Automation with Ansible

References

- Ansible Documentation https://docs.ansible.com/
- Red Hat Ansible Engine 2.7 DO407
- Mastering Ansible Third Edition – James Freeman, Jesse Keating – Packt Publishing – March 2019

Overview of Ansible

What is Ansible?

Ansible is an open source automation platform. It is a simple automation language that can perfectly describe an IT application infrastructure in Ansible Playbooks.

It is also an automation engine that runs Ansible Playbooks.

Why Ansible? Simple

- Human readable automation
- No special coding skills needed
- Task executed in order
- Get productive quickly

Why Ansible? Powerful

- Application deployment
- Configuration management
- Workflow orchestration
- Orchestrate the application lifecycle

Why Containers? Agentless

- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- More efficient & more secure

Ansible Strength

- Cross platform support
- Human-readable automation
- Perfect description of applications
- Easy to manage in version control
- Support for dynamic inventories
- Orchestration that integrates easily with other systems

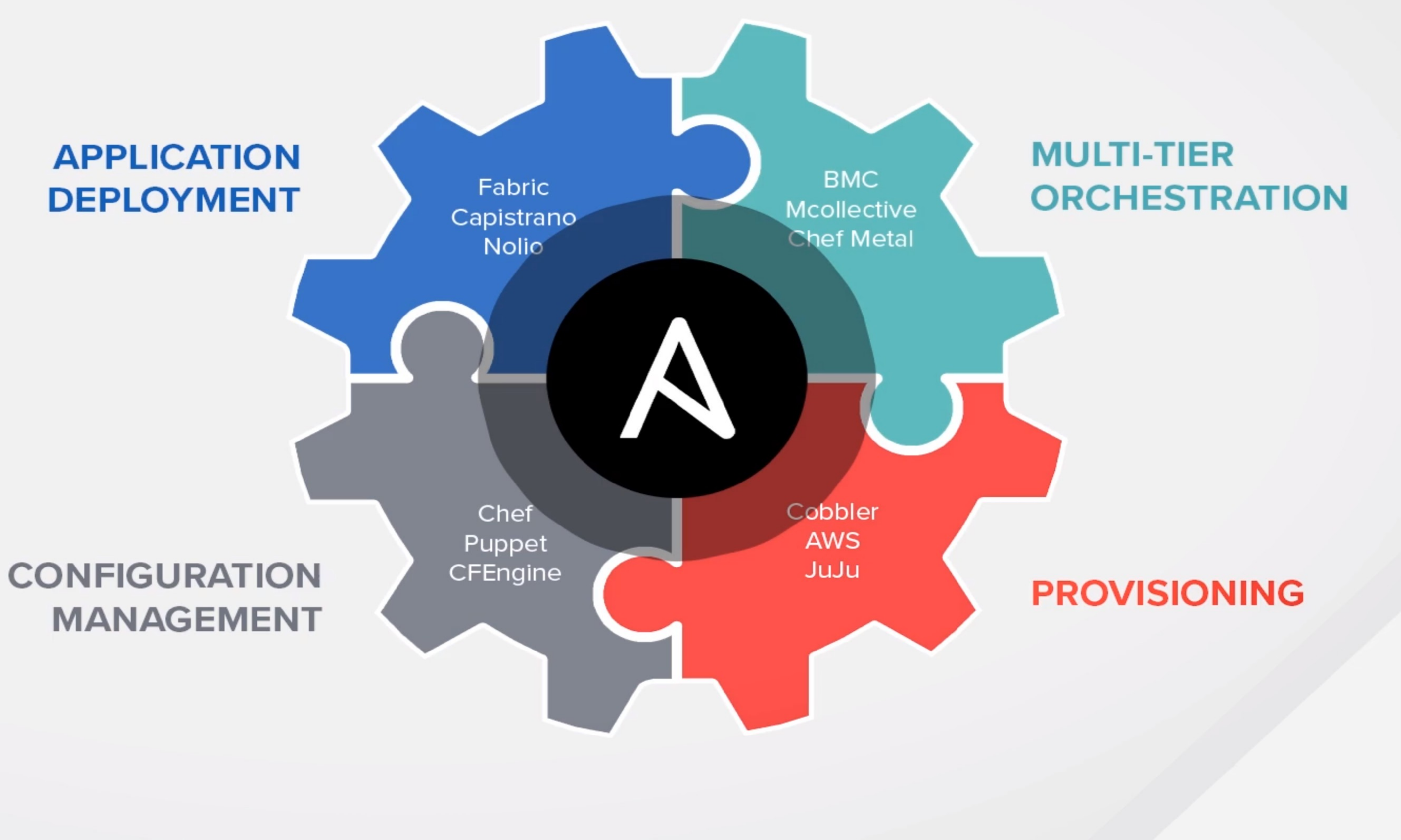
Ansible Concept

- Control node
- Managed nodes
- Inventory
- Modules
- Tasks
- Playbooks

Ansible Comes Bundled with Over 450 Modules

- Cloud
- Containers
- Database
- Files
- Messaging
- Monitoring
- Network
- Notifications
- Packaging
- Source Control
- System
- Testing
- Utilities
- Web Infrastructure

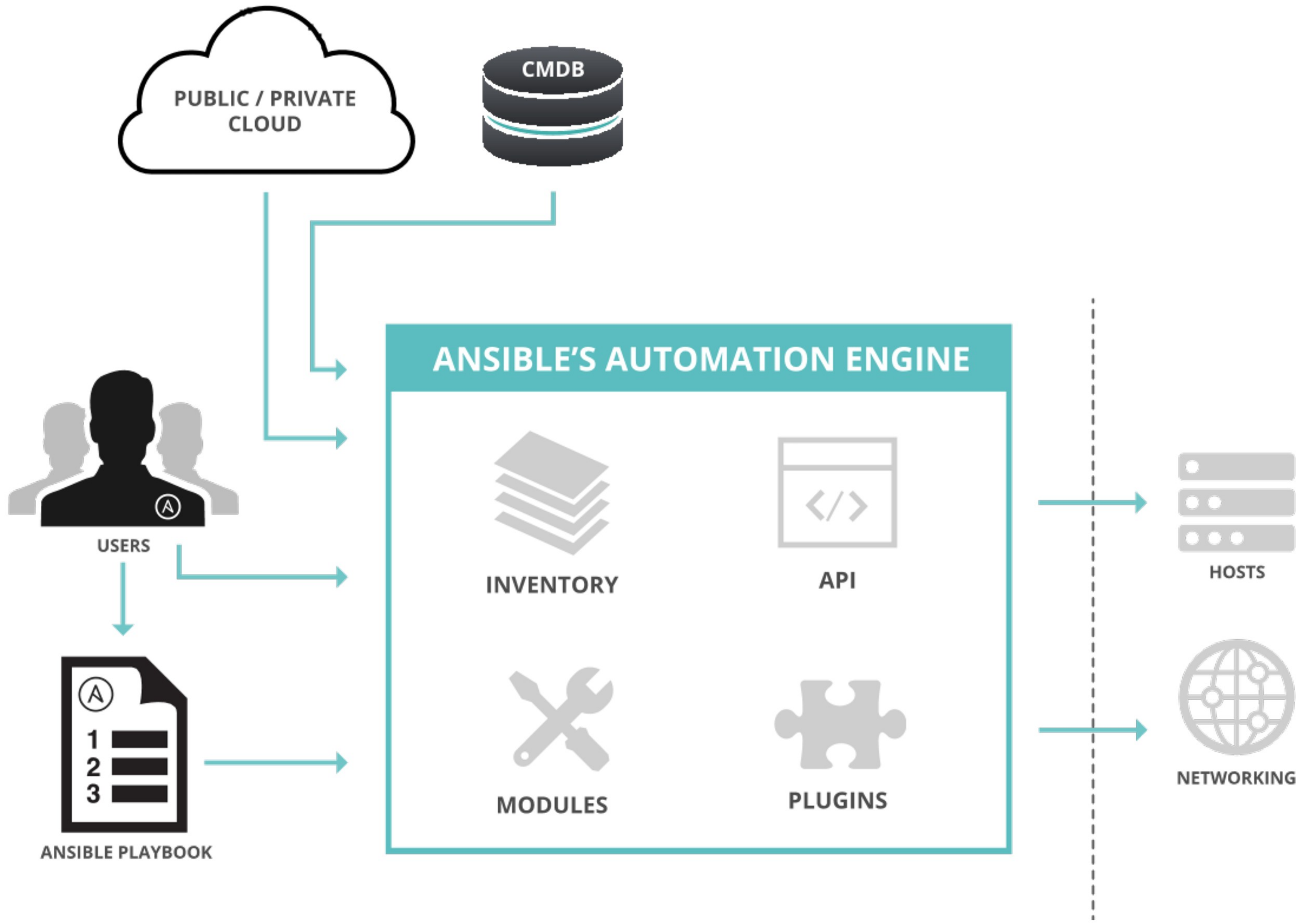
Ansible is Complete Package



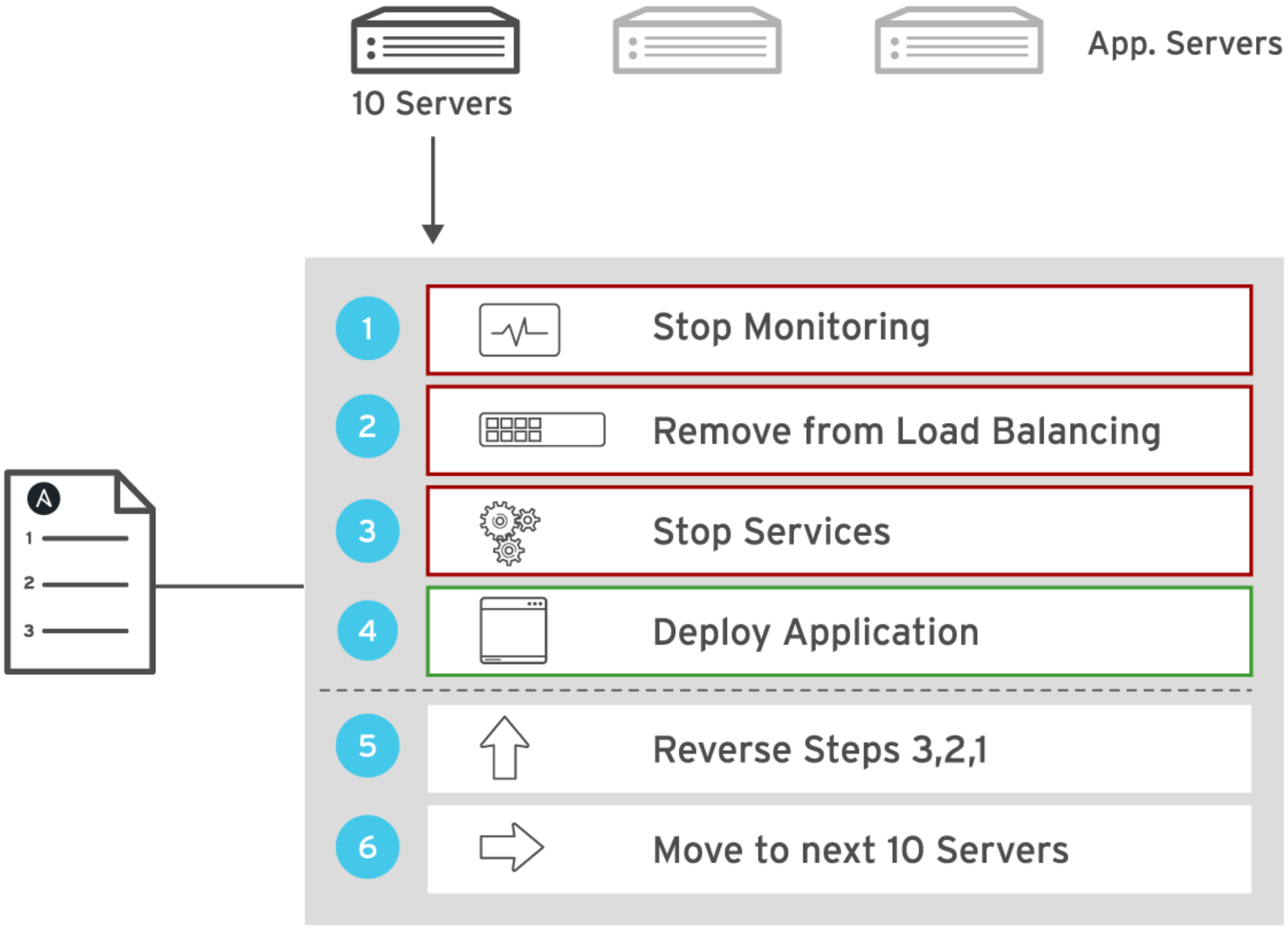
Ansible Use Cases

- Configuration Management
- Security and Compliance
- Application Deployment
- Orchestration
- Continuous Delivery
- Provisioning

Ansible Architecture



The Ansible Way



The Ansible Way (1)

- Complexity Kills Productivity
- Optimize For Readability
- Think Declaratively

Installing Ansible

Ansible Requirements

- Python 2.6 or higher
- paramiko
- PyYAML
- Jinja2
- httplib2
- Unix-based OS

Install Ansible

- CentOS

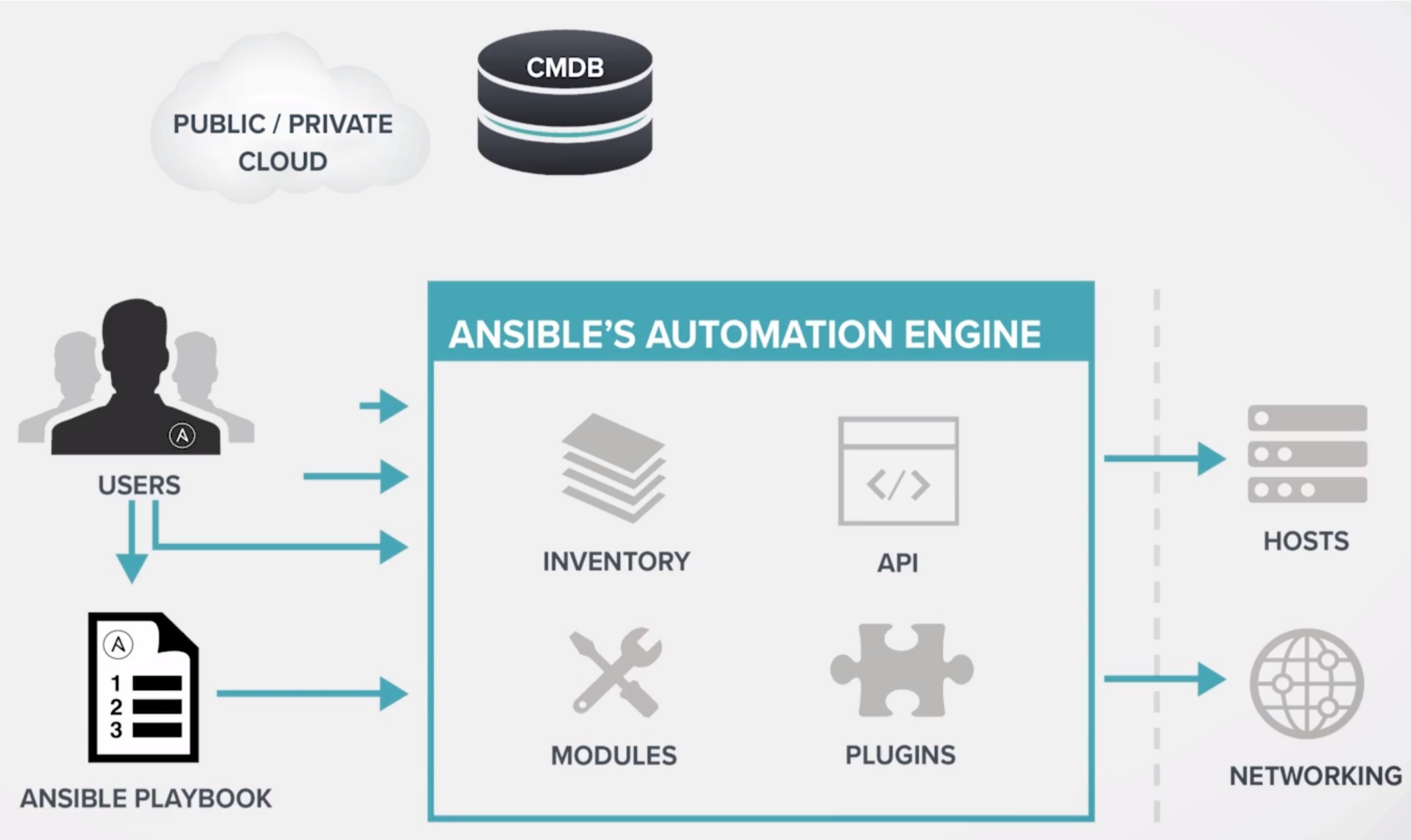
```
sudo yum install ansible
```

- Ubuntu

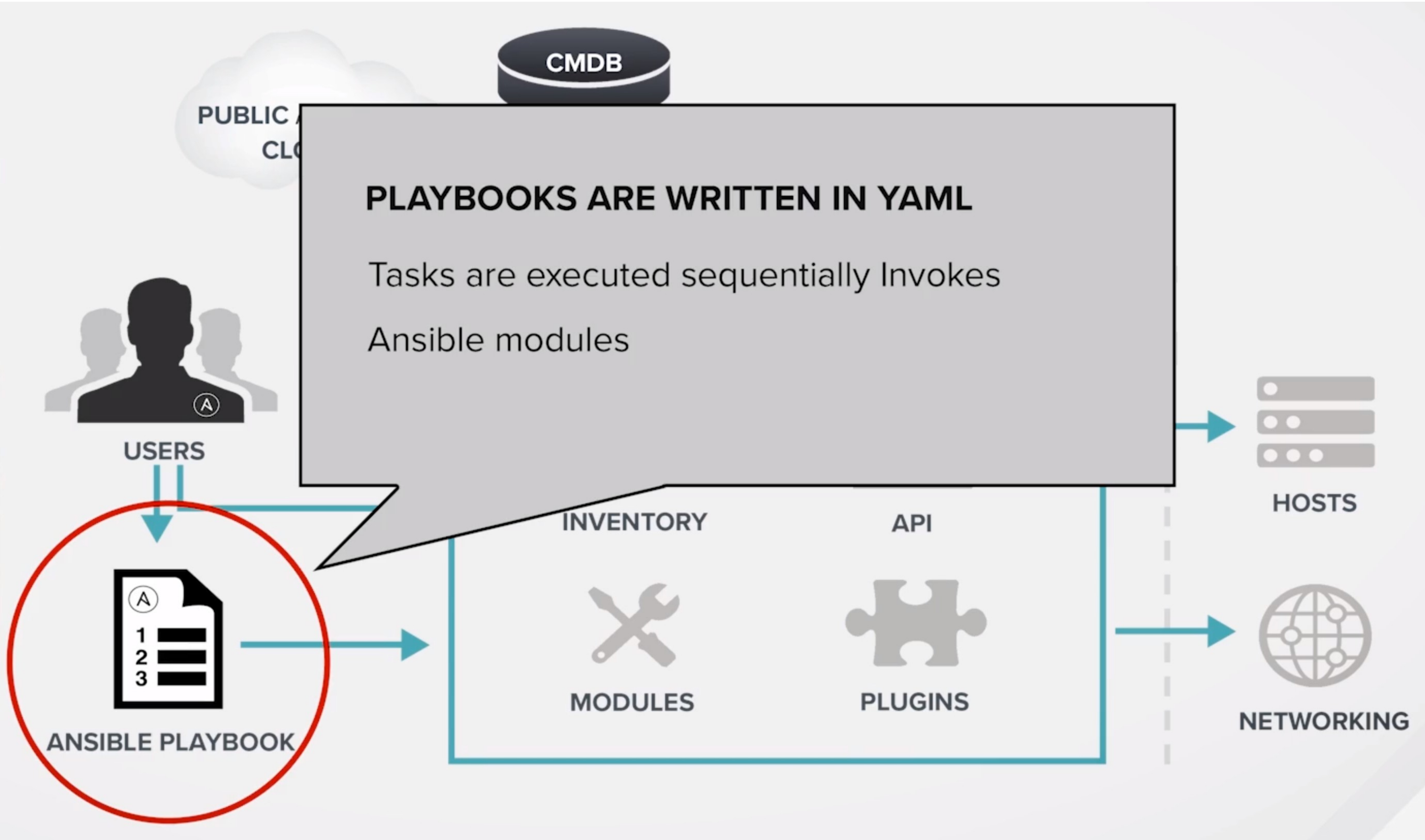
```
sudo apt-get update
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

How Ansible Works?

How Ansible Works



How Ansible Works (1)

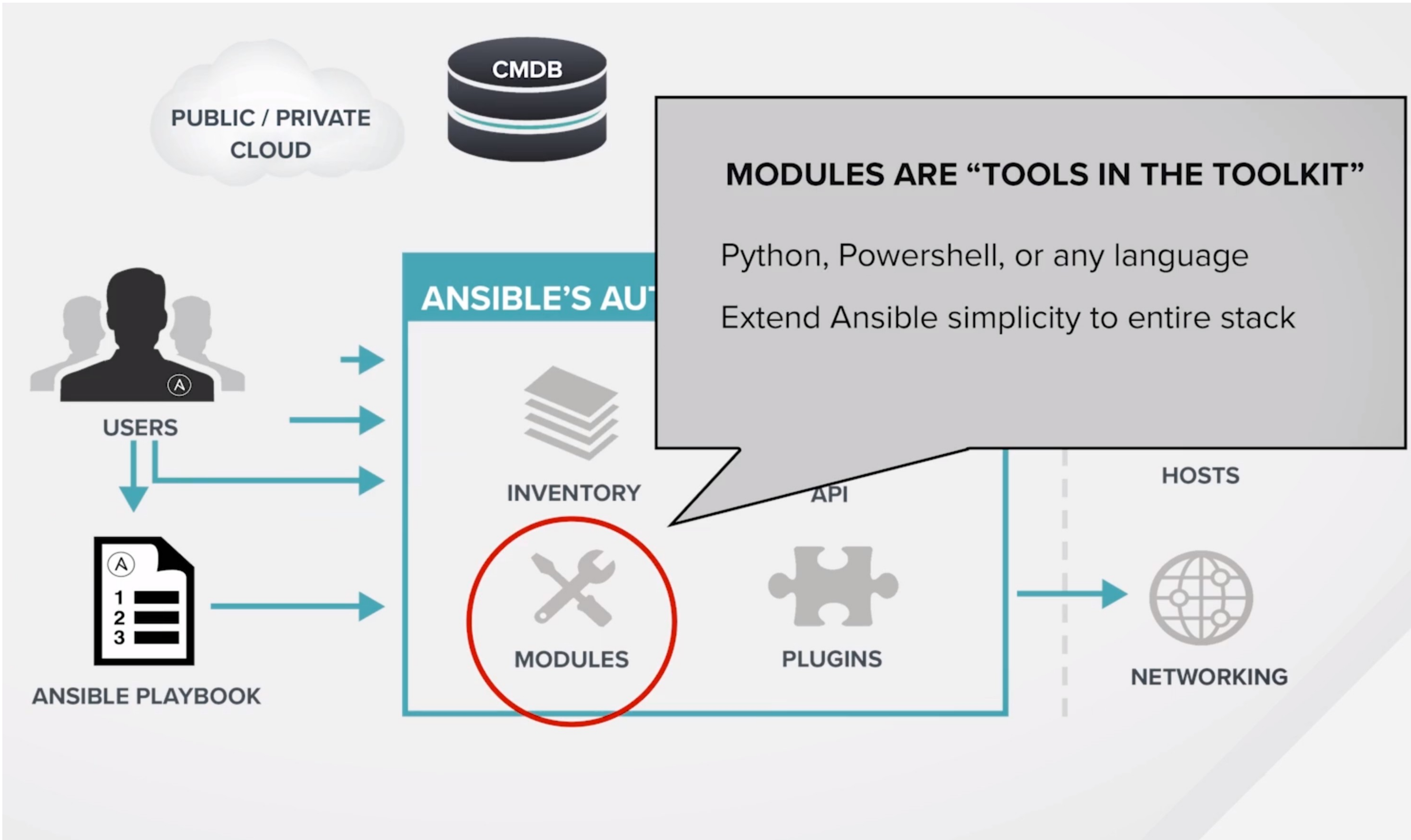


PLAYBOOKS ARE WRITTEN IN YAML

Tasks are executed sequentially Invokes

Ansible modules

How Ansible Works (2)

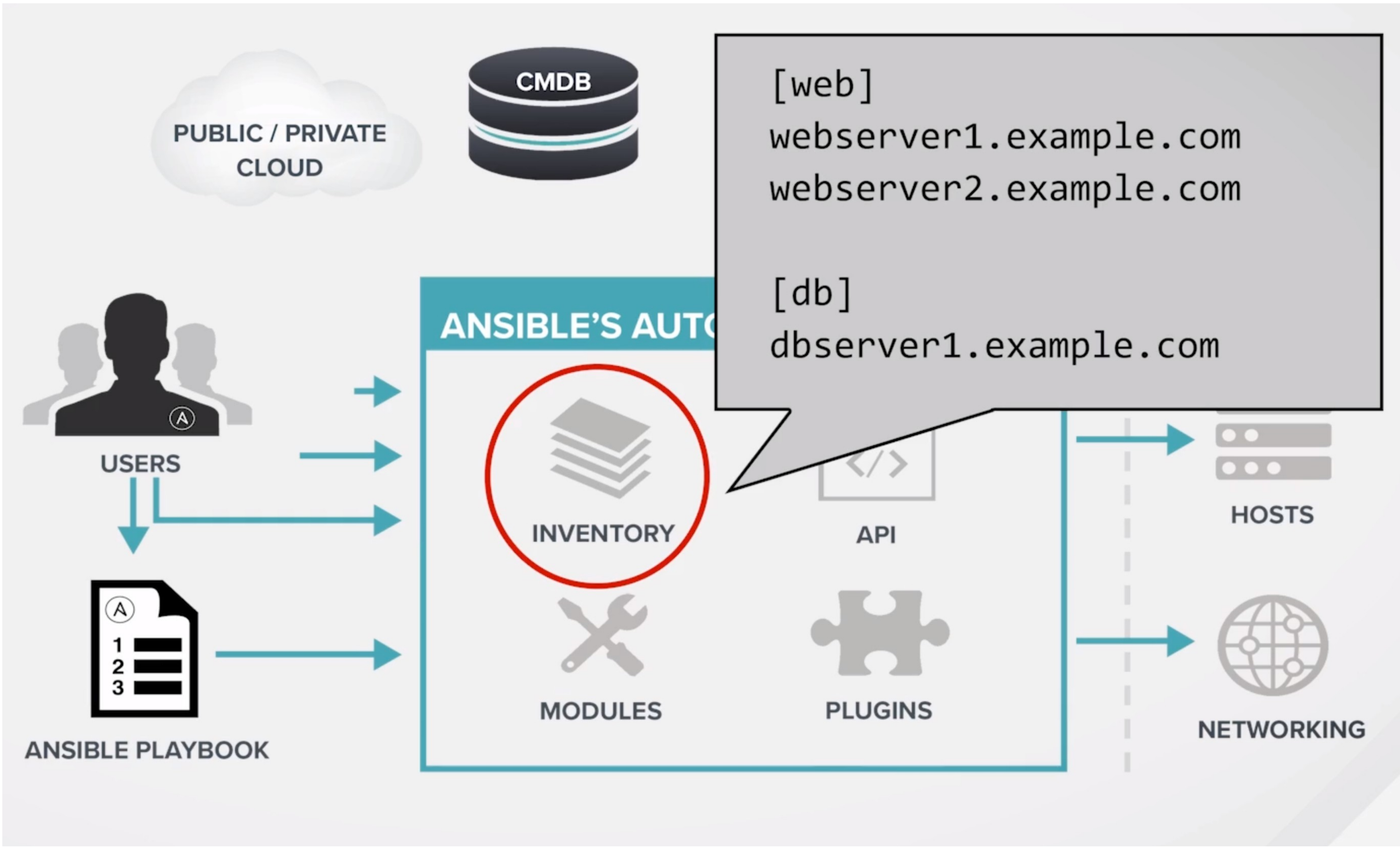


MODULES ARE "TOOLS IN THE TOOLKIT"

Python, Powershell, or any language

Extend Ansible simplicity to entire stack

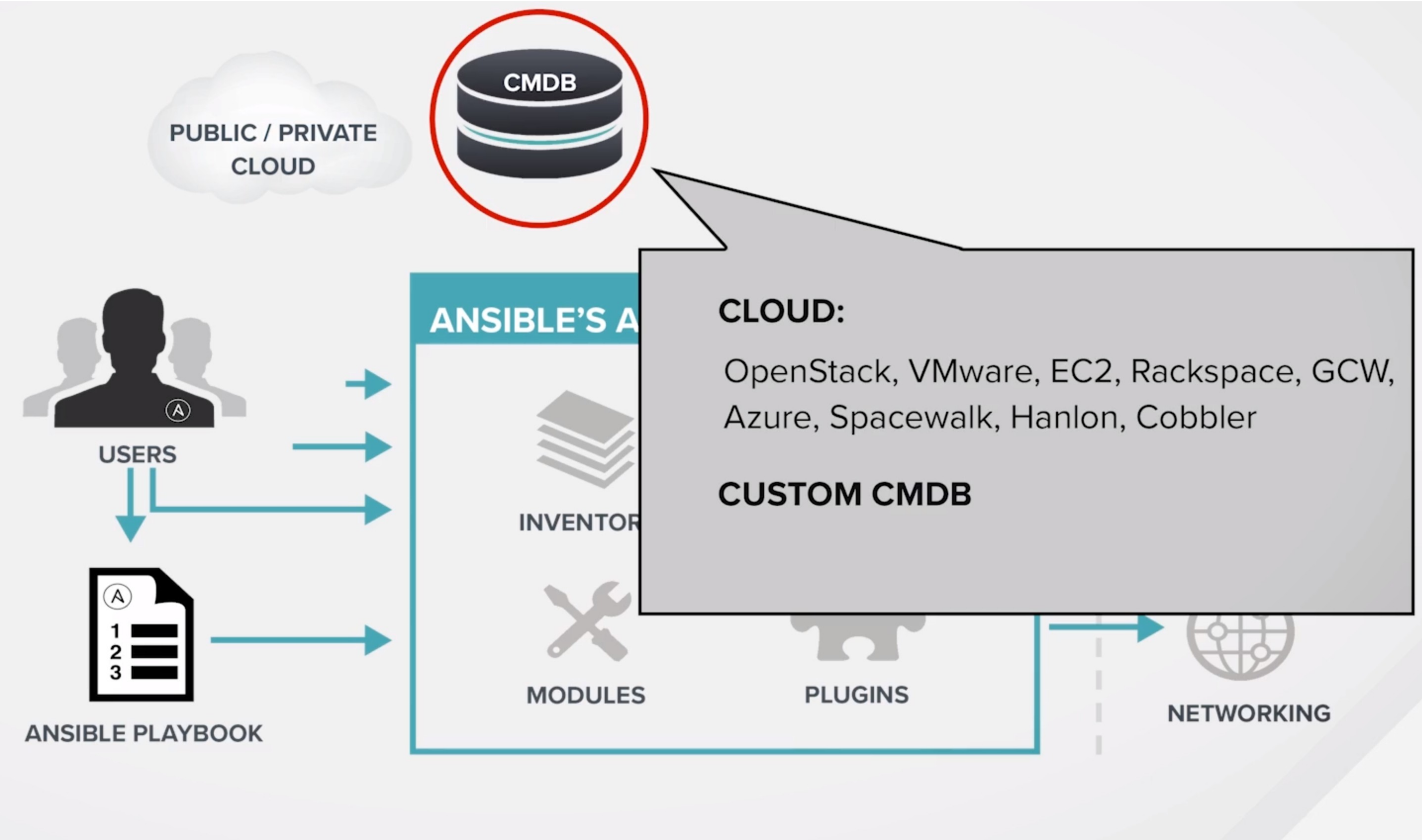
How Ansible Works (3)



```
[web]
webserver1.example.com
webserver2.example.com

[db]
dbserver1.example.com
```

How Ansible Works (4)

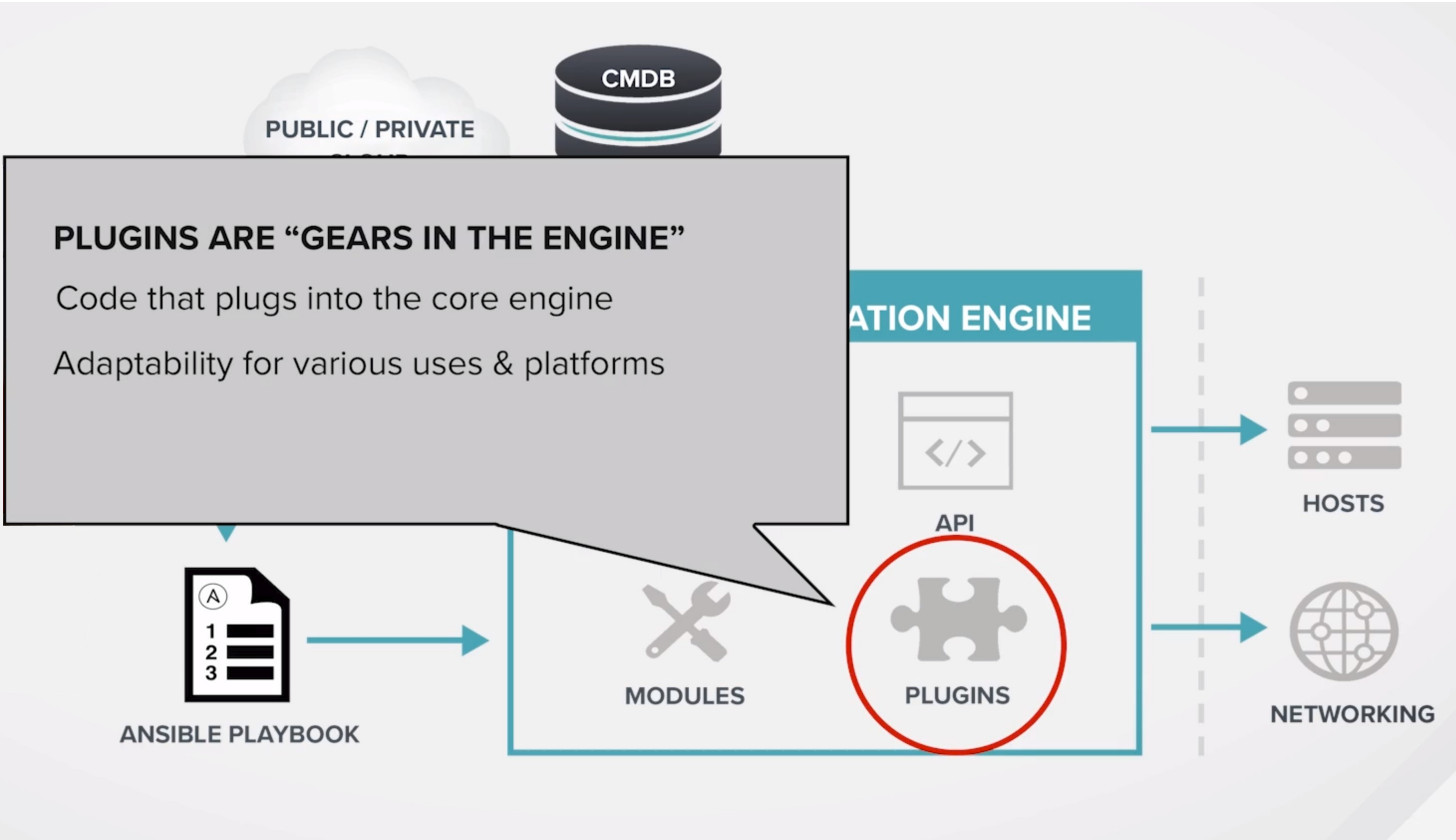


CLOUD:

OpenStack, VMware, EC2, Rackspace, GCW, Azure, Spacewalk, Hanlon, Cobbler

CUSTOM CMDB

How Ansible Works (5)



PLUGINS ARE "GEARS IN THE ENGINE"

Code that plugs into the core engine

Adaptability for various uses & platforms

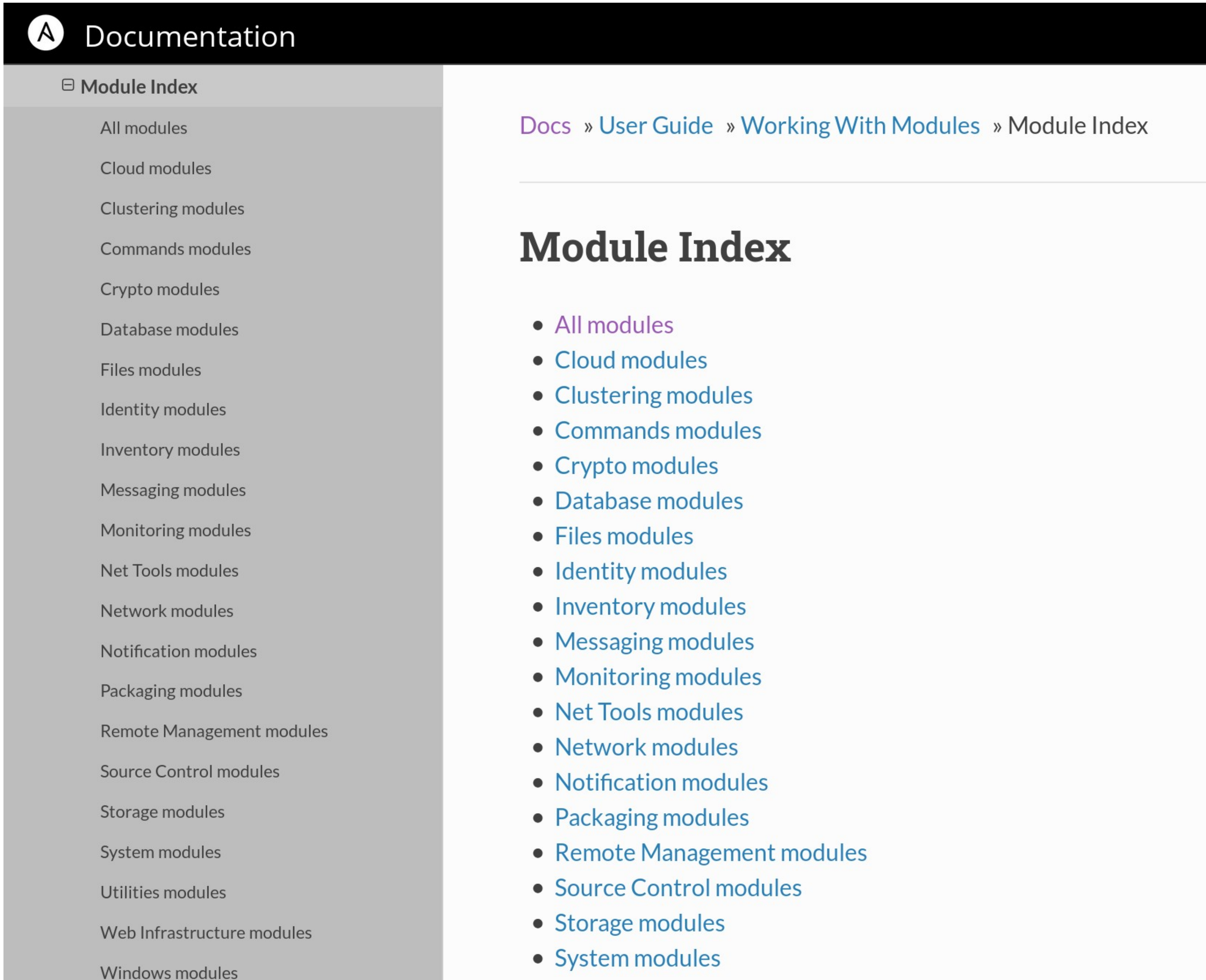
Modules

- apt/yum
- copy
- file
- get_url
- git
- ping
- debug
- service
- synchronize
- template
- uri
- user
- wait_for
- assert

Ansible Modules Category

MODULES CATEGORY	MODULES
Files Modules	• copy: Copy a local file to the managed host • file: Set permissions and other properties of files • lineinfile: Ensure a particular line is or is not in a file • synchronize: Synchronize content using rsync
Software Package Modules	• package: Manage packages using autodetected package manager native to the operating system • yum: Manage packages using the YUM package manager • apt: Manage packages using the APT package manage • dnf: Manage packages using the DNF package manager • gem: Manage Ruby gems • pip: Manage Python packages from PyPI • yum: Manage packages using the YUM package manager
System Modules	• firewalld: Manage arbitrary ports/services using firewalld • reboot: Reboot a machine • service: Manage services • user: Add, remove, and manage user accounts
Net Tools Modules	• get_url: Download files via HTTP, HTTPS, or FTP • nmcli: Manage networking • uri: Interact with web services

Modules Index



Documentation

Module Index

- All modules
- Cloud modules
- Clustering modules
- Commands modules
- Crypto modules
- Database modules
- Files modules
- Identity modules
- Inventory modules
- Messaging modules
- Monitoring modules
- Net Tools modules
- Network modules
- Notification modules
- Packaging modules
- Remote Management modules
- Source Control modules
- Storage modules
- System modules
- Utilities modules
- Web Infrastructure modules
- Windows modules

Docs » User Guide » Working With Modules » Module Index

Module Index

- All modules
- Cloud modules
- Clustering modules
- Commands modules
- Crypto modules
- Database modules
- Files modules
- Identity modules
- Inventory modules
- Messaging modules
- Monitoring modules
- Net Tools modules
- Network modules
- Notification modules
- Packaging modules
- Remote Management modules
- Source Control modules
- Storage modules
- System modules

Modules Run Command

- **command**: Takes the command and execute it. The most secure and predictable
- **shell**: Executes through a shell like /bin /sh so you can use pipes etc
- **script**: runs a local script   on a remote node after transferring it.
- **raw**: Executes a command without going through the Ansible module subsystem

Deploying Ansible

Managing Ansible Configuration Files

- **Using /etc/ansible/ansible.cfg**

  # The ansible package provides a base configuration file located at /etc/ansible/ansible.cfg. This file is used if no other configuration file is found.

- **Using ~/.ansible.cfg**

  # Ansible looks for a .ansible.cfg file in the user's home directory. This configuration is used instead of the /etc/ansible/ansible.cfg if it exists and if there is no ansible.cfg file in the current working directory

- **Using ./ansible.cfg .**

  # If an ansible.cfg file exists in the directory in which the ansible command is executed, it is used instead of the global file or the user's personal file.

- **Using the ANSIBLE_CONFIG environment variable**

  # You can use different configuration files by placing them in different directories and then executing Ansible commands from the appropriate directory, but this method can be restrictive and hard to manage as the number of configuration files grows.

Managing Setting in The Configuration File

For basic operation use the following two sections:

- [defaults] sets defaults for Ansible operation
- [privilege_escalation] configures how Ansible performs privilege escalation on managed hosts

- For example, the following is a typical ansible.cfg file:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Configuring Connections

- Ansible needs to know how to communicate with its managed hosts.
- One of the most common reasons to change the configuration file is to control which methods and users Ansible uses to administer managed hosts. Some of the information needed includes:

• The location of the inventory that lists the managed hosts and host groups

• Which connection protocol to use to communicate with the managed hosts (by default, SSH), and whether or not a nonstandard network port is needed to connect to the server

• Which remote user to use on the managed hosts; this could be root or it could be an unprivileged user

• If the remote user is unprivileged, Ansible needs to know if it should try to escalate privileges to root and how to do it (for example, by using sudo)

• Whether or not to prompt for an SSH password or sudo password to log in or gain privileges

Inventory Location

- In the **[defaults]** section, the **inventory** directive can point directly to a static inventory file, or to a directory that contains multiple static inventory files and/or dynamic inventory scripts.

```
[defaults]

inventory = ./inventory
```

Connection Settings

- Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the [defaults] section.
- By default, Ansible attempts to connect to the managed host using the same username as the local user running the Ansible commands. To specify a different remote user, set the **remote_user** parameter to that username.
- If the local user running Ansible has private SSH keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in.
- Otherwise you can configure Ansible to prompt the local user for the password used by the remote user by setting the directive **ask_pass = true**

```
[defaults]
inventory = ./inventory

remote_user = root
ask_pass = false
```

Privilege Escalation

- For security and auditing reasons, Ansible might need to connect to remote hosts as an unprivileged user before escalating privileges to get administrative access as root. This can be set up in the **[privilege_escalation]** section of the Ansible configuration file.
- To enable privilege escalation by default, set the directive **become = true** in the configuration file.
- The **become_method** directive specifies how to escalate privileges. Several options are available, but the default is to use **sudo**. Likewise, the **become_user** directive specifies which user to escalate to, but the default is root.
- If the become_method mechanism chosen requires the user to enter a password to escalate privileges, you can set the **become_ask_pass = true** directive in the configuration file.

Privilege Escalation (1)

- The following example **ansible.cfg** file assumes that you can connect to the managed hosts as someuser using SSH key-based authentication, and that someuser can use sudo to run commands as root without entering a password:

Example:
```
[defaults]
inventory = inventory.yaml
remote_user = someuser
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Ansible Configuration

DIRECTIVE	DESCRIPTION
inventory	Specifies the path to the inventory file.
remote_user	The name of the user to log in as on the managed hosts. If not specified, the current user's name is used.
ask_pass	Whether or not to prompt for an SSH password. Can be false if using SSH public key authentication.
become	Whether to automatically switch user on the managed host (typically to root) after connecting. This can also be specified by a play.
become_method	How to switch user (typically sudo, which is the default, but su is an option).
become_user	The user to switch to on the managed host (typically root, which is the default).
become_ask_pass	Whether to prompt for a password for your become_method. Defaults to false.

Ad-Hoc Commands

- ansible host-pattern -m module [-a 'module arguments'] [-i inventory]

Example:

- # check all my inventory hosts are ready to be
  # manage by Ansible
  $ ansible all -m ping

- # run the uptime command on all hosts in the web
  # group
  $ ansible web -m command -a “uptime”

- # collect and display the discovered for the localhost
  $ ansible localhost -m setup

Sidebar: Discovered Facts

```
$ ansible localhost -m setup
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.1.6"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::d0d8:fcbf:a20e:c97f"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
```

Configuring Connections for Ad Hoc Commands

- The directives for managed host connections and privilege escalation can be configured in the Ansible configuration file, and they can also be defined using options in ad hoc commands.
- The following table shows the analogous command-line options for each configuration file directive.

CONFIGURATION FILE DIRECTIVE	COMMAND LINE OPTION
inventory	-i
Remote User	-u
become	--become, -b
become_method	--become-method
become_user	--become-user
become_ask_pass	--ask-become-pass, -K

The Inventory

- Inventory is a collection of nodes or“hosts” againts which Ansible can work with.

The Inventory is consist of:

- Hosts
- Groups Sources
- Inventory-Specific data
- Static or dynamic

The Inventory

- Hosts and Groups
- Host Variables
- Group Variables
- Groups of Groups, and Group Variables
- Default groups
- Splitting Out Host and Group Specific Data
- List of Behavioral Inventory Parameters
- Non-SSH connection types

Static Inventory

```
10.1.1.10
10.1.1.20
10.1.1.30
10.1.1.40
```

```
[control]
control-node.example.com ansible_host=10.1.1.10

[webserver]
servera.example.com ansible_host=10.1.1.20

[dbserver]
serverb.example.com ansible_host=10.1.1.30

[haproxy]
haproxy.example.com ansible_host=10.1.1.40

[all:vars]
ansible_user: root
ansible_ssh_private_key_file: /root/.ssh/id_rsa
```

Defining Nested Group

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

```
deploy.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Simplifying Host Specifications with Ranges

- If you are adding a lot of hosts following similar patterns, you can do this rather than listing each hostname:

```
example:

[webservers]

server[a:e].example.com
```

- Output:

```
servera.example.com

serverb.example.com

serverc.example.com

serverd.example.com

servere.example.com
```

Simplifying Host Specifications with Ranges (1)

- For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive. You can also define alphabetic ranges:

```
example:
[databases]
db-[1:4].example.com
```

- Output :

```
[databases]
db-1.example.com
db-2.example.com
db-3.example.com
db-4.example.com
```

Verifying the Inventory

- $ ansible servera.example.com –list-hosts

```
hosts:
        servera.example.com
```

- $ ansible webservers –list-hosts

```
hosts :
        servera.example.com
        serverb.example.com
        serverc.example.com
        serverd.example.com
        servere.example.com
```

DEMO Ad-Hoc & Inventory

Introducing to Playbooks

Overview of Variable

- Ansible can work with metadata from various sources and manage their context in the form of variables.
- Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project. Examples of values that variables might contain include:
  - Users to create
  - Packages to install
  - Services to restart
  - Files to remove
  - Archives to retrieve from the internet

Naming Variable

INVALID VARIABLE NAMES	VALID VARIABLE NAMES
web server	web_server
remote.file	remote_file
1st file	file_1 / file1
remoteserver$1	remote_server_1 / remote_server1

Variable Precedence

1. Extra vars
2. Task vars (only for the task)
3. Block vars (only for tasks in the block)
4. Role and include vars
5. Play vars_files
6. Play vars_prompt
7. Play vars
8. Set_facts
9. Registered vars
10. Host facts
11. Playbook host_vars
12. Playbook group_vars
13. Inventory host_vars
14. Inventory group_vars
15. Inventory vars
16. Role defaults

Variables

- **File**: A directory should exist
- **Yum**: A package should be installed
- **Service**: A service should be running
- **Template**: Render a config file from a template
- **Get_url**: fetch an archive file from a URL
- **Git**: Clone a source code repository

Example Task in a Play

```
tasks:
  - name: add cache dir
    file:
      path: /opt/cache
      state: directory

  - name: install nginx
    yum:
      name: nginx
      state: latest

  - name: restart nginx
    service:
      name: nginx
      state: restarted
```

Handler Tasks

- Handlers are special tasks that run at the end of a play if notified by another task.
- If a configuration file gets changed notify a service restart task it needs to run.

Example Handler in a Play

```
tasks:
  - name: add cache dir
    file:
      path: /opt/cache
      state: directory

  - name: install nginx
    yum:
        name: nginx
        state: latest
    notify: restart nginx

handlers:
  - name: restart nginx
    service:
        name: nginx
        state: restarted
```

Play and Playbooks

- Plays are ordered sets of tasks to execute against host selections from your inventory.

- A playbook is a file containing one or more plays.

Playbook Example

```
---
- 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 log file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Playbook Example (1)

```
---
- 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 log file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Playbook Example (2)

```
---
- 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 log file
     template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
     service: name=httpd state=started
```

Playbook Example (3)

```
---
- 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 log file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Playbook Example (4)

```
---
- 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 log file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Playbook Example (5)

```
---
- 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 log file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

Writing Multiple Plays

- A playbook is a YAML file containing a list of one or more plays.
- Remember that a single play is an ordered list of tasks to execute against hosts selected from the inventory.
- Therefore, if a playbook contains multiple plays, each play may apply its tasks to a separate set of hosts.

Example Simple Playbook with Two Plays

```
---
# This is a simple playbook with two plays
- name: first play
  hosts: web.example.com
  tasks:

  - name: first task
     yum:
         name: httpd
         status: present

- name: second task
   service:
       name: httpd
       enabled: true

- name: second play
  hosts: database.example.com
  tasks:

- name: first task
  service:
      name: mariadb
      enabled: true
```

DEMO Playbook

Managing Secrets

Ansible Vault

- Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles.
- These vault files can then be distributed or placed in source control.
- To use Ansible Vault, a command-line tool named ansible-vault is used to create, edit, encrypt, decrypt, and view files.
- Ansible Vault does not implement its own cryptographic functions but rather uses an external Python toolkit.
- Files are protected with symmetric encryption using AES256 with a password as the secret key.

Creating an Encrypted File

- To create a new encrypted file, use the **ansible-vault create** *filename* command. The command prompts for the new vault password and then opens a file using the default editor, vi.

```
$ ansible-vault create secret.yml

New Vault password: rahasia

Confirm New Vault password: rahasia
```

- Instead of entering the vault password through standard input, you can use a vault password file to store the vault password. You need to carefully protect this file using file permissions and other means.

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

- The cipher used to protect files is AES256 in recent versions of Ansible, but files encrypted with older versions may still use 128-bit AES.

Viewing an Encrypted File

- You can use the **ansible-vault view** *filename* command to view an Ansible Vault-encrypted file without opening it for editing.

```
$ ansible-vault view secret1.yml
Vault password: rahasia
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
my_secret: "yJJvPqhsiusmmPPZdnjndkdnYNDjdj782meUZcw"
```

Editing an Existing Encrypted File

- To edit an existing encrypted file, Ansible Vault provides the **ansible-vault edit** *filename* command.
- This command decrypts the file to a temporary file and allows you to edit it.
- When saved,it copies the content and removes the temporary file.

```
$ ansible-vault edit secret.yml

Vault password: rahasia
```

Encrypting an Existing File

- To encrypt a file that already exists, use the **ansible-vault encrypt** *filename* command.
- This command can take the names of multiple files to be encrypted as arguments.

```
$ ansible-vault encrypt secret1.yml secret2.yml

New Vault password: rahasia

Confirm New Vault password: redhat

Encryption successful
```

- Use the **--output=OUTPUT_FILE** option to save the encrypted file with a new name.
- At most one input file may be used with the **--output** option.

Decrypting an Existing File

- An existing encrypted file can be permanently decrypted by using the **ansible-vault decrypt** *filename* command. When decrypting a single file, you can use the --output option to save the decrypted file under a different name.

```
$ ansible-vault decrypt secret1.yml --output=secret1-decrypted.yml
Vault password: rahasia
Decryption successful
```

Changing The Password of an Encrypted File

- You can use the **ansible-vault rekey** *filename* command to change the password of an encrypted file. This command can rekey multiple data files at once. It prompts for the original password and then the new password.

```
$ ansible-vault rekey secret.yml
Vault password: rahasia
New Vault password: RaHaSia
Confirm New Vault password: RaHaSia
Rekey successful
```

Playbooks and Ansible Vault

- To run a playbook that accesses files encrypted with Ansible Vault, you need to provide the encryption password to the ansible-playbook command. If you do not provide the password, the playbook returns an error:

```
$ ansible-playbook site.yml

ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

- To provide the vault password to the playbook, use the --vault-id option. For example, to provide the vault password interactively, use --vault-id @prompt as illustrated in the following example:

```
$ ansible-playbook --vault-id @prompt site.yml

Vault password (default): rahasia
```

Managing Facts

Ansible Facts

- Ansible facts are variables that are automatically discovered by Ansible on a managed host.

- Facts contain host-specific information that can be used just like regular variables in:

  Plays, conditionals, loops, or any other statement that depends on a value collected from a managed host.

Some of The Facts Gathered for a Managed Host

- The host name.
- The kernel version.
- The network interfaces.
- The IP addresses.
- The version of the operating system.
- Various environment variables.
- The number of CPUs.
- The available or free memory.
- The available disk space.

Another Function of Facts

Facts are a convenient way to retrieve the state of a managed host and to determine what action to take based on that state. For example:

- A server can be restarted by a conditional task which is run based on a fact containing the managed host's current kernel version.
- The MySQL configuration file can be customized depending on the available memory reported by a fact.
- The IPv4 address used in a configuration file can be set based on the value of a fact.

Viewing All Facts

- A simple Playbook gathers facts and uses the debug module to print the value of the ansible_facts variable.

```
- name: Fact dump

  hosts: all

  tasks:
    - name: Print all facts

      debug:

      var: ansible_facts
```

- Ad-Hoc Command

```
$ ansible hostname -m setup
```

Viewing a Subset of Facts

- Because Ansible collects many facts, the setup module supports a filter parameter that lets you filter by fact name by specifying a glob.

  Example

```
$ ansible web -m setup -a 'filter=ansible_eth*'
```

Turn Off Fact Gathering

- To disable fact gathering for a play, set the **gather_facts** keyword to **no:**

```
---
- name: This play gathers no facts automatically
  hosts: large_farm
  gather_facts: no
```

Example Static Custom Facts

- This is an example of a static custom facts file written in INI format.
- An INI-formatted custom facts file contains a top level defined by a section, followed by the key-value pairs of the facts to define:

```
[packages]

web_package = httpd

db_package = mariadb-server


[users]

user1 = joe

user2 = jane
```

Example Static Custom Facts in JSON

- The following JSON facts are equivalent to the facts specified by the INI format in the preceding example.

```
{
   "packages":{
      "web_package":"httpd",
      "db_package":"mariadb-server"


},
   "users":{
      "user1":"joe",
      "user2":"jane"


}
}
```

Verify Custom Facts

```
$ **ansible demo1.example.com -m setup**
demo1.example.com | SUCCESS => {
        "ansible_facts": {
...output omitted...
          "ansible_local": {
                "custom": {
                     "packages": {
                           "db_package": "mariadb-server",
                           "web_package": "httpd"
                     },
                            "users": {
                                     "user1": "joe",
                                     "user2": "jane"
                            }
                        }
                 },
...output omitted...
     },
     "changed": false
}
```

LAB 1

Modifying and Copying Files to Hosts

Describing Files Module

- The Files modules library includes modules that allow you to accomplish most tasks related to Linux file management, such as creating, copying, editing, and modifying permissions and other attributes of files.

Common Files Modules

Module Name	Module Description
blockinfile	Insert, update, or remove a block of multiline text surrounded by customizable marker lines.
copy	Copy a file from the local or remote machine to a location on a managed host. Similar to the file module, the copy module can also set file attributes, including SELinux context.
fetch	This module works like the copy module, but in reverse. This module is used for fetching files from remote machines to the control node and storing them in a file tree, organized by host name.
file	Set attributes such as permissions, ownership, SELinux contexts, and time stamps of regular files, symlinks, hard links, and directories. This module can also create or remove regular files, symlinks, hard links, and directories. A number of other file-related modules support the same options to set attributes as the file module, including the copy module.
lineinfile	Ensure a particular line is in a file, or replace an existing line using a backreference regular expression. This module is primarily useful when you want to change a single line in a file.
stat	Retrieve status information for a file, similar to the Linux stat command.
synchronize	A wrapper around the **rsync** command to make common tasks quick and easy. The synchronize module is not intended to provide access to the full power of the **rsync** command, but does make the most common invocations easier to implement. You may still need to call the **rsync** command directly via a run command module depending on your use case.

Ensuring a File Exists on Managed Hosts

- In this example, in addition to touching the file, Ansible makes sure that the owning user, group, and permissions of the file are set to specific values.

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

- Outcome

```
$ ls -l file
-rw-r----- user1 group1 0 Nov 25 08:00 file
```

Copying and Editing Files on Managed Hosts

- The **copy** module is used to copy a file located in the Ansible working directory on the control node to selected managed hosts.
- By default this module assumes **force: yes** is set. That forces the module to overwrite the remote file if it exists but contains different contents from the file being copied.
- **If force: no** is set, then it only copies the file to the managed host if it does not already exist.

```
- name: Copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

Copying and Editing Files on Managed Hosts (1)

- To ensure a specific single line of text exists in an existing file, use the lineinfile module:

```
- name: Add a line of text to a file
  lineinfile:
    path: /path/to/file
    line: 'Add this line to the file'
    state: present
```

- To add a block of text to an existing file, use the blockinfile module:

```
- name: Add additional lines to a file
  blockinfile:
     path: /path/to/file
     block: |
         First line in the additional block of text
         Second line in the additional block of text
state: present
```

Removing a File from Managed Hosts

- A basic example to remove a file from managed hosts is to use the **file** module with the **state: absent** parameter.
- The **state** parameter is optional to many modules.
- You should always make your intentions clear whether you want **state: present** or **state: absent** for several reasons.

```
- name: Make sure a file does not exist on managed hosts
  file:
    dest: /path/to/file
    state: absent
```

Retrieving the Status of a File on Managed Hosts

- The **stat** module retrieves facts for a file, similar to the Linux **stat** command.
- The **stat** module returns a hash/dictionary of values containing the file status data, which allows you to refer to individual pieces of information using separate variables.
- The following example registers the results of a stat module and then prints the MD5 checksum of the file that it checked.

```
- name: Verify the checksum of a file
  stat:
    path: /path/to/file
    checksum_algorithm: md5
  register: result
- debug
     msg: "The checksum of the file is {{ result.stat.checksum }}"
```

- The outcome:

```
TASK [Get md5 checksum of a file] *****************************************
ok: [hostname]
TASK [debug] ***************************************************************
ok: [hostname] => {
     "msg": "The checksum of the file is 5f76590425303022e933c43a7f2092a3"
}
```

Synchronizing Files Between the Control Node and Managed Hosts

- The **synchronize** module is a wrapper around the **rsync** tool, which simplifies common file management tasks in your playbooks.
- The **rsync** tool **must be installed on both** the local and remote host

```
- name: synchronize local file to remote files
  synchronize:
      src: file
      dest: /path/to/file
```

Deploying Custom Files with Jinja2 Templates

Templating Files

- A much more powerful way to manage files is to template them.
- With this method, you can write a template configuration file that is automatically customized for the managed host when the file is deployed, using Ansible variables and facts.
- This can be easier to control and is less error-prone.

Introduction to Jinja2

- Ansible uses the Jinja2 templating system for template files.
- Ansible also uses Jinja2 syntax to reference variables in playbooks, so you already know a little bit about how to use it.

Building a Jinja2 Template

- A Jinja2 template is composed of multiple elements: data, variables, and expressions.
- Those variables and expressions are replaced with their values when the Jinja2 template is rendered.
- The variables used in the template can be specified in the vars section of the playbook.
- The following example shows how to create a template with variables using two of the facts retrieved by Ansible from managed hosts: **ansible_facts.hostname** and **ansible_facts.date_time.date**

```
Welcome to {{ ansible_facts.hostname }}.
Today's date is: {{ ansible_facts.date_time.date }}.
```

Deploying Jinja2 Template

- Jinja2 templates are a powerful tool to customize configuration files to be deployed on the managed hosts.
- When the Jinja2 template for a configuration file has been created, it can be deployed to the managed hosts using the template module.

```
tasks:
  - name: template render
    template:
      src: /tmp/j2-template.j2
      dest: /tmp/dest-config-file.txt
```

Managing Templated Files

- To avoid having system administrators modify files deployed by Ansible, it is a good practice to include a comment at the top of the template to indicate that the file should not be manually edited.
- One way to do this is to use the 'Ansible managed' string set in the ansible_managed directive.
- The ansible_managed directive is set in the ansible.cfg file:

```
ansible_managed = Ansible managed
```

- To include the ansible_managed string inside a Jinja2 template, use the following syntax:

```
{{ ansible_managed }}
```

Managing Large Projects

Selecting Host with Host Patterns

- Host patterns are used to specify the hosts to target by a play or ad hoc command.
- Host patterns are important to understand. It is usually easier to control what hosts a play targets by carefully using host patterns and having appropriate inventory groups, instead of setting complex conditionals on the play's tasks.

```
$ cat myinventory
web.example.com
data.example.com

[lab]
labhost1.example.com
labhost2.example.com

[test]
test1.example.com
test2.example.com

[datacenter1]
labhost1.example.com
test1.example.com

[datacenter2]
labhost2.example.com
test2.example.com

[datacenter:children]
datacenter1
datacenter2

[new]
192.168.2.1
192.168.2.2
```

Manage Hosts

- The most basic host pattern is the name for a single managed host listed in the inventory. This specifies that the host will be the only one in the inventory that will be acted upon by the **ansible** command.
- When the playbook runs, the first Gathering Facts task should run on all managed hosts that match the host pattern.
- The following example shows how a host pattern can be used to reference an IP address contained in an inventory

```
$ cat playbook.yml
---
- hosts: 192.168.2.1
...output omitted...

[student@controlnode ~]$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ************************************************
TASK [Gathering Facts] ***************************************************
ok: [192.168.2.1]
...output omitted..
```

Groups

- When a group name is used as a host pattern, it specifies that Ansible will act on the hosts that are members of the group.

```
$ cat playbook.yml
---
- hosts: lab
...output omitted..
.
[student@controlnode ~]$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ***************************************************
TASK [Gathering Facts] ******************************************************
ok: [labhost1.example.com]
ok: [labhost2.example.com]
...output omitted...
```

- Remember that there is a special group named all that matches all managed hosts in the inventory.

```
$ cat playbook.yml
---
- hosts: all
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ***************************************************
TASK [Gathering Facts] ******************************************************
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

Groups (1)

- There is also a special group named ungrouped which matches all managed hosts in the inventory that are not members of any other group

```
$ cat playbook.yml
---
- hosts: ungrouped
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] **************************************************
TASK [Gathering Facts] *****************************************************
ok: [web.example.com]
ok: [data.example.com]
```

Wildcards

- Another method of accomplishing the same thing as the all host pattern is to use the asterisk (*) wildcard character, which matches any string. If the host pattern is just a quoted asterisk, all hosts in the inventory will match.

```
$ cat playbook.yml
---
- hosts: '*'
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ***********************************************
TASK [Gathering Facts] ****************************************************
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

Wildcards (1)

- The asterisk character can also be used like file globbing to match any managed hosts or groups that contain a particular substring.
- For example, the following wildcard host pattern matches all inventory names that end in

  .example.com:

```
$ cat playbook.yml
---
- hosts: '*.example.com'
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] **************************************************
TASK [Gathering Facts] *****************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
```

Wildcards (2)

- he following example uses a wildcard host pattern to match the names of hosts or host groups that start with **192.168.2.**:

```
$ cat playbook.yml
---
- hosts: '192.168.2.*'
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ***********************************************************
TASK [Gathering Facts] **************************************************************
ok: [192.168.2.1]
ok: [192.168.2.2]
```

- The next example uses a wildcard host pattern to match the names of hosts or host groups that begin with **datacenter**.

```
$ cat playbook.yml
---
- hosts: 'datacenter*'
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ***********************************************************
TASK [Gathering Facts] **************************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
```

List

- Multiple entries in an inventory can be referenced using logical lists. A comma-separated list of host patterns matches all hosts that match any of those host patterns.
- If you provide a comma-separated list of managed hosts, then all those managed hosts will be targeted:

```
$ cat playbook.yml
---
- hosts: labhost1.example.com,test2.example.com,192.168.2.2
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] *****************************************************
TASK [Gathering Facts] ***********************************************************
ok: [labhost1.example.com]
ok: [test2.example.com]
ok: [192.168.2.2]
```

- If you provide a comma-separated list of groups, then all hosts in any of those groups will be targeted:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,datacenter1
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] *****************************************************
TASK [Gathering Facts] ***********************************************************
ok: [labhost1.example.com]
ok: [labhost2.example.com]
ok: [test1.example.com]
```

List (1)

- You could also specify that machines in the **datacenter1** group match only if they are in the lab group with the host patterns **&lab,datacenter1** or **datacenter1,&lab**.

- You can exclude hosts that match a pattern from a list by using the exclamation point or "bang" character (!) in front of the host pattern. This operates like a logical NOT.

- This example, given our test inventory, matches all hosts defined in the datacenter group, with the exception of **test2.example.com**:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: datacenter,!test2.example.com
...output omitted...

$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ****************************************************
TASK [Gathering Facts] *****************************************************
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
```

- The final example shows the use of a host pattern that matches all hosts in the test inventory, with the exception of the managed hosts in the datacenter1 group.

```
$ cat playbook.yml
---
- hosts: all,!datacenter1
...output omitted...
$ ansible-playbook playbook.yml
PLAY [Test Host Patterns] ****************************************************
TASK [Gathering Facts] *****************************************************
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [192.168.2.1]
ok: [192.168.2.2]
```

Introducing to Roles

Roles

- Roles are a package of closely related Ansible content that can be shared more easily than plays alone.
- Ansible roles provide a way for you to make it easier to reuse Ansible code in a generic way.
- You can package, in a standardized directory structure, all the tasks, variables, files, templates, and other resources needed to provision infrastructure or deploy applications.

Benefit of Ansible Roles

- Roles group content, allowing easy sharing of code with others
- Roles can be written that define the essential elements of a system type: web server, database server, Git repository, or other purpose
- Roles make larger projects more manageable
- Roles can be developed in parallel by different administrators

Examining The Ansible Role Structure

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Roles Directory

- Roles expect files to be in certain directory names.
- Roles must include at least one of these directories, however it is perfectly fine to exclude any which are not being used.
- When in use, each directory must contain a **main.yml** file, which contains the relevant content:

Roles Directory (1)

- **defaults -** The main.yml file in this directory contains the default values of role variables that can be overwritten when the role is used.
- **tasks** - contains the main list of tasks to be executed by the role.
- **handlers** - contains handlers, which may be used by this role or even anywhere outside this role.
- **defaults** - default variables for the role (see Using Variables for more information).
- **vars** - other variables for the role (see Using Variables for more information).
- **files** - contains files which can be deployed via this role.
- **templates** - contains templates which can be deployed via this role.
- **meta** - defines some meta data for this role. See below for more details.
- **meta** - This directory can contain an inventory and test.yml playbook that can be used to test the role..

Playbooks with Roles

```
# site.yml
---
- hosts: web
  Roles:
        - common
        - webservers
```

Creating a New Role

- Creating roles in Ansible requires no special development tools.
- Creating and using a role is a three step process:
  - Create the role directory structure.
  - Define the role content.
  - Use the role in a playbook

Creating The Role Directory Structure

- By default, Ansible looks for roles in a subdirectory called roles in the directory containing your Ansible Playbook.
- You can create all of the subdirectories and files needed for a new role using standard Linux commands.
- Alternatively, command line utilities exist to automate the process of new role creation.
- The **ansible-galaxy** command line tool (covered in more detail later in this course) is used to manage Ansible roles, including the creation of new roles.

```
[user@host playbook-project]$ cd roles

[user@host roles]$ ansible-galaxy init my_new_role

- my_new_role was created successfully



[user@host roles]$ ls my_new_role/

defaults files handlers meta README.md tasks templates tests vars
```

Defining Role Content

- Once you have created the directory structure, you must write the content of the role. A good place to start is the ***ROLENAME*/tasks/main.yml** task file, the main list of tasks run by the role

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd
- name: deliver motd file
  template:
    src: motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

- The following command displays the contents of the **motd.j2** template of the motd role. It references Ansible facts and a **system_owner** variable

```
[user@host ~]$ cat roles/motd/templates/motd.j2
This is the system {{ ansible_facts['hostname'] }}.
Today's date is: {{ ansible_facts['date_time']['date'] }}.
Only use this system with permission.
You can ask {{ system_owner }} for access.
```

Defining Role Dependencies

- Role dependencies allow a role to include other roles as dependencies.
- For example, a role that defines a documentation server may depend upon another role that installs and configures a web server.
- Dependencies are defined in the meta/main.yml file in the role directory hierarchy.
- The following is a sample meta/main.yml file.

```
---
dependencies:
  - role: apache
    port: 8080
  - role: postgres
    dbname: serverlist
    admin_user: felix
```

- By default, roles are only added as a dependency to a playbook once

Using The Role in a Playbook

- To access a role, reference it in the **roles**: section of a playbook. The following playbook refers to the **motd** role. Because no variables are specified, the role is applied with its default variable values.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true
  roles:
    - motd
```

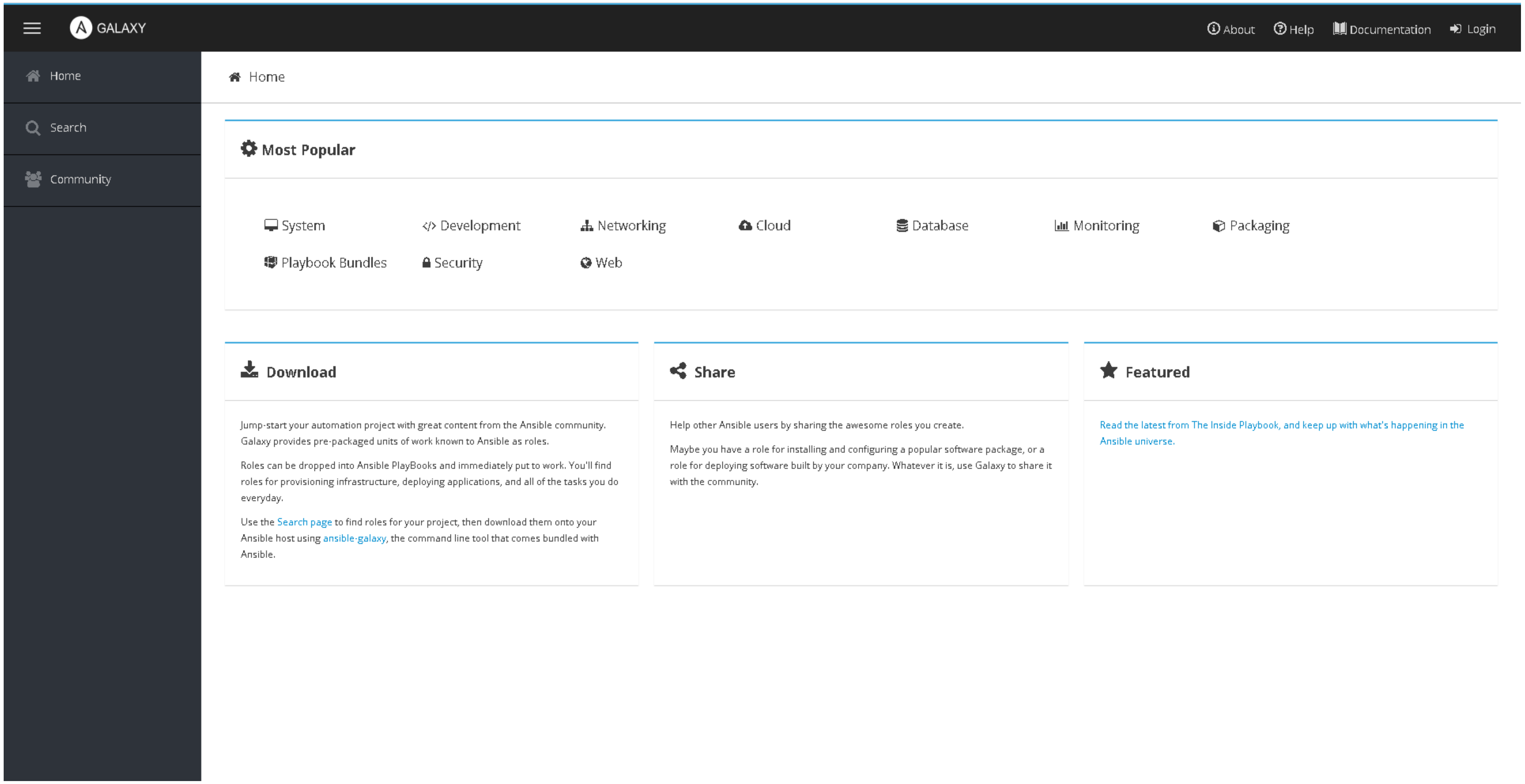
- When the playbook is executed, tasks performed because of a role can be identified by the role name prefix. The following sample output illustrates this with the motd : prefix in the task name:

```
[user@host ~]$ ansible-playbook -i inventory use-motd-role.yml
PLAY [use motd role playbook] ******************************************************
TASK [setup] ***********************************************************************
ok: [remote.example.com]
TASK [motd : deliver motd file] ****************************************************
changed: [remote.example.com]
PLAY RECAP *************************************************************************
remote.example.com : ok=2 changed=1 unreachable=0 failed=0
```

- The above scenario assumes that the **motd** role is located in the **roles** directory

Ansible Galaxy

- https://galaxy.ansible.com



Most Popular

System, Development, Networking, Cloud, Database, Monitoring, Packaging, Playbook Bundles, Security, Web

Download

Jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work known to Ansible as roles.

Roles can be dropped into Ansible PlayBooks and immediately put to work. You'll find roles for provisioning infrastructure, deploying applications, and all of the tasks you do everyday.

Use the Search page to find roles for your project, then download them onto your Ansible host using ansible-galaxy, the command line tool that comes bundled with Ansible.

Share

Help other Ansible users by sharing the awesome roles you create.

Maybe you have a role for installing and configuring a popular software package, or a role for deploying software built by your company. Whatever it is, use Galaxy to share it with the community.

Featured

Read the latest from The Inside Playbook, and keep up with what's happening in the Ansible universe.

Create The Roles Structure with Ansible Galaxy

ansible-galaxy init --help

```
Usage: ansible-galaxy init [options] role_name


Initialize new role with the base structure of a role.


Options:
  -f, --force           Force overwriting an existing role
  -h, --help            show this help message and exit
  -c, --ignore-certs    Ignore SSL certificate validation errors.
  --init-path=INIT_PATH
                        The path in which the skeleton role will be created.
                        The default is the current working directory
   And more options …
```

Automating Linux Administration Task

Example Administration Tasks

- Managing Softwares
- Managing Users

LAB 2