Red Hat
Training and
Certification

Ansible Automation Platform 2.x Webinar

Travis Michette

Version 1.0

Table of Contents

Introduction to Ansible Automation

1. Ansible & Ansible Automation Engine (Past)

1.1. Ansible Infrastructure

The Ansible infrastructure and Ansible Automation consists of multiple components. The initial main foundation of Ansible is the Ansible Automation Engine.

For the most part, Ansible is a declarative automation platform that is considered idempotent meaning that Ansible will only execute tasks and plays if the item needs to be changed/modified. Otherwise, Ansible will skip to the next task or play in a playbook.

Ansible Components

- **Control Node**: System with Ansible installed, contains Ansible inventory files, **ansible.cfg**, and playbooks. This system manages and controls other managed hosts/nodes.
- **Managed Host/Managed Node**: System or node being managed in the Ansible environment. The **Control Node** executes various Ansible modules against these devices.

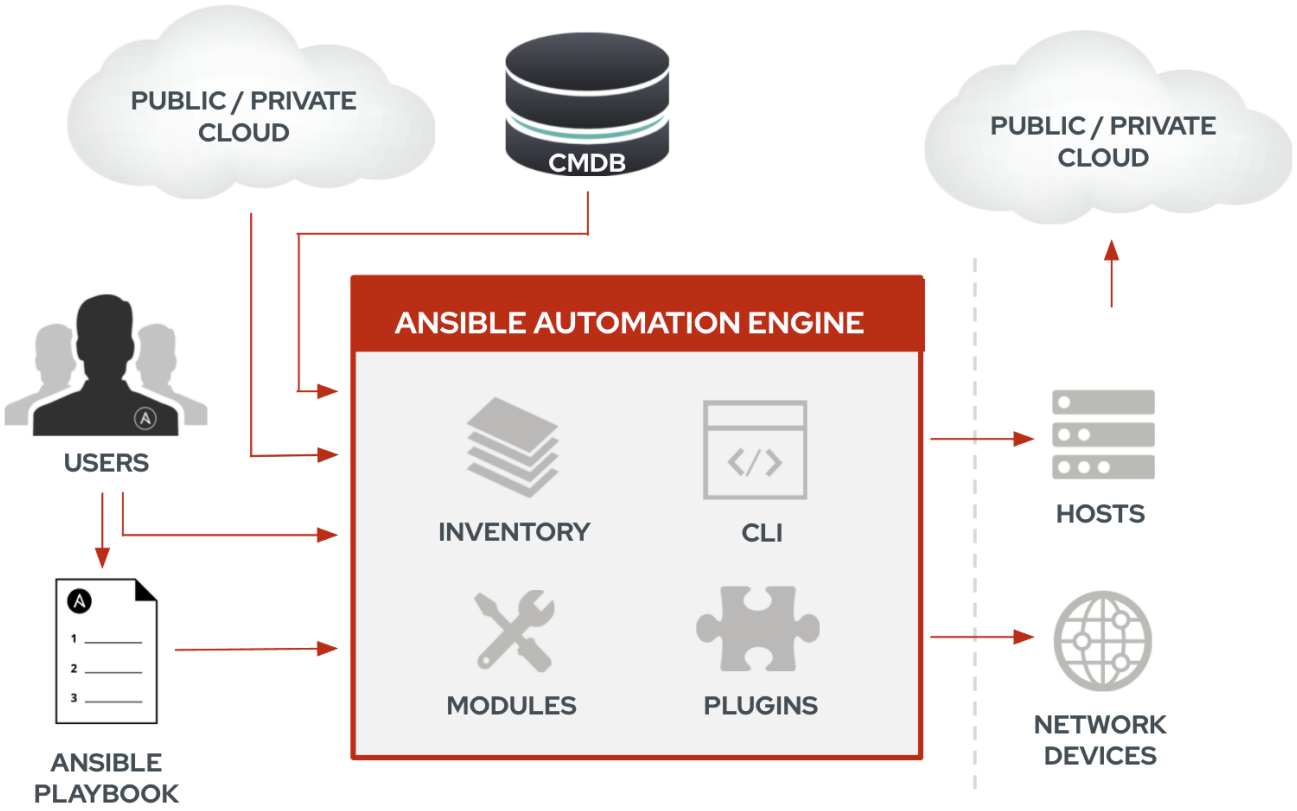

Figure 1. Ansible Automation

Ansible Automation Engine

- Inventory
- Command-Line Interface (CLI)
- Modules (Generally Python/Powershell)
- Plugins

Ansible Automation Engine utilizes the **ansible** command for Ad-Hoc Ansible Automation or the **ansible-playbook** command for running multiple tasks by leveraging and Ansible playbook containing one or more plays consisting of one or more tasks.

1.1.1. Inventory

List of systems in the infrastructure to be managed. Inventories can be static, dynamic, or a combination of both static and dynamic. Ansible also allows inventories to contain variables for the devices being managed. Devices must exist in inventory in order for Ansible to be capable of managing the devices.

1.1.2. Modules

Code utilized by the Ansible core engine which is used to perform a given tasks. Most modules are written in Python for Linux and Powershell for Windows. Modules can extend Ansible automation to multiple platforms simplifying and extending the automation to the entire stack.

Non-Idempotent Modules

There are some Ansible modules that aren't idempotent. Modules such as **commmand**, **shell**, and **raw** to name a few will execute regardless of the state. It is possible to use these modules with logic to make a playbook idempotent, but it is recommended to find an actual Ansible module to perform the task. These modules should be used as a last resort when no other module exists to perform a task.

1.1.3. Plugins

Code utilized by the Ansible core engine which is used to manipulate, transform, or otherwise modify either data in the playbook or items captured by the playbook and modules so that it is adaptable and usable on different platforms.

1.1.4. Playbooks

List of sequential tasks to allowing individual Ansible modules to be executed to perform a sequence of steps in an automation task. Playbooks are written in YAML and are simple easy-to-read steps on the end state of the system.

1.1.5. Ansible Tower

Ansible Tower delivers enterprise management and features to the Ansible family. Through Tower, Ansible can provide the following:

- Role-Based Access Control (RBAC)
- Restful API
- Push button deployment

- Workflows
- Credential and Secret Management
- Integration into SCM systems
- Integration into other management systems for dynamic inventory
- WebUI
- … and more

Ansible Tower allows enterprises to manage their IT environment by providing a centralized web solution to end-users and administrators to perform automation and self-service tasks. :pygments-style: tango :source-highlighter: pygments :toc: :toclevels: 7 :sectnums: :sectnumlevels: 6 :numbered: :chapter-label: :icons: font :icons: font :imagesdir: ../images/

1.2. Ansible Inventory, Ansible Config File, and Ansible Ad-Hoc Commands

1.2.1. Ansible Inventory

Ansible inventories can be either static, dynamic, or a combination of both static and dynamic. The traditional form of the Ansible inventory file is either YAML or INI. Inventory items consist of either individual managed nodes or groups of managed nodes.

Listing 1. Ansible Inventory File **INI** *Format*

```
servera ①
serverb
serverc
serverd

[load_balancers] ②
servere
serverf
```

① Individual managed host/node

② Inventory Group

Converting INI to YAML Inventory

Ansible provides the **ansible-inventory** command that will easily allow the inventory to be transformed from one form to another.

```
ansible-inventory --list -y
```

Listing 2. Ansible Inventory File **YAML** *Format*

```
all:
  children:
    load_balancers: ①
      hosts:
        servere: {}
        serverf: {}
    ungrouped:
      hosts: ②
        servera: {}
        serverb: {}
        serverc: {}
        serverd: {}
```

① Inventory Hosts in a Group

② Individual managed host/node (ungrouped)

1.2.1.1. Inventory Variables

It is possible for Ansible playbooks and Ansible ad-hoc commands to utilize inventory variables. These variables can be defined directly within the static inventory files themselves or those variables can be defined within the directory structure of the project utilizing either project directories or inventory directories.

Keep Inventory Simple and Organized

It is extremely important not to define variables for inventory in multipe locations as variable precedence and variable merging comes into play. It is equally important to devise an inventory strategy on where/how variables will be defined so that the playbooks and automation goals are kept simple and easy to understand and follow.

Listing 3. Sample Inventory File with Variables

```
[app1srv]
appserver01 ansible_host=10.42.0.2 ①
appserver02 ansible_host=10.42.0.3

[web]
node-[1:30] ansible_host=10.42.0.[31:60]

[web:vars] ②
apache_listen_port=8080
apache_root_path=/var/www/mywebdocs/

[all:vars] ③
ansible_user=kev
ansible_ssh_private_key_file=/home/kev/.ssh/id_rsa
```

① Defined variable at a **host** level

② Defined variables at a **group** level

③ Defined variables for all inventory items

1.2.2. Ansible Config

The **ansible.cfg** file controls how the **ansible** and **ansible-playbook** commands are run and interpreted. The configuration file has two (2) main sections that are commonly used, but include other sections as well. For the purpose of understanding how Ansible works, we will examine both the **[defaults]** section and the **[privilege_escalation]** section.

Listing 4. **ansible.cfg** *Defaults Section*

```
[defaults]
inventory = inventory ①
remote_user = devops ②
```

① Specifies which inventory file Ansible will use

② Specifies the remote user to be used by **ansible** or **ansible-playbook** commands.

A perfectly acceptable **ansible.cfg** might only have a [defaults] section specifying the inventory to be used.

Listing 5. **ansible.cfg** *Privilege Escalation Section*

```
[privilege_escalation]
become = False ①
become_method = sudo ②
become_user = root ③
become_ask_pass = False ④
```

① Sets default behavior whether to elevate privileges

② Sets method for privilege escalation

③ Sets username of privileged user

④ Sets option on whether or not user is prompted for password when perfoming privilege escalation.

Ansible Config File Precedence

- **ANSIBLE_CONFIG** - Environment Variable (highest)
- **ansible.cfg** - Config file in current working directory (most common and recommended)
- **~/.ansible.cfg** - Ansible config file in the home directory
- **/etc/ansible/ansible.cfg** - Ansible's installed default location (lowest)

1.2.3. Ansible Ad-Hoc Commands

Ansible Ad-Hoc commands are most often used to quickly perform an automation task using a single Ansible module. These commands can be executed against one or more hosts in the Ansible inventory file.

Table 1. Ansible **Ad-Hoc** *Command Arguments*

Command Argument	Description
-m MODULE_NAME	Module name to execute for the ad-hoc command

Command Argument	Description
-a MODULE_ARGS	Module arguments needed for the ad-hoc command
-b	Runs ad-hoc command as a privileged user
-K	Runs ad-hoc command as a privileged user and requests the **become** password
-e EXTRA_VARS	Provides extra variables as **KEY=VALUE** to be used for the execution of the ad-hoc command

1.2.4. DEMO - Ansible Ad-Hoc Commands

Demonstration and hands-on workshop for Ad-Hoc commands. The demo will utilize the **ping** module to ensure that the **ansible.cfg** and the **inventory** file are correctly setup and working within the Ansible environment.

Example 1. **DEMONSTRATION - Ansible Ping**

1. Change to correct directory

```
[student@workstation ~]$ cd /home/student/Github/AAP_Webinar/Past/ad-hoc
```

2. Run the **ansible** ad-hoc command

```
[student@workstation ad-hoc]$ ansible -m ping all
servere | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
servera | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverc | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverb | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverd | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverf | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Checking Sudoers Ability and Setup

Listing 6. Checking **ansible.cfg** *for Ability to* **BECOME** *without* **sudo** *Password*

```
[student@workstation ad-hoc]$ ansible -m ping all --become
```

Listing 7. Checking **ansible.cfg** *for Ability to* **BECOME** *with* **sudo** *and Prompting for Password*

```
[student@workstation ad-hoc]$ ansible -m ping all --become -K
BECOME password:
```

The next demonstration will use the **copy** module to create a user in the managed systems making an entry to the **sudoers** file.

Example 2. **DEMONSTRATION - Ansible Ad-Hoc Command to Create User and Sudoers File**

1. Change to correct directory

```
[student@workstation ~]$ cd /home/student/Github/AAP_Webinar/Past/ad-hoc
```

2. Run the **ansible** commands to create the user and update the **sudoers** file.

   a. Create the user on the remote system.

```
[student@workstation ad-hoc]$ ansible -m user -a 'name=travis uid=1040 comment="Travis Michette" group=wheel' servera -b
servera | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "append": false,
    "changed": false,
    "comment": "Travis Michette",
    "group": 10,
    "home": "/home/travis",
    "move_home": false,
    "name": "travis",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 1040
}
```

   b. Create the user in a **sudoers** file.

```
[student@workstation ad-hoc]$ ansible -m copy -a 'content="travis ALL=(ALL) NOPASSWD:ALL" dest=/etc/sudoers.d/travis' servera
-b
```

3. Test new user and sudo rights

   a. SSH to **servera**

```
[student@workstation ad-hoc]$ ssh travis@servera
```

   b. **sudo** without a password

```
[travis@servera ~]$ sudo -i
[root@servera ~]#
```

1.3. Ansible Playbooks

Section Info Here

1.3.1. Playbook Basics

1.3.2. Running Playbooks

1.4. Ansible Roles

Section Info Here

1.4.1. Ansible Role Overview

1.4.2. Using Roles

2. Ansible Automation Platform 1.x (Present)

2.1. <SECTION TITLE>

Section Info Here

2.1.1. <Section_Sub_Intro_Here>

3. Ansible Automation Platform 2.x (Future)

3.1. <SECTION TITLE>

Section Info Here

3.1.1. <Section_Sub_Intro_Here>