ANSIBLE

# AUTOMATION FOR EVERYONE

Ansible Essentials Workshop

redhat

# WHAT YOU WILL LEARN



Ansible is capable of handling many powerful automation tasks with the flexibility to adapt to many environments and workflows. With Ansible, users can very quickly get up and running to do real work. This 2 hour lab will prove that.

- What is Ansible and The Ansible Way
- Installing Ansible
- How Ansible Works and its Key Components
- Ad-Hoc Commands
- Playbook Basics
- Reuse and Redistribution of Ansible Content with Roles
- Ansible Tower

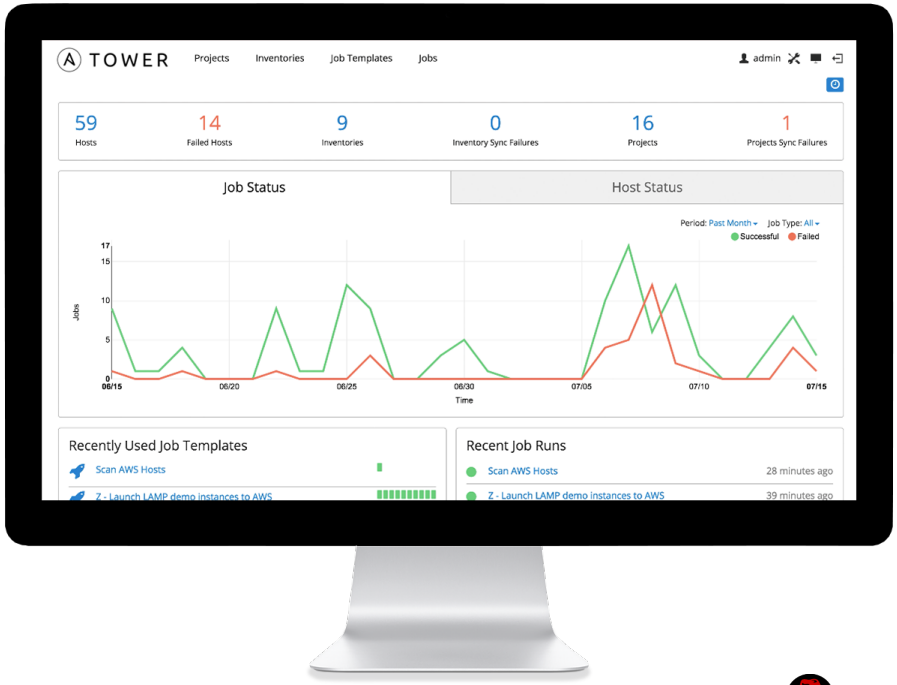
# WHAT IS ANSIBLE?



It's a **simple automation language** that can perfectly describe an IT application infrastructure in Ansible Playbooks.

It's an **automation engine** that runs Ansible Playbooks.

Ansible Tower is an **enterprise framework** for controlling, securing and managing your Ansible automation with a **UI and RESTful API.**



## 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**

# THE ANSIBLE WAY



## CROSS PLATFORM – Linux, Windows, UNIX

Agentless support for all major OS variants, physical, virtual, cloud and network

## HUMAN READABLE – YAML

Perfectly describe and document every aspect of your application environment

## PERFECT DESCRIPTION OF APPLICATION

Every change can be made by playbooks, ensuring everyone is on the same page

## VERSION CONTROLLED

Playbooks are plain-text. Treat them like code in your existing version control.

## DYNAMIC INVENTORIES

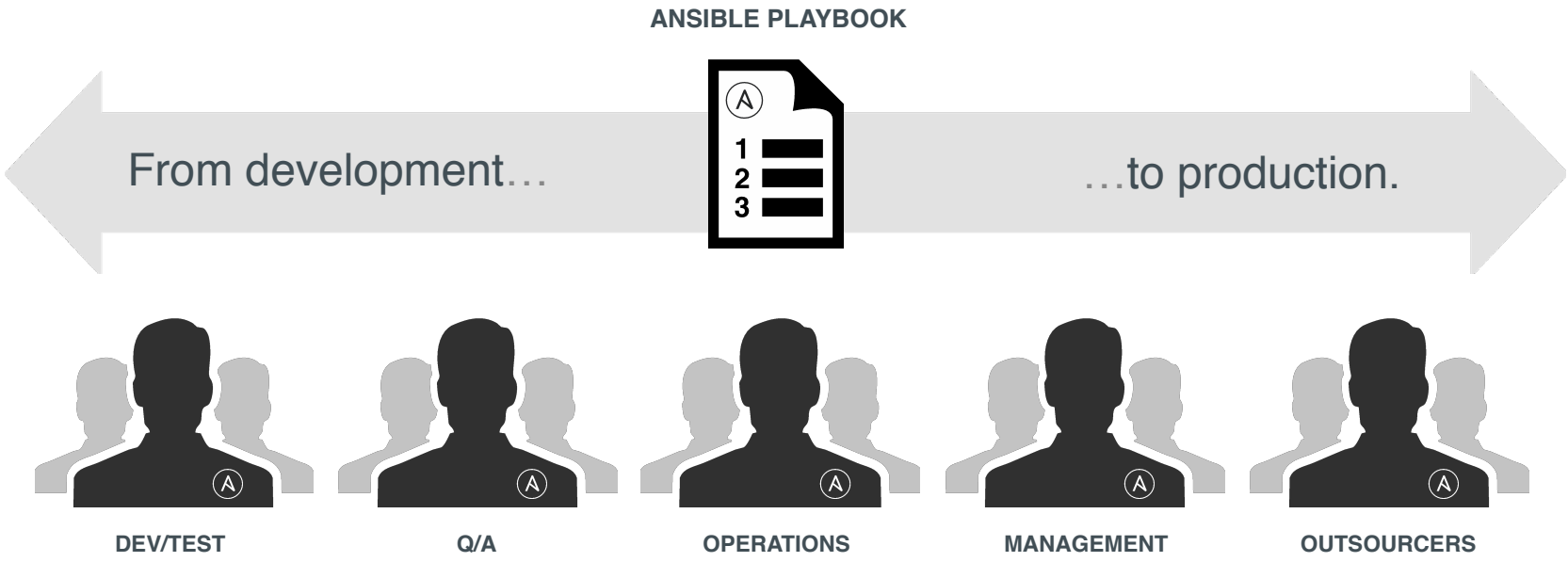
Capture all the servers 100% of the time, regardless of infrastructure, location, etc.

## ORCHESTRATION THAT PLAYS WELL WITH OTHERS – HP SA, Puppet, Jenkins, RHNSS, etc.

Homogenize existing environments by leveraging current toolsets and update mechanisms.

# ANSIBLE: THE LANGUAGE OF DEVOPS






## COMMUNICATION IS THE KEY TO DEVOPS.

Ansible is the first **automation language** that can be read and written across IT.

Ansible is the only **automation engine** that can automate the entire **application lifecycle** and **continuous delivery** pipeline.

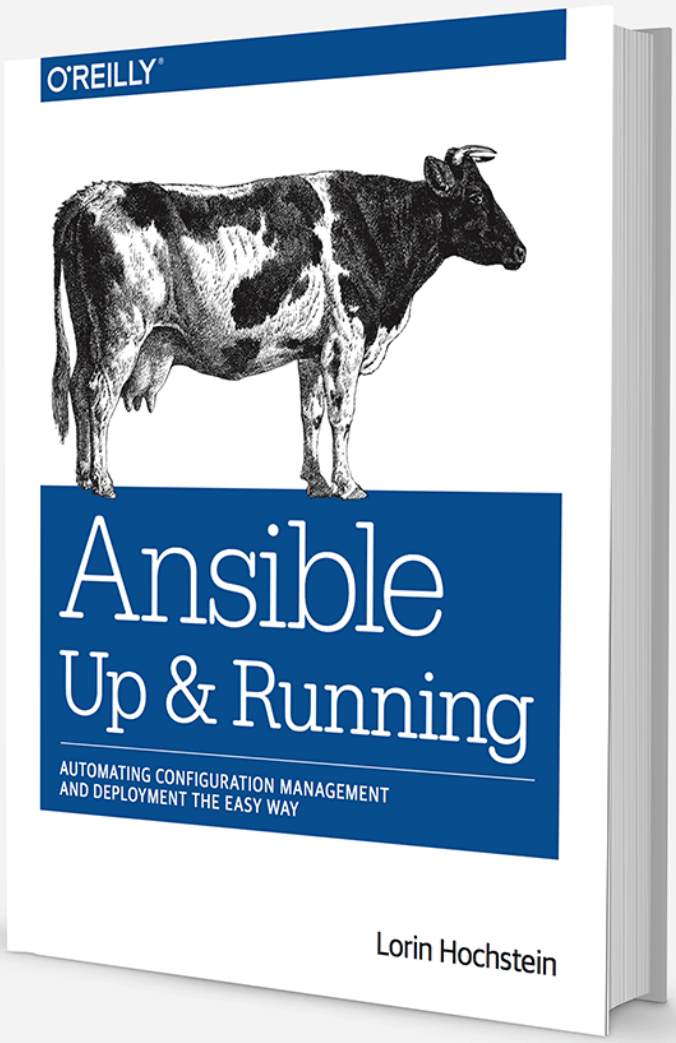
## Ansible comes bundled with hundreds of modules for a wide variety of automation tasks

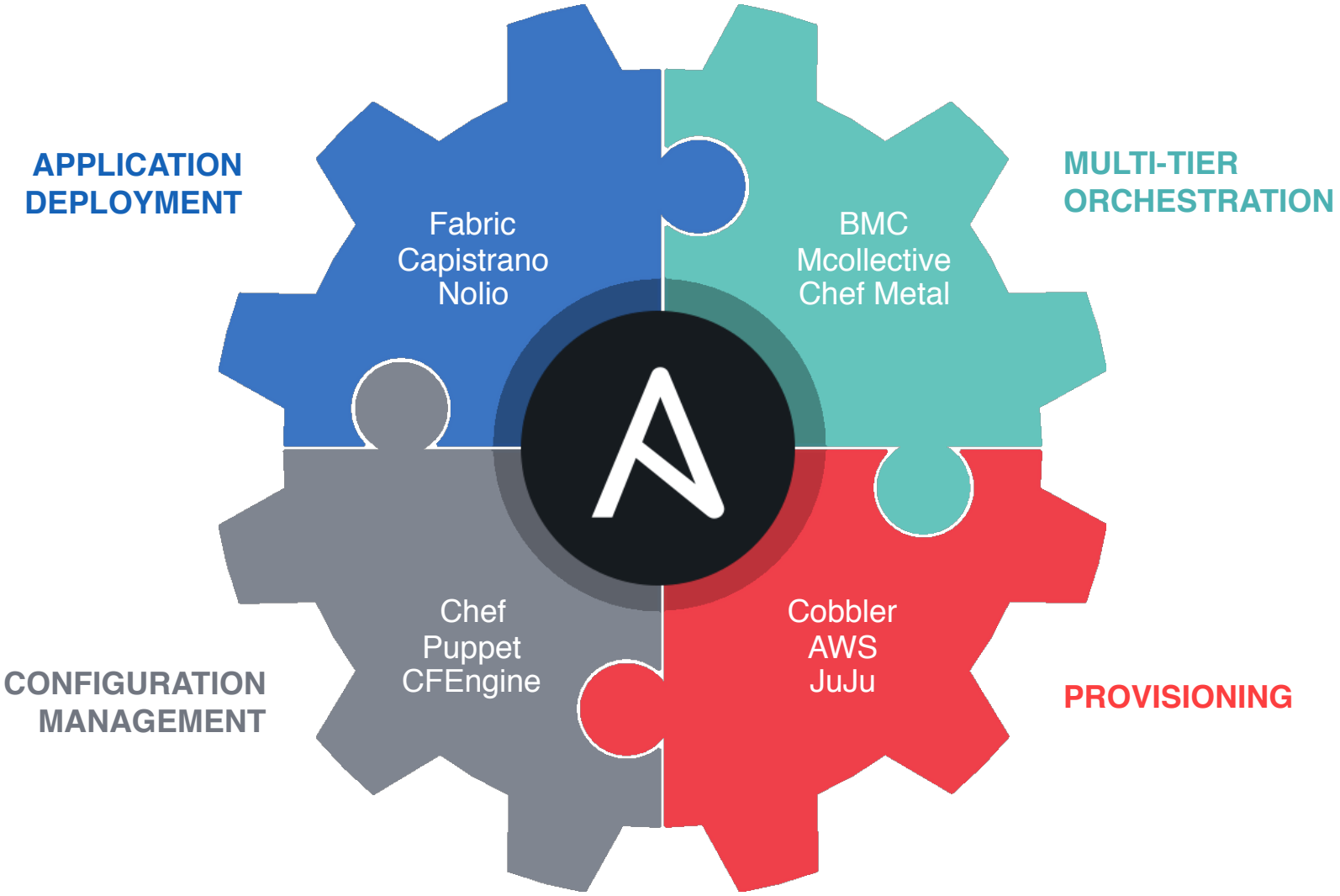
- cloud
- containers
- database
- files
- messaging
- monitoring
- network
- notifications
- packaging
- source control
- system
- testing
- utilities
- web infrastructure

## THE MOST POPULAR OPEN-SOURCE AUTOMATION COMMUNITY ON GITHUB

- 13,000+ stars & 4,000+ forks on GitHub
- 2000+ GitHub Contributors
- Over 450 modules shipped with Ansible
- New contributors added every day
- 1200+ users on IRC channel
- Top 10 open source projects in 2014
- World-wide meetups taking place every week
- Ansible Galaxy: over 18,000 subscribers
- 250,000+ downloads a month
- AnsibleFests in NYC, SF, London

**http://ansible.com/community**




APPLICATION DEPLOYMENT
Fabric
Capistrano
Nolio
MULTI-TIER ORCHESTRATION
BMC
Mcollective
Chef Metal
CONFIGURATION MANAGEMENT
Chef
Puppet
CFEngine
PROVISIONING
Cobbler
AWS
JuJu

# USE CASES



## CONFIG MANAGEMENT

Centralizing configuration file management and deployment is a common use case for Ansible, and it's how many power users are first introduced to the Ansible automation platform.

## APP DEPLOYMENT

When you define your application with Ansible, and manage the deployment with Tower, teams are able to effectively manage the entire application lifecycle from development to production.

## PROVISIONING

Your apps have to live somewhere. If you're PXE booting and kickstarting bare-metal servers or VMs, or creating virtual or cloud instances from templates, Ansible and Ansible Tower help streamline the process.

## CONTINUOUS DELIVERY

Creating a CI/CD pipeline requires buy-in from numerous teams. You can't do it without a simple automation platform that everyone in your organization can use. Ansible Playbooks keep your applications properly deployed (and managed) throughout their entire lifecycle.

## SECURITY & COMPLIANCE

When you define your security policy in Ansible, scanning and remediation of site-wide security policy can be integrated into other automated processes and instead of being an afterthought, it'll be integral in everything that is deployed.

## ORCHESTRATION

Configurations alone don't define your environment. You need to define how multiple configurations interact and ensure the disparate pieces can be managed as a whole. Out of complexity and chaos, Ansible brings order.

# INSTALLING ANSIBLE



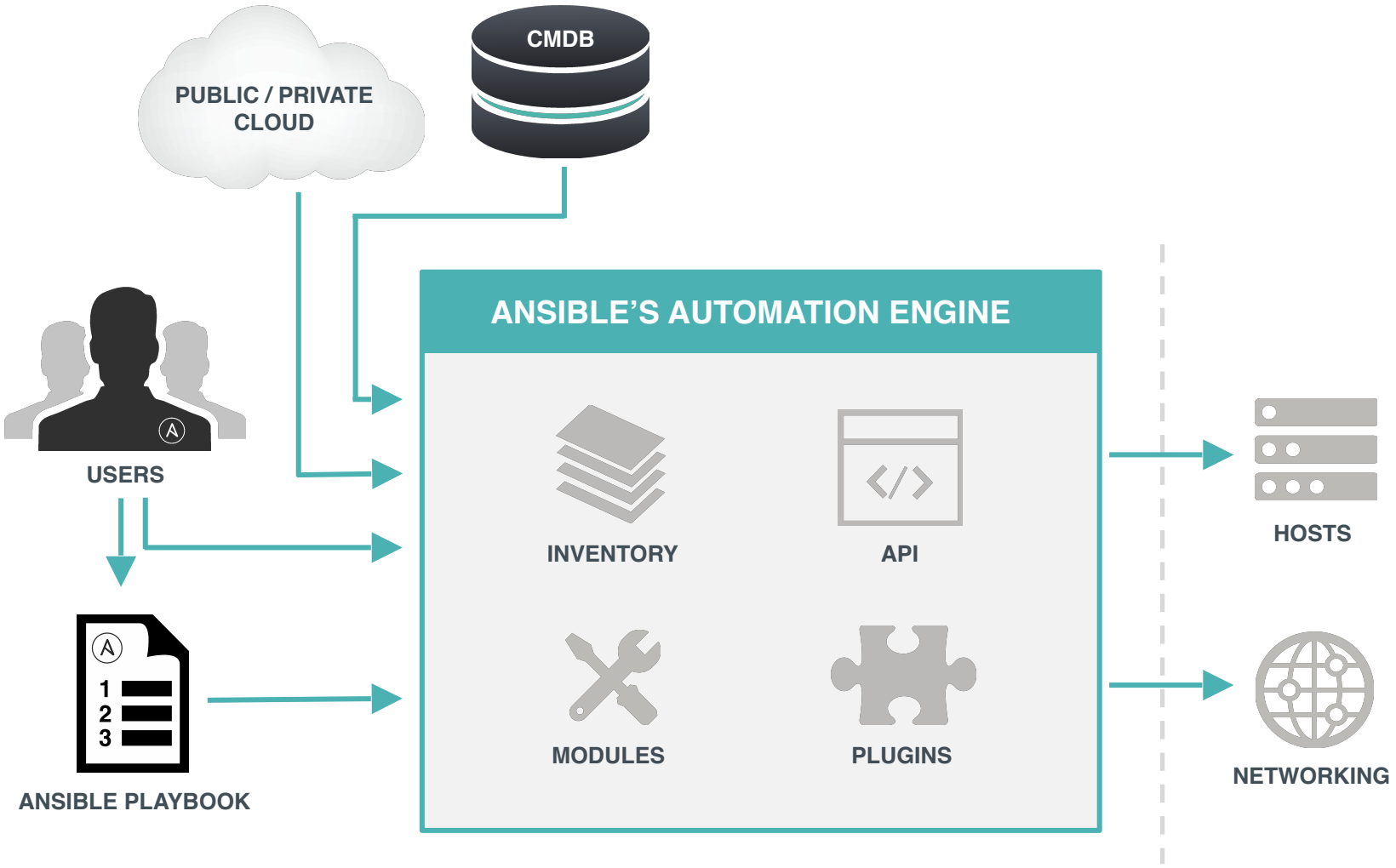
```
# the most common and preferred way of
# installation
$ pip install ansible

# install the epel-release RPM if needed on
# CentOS, RHEL, or Scientific Linux
$ sudo yum install ansible

# you will need the PPA repo configured
$ sudo apt-get install ansible
```

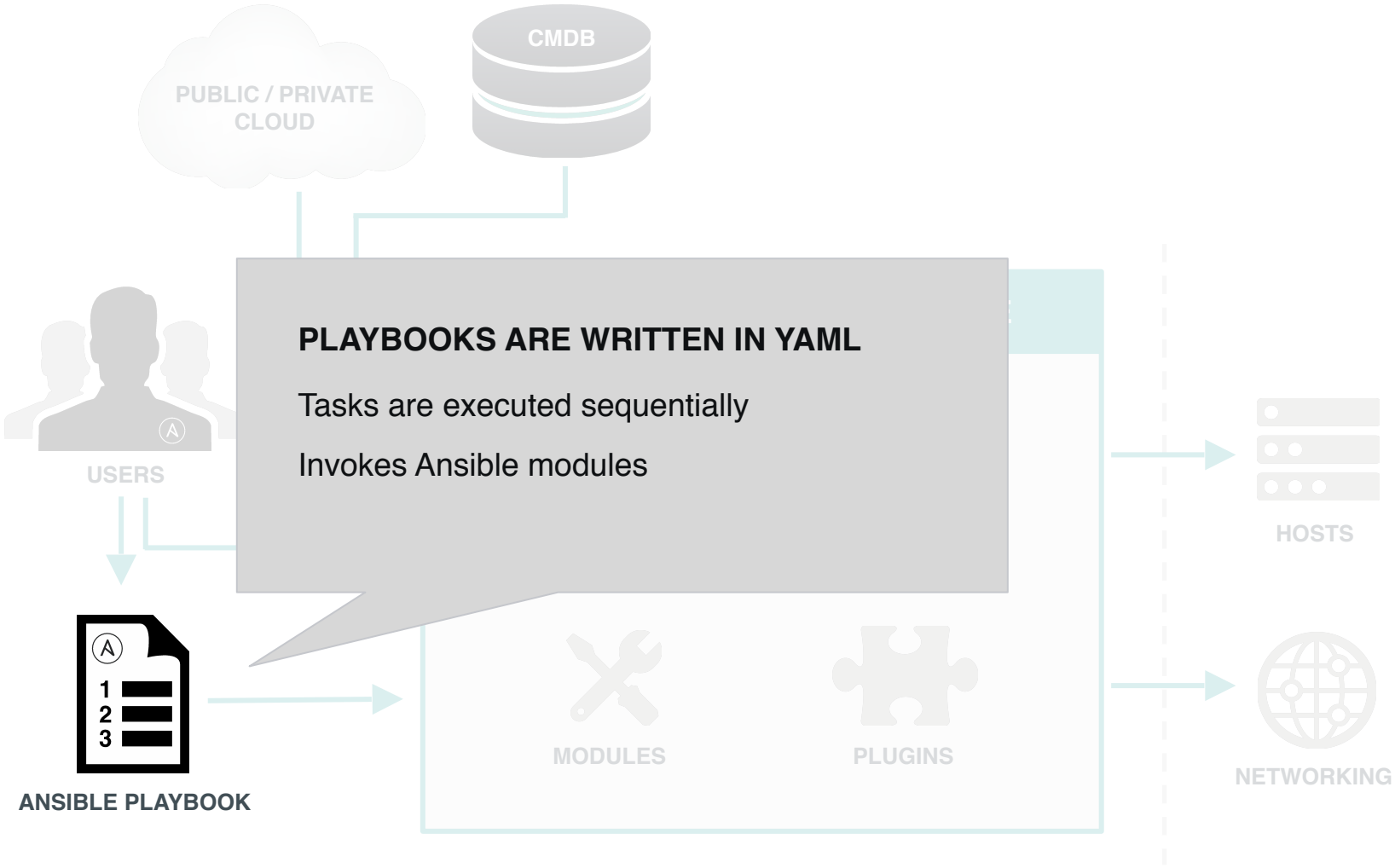
# HOW ANSIBLE WORKS





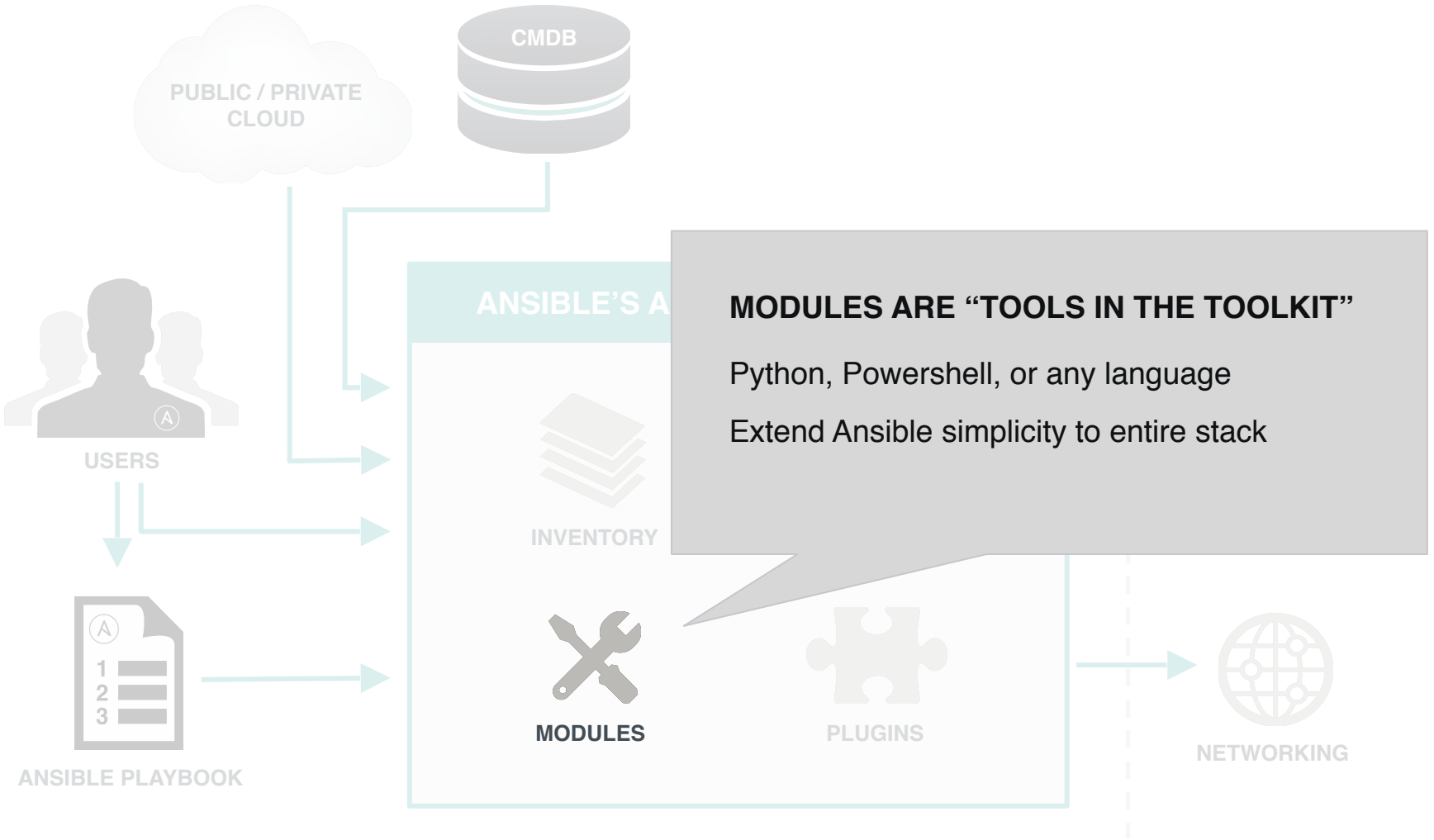
# KEY COMPONENTS: PLAYS & PLAYBOOKS





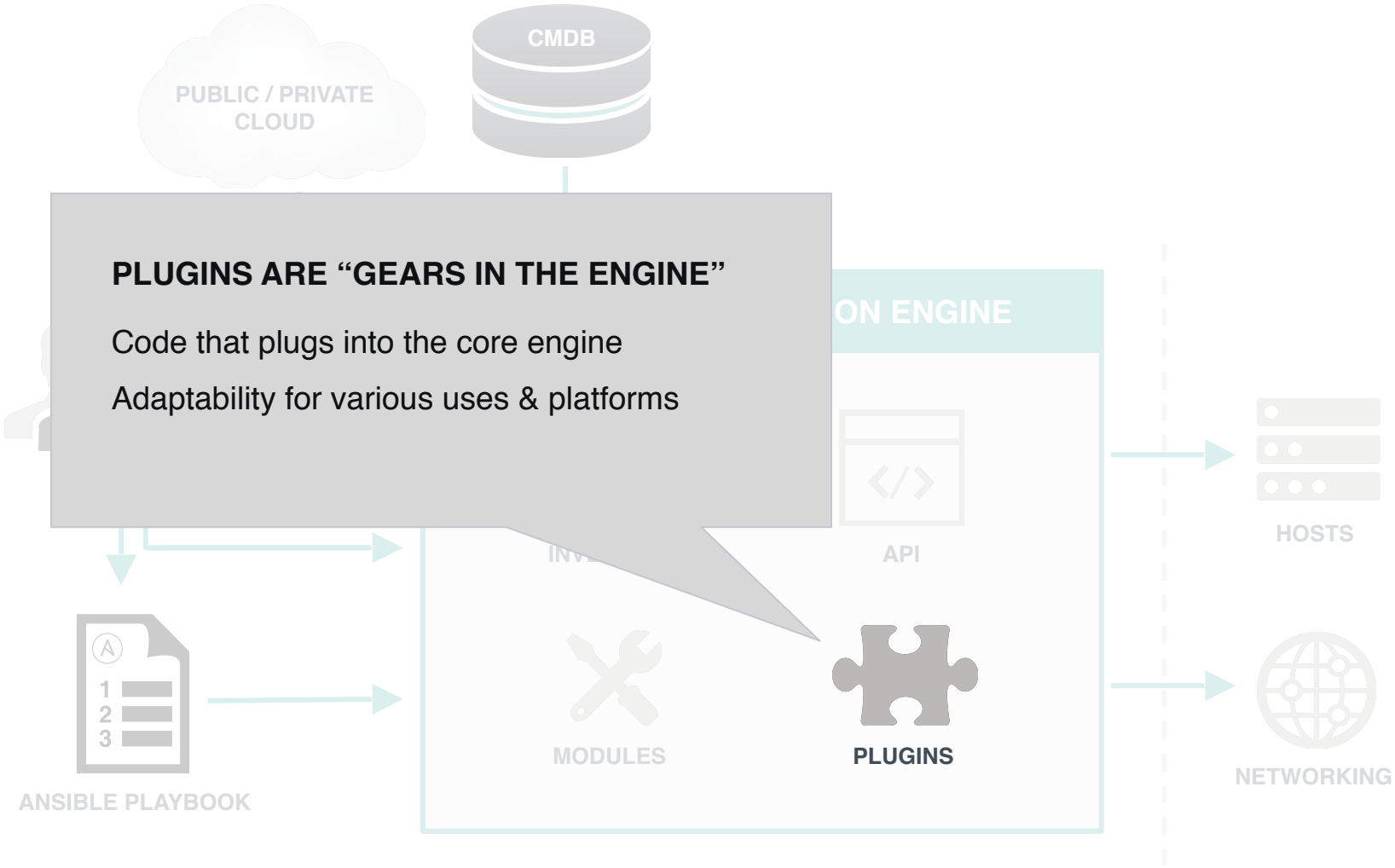
# KEY COMPONENTS: MODULES & TASKS





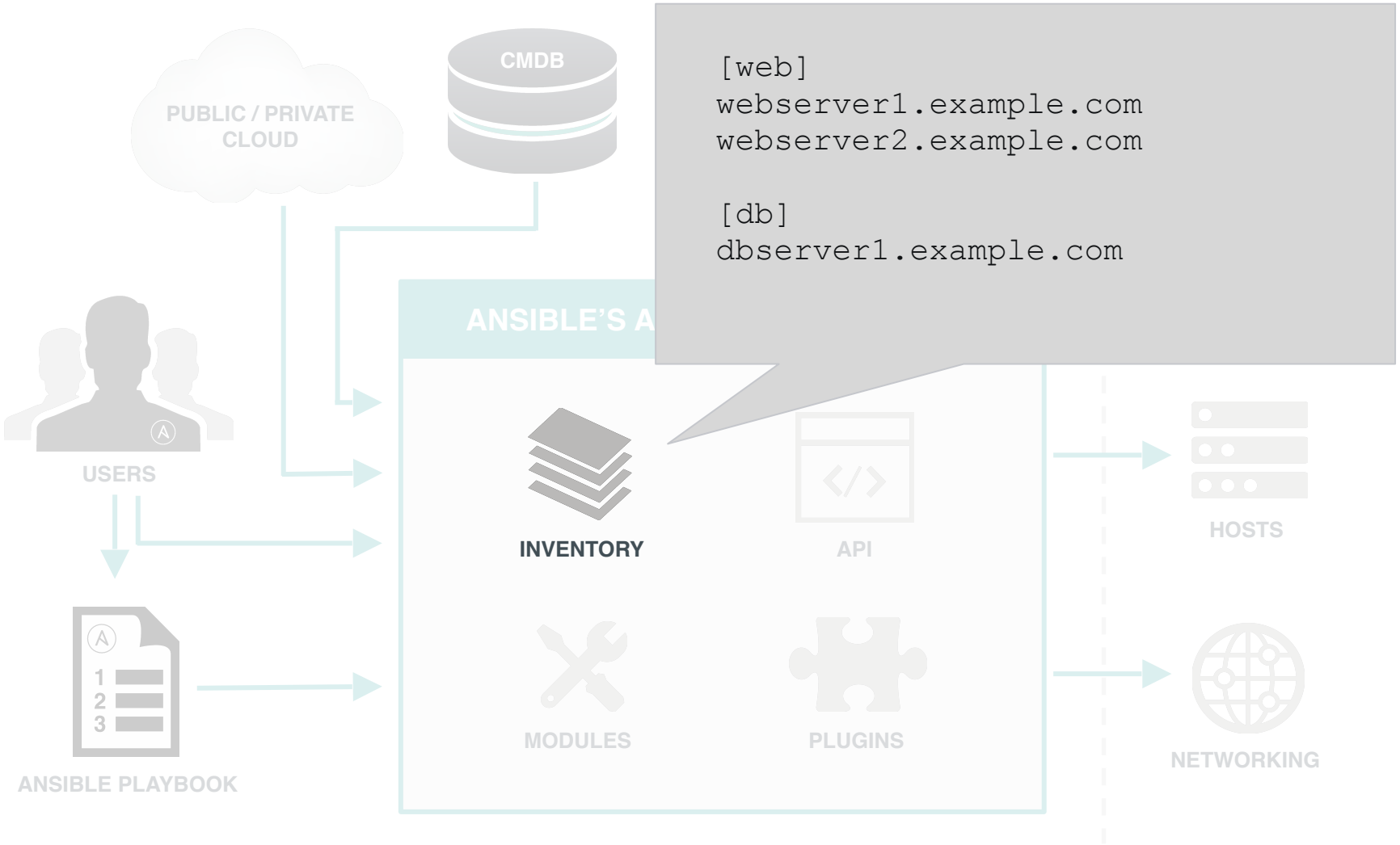
# KEY COMPONENTS: PLUGINS





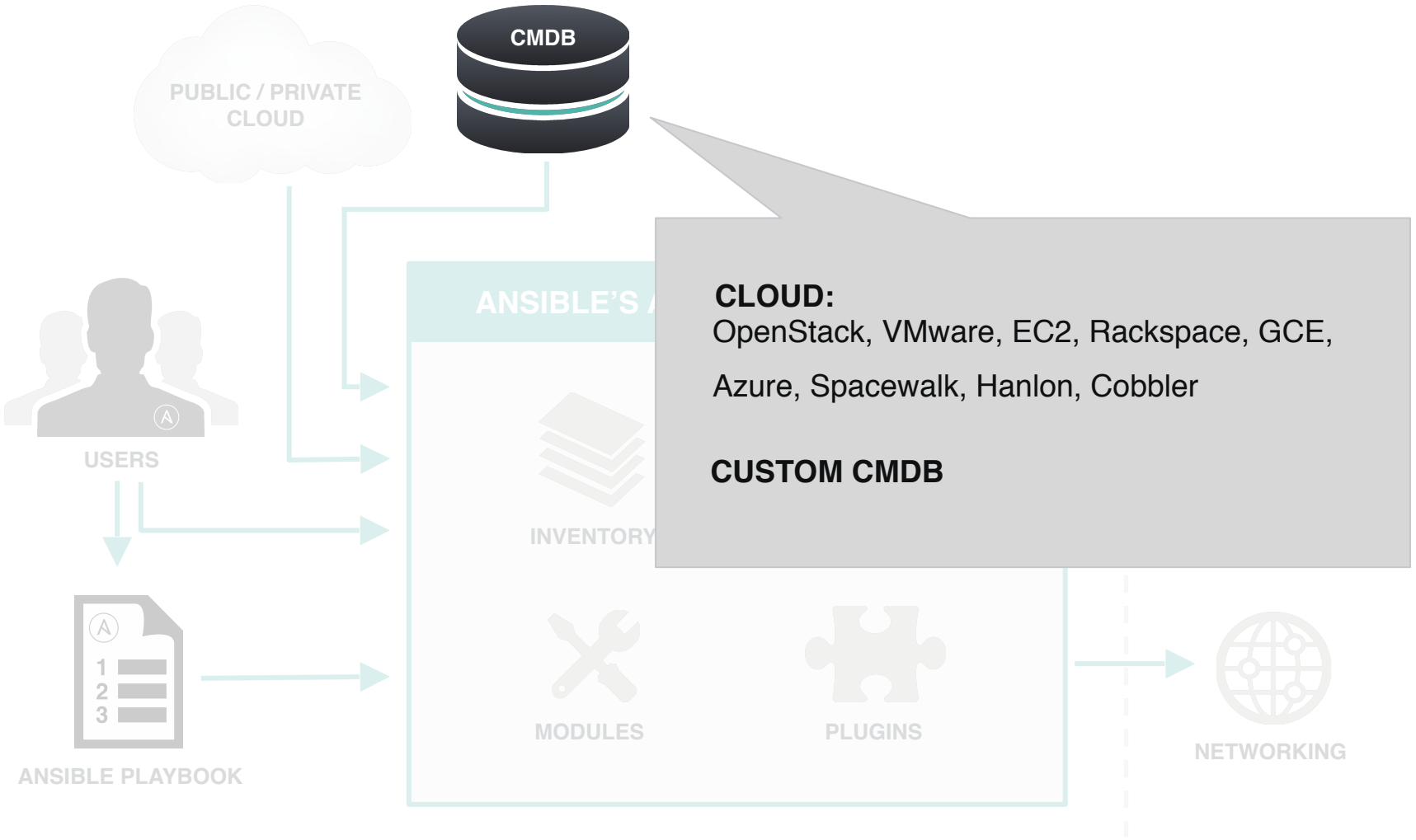
# KEY COMPONENTS: INVENTORY





# KEY COMPONENTS: INVENTORY





## Modules are bits of code transferred to the target system and executed to satisfy the task declaration.

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

Docs » Module Index

# Module Index

- All Modules
- Cloud Modules
- Clustering Modules
- Commands Modules
- Database Modules
- Files Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Source Control Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

http://docs.ansible.com

## service - Manage services.

- Synopsis
- Options
- Examples
- This is a Core Module

### Synopsis

Controls services on remote hosts. Supported init systems include BSD init, OpenRC, SysV, Solaris SMF, systemd, upstart.

### Options

| parameter | required | default | choices | comments |
|---|---|---|---|---|
| arguments | no | | | Additional arguments provided on the command line<br>aliases: args |
| enabled | no | | • yes<br>• no | Whether the service should start on boot. **At least one of state and enabled are required.** |
| name | yes | | | Name of the service. |
| pattern | no | | | If the service does not respond to the status command, name a substring to look for as would be found in the output of the *ps* command as a stand-in for a status result. If the string is found, the service will be assumed to be running. |
| runlevel | no | default | | For OpenRC init scripts (ex: Gentoo) only. The runlevel that this service belongs to. |
| sleep (added in 1.3) | no | | | If the service is being `restarted` then sleep this many seconds between the stop and start command. This helps to workaround badly behaving init scripts that exit immediately after signaling a process to stop. |
| state | no | | • started<br>• stopped<br>• restarted<br>• reloaded | `started` / `stopped` are idempotent actions that will not run commands unless necessary. `restarted` will always bounce the service. `reloaded` will always reload. **At least one of state and enabled are required.** |

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

- **command:** Takes the command and executes it. The most secure and predictable.
- **shell:** Executes through a shell like `/bin/sh` so you can use pipes etc. Be careful.
- **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

# AD-HOC COMMANDS



```
# check all my inventory hosts are ready to be
# managed 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_default_ipv4": {
            "address": "192.168.1.37",
            "alias": "wlan0",
            "gateway": "192.168.1.1",
            "interface": "wlan0",
            "macaddress": "c4:85:08:3b:a9:16",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.168.1.0",
            "type": "ether"
      },
```

## Inventory is a collection of hosts (nodes) against which Ansible can work with.

- Hosts
- Groups
- Inventory-specific data (variables)
- Static or dynamic sources

# STATIC INVENTORY EXAMPLE



```
10.42.0.2
10.42.0.6
10.42.0.7
10.42.0.8
10.42.0.100
```

# STATIC INVENTORY EXAMPLE



```
[control]
control ansible_host=10.42.0.2

[web]
node-1 ansible_host=10.42.0.6
node-2 ansible_host=10.42.0.7
node-3 ansible_host=10.42.0.8

[haproxy]
haproxy ansible_host=10.42.0.100

[all:vars]
ansible_user=vagrant
ansible_ssh_private_key_file=~/.vagrant.d/insecure_private_key
```

## Ansible can work with metadata from various source and manage their context in the form of variables.

- command line parameters
- plays and tasks
- files
- inventory
- discovered facts
- roles

The order in which the same variable from different sources will override each other.

1. extra vars
2. task vars (only for the task)
3. block vars (only for tasks in 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

The order in which the same variable from different sources will override each other.

1. extra vars
2. task vars (only for the task)
3. block vars (only for tasks in 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

# DEMO TIME

## AD-HOC COMMANDS AND INVENTORY

## Tasks are the application of a module to perform a specific unit of work.

- **`file`**: A directory should exist
- **`yum`**: A package should be installed
- **`service`**: A service should be running
- **`template`**: Render a configuration file from a template
- **`get_url`**: Fetch an archive file from a URL
- **`git`**: Clone a source code repository

# EXAMPLE TASKS 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
```

## 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
```

Plays are ordered sets of tasks to execute against host selections from your inventory. A playbook is a file containing one or more plays.

If a configuration file gets changed notify a service restart task it needs to run.

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

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

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

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

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

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

# DEMO TIME

## A BASIC PLAYBOOK RUN

## Roles are a packages of closely related Ansible content that can be shared more easily than plays alone.

- Improves readability and maintainability of complex plays
- Eases sharing, reuse and standardization of automation processes
- Enables Ansible content to exist independently of playbooks, projects -- even organizations
- Provides functional conveniences such as file path resolution and default values

# PROJECT WITH EMBEDDED ROLES EXAMPLE



```
site.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

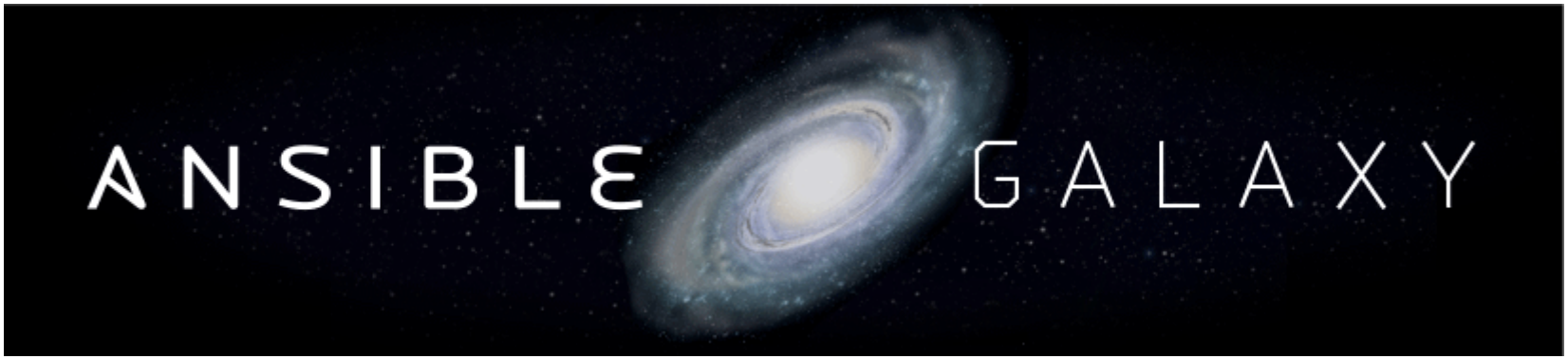
# PLAYBOOK WITH ROLES EXAMPLE



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

# ANSIBLE GALAXY






**http://galaxy.ansible.com**

Ansible Galaxy is a hub for finding,
reusing and sharing Ansible content.

Jump-start your automation project with
content contributed and review
by the Ansible community.

# DEMO TIME

## A PLAYBOOK USING ROLES

ANSIBLE

# GETTING STARTED

Would you like to learn Ansible?  It's easy to get started:
**ansible.com/get-started**

Want to learn more about Ansible? Check out our documentation:
**docs.ansible.com**

Want to learn more?
**ansible.com/whitepapers**

redhat