ANSIBLE BASICS

OHIO LINUXFEST INSTITUTE

SCOTT SEIGHMAN
SOLUTIONS ARCHITECT
SSEIGHMA@REDHAT.COM

OCTOBER 7, 2016

Cleveland Indians

Lead MLB Division Series against Boston Red Sox 1-0

Game 1, **Final** - Yesterday, 8:08 PM
Progressive Field, Cleveland, Ohio

Boston Red Sox	4	
Cleveland Indians	5	

Box Score

All times are in Eastern Time

Schedule and scores

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

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.

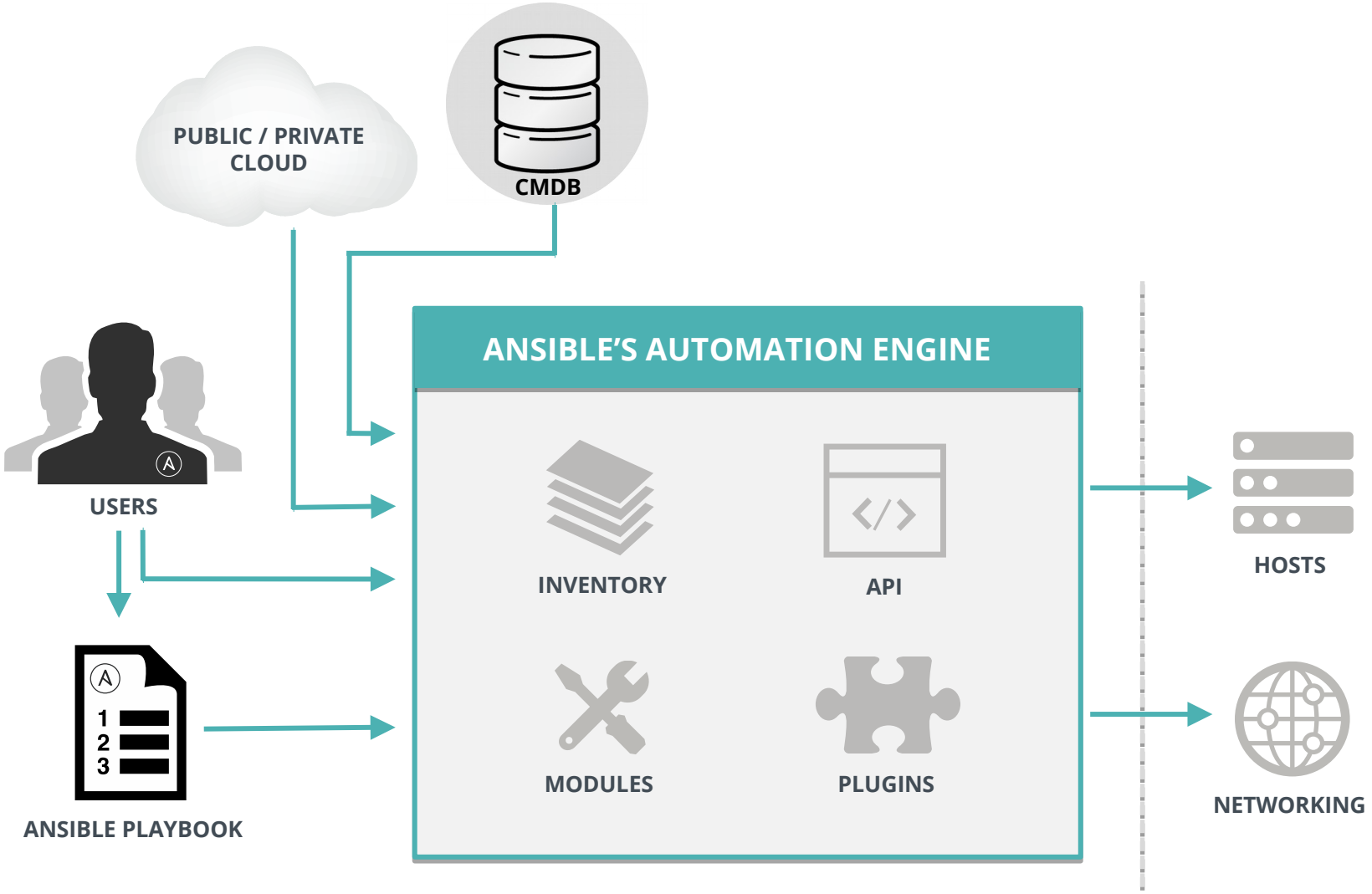
DYNAMIC INVENTORIES

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

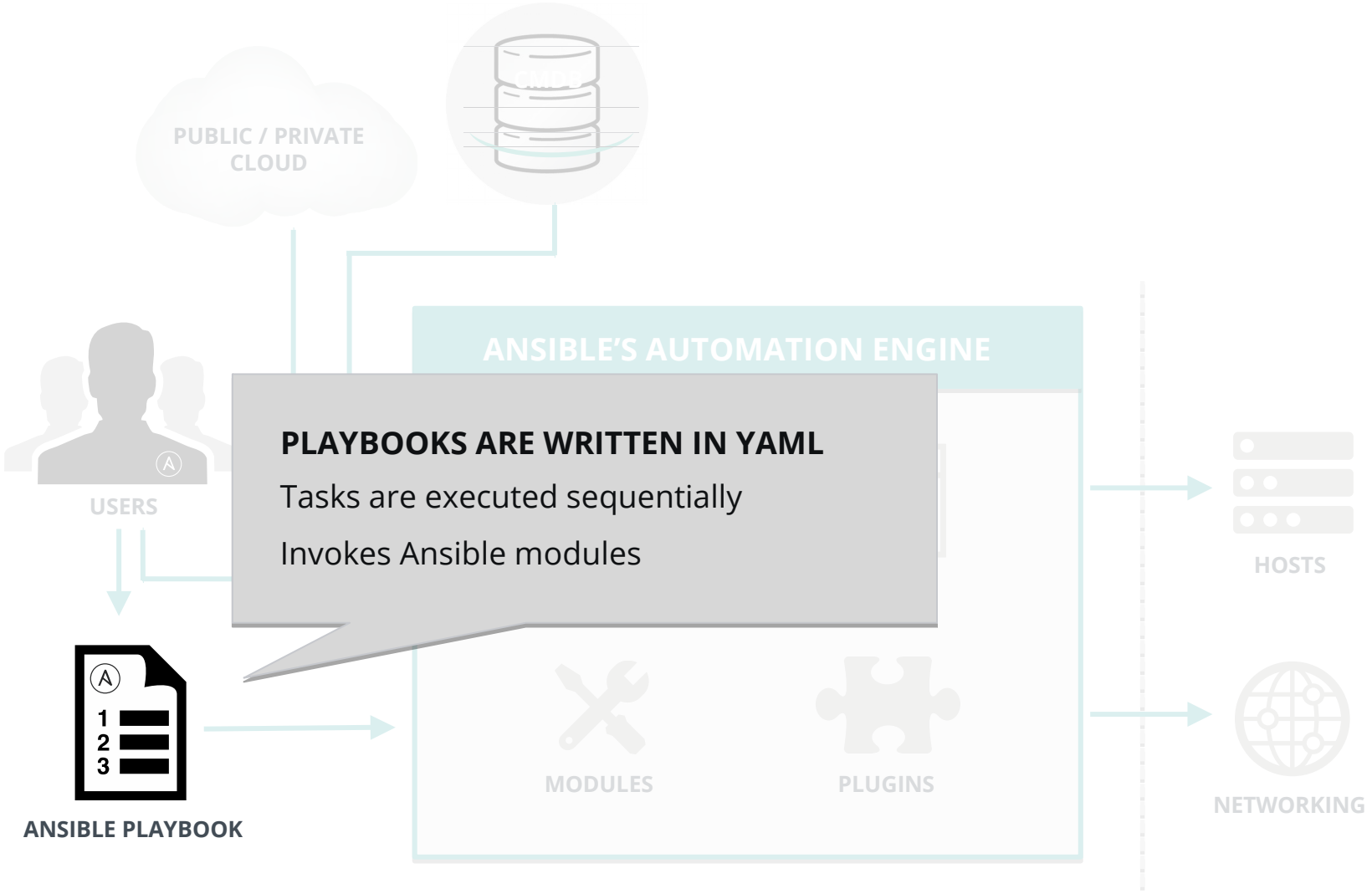
ORCHESTRATION THAT PLAYS WELL WITH OTHERS (HP SA, Puppet, Jenkins)

Homogenize existing environments by leveraging current toolsets and update mechanisms.

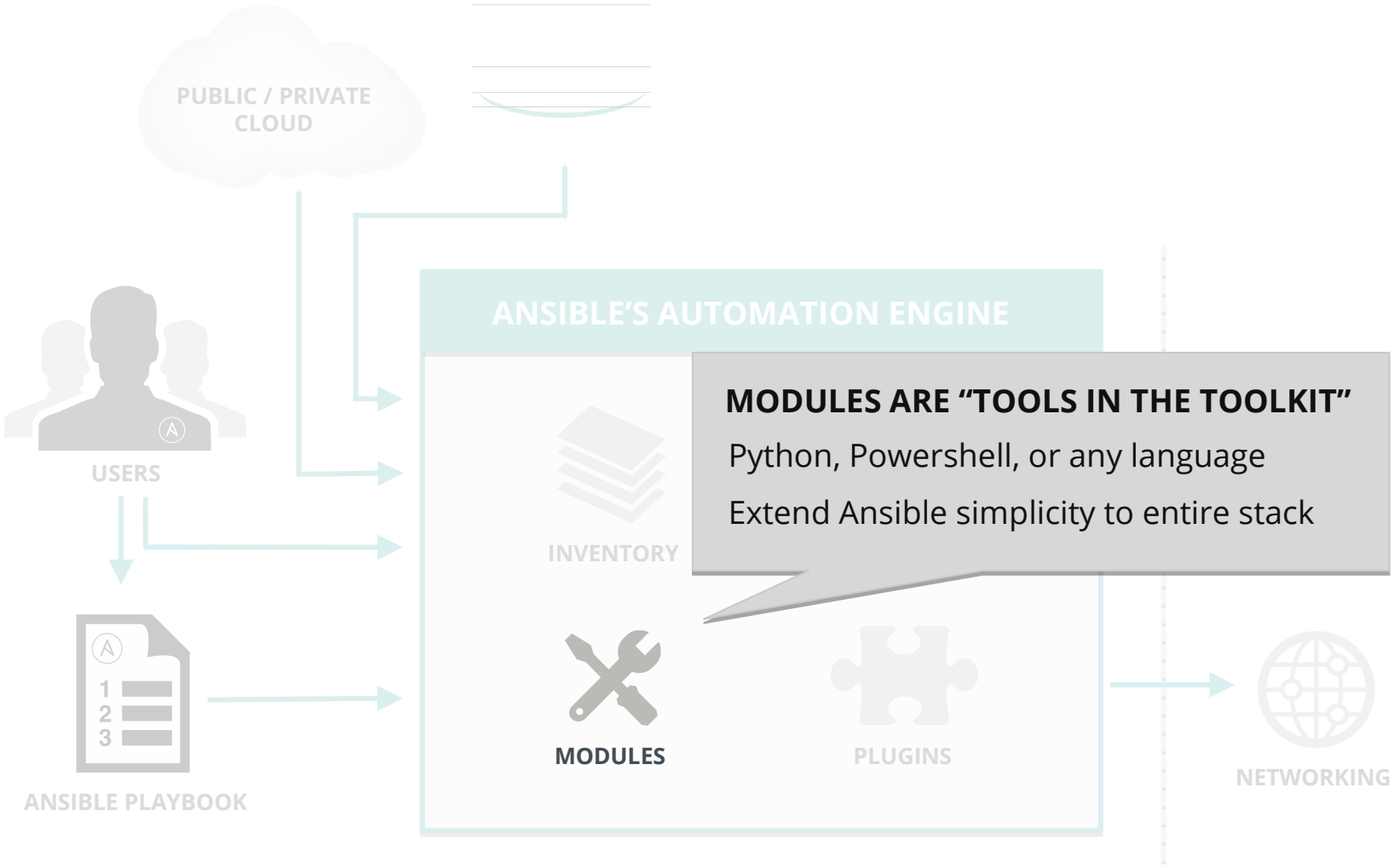
HOW ANSIBLE WORKS



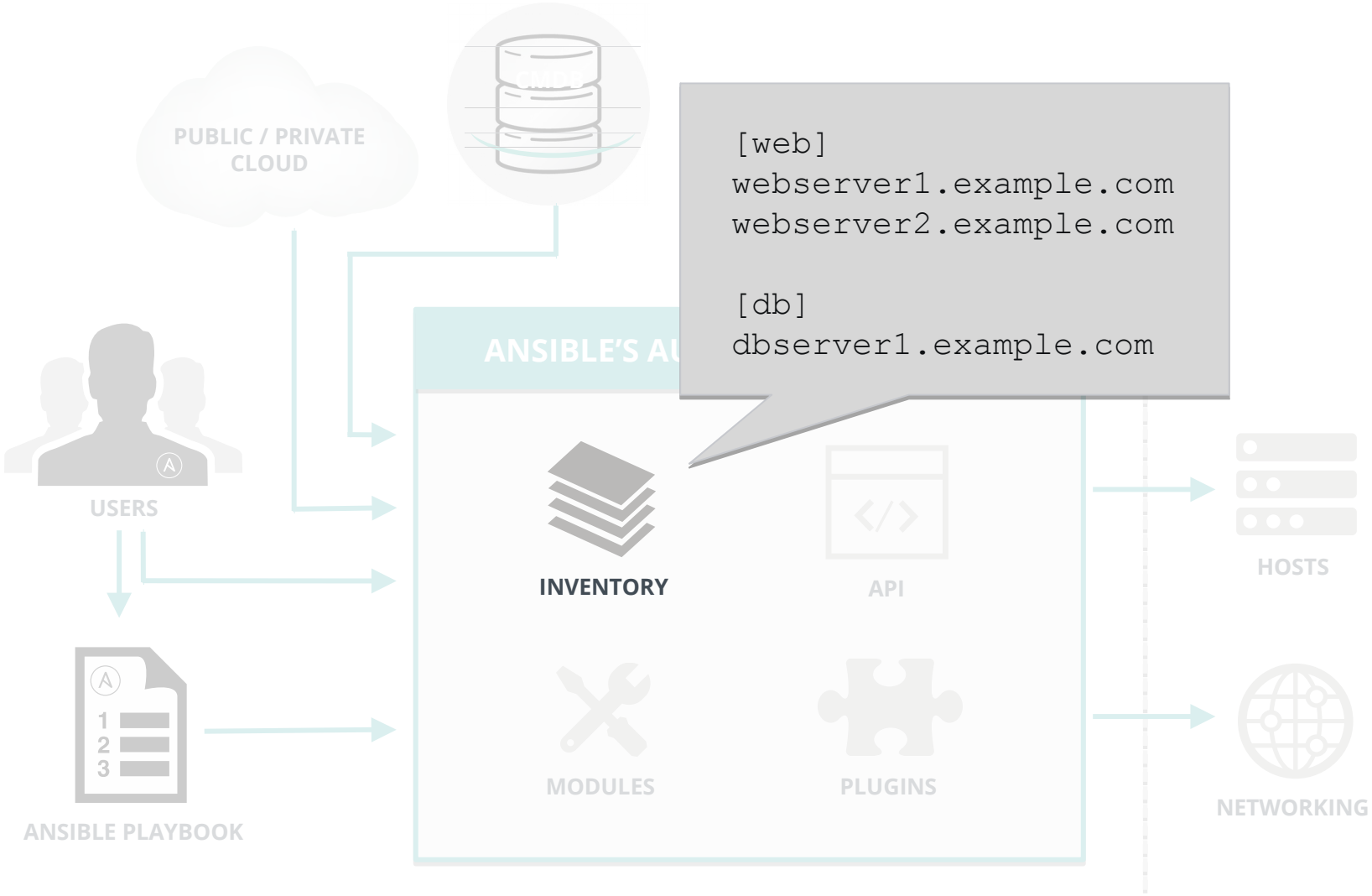
HOW ANSIBLE WORKS



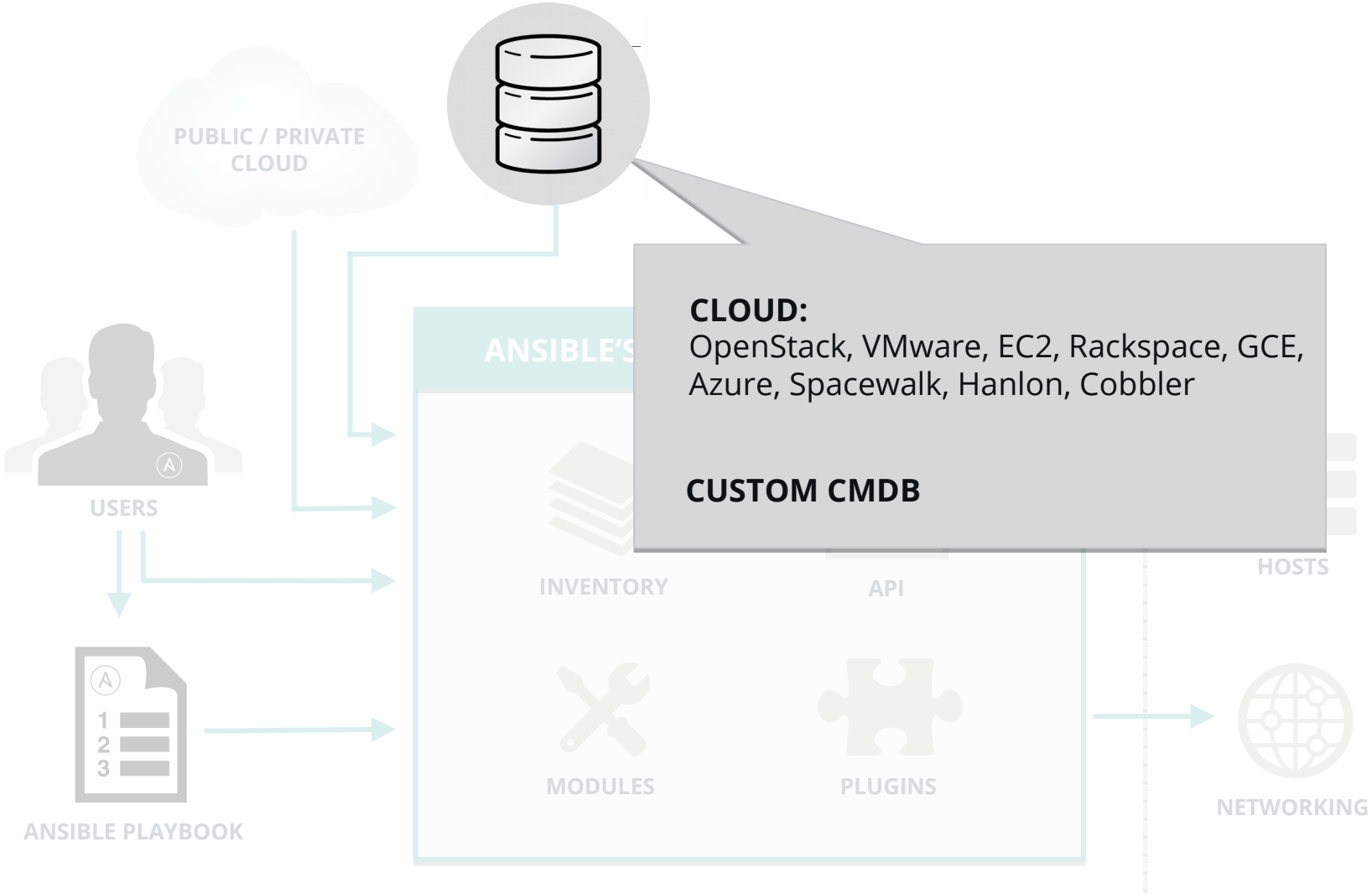
HOW ANSIBLE WORKS



HOW ANSIBLE WORKS



HOW ANSIBLE WORKS



INSTALLING ANSIBLE

RECOMMENDATIONS/REQUIREMENTS

- For the latest released version of Ansible (if you are running Red Hat Enterprise Linux, CentOS, Fedora, Debian, or Ubuntu) we recommend using the OS package manager

- For other installation options (like MacOS), we recommend installing via "**pip**", which is the Python package manager, though other options are also available

- Currently Ansible can be run from any machine with **Python 2.6 or 2.7 installed**, this includes Red Hat, Debian, CentOS, OS X, any of the BSDs, and so on …

- **Windows isn't supported for the control machine**

INSTALLING ANSIBLE

USING YUM

Download the latest EPEL-Release RPM from:

http://fedoraproject.org/wiki/EPEL

Install the RPM and update:

```
$ sudo rpm -ihv epel-release-latest-7.noarch.rpm

$ sudo yum update
```

Install Ansible:

```
$ sudo yum install -y ansible

$ ansible --version
  ansible 2.1.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

USING APT

Add the PPA to your system and update:

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

Install Ansible:

```
$ sudo apt-get install ansible
```

```
$ ansible --version
  ansible 2.1.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

INSTALLING ANSIBLE

USING PIP

If '**pip**' isn't installed as part of your Python environment, add it:

```
$ sudo easy_install pip
```

Install Ansible:

```
$ sudo pip install ansible
```

```
$ ansible --version
  ansible 2.1.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

CONFIGURING SSH

Generate your SSH keys (n*ot necessary if you have an existing key*):

```
# ssh-keygen -t rsa -b 2048 -C "your_email@example.com"
```

Add your key to 'ssh-agent':

```
# ssh-agent bash
# ssh-add ~/.ssh/id_rsa
```

Copy your key to the remote server:

```
# ssh-copy-id username@test.example.com
```

ANSIBLE CONFIGURATION FILES

ANSIBLE CONFIGURATION FILES

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

This allows administrators to create a directory structure where different environments or projects are housed in separate directories with each directory containing a configuration file tailored with a unique set of settings

If there is no **ansible.cfg** in the current working directory, Ansible will look for a **~/.ansible.cfg** in the user's home directory

USING /etc/ansible/ansible.cfg

When installed, the ansible package provides a base configuration file located at **`/etc/ansible/ansible.cfg`**

This file will be used if no other configuration file is found

USING $ANSIBLE_CONFIG

Users can make use of 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

A more flexible option is to define the location of the configuration file with the **$ANSIBLE_CONFIG** environment variable

When this variable is defined, Ansible uses the configuration file the variable specifies instead of any of the previously mentioned configuration files

ANSIBLE CONFIGURATION FILES

CONFIGURATION FILE PRECEDENCE

1) **ansible.cfg** *(in current working directory)*
2) **~/.ansible.cfg** *(in user's home directory)*
3) **/etc/ansible/ansible.cfg**
4) **$ANSIBLE_CONFIG**

The first file located in the above search order is the one from which Ansible will use configuration settings

Ansible will only use settings from this configuration file

Even if other files with lower precedence exist, their settings will be ignored and not combined with those in the selected configuration file

ANSIBLE CONFIGURATION FILE

CONFIGURATION FILE SECTIONS

```
$ grep "^\[" /etc/ansible/ansible.cfg
[defaults]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[accelerate]
[selinux]
[colors]
```

CONFIGURATION FILE SECTIONS

Most of the settings in the configuration file are grouped under the **[defaults]** section

The **[privilege_escalation]** section contains settings for defining how operations which require escalated privileges will be executed on managed hosts

The **[paramiko_connection]**, **[ssh_connection]**, and **[accelerate]** sections contain settings for optimizing connections to managed hosts

The **[selinux]** section contains settings for defining how SELinux interactions will be configured.

ANSIBLE CONFIGURATION FILE

The following table highlights some commonly modified Ansible configuration settings:

SETTING	DESCRIPTION
inventory	Location of the Ansible inventory file
remote user	The remote user account used to establish connections to managed hosts
become	Enables or disables privilege escalation for operations on managed hosts
become_method	Defines the privilege escalation method on managed hosts
become_user	The user account to escalate privileges to on managed hosts
become_ask_pass	Defines whether privilege escalation on managed hosts should prompt for a password

ANSIBLE FUNDAMENTALS

KEY COMPONENTS OF ANSIBLE

Modules/Tasks

Inventory

Plays

Playbooks

ANSIBLE MODULES

KEY COMPONENTS OF ANSIBLE: MODULES

Modules/Tasks

Code copied to the target system

Executed to satisfy the task declaration

Customizable

KEY COMPONENTS OF ANSIBLE: MODULES

There are three types of Ansible modules:

Core modules are included with Ansible and are written and maintained by the Ansible development team. Core modules are the most important modules and are used for common administration tasks

Extras modules are currently included with Ansible but may be promoted to core separately in the future. They are generally not maintained by the Ansible team but by the community. Typically, these modules implement features for newer technologies

Custom modules are developed by end users and not shipped with Ansible. Administrators can write a new modules to implement features not available in existing modules

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

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 aliases: args
enabled	no		• yes • 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 • stopped • restarted • 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.**

http://docs.ansible.com/ansible/modules_by_category.html

MODULES HELP

You can learn more about modules by using the documentation:

```
$ ansible-doc -l | grep yum
yum             Manages packages with the `yum' package manager
yum_repository      Add and remove YUM repositories
```

```
$ ansible-doc yum

> YUM
Installs, upgrade, removes, and lists packages and groups with
the `yum' package manager.
…
```

$ ansible-doc -s yum *(snippets)*

```
name: Manages packages with the `yum' package manager
action: yum
conf_file
# The remote yum configuration ...
```

EXECUTING COMMANDS

MODULES: RUN COMMANDS

command

Takes the command and executes it
Most secure and predictable

shell

Executes command line via a shell (like /bin/sh), you can use pipes etc.
Be careful!

script

Runs a local script on a remote node after transferring it

raw

Executes a straight-up SSH command without going through the Ansible module subsystem

MODULES: RUN COMMANDS

BEST PRACTICES

Always use '**command**' first

shell opens an interactive terminal dictated by target's SHELL

raw uses no Python

RUNNING AD-HOC ANSIBLE COMMANDS

An **ad-hoc** command is something that you might type in to do something really quick, but don't want to save for later

ad-hoc commands can also be used perform quick tasks not necessarily fit for a full playbook

Why would you use ad-hoc tasks vs playbooks?

> For instance, if you wanted to power off all of your lab systems for vacation, you could execute a quick one-liner in Ansible without writing a entire playbook

Generally speaking, the true power of Ansible lies in playbooks

RUNNING AD-HOC ANSIBLE COMMANDS

The syntax for ad-hocs commands is:

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

The **`host-pattern`** is used to define the list of managed hosts for Ansible to perform the ad hoc command on

RUNNING AD-HOC ANSIBLE COMMANDS

Administrators have the option of defining a default module using the **module_name** setting under the **[defaults]** section of the **/etc/ansible/ansible.cfg** file

```
# default module name for /usr/bin/ansible
#module_name = command
```

When the **-m** option is omitted in the execution of an ad hoc command, Ansible will consult the Ansible configuration file and use the module defined there

If no module is defined, Ansible uses the internally predefined command module

RUNNING AD-HOC ANSIBLE COMMANDS

Configuration settings to enable privilege escalation are located under the **[privilege_escalation]** section of the **ansible.cfg** configuration file:

```
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

Privilege escalation is not enabled by default. To enable privilege escalation, the become parameter must be uncommented and defined as True

```
become=True
```

RUNNING AD-HOC ANSIBLE COMMANDS

When privilege escalation is enabled, the **become_method**, **become_user**, and **become_ask_pass** parameters come into play

This is true even if these parameters are commented out in /etc/ansible/ansible.cfg since they are predefined internally within Ansible

Their predefined values are as follows:

become_method=sudo
become_user=root
become_ask_pass=False

RUNNING AD-HOC ANSIBLE COMMANDS

The following table shows the analogous Ansible command line options for each configuration file setting:

SETTING	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

INVENTORIES

ANSIBLE INVENTORIES

A **host** can be a physical or virtual server, container, application or network switch

Defines:

- Port and address
- Remote/sudo usernames
- Connection type

```
web1.example.com  ansible_port=5555  ansible_host=192.168.124.221
```

Inventory name

SSH Port

Connection Address

ANSIBLE INVENTORIES

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, which defaults to being saved in the location **/etc/ansible/hosts**

The format for **/etc/ansible/hosts** is an INI-like format:

```
mail.example.com

[web]
foo.example.com
bar.example.com

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

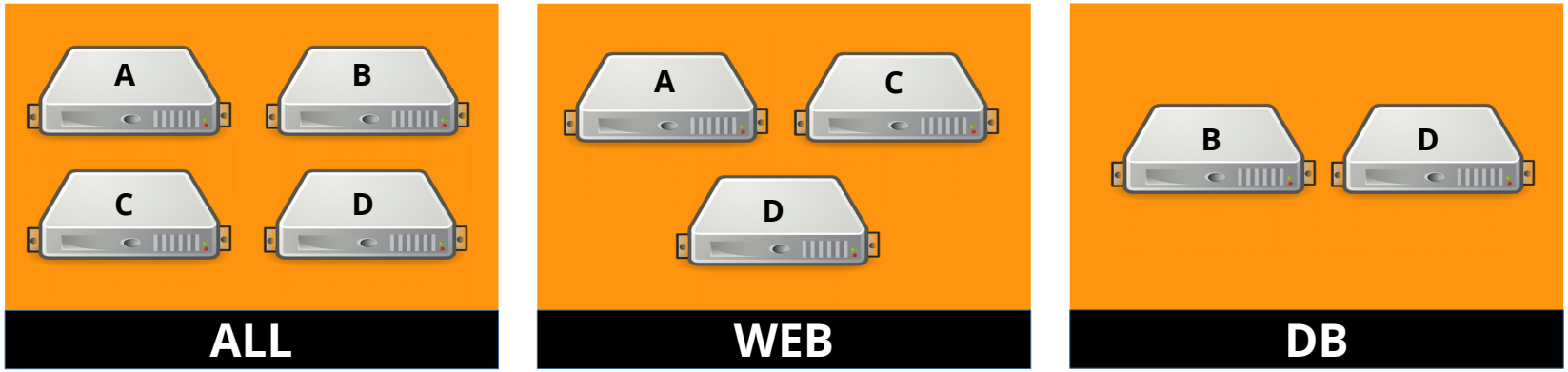
Group

Group

ANSIBLE INVENTORIES

Groups can include hosts from other groups

For instance a server could be both a webserver and a dbserver:



ANSIBLE INVENTORIES

If you're adding many hosts following similar patterns, you can can use ranges rather than listing each hostname:

[webservers]
www[01:50].example.com

For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive

You can also define alphabetic ranges:

[databases]
db-[a:f].example.com

ANSIBLE HOST VARIABLES

It is easy to assign variables to hosts that will be used later in playbooks:

```
[web]
host1    http_port=80     maxRequestsPerChild=808
host2    http_port=8080 maxRequestsPerChild=909
```

Variables can also be applied to an entire group at once:

```
[columbus]
host1
host2

[columbus:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

ANSIBLE DYNAMIC INVENTORIES

A number of existing scripts are available from Ansible's GitHub site:
https://github.com/ansible/ansible/tree/devel/contrib/inventory

These scripts support the dynamic generation of an inventory based on host information available from a large number of platforms, including:

- Private cloud platforms, such as Red Hat OpenStack Platform
- Public cloud platforms, such as Rackspace Cloud, AWS, or Google Compute Engine
- Virtualization platforms, such as Ovirt (upstream of Red Hat Enterprise Virtualization)
- Platform-as-a-Service solutions, such as OpenShift
- Life cycle management tools, such as Spacewalk (upstream of Red Hat Satellite)
- Hosting providers, such as Digital Ocean or Linode

ANSIBLE TASKS

ANSIBLE TASKS

Each play contains a list of tasks

Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task

It is important to understand that, within a play, all hosts are going to get the same task directives

Every task should have a **name**, which is included in the output from running the playbook

This is human readable output, and so it is useful to have provide good descriptions of each task step

ANSIBLE TASKS

Task examples:

- Directory should exist
- Package should be installed
- Service should be running

```
$ ansible web-hosts -m file -a "path=/opt/cache \
state=directory"

$ ansible web-hosts -m yum -a "name=nginx state=present"

$ ansible web-hosts -m service -a "name=nginx enabled=yes \
state=started"
```

ANSIBLE TASKS

Here is a basic task. As with most modules, the service module takes key=value arguments:

```
tasks:
  - name: Make certain Apache is running
    service: name=httpd state=started
```

The command and shell modules are the only modules that just take a list of arguments and don't use the key=value form. This makes them work as simply as you would expect:

```
tasks:
  - name: Disable SElinux
    command: /sbin/setenforce 0
```

ANSIBLE HANDLERS

Modules are written to be '**idempotent**' and can relay when they have made a change on the remote system

Playbooks recognize this and have a basic event system that can be used to respond to change

These '**notify**' actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks

For instance, multiple resources may indicate that Apache needs to be restarted because they have changed a config file, but Apache will only be bounced once to avoid unnecessary restarts once, after all of the tasks complete in a particular play

ANSIBLE HANDLERS

Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name, and are **notified by notifiers**

If nothing notifies a handler, it will not run. Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play

Here's an example handlers section:

```
handlers:
  - name: restart memcached
    service: name=memcached state=restarted
  - name: restart apache
    service: name=apache state=restarted
```

ANSIBLE HANDLERS

Here's an example of restarting two services when the contents of a file change, but only if the file changes:

```
- name: template configuration file
  template: src=template.j2   dest=/etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

The items listed in the **notify** section of a task are called **handlers**

ANSIBLE HANDLERS

As of Ansible 2.2, handlers can also “listen” to generic topics, and tasks can notify those topics as follows:

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
      listen: "restart web services"
    - name: restart apache
      service: name=apache state=restarted
      listen: "restart web services"

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

ANSIBLE PLAYBOOKS

ANSIBLE PLAYBOOKS

Ansible playbooks are written in **YAML** in a list format

The items in the list are **key/value pairs**

Composing a playbook requires only a basic knowledge of YAML syntax

Start and end of file markers YAML files are initiated with a beginning of document marker made up of three dashes **(---)**

Playbooks can be initiated with this line. However, it is optional and does not affect the execution of a playbook

YAML files are terminated with an end of document marker made up of three periods **(...)**. This is also optional in a playbook

ANSIBLE PLAYBOOKS

Playbooks are files which describe the desired configurations or procedural steps to implement on managed hosts

Playbooks offer a powerful and flexible solution for configuration management and deployment

Playbooks can change lengthy, complex administrative tasks into easily repeatable routines with predictable and successful outcomes

Each playbook can contain one or more **plays**

Because a play applies a **set of tasks** to a set of hosts, multiple plays are required when a situation calls for a different set of tasks to be performed on one set of hosts and a different set of tasks to be performed on another set of hosts

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE


```
---
- name: Install and start Apache
  hosts: all
  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=running
```

PLAYBOOK EXAMPLE: USING VARIABLES


```
---
- hosts: all
  become: true
  vars:
     package: vim
  tasks:
     - name: Install Package
       apt: name={{ package }} state=latest
```

PLAYBOOK EXAMPLE: USING VARIABLES/LOOPS


```
- name: Install Packages
  apt: name={{ item }} state=latest
  with_items:
     - vim
     - git
     - curl
```

PLAYBOOK EXAMPLE: USING VARIABLES/ARRAYS


```
---
- hosts: all
  sudo: true
  vars:
     packages: [ 'vim', 'git', 'curl' ]
  tasks:
     - name: Install Package
       apt: name={{ item }} state=latest
       with_items: packages
```

PLAYBOOK EXAMPLE: CONDITIONALS


```
- name: Shutdown Fedora Based Systems
  command: /usr/bin/systemctl poweroff
  when: ansible_os_family == "Fedora"



- name: This is a Play
  hosts: web-servers
  tasks:
    - name: install nginx
      yum:
        name: httpd
        state: installed
      when: ansible_os_family == "RedHat"
```

ANSIBLE PLAYBOOKS: TAGS

Tags enable you to to run specific parts of a larger playbook without running the entire playbook

```
tasks:
    - yum: name={{ item }} state=installed
      with_items:
         - httpd
         - memcached
      tags:
         - packages

    - template: src=templates/src.j2 dest=/etc/httpd.conf
      tags:
         - configuration
```

```
$ ansible-playbook example.yml --tags “configuration”

$ ansible-playbook example.yml --skip-tags "notification"
```

RUNNING ANSIBLE PLAYBOOKS

ANSIBLE YAML SYNTAX VALIDATION

YAML syntax errors in a playbook will cause its execution to fail

Upon failure, the execution output may or may not assist in pinpointing the exact source of the syntax error

Therefore, it is advisable that the YAML syntax in a playbook be verified prior to its execution

There are several ways to accomplish validation. The following example which requires the installation of the PyYAML package:

```
$ python -c 'import yaml, sys; print \
yaml.load(sys.stdin)' < myyaml.yml
```

If no syntax error exists, Python prints the contents of the YAML file to stdout in JSON format

ANSIBLE YAML SYNTAX VALIDATION

For administrators not familiar with Python, there are many online YAML syntax verification tools available

One example is the YAML Lint (http://yamllint.com/) website

Copy and paste the YAML contents in a playbook into the form on the home page and then submit the form

The web page reports the results of the syntax verification, as well as display a formatted version of the originally submitted content

The YAML Lint website reports the error message "Valid YAML!" when the following correct YAML contents are submitted

ANSIBLE YAML SYNTAX VALIDATION

There's another method we'll cover in a few moments ... :)

ANSIBLE PLAYBOOKS

Playbooks are executed using the **`ansible-playbook`** command

The command is executed on the **control node** and the name of the playbook to be executed is passed as an argument

When the playbook is executed, output is generated to show the play and tasks being executed

The output also reports the results of each task executed

Syntax:

```
$ ansible-playbook webserver.yml
```

ANSIBLE PLAYBOOKS

Prior to executing a playbook, it is good practice to perform a verification to ensure that the syntax of its contents is correct

The **`ansible-playbook`** command offers a **`–syntax-check`** option which can be used to verify the syntax of a playbook file

The following example shows the successful syntax verification of a playbook:

```
$ ansible-playbook --syntax-check webserver.yml

playbook: webserver.yml
```

ANSIBLE PLAYBOOKS: EXECUTING A DRY-RUN

Another helpful option is the **–C** option

This causes Ansible to report **what changes would have occurred** if the playbook were executed, but does not make any actual changes to managed hosts

ANSIBLE PLAYBOOKS: EXECUTING A DRY-RUN

The following example shows the dry run of a playbook containing a single task for ensuring that the latest version of httpd package is installed on a managed host

```
$ sudo ansible-playbook -C webserver.yml

PLAY [play to setup web server]
*******************************************

TASK [setup] *******************************************************
ok: [fedora24.example.com]

TASK [latest httpd version installed]
*******************************************
ok: [fedora24.example.com]

PLAY RECAP *********************************************************
fedora24.example.com: ok=2   changed=0   unreachable=0   failed=0
```

ANSIBLE PLAYBOOKS: STEP-BY-STEP

When developing new playbooks, it may be helpful to execute the playbook interactively

The **`ansible-playbook`** command offers the **`--step`** option for this purpose

When executed with this option, Ansible steps through each task in the playbook

Prior to executing each task, it prompts the user for input. User can choose '**y**' to execute the task, '**n**' to skip the task, or '**c**' to exit step-by-step execution and execute the remaining tasks non-interactively

ANSIBLE PLAYBOOKS: STEP-BY-STEP

The following example shows the step-by-step run of a playbook containing a task for ensuring that the latest version of the httpd package is installed on a managed host:

```
$ ansible-playbook --step webserver.yml
```

HANDLING ERRORS

ANSIBLE PLAYBOOKS: HANDLING ERRORS

If a node fails, Ansible stops attempting actions on the failed node, but will continue with the others...

If failure is not to be tolerated, abort the entire playbook if something fails with any host:

```
- hosts: all
  any_errors_fatal: true
  tasks:
    ...
```

Abort the run after a given percentage of hosts have failed:

```
- hosts: all
  max_fail_percentage: 20
  tasks:
    ...
```

ANSIBLE PLAYBOOKS: HANDLING ERRORS

Execute your command again with **-vvv** switch (verbose, level 3)

For ad-hoc:

```
$ ansible myhosts -vvv -m ping
```

For playbook:

```
$ ansible-playbook -vvv myplaybook.yml
```

There is also a debug mode in Ansible, you can enable it with Ansible configuration option

The easiest way to do so is to use environment variable, execute your command as follows:

```
$ ANSIBLE_DEBUG=1 ansible managed-host -m ping
```

ANSIBLE ROLES

ANSIBLE ROLES

Data centers have a variety of different types of hosts. Some serve as web servers, others as database servers, and others can have software development tools installed and configured on them

An Ansible playbook, with tasks and handlers to handle all of these cases, would become large and complex over time

Ansible roles allow administrators to organize their playbooks into separate, smaller playbooks and files

Roles provide Ansible with a way to load tasks, handlers, and variables from external files

Static files and templates can also be associated and referenced by a role

ANSIBLE ROLES

The files that define a role have specific names and are organized in a rigid directory structure, which will be discussed later

Roles can be written so they are general purpose and can be reused

Use of Ansible roles has the following benefits:

- Roles group content, allowing easy sharing of code
- Roles can be written that define the essential system elements
- Roles make larger projects more manageable
- Roles can be developed in parallel by different administrators

ANSIBLE TOWER

ANSIBLE TOWER

TOWER EMPOWERS TEAMS TO AUTOMATE

CONTROL

Scheduled and centralized jobs

KNOWLEDGE

Visibility and compliance

DELEGATION

Role-based access and self-service

SIMPLE

Everyone speaks the same language

POWERFUL

Designed for multi-tier deployments

AGENTLESS

Predictable, reliable, and secure

AT ANSIBLE'S CORE IS AN **OPEN-SOURCE** AUTOMATION ENGINE

TOWER TAKES TEAMS FURTHER, FASTER

ACCELERATED INNOVATION

- Automation enables IT to drive innovation across the business

SCALABLE SIMPLICITY

- Reduction of manual effort speeds work with fewer errors

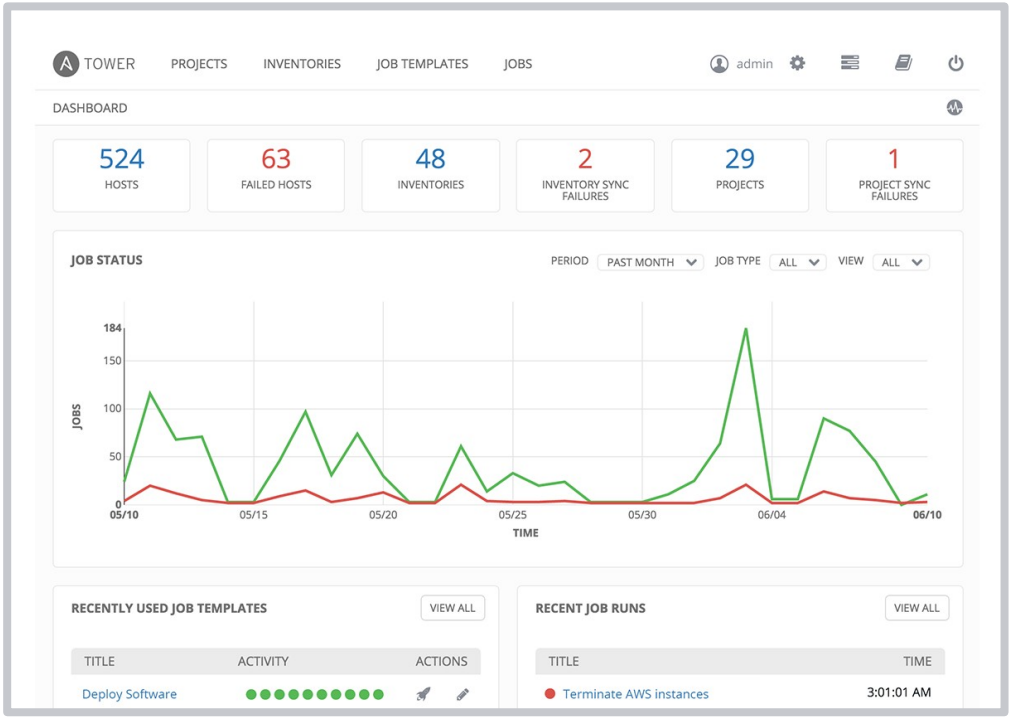
ACCOUNTABLE AUTOMATION

- Achieve compliance without being held back by it

COMMUNITY DRIVEN

- Unify teams with tooling built to drive culture

WHAT IS ANSIBLE TOWER?

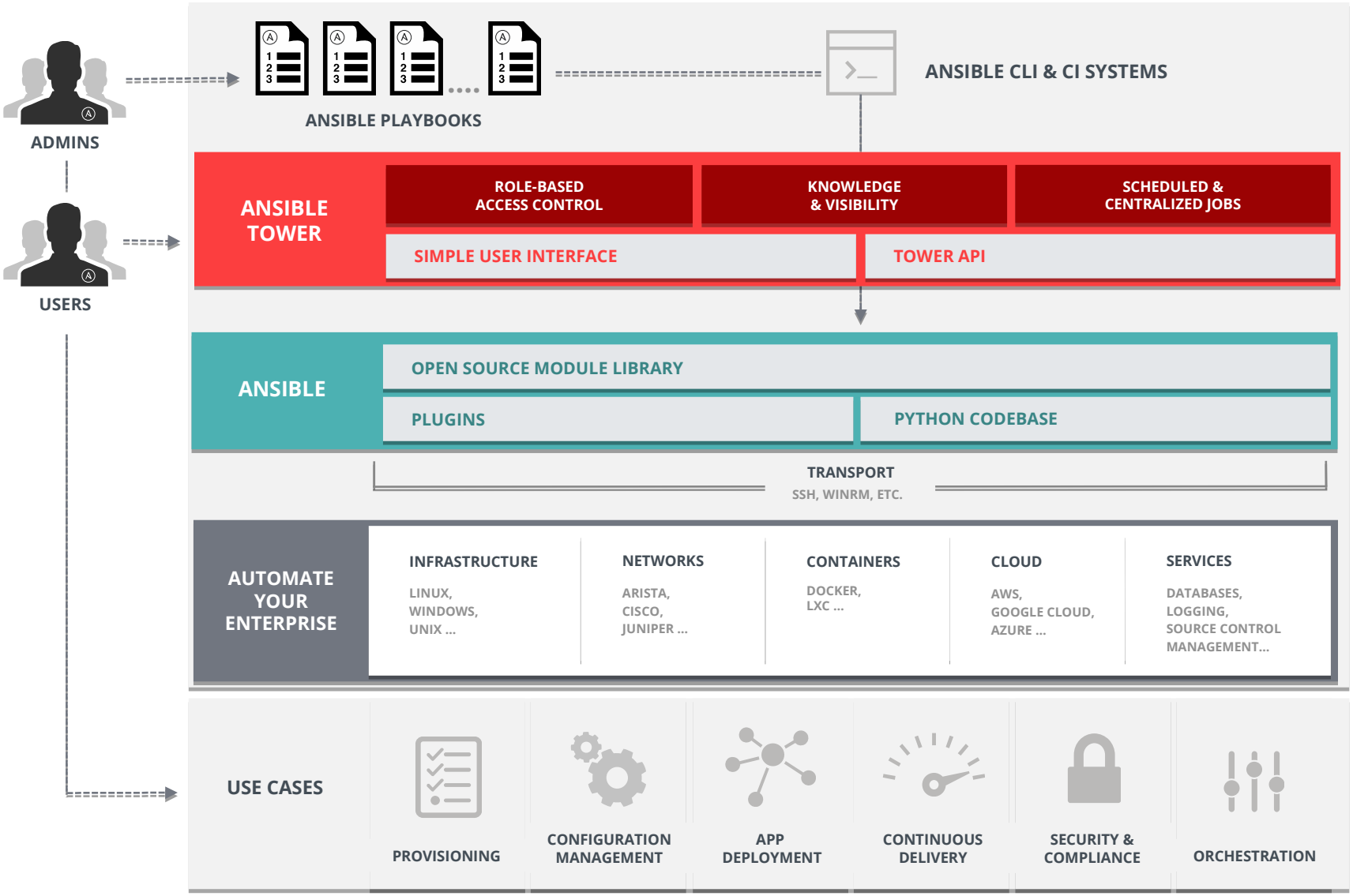


Ansible tower is an **enterprise framework** for controlling, securing and managing your Ansible automation – with a **UI and restful API.**

- **Role-based access control** keeps environments secure, and teams efficient.
- Non-privileged users can **safely deploy** entire applications with **push-button deployment** access.
- All Ansible automations are **centrally logged,** ensuring **complete auditability and compliance**.

PLATFORM OVERVIEW

ANSIBLE



OHIO LINUX FEST

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

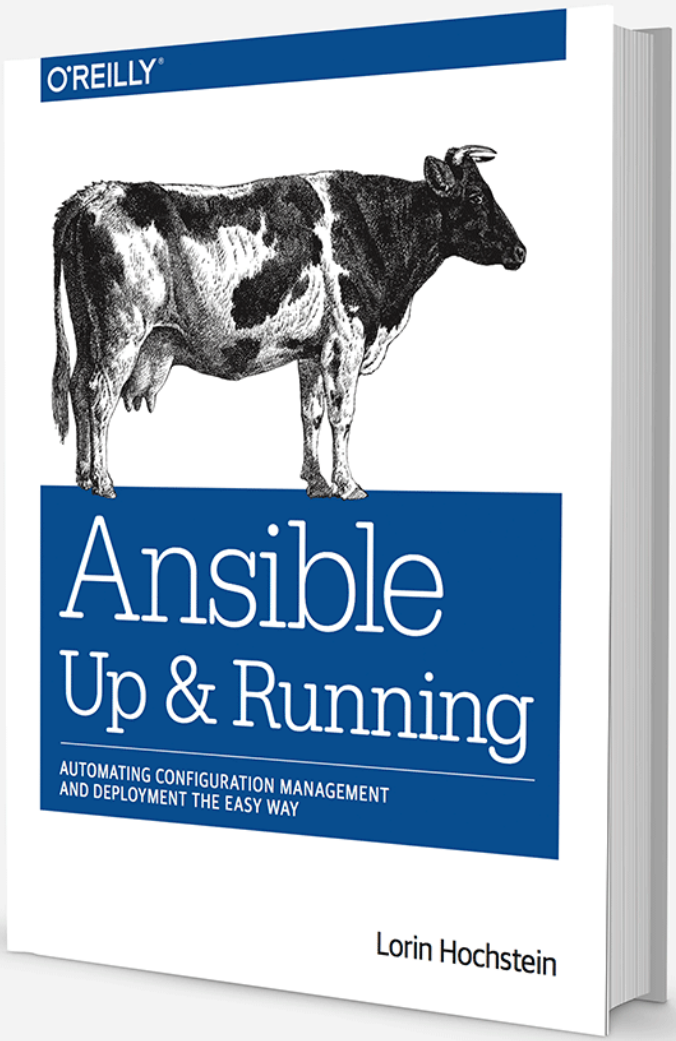
When you define your security policy in Ansible, scanning and remediation of site-wide security policy can be integrated into other automated processes and instead of being an afterthought, it'll be integral in everything that is deployed.

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.

THE MOST POPULAR OPEN-SOURCE AUTOMATION COMMUNITY ON GITHUB

- 17,500+ stars & 5,300+ forks on GitHub
- 2000+ GitHub Contributors
- Over 450 modules shipped with Ansible
- New contributors added every day
- 1400+ users on IRC channel
- Top 10 open source projects in 2014
- World-wide meetups taking place every week
- Ansible Galaxy: over 7,000 Roles
- 250,000+ downloads a month
- AnsibleFests in NYC, SF, London



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.

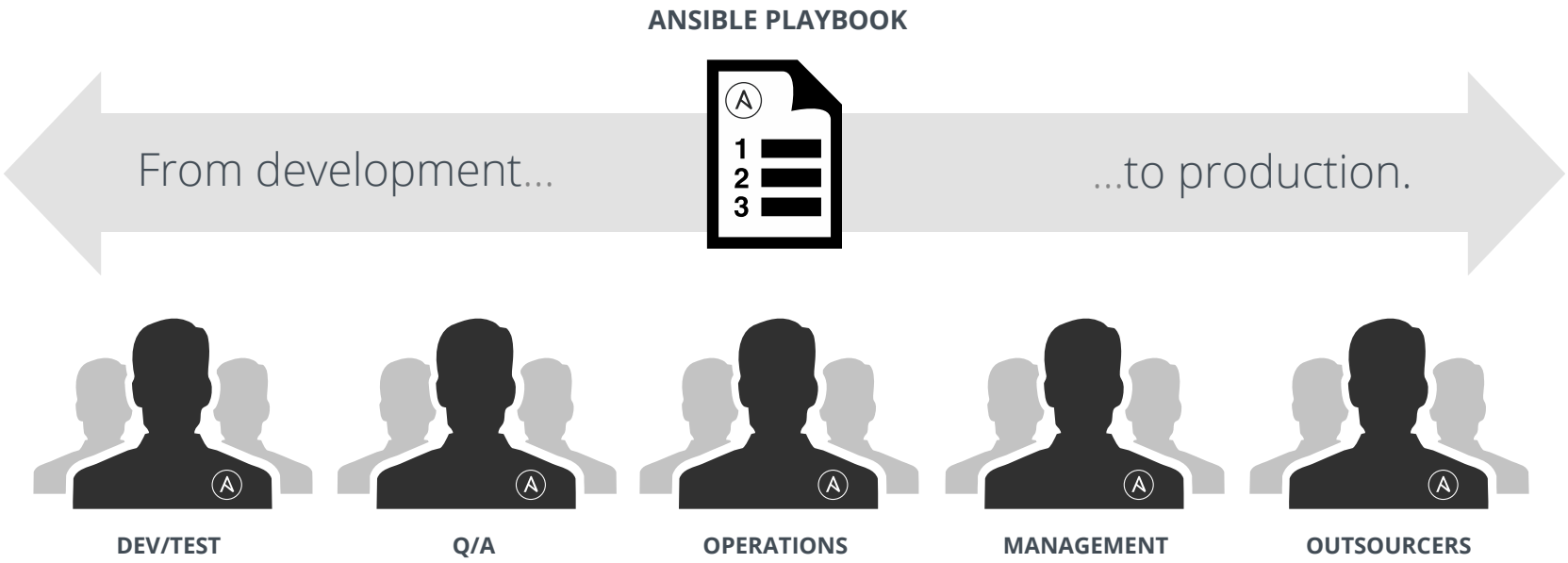
DYNAMIC INVENTORIES

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



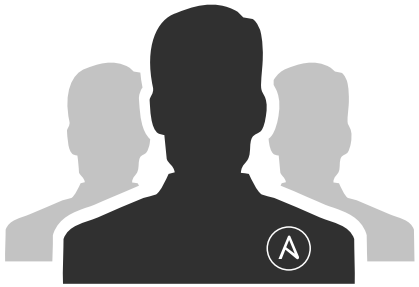
COMMUNICATION IS THE KEY TO DEVOPS.

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: ENTERPRISE IMPACT

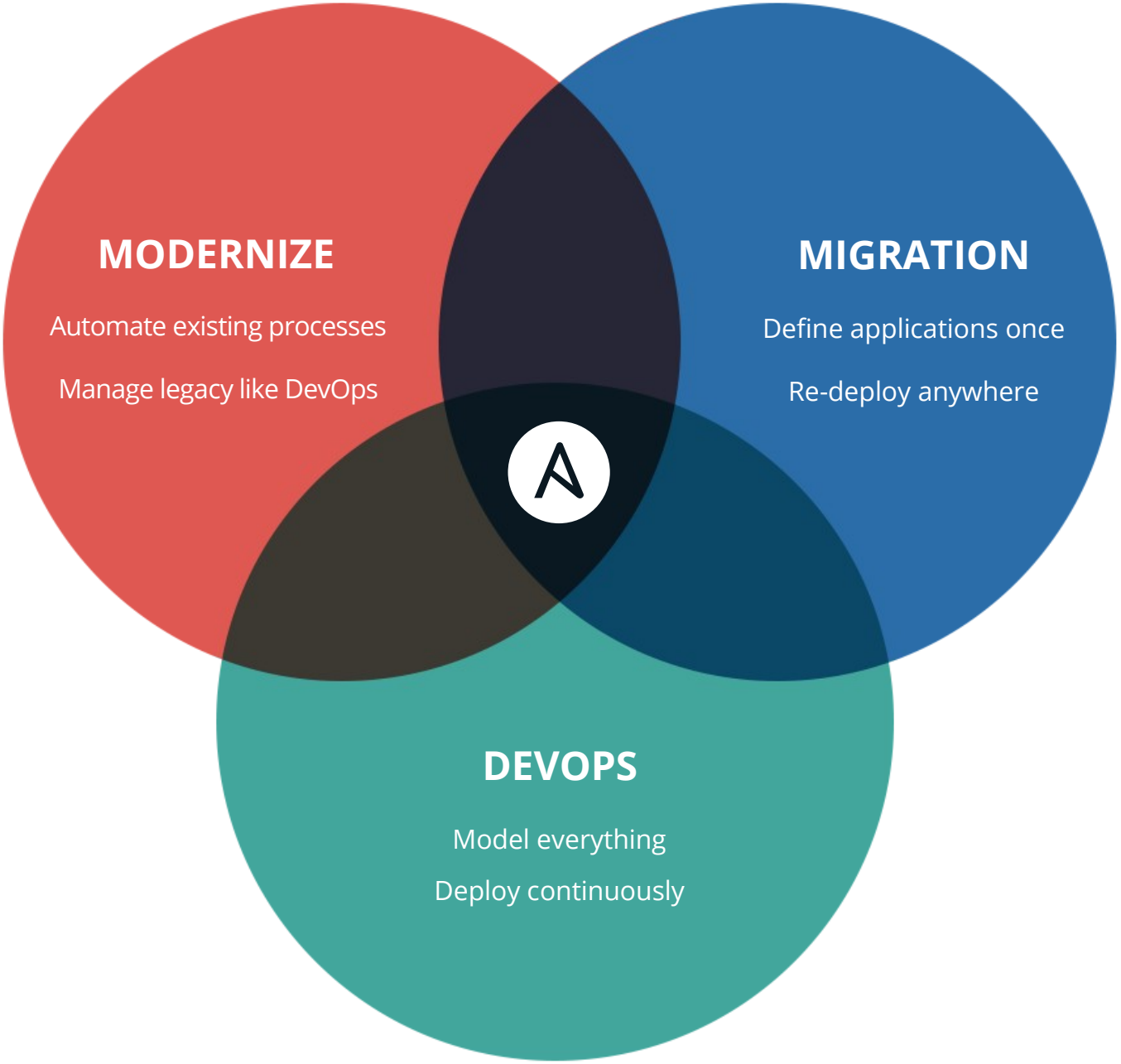
ANSIBLE



TEAM IMPACT

ENTERPRISE IMPACT

+ Save time and be more productive	+ Overcome complexity
+ Eliminate repetitive tasks	+ More resources for innovation
+ Fewer mistakes & errors	+ Increase accountability and compliance
+ Improve collaboration and job satisfaction	+ A culture of success


MODERNIZE
Automate existing processes
Manage legacy like DevOps
MIGRATION
Define applications once
Re-deploy anywhere
DEVOPS
Model everything
Deploy continuously

PLAYBOOK EXAMPLES

LAMP + HA Proxy + Nagios:

https://github.com/ansible/ansible-examples/tree/master/lamp_haproxy

JBoss Application Server:

https://github.com/ansible/ansible-examples/tree/master/jboss-standalone

RHEL DISA STIG Compliance:

http://www.ansible.com/security-stig

Many more examples at:

- http://galaxy.ansible.com
- https://github.com/ansible/ansible-examples

GETTING STARTED

Have you used Ansible already? Try Tower for free:
ansible.com/tower-trial

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

Want to learn more?
ansible.com/whitepapers