redhat.

Axcelinno

Ansible and Ansible Tower

May 23, 2019

Jim Garrett
Senior Solution Architect
jgarrett@redhat.com

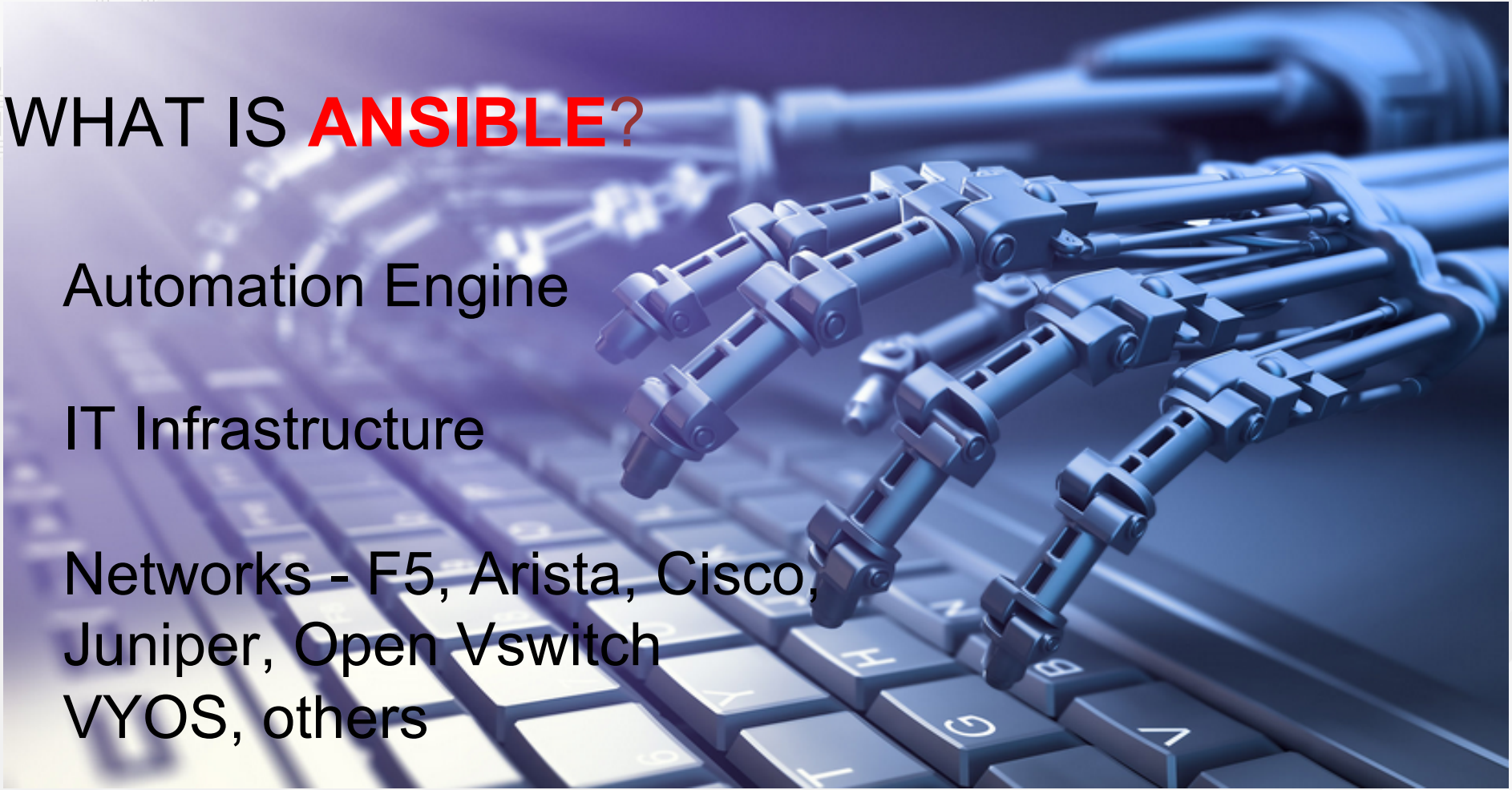
Richard Abdullah
Solutions and Innovation Consultant
richard.abdullah@axcelinno.io

Dee Ray
Solutions and Innovation Consultant
dee.ray@axcelinno.io

What You Will Learn

- What is Ansible and The Ansible Way
- 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**?



Automation Engine

IT Infrastructure

Networks - F5, Arista, Cisco,
Juniper, Open Vswitch
VYOS, others

WHAT IS **ANSIBLE**?

- It’s an automation engine - **Ansible**

Automate What????

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 kick-starting 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.

WHAT IS ANSIBLE?

- It's an automation engine - **Ansible**
- It's an automation language - **Ansible Playbooks**.
- Enterprise framework for controlling, securing and managing your automation – **Ansible Tower**

SIMPLE

Human readable automation

No special coding skills needed

Tasks executed in order

Get productive quickly

POWERFUL

Configuration management

App deployment

Workflow orchestration

Orchestrate the app lifecycle

AGENTLESS

Agentless architecture

Uses OpenSSH & WinRM

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.

Installing Ansible

```
# the best way to install Ansible on
# CentOS, RHEL, or Scientific Linux
# is to configure the EPEL repository
# and install Ansible directly
$ sudo yum install ansible

# on Debian or Ubuntu you will need the PPA
# repo configured
$ sudo apt-get install ansible

# on all other platforms it can be
# installed via pip
$ sudo pip install ansible
```

Lab Location

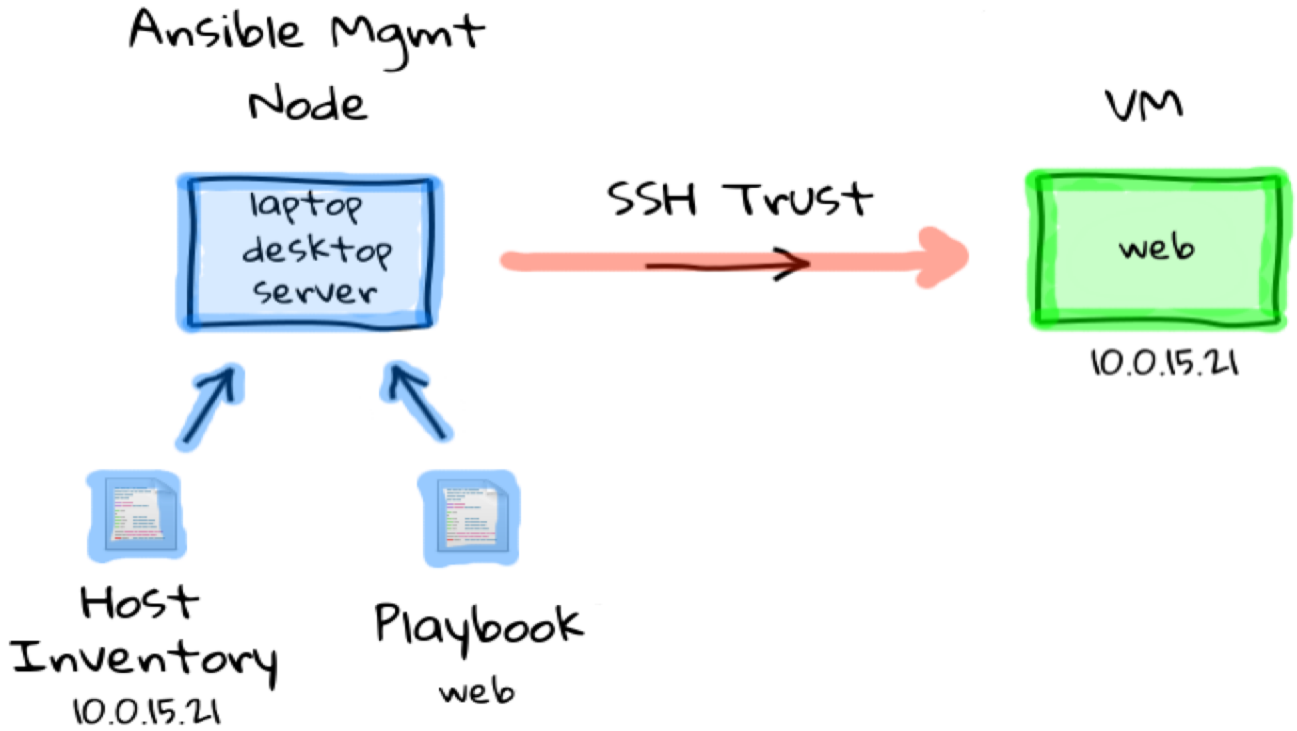
https://github.com/jimgarrett123/ansible-workshop

Lab 1 : Ansible_Install

How ANSIBLE Works – Simplicity – Command Line

$ ansible all -m ping

How ANSIBLE Works – Simplicity - playbook



$ ansible-playbook simple_playbook.yml

How ANSIBLE Works – Inventory file example

[web]

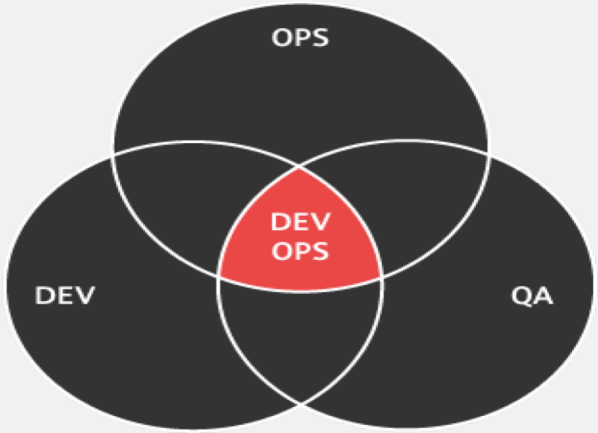
r2d2.example.com

c3po.example.com

[dbservers]

mysql.example.com

postgre.example.com



How ANSIBLE Works – Playbook

```
---
- 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=running
```

How ANSIBLE Works – Playbook - hosts

```
---
- 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=running
```

How ANSIBLE Works – Playbook - Variables

```
---
- 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=running
```

How ANSIBLE Works – Playbook – remote user

```
---
- 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=running
```

How ANSIBLE Works – Playbook - Tasks

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

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

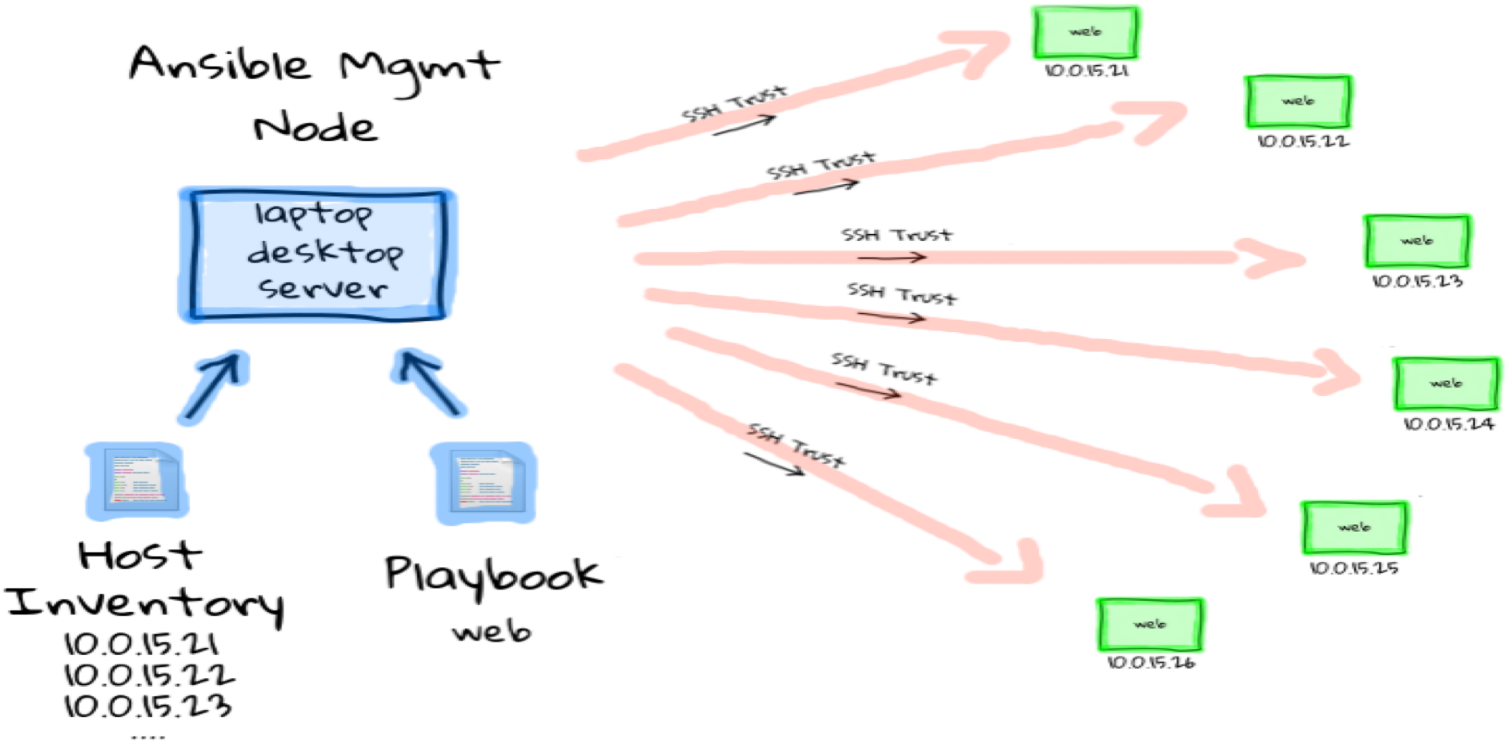
How ANSIBLE Works – Playbook - Tasks

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

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

How ANSIBLE Works – Playbook - Tasks

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

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

How ANSIBLE Works – at scale




```
$ ansible-playbook simple_playbook.yml
```

Modules

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

$ ansible all -m ping

Modules Documentation

http://docs.ansible.com/

Docs » 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
- Network Modules
- Notification Modules
- Packaging Modules
- Remote Management Modules
- Source Control Modules
- Storage Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

service - Manage services.

- Synopsis
- Options
- Examples
  - Status
  - Support

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.** Note that reloaded will start the service if it is not already started, even if your chosen init system wouldn't normally.
use (added in 2.2)	no	auto		The service module actually uses system specific modules, normally through auto detection, this setting can force a specific module. Normally it uses the value of the 'ansible_service_mgr' fact and falls back to the old 'service' module when none matching is found.

Modules Documentation

```
# List out all modules installed
$ ansible-doc -l
...
copy
cron
...

# Read documentation for installed module
$ ansible-doc copy
> COPY

  The [copy] module copies a file on the local box to remote locations. Use the [fetch] module to copy files from remote locations to the loca
  box. If you need variable interpolation in copied files, use the [template] module.

  * note: This module has a corresponding action plugin.

Options (= is mandatory):
```

Modules: Run Commands

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

- **command**: Takes the command and executes it on the host. The most secure and predictable.
- **shell**: Executes through a shell like `/bin/sh` 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

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

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

Sidebar: Discovered Facts

Facts are bits of information derived from examining a host systems that are stored as variables for later use in a play.

```
$ 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"
      },
```

Lab 2 :Adhoc_Commands

Variables

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

- Command line parameters
- Plays and tasks
- Files
- Inventory
- Discovered facts
- Roles

Tasks

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: Ensure httpd package is present
  yum:
    name: httpd
    state: latest

- name: Ensure latest index.html file is present
  copy:
    src: files/index.html
    dest: /var/www/html/

- name: Restart httpd
  service:
    name: httpd
    state: restarted
```

Handler Tasks

Handlers are special tasks that run at the end of a play if notified by another task when a change occurs.

If a package gets installed or updated, notify a service restart task that it needs to run.

Example Handler Task in a Play

```
tasks:
- name: Ensure httpd package is present
  yum:
    name: httpd
    state: latest
  notify: restart httpd

- name: Ensure latest index.html file is present
  copy:
    src: files/index.html
    dest: /var/www/html/

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

Lab 3 : Simple_Playbook (nginx)

Doing More with Playbooks

Here are some more essential playbook features that you can apply:

- Templates
- Loops
- Conditionals
- Tags
- Blocks

Lab 4 : Basic_Playbook (a more practical playbook)

Roles

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/
   apache/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

Project with Embedded Roles Example

```
# site.yml
---
- name: Execute common and apache role
  hosts: web
  roles:
     - common
     - apache
```

Lab 5 : Roles

Next Steps

- **It's easy to get started**
  ansible.com/get-started
- **Join the Ansible community**
  ansible.com/community
- **Would you like to learn a lot more?**
  redhat.com/en/services/training/do407-automation-ansible

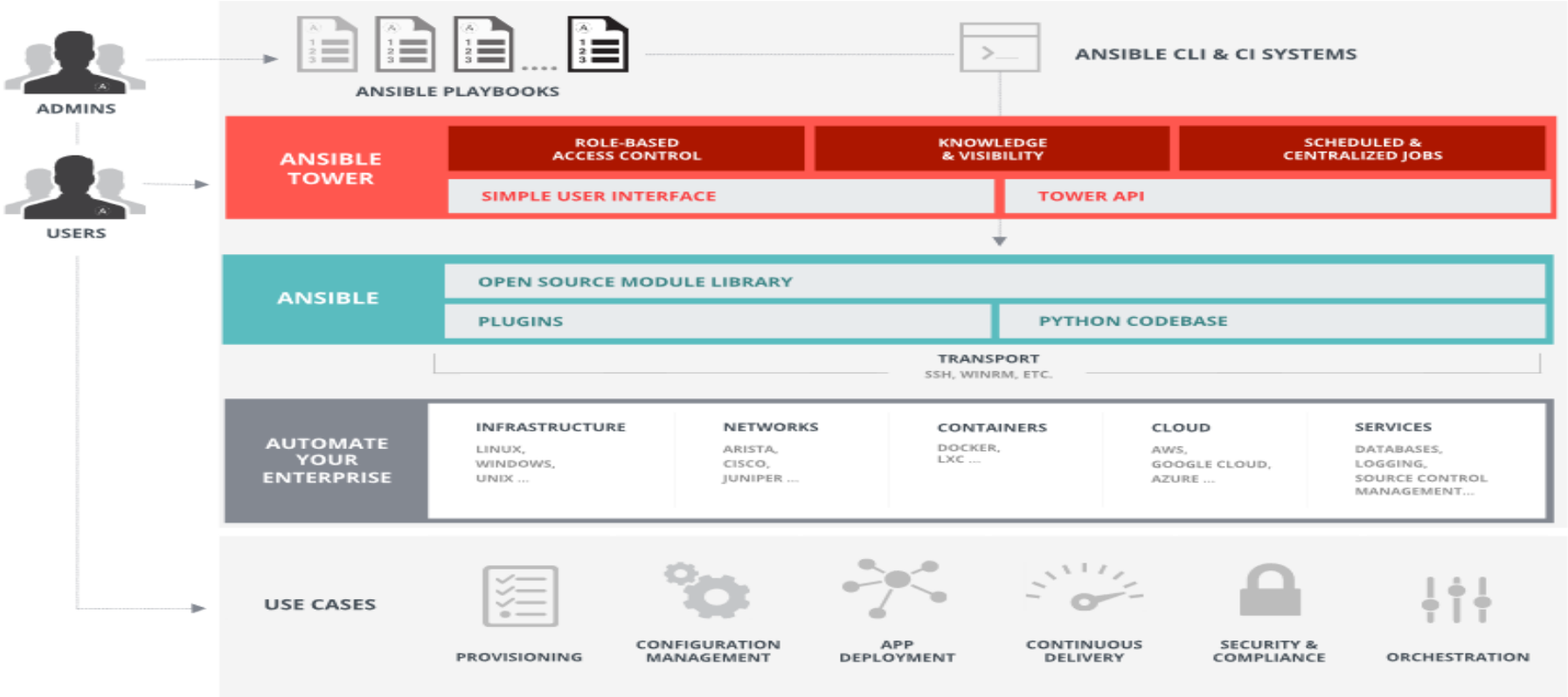
Introduction to Ansible Tower

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



Installing Ansible Tower

Download the latest Ansible Tower package
$ curl -O https://releases.ansible.com/ansible-tower/setup-bundle/ansible-tower-setup-bundle-latest.el7.tar.gz

Untar and unzip the package file
$ tar xvfz /tmp/ansible-tower-setup- (Name of tar file that was downloaded)*

Change directories into the ansible tower package
$ cd /tmp/ansible-tower-setup

Using an editor of your choice, open the inventory file
$ vi inventory
Fill a few variables out in an inventory file: **admin_password, rabbitmq_password, pg_password**

Run the Ansible Tower setup script
$ sudo ./setup.sh

Server Requirements

- Red Hat Enterprise Linux (RHEL) 7 (and select derivatives), Ubuntu 14.04 64-bit, and Ubuntu 16.04 LTS 64-bit support required (kernel and runtime).
- A currently supported version of Mozilla Firefox or Google Chrome.
- 2 GB RAM minimum (4+ GB RAM highly recommended)
- 20 GB of dedicated hard disk space

Lab : Installing Ansible Tower

Key Features of Ansible Tower

- Dashboard and User Interface
- User Base -- Organizations, Teams & Users
- Credentials
- Inventories
- Projects
- Job Templates & Jobs
- Role Based Access Control (RBAC)

Dashboard and User Interface

User Base

- A **user** is an account to access Ansible Tower and its services given the permissions granted to it.
- An **organization** is a logical collection of users, teams, projects, inventories and more. All entities belong to an organization with the exception of users.
- **Teams** provide a means to implement role-based access control schemes and delegate responsibilities across organizations.

Credentials

Credentials are utilized by Ansible Tower for authentication with various external resources:

- Connecting to **remote machines** to run jobs
- Syncing with **inventory** sources
- Importing project content from **version control systems**
- Connecting to and managing **networking devices**

Centralized management of various credentials allows end users to leverage a secret without ever exposing that secret to them.

Inventory

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

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

Projects

A Project is a logical collection of Ansible Playbooks, represented in Ansible Tower.

You can manage Playbooks and Playbook directories by placing them in a **source code management system** supported by Ansible Tower, including Git, Subversion, and Mercurial.

Job Templates

A job template is a definition and set of parameters for running an Ansible Playbook.

Job templates are useful to **execute** the same job many times and encourage the **reuse** of Ansible Playbook content and collaboration between teams.

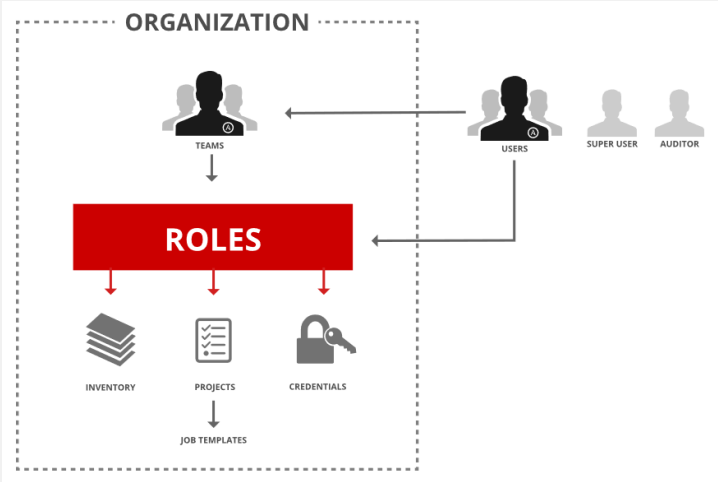
Jobs

A job is an instance of Ansible Tower launching an Ansible Playbook against an inventory of hosts.

- Job results can be easily viewed
- View the standard out for a more in-depth look

Role Based Access Control (RBAC)

Role-Based Access Controls (RBAC) are built into Ansible Tower and allow administrators to **delegate access** to server inventories, organizations, and more. These controls allow Ansible Tower to help you **increase security** and **streamline management** of your Ansible automation.



Lab : Configuring Ansible Tower

Dynamic Inventory in Ansible Tower

Dynamic inventory is a script that queries a service, like a cloud provider API or a management application. This data is formatted in an Ansible-specific JSON data structure and is used in lieu of static inventory files.

- Groups are generated based on host metadata
- Single source of truth saves time, avoids duplication and reduces human error
- Dynamic and static inventory sources can be used together

Next Steps

- **It's easy to get started**
  ansible.com/get-started
- **Try Ansible Tower for free:**
  ansible.com/tower-trial
- **Would you like to learn a lot more?**
  redhat.com/en/services/training/do409-automation-ansible-ii-ansible-tower

Lab : Creating and Running Job Template

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHatNews


youtube.com/user/RedHatVideos

Community

THE MOST POPULAR OPEN-SOURCE AUTOMATION COMMUNITY ON GITHUB

- 28,000+ stars & 10,000+ forks on GitHub
- 3200+ GitHub Contributors
- Over 1300 modules shipped with Ansible
- New contributors added every day
- 1200+ users on IRC channel
- Top 10 open source projects in 2017
- World-wide meetups taking place every week
- Ansible Galaxy: over 18,000 subscribers
- 250,000+ downloads a month
- AnsibleFest and Ansible Automates events across the globe

http://ansible.com/community