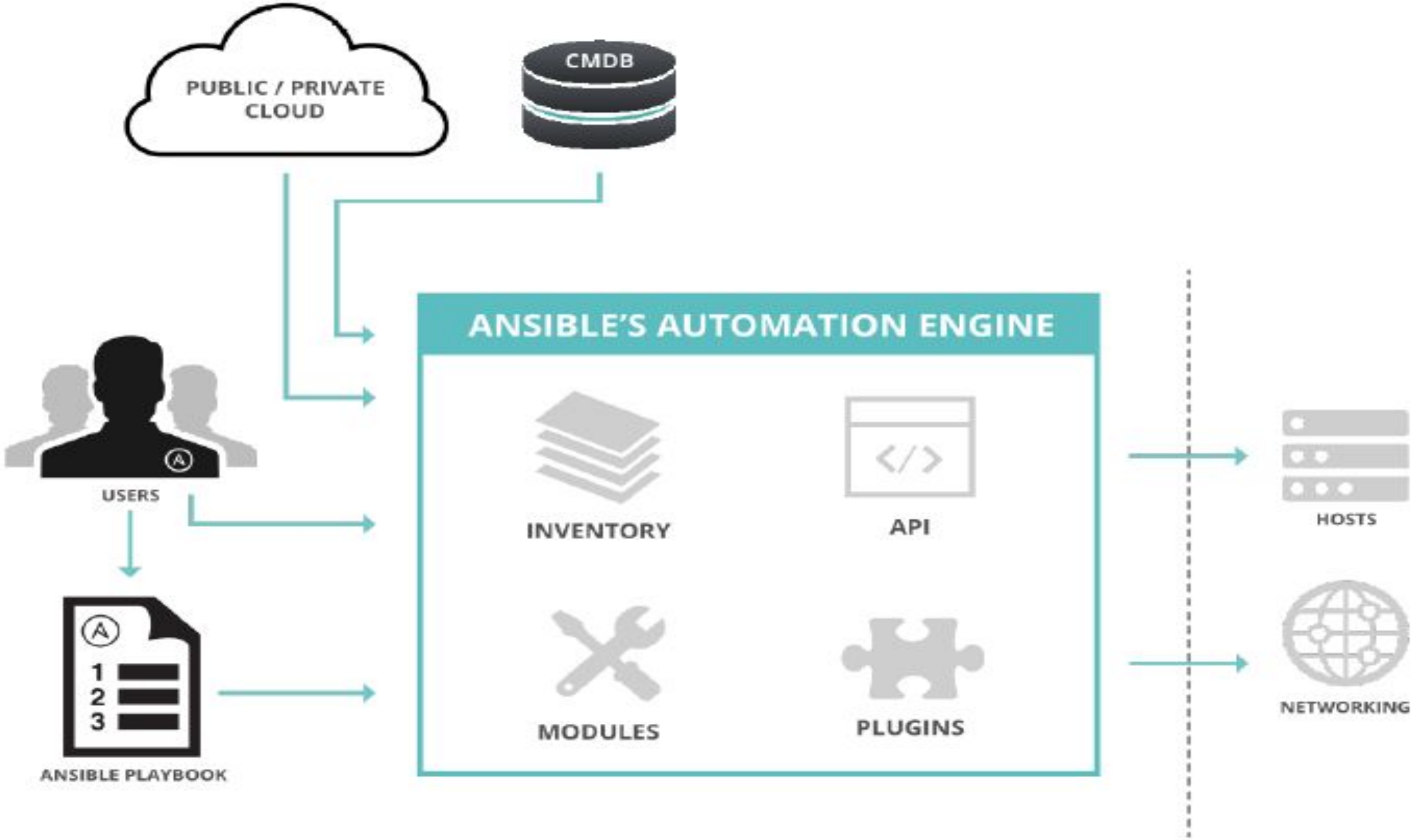
Managed Hosts

- One of the benefits of Ansible is that managed hosts do not need to have a special agent installed.
- Python should be present on Managed Nodes.
- The Ansible control node connects to managed hosts using a standard network protocol to ensure that the systems are in the specified state.
  - Linux Protocol from Control to Managed(**Linux**): **ssh**
  - WinRM Protocol from Control to Managed(**Windows**): **winrm**



APP SERVERS
10 SERVERS
1
2
3
1 STOP MONITORING
2 REMOVE FROM LOAD BALANCING
3 STOP SERVICES
4 DEPLOY APPLICATION
5 REPEAT STEPS 3, 2, 1
6 MOVE TO NEXT 10 SERVERS

How Ansible Works?



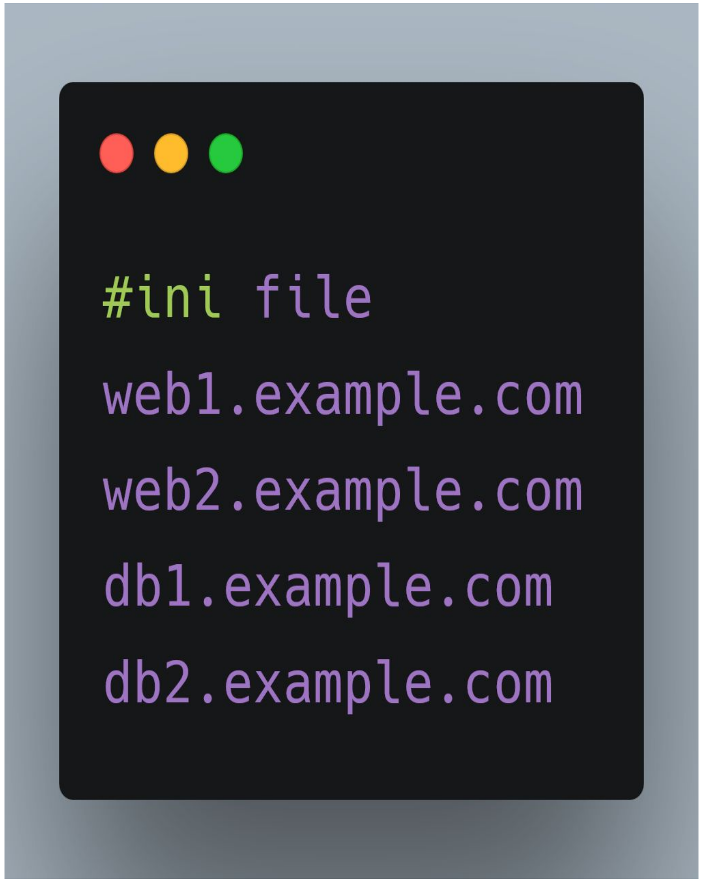
Inventory

Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible can connect and manage.

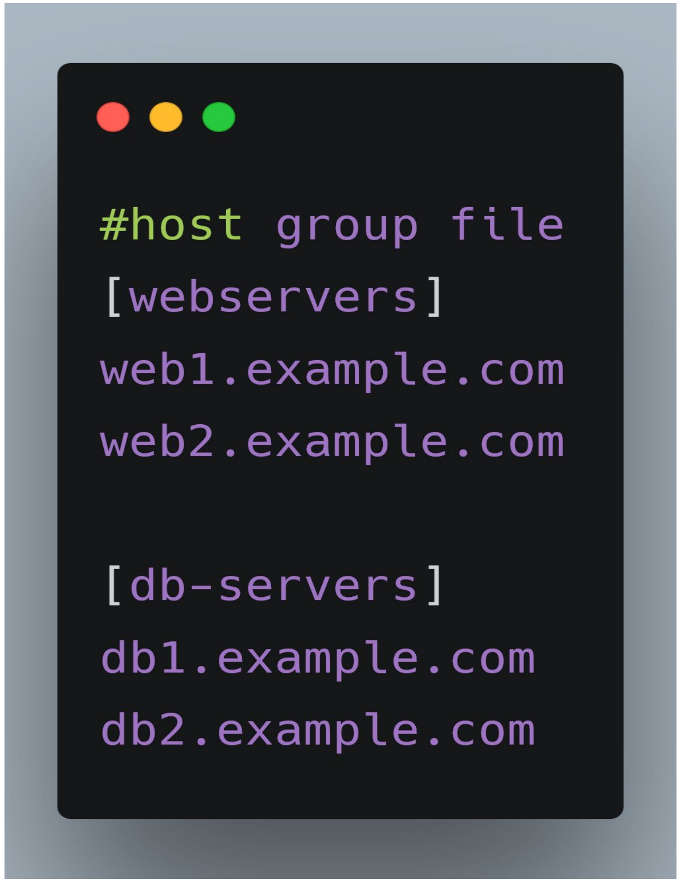
- Hosts (nodes)
- Groups
- Inventory-specific data (variables)
- It does this by selecting portions of systems listed in Ansible's inventory, which defaults to being saved in the location **/etc/ansible/hosts**.
- You can specify a different inventory file using the **-i <path>** option on the command line.

- Ansible reads the inventory file that is specified and performs operations on specific group of hosts inside the inventory.
- **/etc/hosts** on control node should have hostname and IP mapping.

Inventory File



```
#ini file
web1.example.com
web2.example.com
db1.example.com
db2.example.com
```



```
#host group file
[webservers]
web1.example.com
web2.example.com

[db-servers]
db1.example.com
db2.example.com
```

Static Inventory

A static inventory file is an INI-like text file that specifies the managed hosts that Ansible targets.

Host Variables and Group Variables

Host variables

- You can easily assign a variable to a single host, then use it later in playbooks.

Group variables

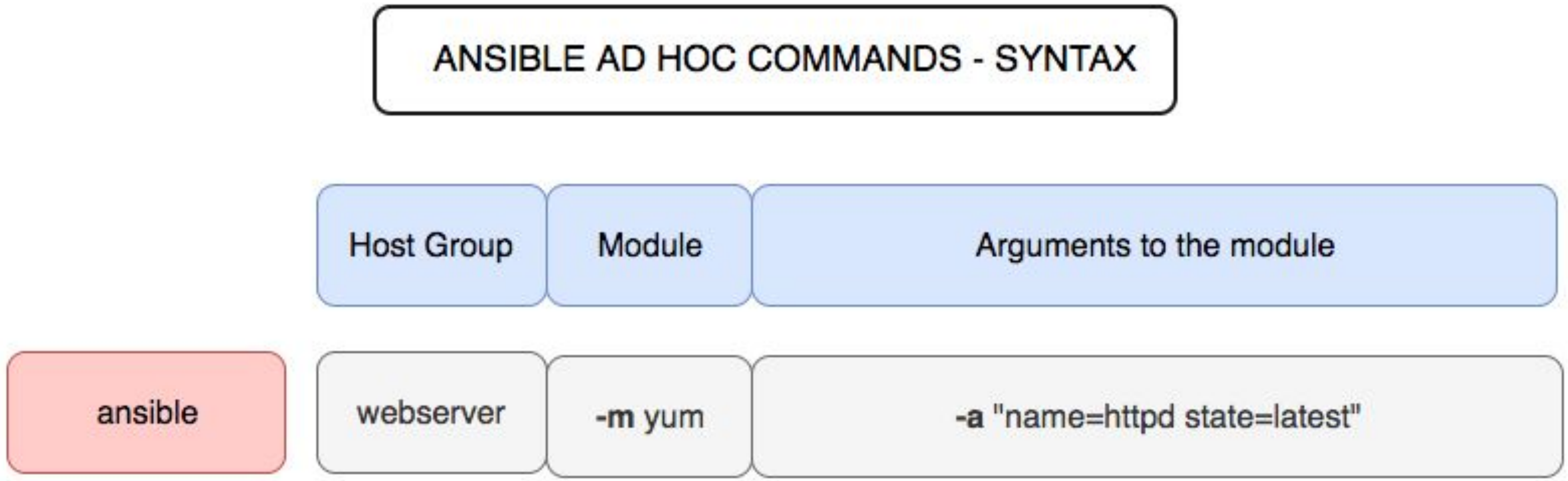
- If all hosts in a group share a variable value, you can apply that variable to an entire group at once

Ansible Modules

Modules are bits of code transferred to the target system and executed to satisfy the task declaration.
All Ansible modules technically **return JSON** format data.

- **ping** - Try to connect to host, verify a usable python and return pong on success
- **copy** - copies a file from Control Node to a location on the Managed Node.
- **user** - Manage user accounts and user attributes in Linux Server.
- **file** - Sets attributes of files and directories, or removes files/symlinks/directories.
- **yum** - Installs, upgrade, downgrades, removes, and lists packages and groups with the yum package manager.
- **fetch** - fetch files from managed nodes and storing them locally, same as copy in reverse.
- **debug** - Print statements during execution

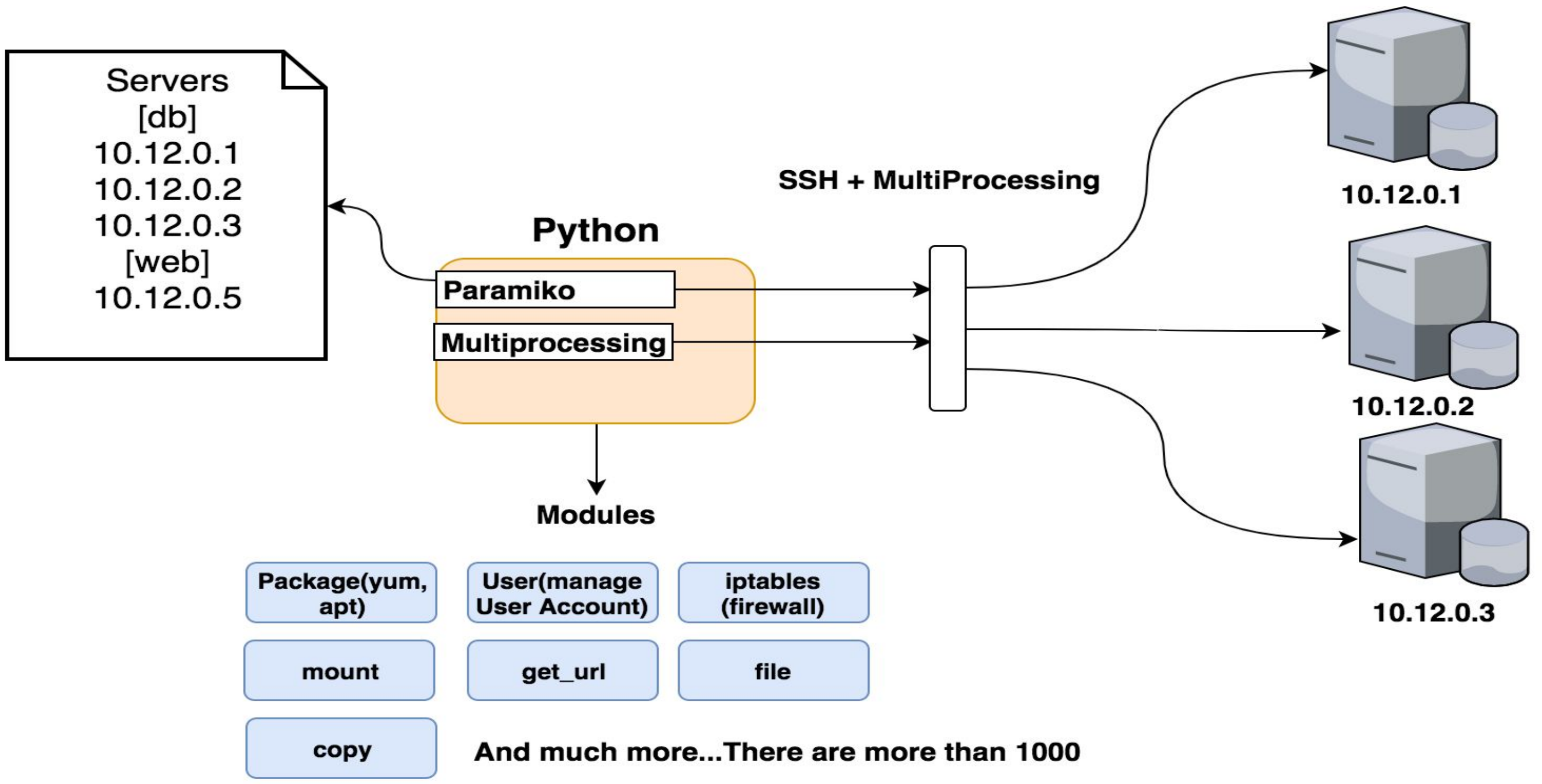
Ansible Adhoc Command



ANSIBLE AD HOC COMMANDS - SYNTAX

	Host Group	Module	Arguments to the module
ansible	webserver	-m yum	-a "name=httpd state=latest"

Ansible Setup



Modules : Run Commands

If Ansible doesn't have a module that suits your needs, there are the "**run command**" modules:

command: Takes the command and executes it on the host. The most secure and predictable.

shell: Executes through a shell like /bin/sh.

script: Runs a local script on a remote node after transferring it.

raw: Executes a command without going through the Ansible module subsystem.

NOTE: Unlike standard modules, run commands have no concept of desired state and should only be used as a last resort.

Ansible Playbooks

- **Playbook** - A single YAML file
  - **Play** - Define a set of activities (tasks) to be run on hosts
    - Task - an action to be performed on host
      - Execute a command
      - Execute a Script
      - Install linux packages
      - Start/Stop/Restart service

Ansible Playbooks

- **Playbooks** are a way to combine many tasks, written in **YAML**, to be carried out against one or many hosts specified in **inventory**.
- A playbook is a text file in YAML format, and is normally saved with the extension **.yml**.

Playbook contains Plays where each play consists of :

- **What to configure**: We need to configure a host or group of hosts to run the play against, that is defined in inventory file.
- **What to run**: This includes the specification of tasks to be run, including which system components to modify and which state they should be in, for example, installed, started, or latest.

Ansible Playbooks

- **Running Playbooks**

The **ansible-playbook** command is used to run playbooks.

A **playbook** is a text file that contains a list of one or more plays to run in order. A **play** is an ordered set of tasks which should be run against hosts selected from your inventory.

- **Playbooks can contain one or many plays**
- A **play** operates on a set of hosts
- **Tasks** are performed via **modules**
- **Tasks are grouped together via plays**

Ansible Playbook

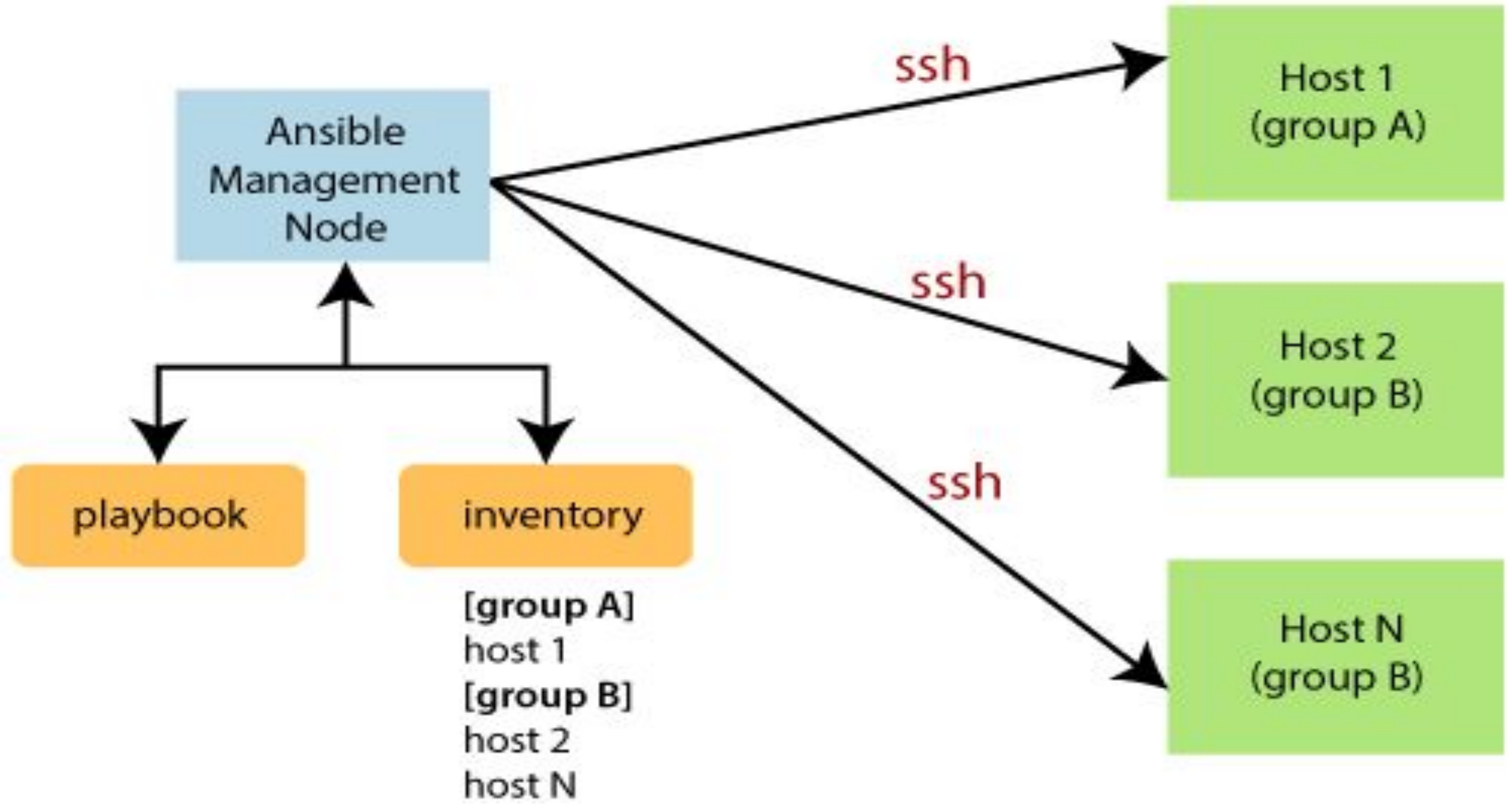
<u>In Simple Terms:</u>

- **Playbook** contains **Plays**
- **Plays** contains **Tasks**
- **Tasks** run **Modules.**

Includes

- **Include** file defines a **set of tasks** that can be **included by a playbook**, this allows sharing sets of tasks without copy/pasting everywhere.
- Playbooks can also include other playbooks

Ansible Working



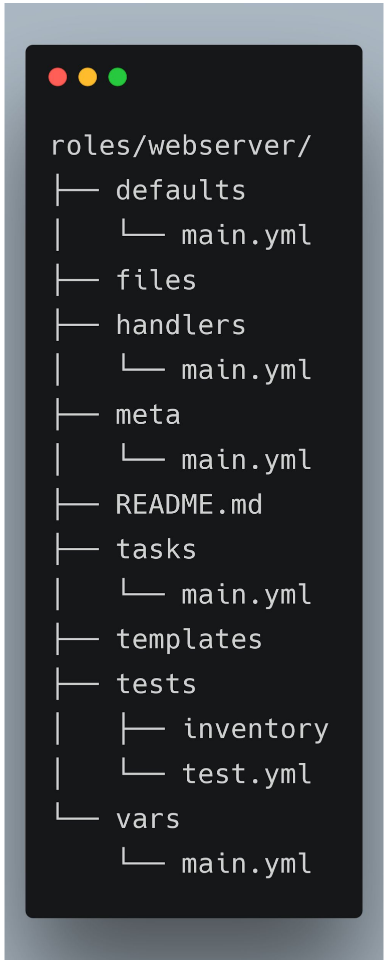
Ansible Roles

- An **Ansible role** is a collection of **files, tasks, templates, variables, and handlers** that together serve a certain purpose like configuring a service.
- Ansible Roles allows you to **easily re-use code** and **share Ansible solutions with other users** which makes working with large environments more manageable.
- Instead of requiring you to explicitly include certain files and playbooks in a role, Ansible **automatically includes any main.yml files** inside specific directories that make up the role.
- There are only two directories required to make a working Ansible role:

```
rolename/
  meta/
  tasks/
```

```
mkdir roles
ansible-galaxy init roles/webserver
```

Role Directory Structure



```
roles/webserver/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

- **defaults** → contains default variables for the role that are meant to be easily overwritten.
- **vars** → contains standard variables for the role that are not meant to be overwritten in your playbook.
- **tasks** → contains a set of tasks to be performed by the role.
- **handlers** → contains a set of handlers to be used in the role.
- **templates** → contains the Jinja2 templates to be used in the role.
- **files** → contains static files which are accessed in the role tasks.
- **meta** → contains role metadata such as author information, license, dependencies, etc.
- **tests** → may contain an optional inventory file, as well as test.yml playbook that can be used to test the role.

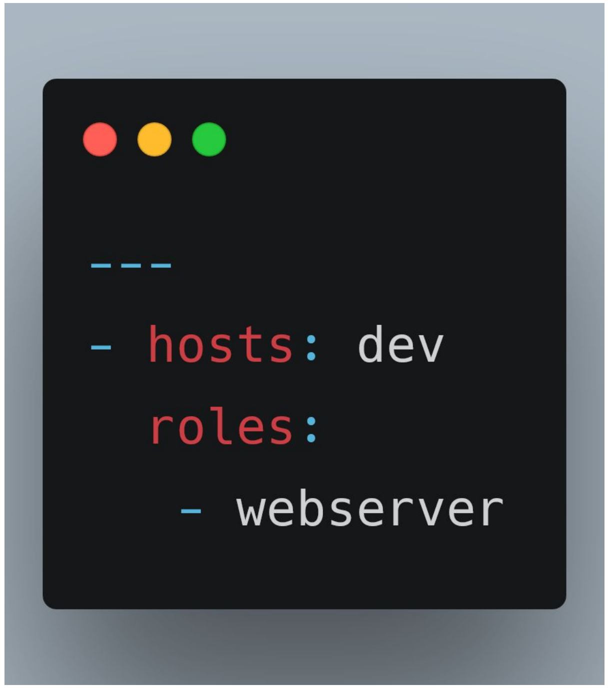
Storing and locating roles

- By default, Ansible looks for roles in two locations:
  - in a directory called **roles/**, relative to the playbook file
  - in **/etc/ansible/roles**

- *By default Ansible will look in each directory within a role for a main.yml file for relevant content (also main.yaml and main)*

Ansible Roles

- If you create a directory structure like the one shown above, with a ***main.yml*** file in each directory, Ansible will run all the tasks defined in ***tasks/main.yml*** if you call the role from your playbook using the following syntax:
- Another simple way to create directory structure for a role is to use the command:
  `ansible-galaxy init role_name`
- Running this command creates an example role in the current working directory, which you can modify to suit your needs.
- Using the **init** command also ensures the role is structured correctly in case you want to someday contribute the role to Ansible Galaxy.



```
---
- hosts: dev
  roles:
    - webserver
```

Variable Precedence

- **Variable Precedence**
  It should be rare that you would need to dig into the details of which variable is used when you define the same variable in five different places, but since there are odd occasions where this is the case, Ansible's documentation provides the following ranking:
  1. **--extra-vars** passed in via the command line
  2. **Task-level vars** (in a task block).
  3. **Block-level vars** (for all tasks in a block).
  4. **Role vars** (e.g. [role]/vars/main.yml) and vars from include_vars module.
  5. **Vars set via set_facts modules**.
  6. **Vars set via register in a task**.
  7. **Individual play-level vars**: 1. vars_files 2. vars_prompt 3. vars
  8. **Host facts**.
  9. **Playbook host_vars**.
  10. **Playbook group_vars**.
  11. **Inventory**: 1. host_vars 2. group_vars 3. vars
  12. **Role default vars** (e.g. [role]/defaults/main.yml).

Ansible Terminologies

- **Controller Machine:** The Controller machine is used to provisioning the servers, which is managed. This is the machine where Ansible is installed.
- **Inventory**: An inventory is an initialization file which has details about the different servers you are managing.
- **Playbook**: It is a code file that is written in the YAML format. A playbook contains the tasks that need to be automated or executed.
- **Task**: Every task represents a single procedure that needs to be executed.
- **Module**: A module is the set of tasks that can be executed. Ansible has 100s of built-in modules, and also you can create custom ones.
- **Role**: The role is a pre-defined way for organizing playbooks and other files to facilitate sharing and reusing portions of provisioning.
- **Play**: The task executed from start to finish, or the execution of a playbook is called the play.
- **Facts**: Facts are global variables which are store details about the system, such as network interfaces or operating system.
- **Handlers**: Handlers are used to trigger the status of a service, such as restarting or stopping a service.

Ansible Ways

- **Complexity Kills Productivity**
  Ansible is designed so that its tools are simple to use and automation is simple to write and read.

- **Optimize For Readability**
  The Ansible automation language is built around simple, declarative, text-based files that are easy for humans to read

- **Think Declaratively**
  Ansible is a desired-state engine. It approaches the problem of how to automate IT Configuration Changes by expressing them in terms of the state that you want your systems to be in.

Ansible Tower

- Ansible Tower is an enterprise framework for controlling, securing and managing your Ansible automation with a UI and RESTful API.
- Ansible Tower adds a user-friendly central dashboard where you can see the status of your servers, manage access control and monitor job runs.
  Ansible Tower