Ansible Advanced

Traditional Datacenter

What is Configuration Management?
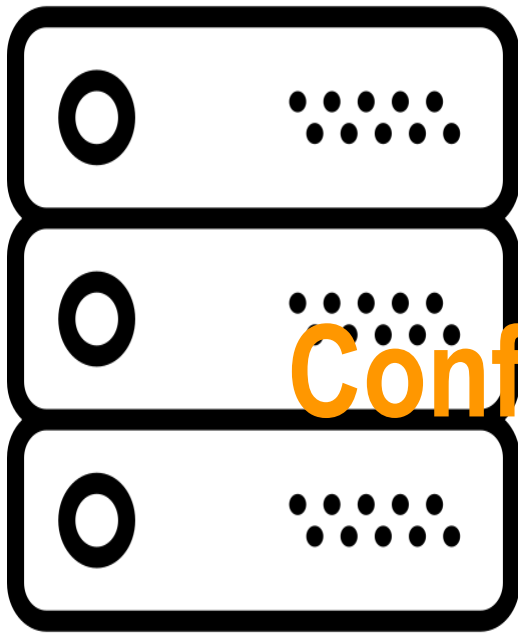
Servers

Network
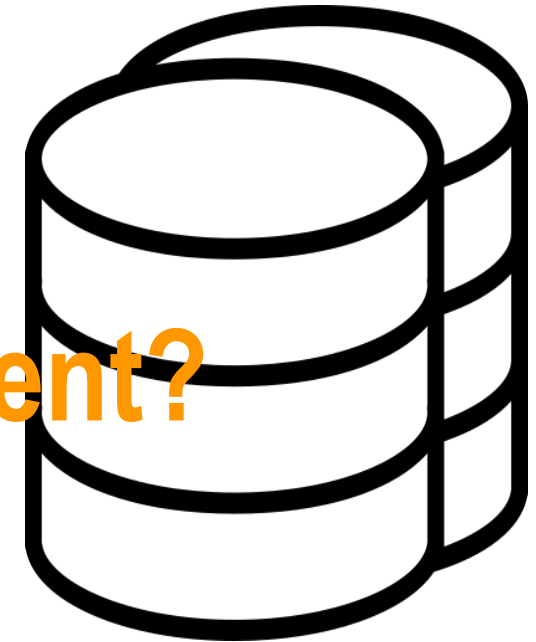
Storage

What is configuration Management

Configuration management is a process for maintaining computer systems, servers, and software in a desired, consistent state. It's a way to make sure that a system performs as it's expected to as changes are made over time.

Configuration management keeps you from making small or large changes that go undocumented. These misconfigurations can lead to poor performance, inconsistencies, or noncompliance and negatively impact business operations and security. When undocumented changes are made across many systems and applications, it adds to instability and downtime.
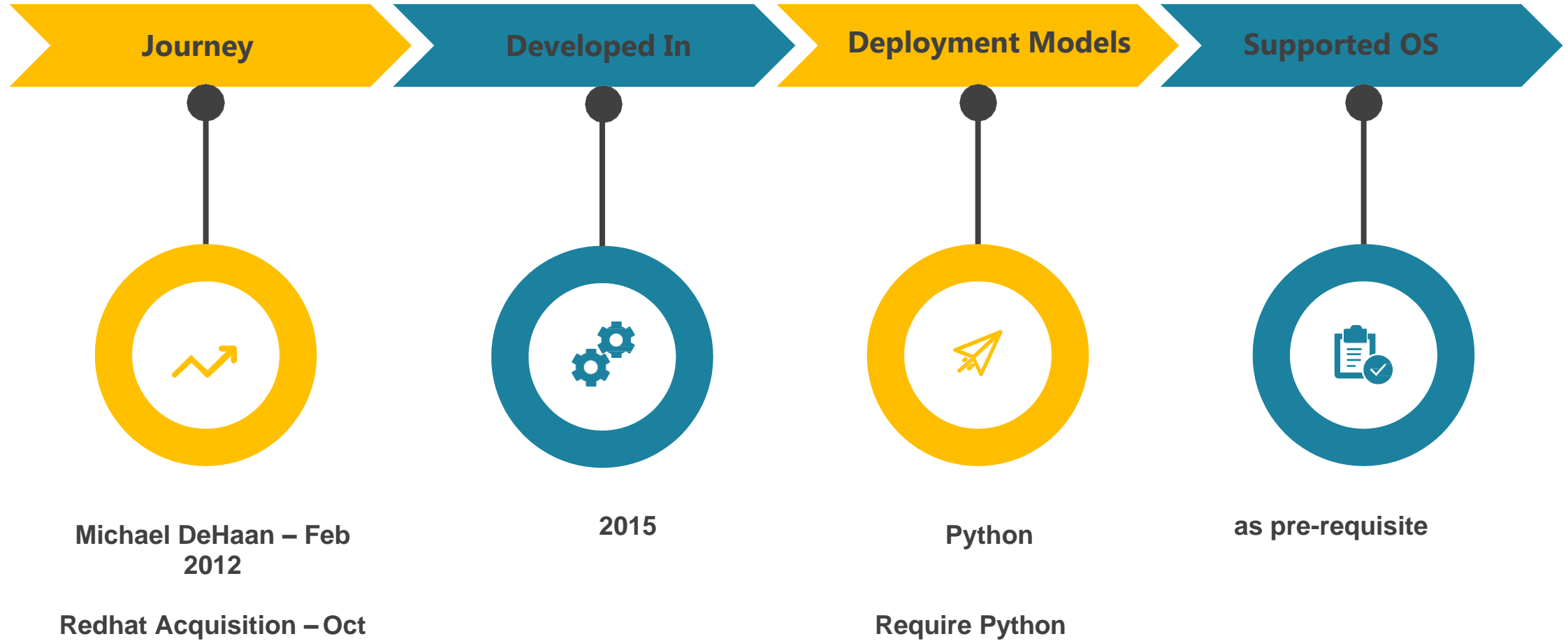
Ansible

GCP

Linux | Windows

AWS

Servers

Network | Storage | Servers

OnPrem Datacentre

Ansible Master
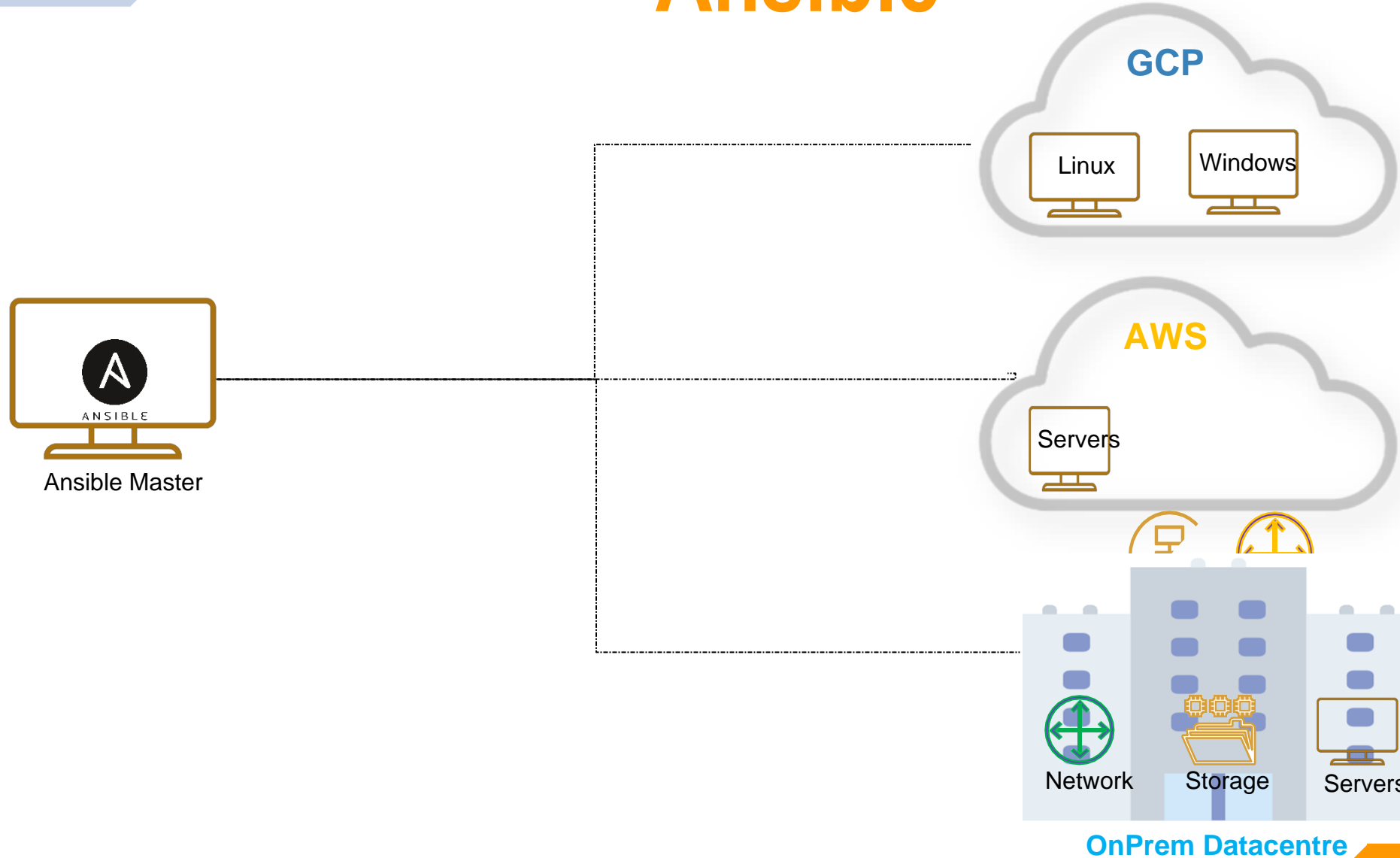
Ansible

Ansible is an easy-to-use IT Automation, Configuration Management & Orchestration Software for System Administrators & DevOps Engineers.

➢ Founded in Feb, 2012

➢ First commercial product release in 2012

➢ Multiple in-built functional modules

➢ Multiple Community Members

➢ 40,000+ Users

➢ 50,000+ Nodes managed in the largest deployments

➢ Support for Red Hat, CentOS, Ubuntu, Oracle Linux, MAC, OS, Solaris 10/11, Windows.

➢ Ansible Controller node Supported on Linux variants only

Ansible Introduction

Journey

Developed In

Deployment Models

Supported OS

Michael DeHaan – Feb 2012

Redhat Acquisition – Oct

2015

Python

Require Python

as pre-requisite

Ansible Introduction

Push Based Architecture

Agentless

MasterNode–Linux

ManagedNode–

Unix, Linux and Windows

Why Ansible?

ANSIBLE

Declarative

Increased Productivity

Agent Less

Simplicity
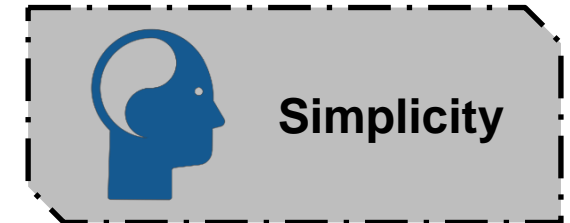
Current IT Automation State

Manually Configure: Literally logging into every node to configure it.

Golden Images: Creating a single copy of a node's software and replicating that across nodes.

Custom One-off Scripts: Custom code written to address a specific, tactical problem.

Software Packages: Typically all or nothing approach.

8

Current IT Automation State

- **Manually Configure:**

  – Difficult to scale.

  – Impossible, for all intents and purposes, to maintain consistency from node-to-node.

Current IT Automation State

- **Golden Images:**

  – Need separate images for different deployment environments, e.g. development, QA, production, or different geo locations.

  – As number of images multiply it becomes very difficult to keep track and keep consistent.

  – Since they're monolithic copies, golden images are rigid and thus difficult to update as the business needs change.

Current IT Automation State

- **Custom One-off Scripts:**

  – No leverage – effort typically cannot be reused for different applications or deployments.

  – Brittle – as needs change, often the entire script must be re-written.

  – Difficult to maintain when the original author leaves the organization.

Current IT Automation State

- **Software Packages:**

  – These packages typically require that all resources be placed under management – cannot selectively adopt and scale automation.

  – As a result, longer deployments times.

  – Dated technology developed before virtualization and cloud computing –lacks responsiveness to changing requirements.

Why Configuration Management?

- To provide optimized level of automated way to configure Applications and Software's inside your system.

- Enable you to Discover, Provision, Configure and Manage the systems.

- Developers should be able to use a single command to build and test software in minutes or even in seconds.

- Maintaining configuration state of all systems simultaneously should be easy.

- Login into every client machine for CM tasks should not be mandate.

- It should be easy to maintain desired state as per policy

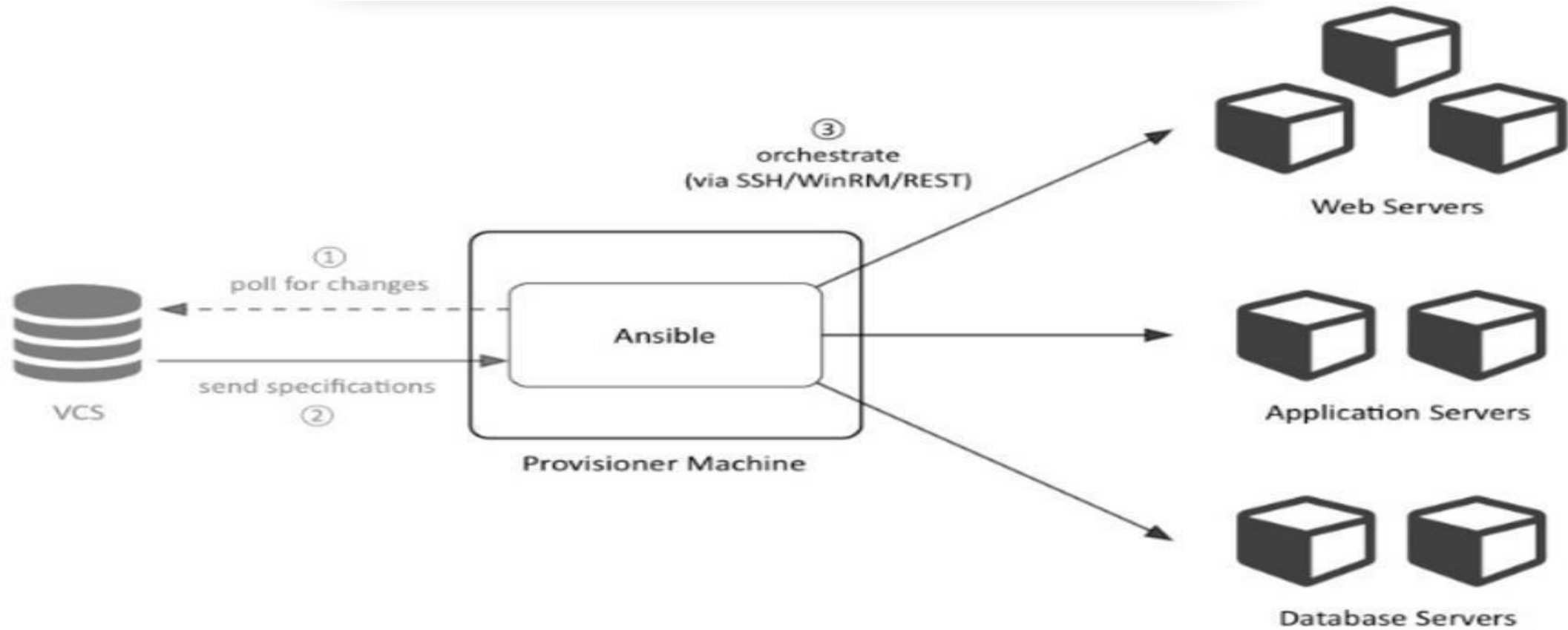# Why Orchestration with Ansible?

- A single tool for deployment and Configuration management

- Easy to manage and use

- Compatible with all major cloud service providers

- Can Orchestrate Infrastructure and Software both

Ansible Components

- Ansible consists of **Agentless Model** and majorly have two parts:

  - ➢ **Controller / Master**: The central configuration server where we will have our all configurations stored.
  - ➢ **Managed Nodes / Clients**: All clients getting configured from Ansible Master.

- **Note**:
- Ansible Master can be run from any Linux machine (Windows not supported) with Python 2 (version 2.7) or Python 3 (versions 3.5 and higher) installed.
- On the managed nodes, you need a way to communicate, which is normally SSH. By default this uses SFTP. If that's not available, you can switch to SCP in ansible.cfg. You also need Python 2 (version 2.6 or later) or Python 3 (version 3.5 or later).

Dataflow

Copyrights
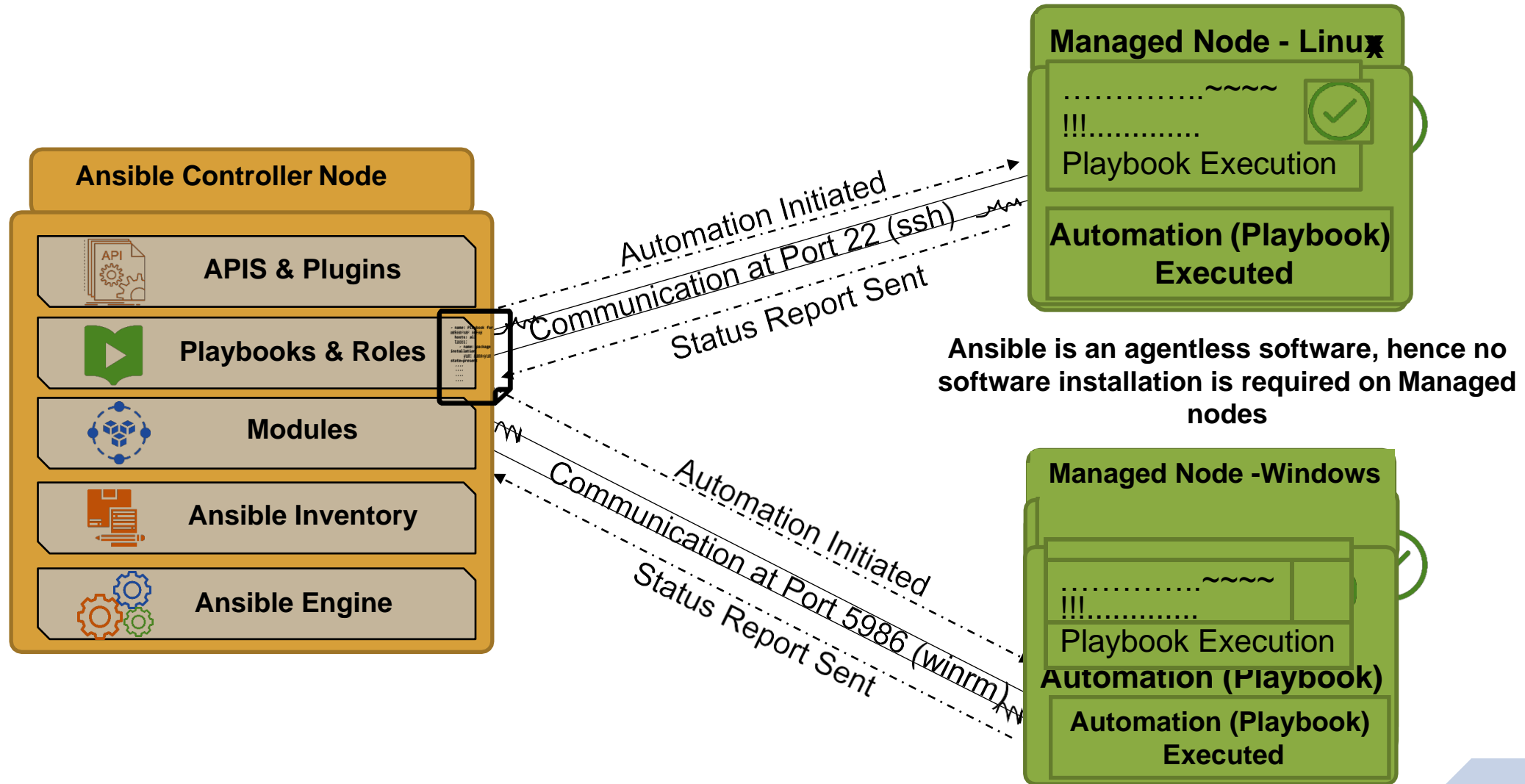
Ansible and its Peers

Many tools available in Market. Few things to consider, before selecting any tool:

- Configuration Management vs Orchestration

- Mutable Infrastructure vs Immutable Infrastructure

- Procedural vs Declarative

- Client/Server Architecture vs Client-Only Architecture

Ansible and its Peers

	Chef	Puppet	Ansible	SaltStack	CloudFormation	Terraform
Code	Open source	Open source	Open source	Open source	Closed source	Open source
Cloud	All	All	All	All	AWS only	All
Type	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	Orchestration	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Mutable	Immutable	Immutable
Language	Procedural	Declarative	Declarative	Declarative	Declarative	Declarative
Architecture	Client/Server	Client/Server	Client-Only	Client/Server	Client-Only	Client-Only

Knowledge Checks

- What is Configuration Management?

- List a few available configuration Management tools.

- What are the Advantages of Ansible?

- Explain Data flow of Ansible.

Ansible Installation

Installation of Ansible

- The Ansible **master** is the machine that controls the infrastructure and dictates policies for the servers it manages.

- Currently Ansible can be run from any machine with Python 2.6 or 2.7 installed (Windows isn't supported for the control machine).

- This includes Red Hat, Ubuntu, Debian, CentOS, OS X, any of the BSDs, and so on.

Lab1: Installation of Ansible

- To install the Ansible Master, we need to install EPEL repository package:

  ◦ Ansible Repository

  http://fedoraproject.org/wiki/EPEL

  Note: If internet connectivity is there just do:

  - wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm

- Pre-installation

- Assign a hostname to your machine(Master) and make that name persist across reboot.

Lab1: Installation of Ansible

- yum install ansible

- rpm –qa | grep –i ansible

- ansible --version

- Default Configuration file is /etc/ansible/ansible.cfg

- Default Inventory file is /etc/ansible/hosts

Ansible Master Configuration

- Edit /etc/ansible/ansible.cfg for any Master Configurations

- Default options are fine

- All parameters can be overridden in ansible-playbook or with command line flags.

Special Shortcut: cat /etc/ansible/ansible.cfg | grep "^\["

Ansible Master Configuration

- Ansible comes with a default Ansible configuration file which can be customized by changing Ansible configuration parameters.

- **/etc/ansible/ansible.cfg** – by default base configuration file location.

- **~/.ansible.cfg** - user specific configuration file, this configuration file will be used by Ansible if Ansible is executed by logged in user.

- **./ansible.cfg** - the precedence will be given to this file, if Ansible run is executed from the directory path where ansible.cfg file is present.

- **ANSIBLE_CONFIG** - configuration file location defined by an environment variable.

Lab2: Working with Ansible.cfg

- Run Ansible --version command and check the outcome of "config file".

- Copy /etc/ansible/ansible.cfg file into your home directory as below:
  - cp /etc/ansible/ansible.cfg ~/.ansible.cfg
- Run Ansible --version command and check the outcome of "config file".

- Switch to /tmp and Copy /etc/ansible/ansible.cfg file into your Current directory as below:
  - cp /etc/ansible/ansible.cfg /tmp/ansible.cfg
- Run Ansible --version command and check the outcome of "config file".

- Export ansible Config file into variable **ANSIBLE_CONFIG** as below:
  - cp /etc/ansible/ansible.cfg /ansible.cfg
  - export ANSIBLE_CONFIG="/ansible.cfg"
- Run Ansible --version command and check the outcome of "config file".

Ansible Clients

- **As Ansible uses Agentless mechanism so no packages are required to be installed on any client**.

- We need a way to communicate, which is normally ssh. By default this uses sftp. If that's not available, you can switch to scp in ansible.cfg.

- We also need Python 2.4 or later. If you are running less than Python 2.5 on the remotes, you will also need below package to be installed on the server:

   "python-simplejson"

- In case of windows client you need to configure winrm and start the service to start listening on port 5985-5986.

Ansible Authentication

- As Ansible is using SSH by default during communication, this communication connection supports both:

- **Password Based Authentication:** Password based authentication is acceptable if your environment is small and easily manageable. But it become very difficult to work with password-based authentications once you scale your environment. Password based authentication is only useful in Engineering Labs or Test Labs or Playbook creations tests.

- **Key Based Authentication:** Key based Authentication is adopted in Enterprise Environments. Here we create one generic user and amend the keys of the generic user in Managed Nodes for Key Based - Password less Authentication. This is a onetime task and can be used with any number of servers.

Ansible Inventory

- Ansible Inventory is a text-based list of individual servers and/or group of multiple servers.

- By default the Ansible Inventory location is **"/etc/ansible/hosts"**.

- You may have multiple Inventory files.

- Ansible Inventory can have Host Name or IP Address or Combination of both.

- Ansible provides the flexibility to pull inventory from Dynamic or Cloud sources with the help of scripts.

- You can specify a different inventory file using the -i <path> option on the command line.

Ansible Fundamentals

Case Study - 1

You need to manage a user, .

You care specifically about:
- his existence
- his primary group
- his home directory

Case Study - 1

- Tools built into most distro's that can help:

- useradd
- usermod
- groupadd
- groupmod
- mkdir
- chmod
- chgrp
- chown

Case Study - 1

- Platform idiosyncrasies:

  o Does this box have 'useradd' or 'adduser'?

  o What was that flag again?

  o What is difference between '–l' and '–L'?

  o What does '–r' means

  o Recursive

  o Remove read privileges

  o System user

- If I run this command again, what will it do?

Case Study - 1

- You could do something like this:

```
#!/bin/sh
USER=$1 ; GROUP=$2 ; HOME=$3
if [ 0 –ne $(getent passwd $USER > /dev/null)$? ]
then useradd $USER –home $HOME –gid $GROUP –n ; fi
OLDGID=`getent passwd $USER | awk –F: '{print $4}'`
OLDGROUP=`getent group $OLDGID | awk –F: '{print $1}'`
OLDHOME=`getent passwd $USER | awk –F: '{print $6}'`
if [ "$GROUP" != "$OLDGID" ] && ["$GROUP" != "$OLDGROUP" ]
then usermod –gid $GROUP $USER; fi
if [ "$HOME" != "$OLDHOME" ]
then usermod –home $HOME $USER; fi
```

Case Study - 1

What About?

- Robust error checking?

- Solaris and Windows support?

- Robust logging of changes?

- Readable code?

- What if need to create in 1000+ Servers?

Case Study – 1

Ansible Way of Configuration Management:

tasks:

 - name: Creating Gagandeep User

   user: name=gagandeep comment="Gagandeep Singh" state=present

tasks:

 - name: Creating Singh Group

   group: name=singh state=present

Ansible Way: Maintaining State

- You(Even Ansible can do it on cloud) provision a node.

- Ansible configures it.

- Ansible maintains the desired state when needed.

Note: You make to sure the state is configured as per environment requirements.

Ansible : Infrastructure as Code

- Descriptive
- Straightforward
- Transparent

```
[root@gagan]# cat ntp.yml
---
# This is my Host section
- hosts: localhost
# This is my Task section
 tasks:
  - name: NTP Installation
    yum: name=ntp state=present
  - name: NTP Service
    service: name=ntpd state=started enabled=yes
```

Ansible : Idempotency

- Ansible enforces in an idempotent way.

- The property of certain operations in mathematics or computer science is that they can be applied multiple times without further changing the result beyond the initial application.

- Able to be applied multiple times with the same outcome.

Ansible Terminology

- **Controller/Master:** The Ansible master is the machine that controls the infrastructure and dictates policies for the servers it manages. It operates both as a repository for configuration data and as the control center that initiates remote commands and ensures the state of your other machines.

- **Managed/Agent Nodes :** The servers that Ansible configures are called Clients/Nodes.

- **Ansible Inventory:** Ansible Inventory represents which machines it should manage using a very simple INI file that puts all of your managed machines in groups of your own choosing.

- **Ansible Adhoc-tasks:** Ansible uses adhoc requests to confirm simple and small tasks on any server right a way without login into the client. The best example is to check the Alive Status for whole managed inventory.

Ansible Terminology

Playbooks: A structured way to put all of the defines tasks for your application or your whole setup.

Modules: In built functions which executes at the backend to perform underlined tasks in Ansible.

*yml files: describe a set of desired states that a system needs to be in, for example "apache needs to be installed and running".

Ansible Tower: Ansible Tower by Red Hat helps is a web-based solution that makes Ansible even more easy to use for IT teams of all kinds. It's designed to be the hub for all of your automation tasks.

Ansible Terminology

- **Ansible Galaxy:** Ansible Galaxy is a free site for finding, downloading, and sharing community developed roles. Downloading roles from Galaxy is a great way to jumpstart your automation projects..

- **Ansible for Unix/Linux:** The Ansible master communicates and manage Unix/Linux Clients using SSH by default.

- **Ansible for Windows:** Starting in version 1.7, Ansible also contains support for managing Windows machines. This uses native PowerShell remoting, rather than SSH. and uses the "winrm" Python module to talk to remote hosts.

Ansible Communications
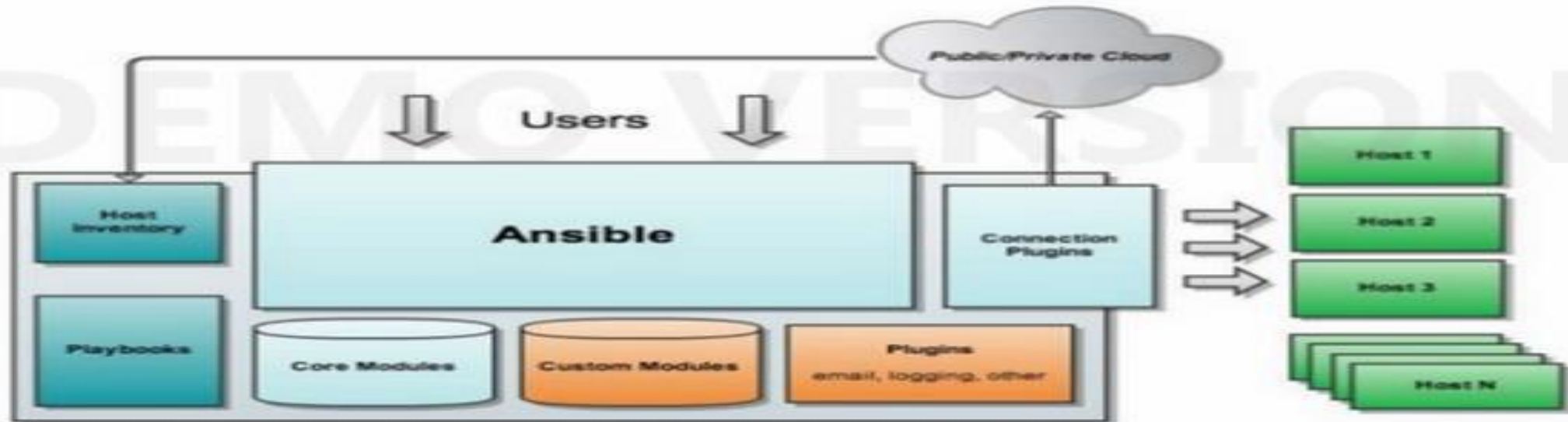
Ansible Communication test

- Communication checks with password authentications:

```
[root@gagan-controller ~]# ansible centos-managed -m ping --ask-pass
SSH password:
centos-managed | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Lab 3: Ansible configuration

- Edit Client IP and Name entry in /etc/hosts file for Name to IP Mapping:

    172.31.19.138 gagan-client   //   /etc/hosts

- Put client name entry in in /etc/ansible/hosts file (under unmanaged section/top) , so that it can be managed with ansible:

    gagan-client   //   /etc/ansible/hosts

- Run ansible commands to check if client host is manageable with ansible or not:

    ansible gagan-client –m ping --ask-pass   // ask-pass as we have not provided password in file

- Provide username and password in /etc/ansible/hosts file and run command without password:

    gagan-client ansible_user=USER ansible_password=PASSWORD   //   /etc/ansible/hosts

- Generate SSH keypair and copy keypair in remote machine for passwordless connection:

    ssh-keygen –t rsa // press enter thrice after this – no passphrase

    ssh-copy-id gagan-client   // hit enter and provide password for next machine

    ansible gagan-client –m ping

    ansible-doc -l

Ansible Modules

- Modules are the Basic Building Block of Ansible.

- These are the readymade tools to perform various tasks and operations on "Managed Nodes".

- Modules can be used with Ansible Ad-Hoc Remote Executions and/or Playbooks as a core building blocks.

- Ansible Ships with multiple in-built Modules (approx. 2000+ in Ansible 2.7).

- Can be used for Standalone servers, Virtual Machines and for any Public/Private Cloud Instances.

Ansible Modules

- Two types of Ansible Modules: Core Modules & Custom Modules.

- Robust Module Documentation on website.

- Command line utility on Module information and usage.

- CLI utility ansible-doc on "Controller Node".

- Execute ansible-doc -l to list all available Modules.

- Execute ansible-doc <module-name> to find all details about Modules.

- For GUI refer **"http://docs.ansible.com/ansible/modules_by_category.html"**

Ansible Adhoc Execution

- Easy to learn ad-hoc command line utility - "ansible".

- Quick On-demand tasks on "Managed Nodes".

- 1 to 1 approach, single ad-hoc command is used to perform single operation.

- Multiple ad-hoc operations require multiple "ansible" ad-hoc run.

- Ad-hoc execution syntax.

- Ansible Ping Communication Test with "Managed Nodes".

- Various real time examples with ad-hoc execution.

Ansible Adhoc Execution

- Let's try executing a remote command, before that make sure you have out an entry of the host in "/etc/ansible/hosts"

- Connect to the master and type:

  ansible <host-name/IP> -m ping --ask-pass

  ansible "*" -m ping --ask-pass

- First argument = target client
- Second argument = function to execute
- Other arguments = params for the function

Ansible Adhoc Execution

Adding Username and Connection method in "/etc/ansible/hosts":

You can specify each host for specific connection type/port and connection username:

- ansible_host :   The name of the host to connect to, if different from the alias you wish to give
- ansible_port: The ssh port number, if not 22
- ansible_user: The default ssh user name to use.
- ansible_ssh_pass: The ssh password to use (never store this variable in plain text; always use a vault. See Variables and Vaults)
- ansible_ssh_private_key_file: Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.

user20-client ansible_connection=ssh   ansible_user=centos

Ansible Adhoc Execution

- There are a bunch of predefined :

«Ad-Hocmodules»

«execution modules»

«Ad-Hocmodules»: ansible all/"Client or Group" -a "<adhoc-command>" --ask-pass

«execution modules» ansible all/"Client or Group" -m <module_name> -a <arguments>

- Note: To list all Ansible Modules run below command:

ansible-doc –l
ansible-doc -t connection -l

Ansible Adhoc Execution

- For example, executing a shell commands:

ansible 192.168.74.51 -a "ls -l /tmp" --ask-pass

ansible 192.168.74.51 -a "uname -a" --ask-pass

ansible 192.168.74.51 -a "cat /etc/redhat-release" --ask-pass

ansible 192.168.74.51 -a 'service ntpd status' --ask-pass

ansible "*" --list-hosts   // Its'll show all the hosts that'll effect with the command

53

Ansible Adhoc Execution

- For example, executing Ansible Modules:

ansible 192.168.74.51 -m ping --ask-pass

ansible 192.168.74.51 -m user -a "name=rahejagagan state=present" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/tmp/gagandeep mode=777 group=rahejagagan state=touch" --ask-pass

ansible 192.168.74.51 -m service -a "name=ntpd state=stopped" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/yog/rah/test mode=777 state=directory" --ask-pass

Lab 4: Ansible Adhoc Commands

- Check the uptime using Ansible Ad-hoc execution with password-less authentication

- Check OS release using Ansible Ad-hoc execution with password-less authentication

- Install a package named "telnet" on managed host

- Create a user named "yourname" with bash shell having user id of 9999 on managed host

- Create a file named "/tmp/myfile" with permission 777 and user + owner as root on managed host

- Copy a local file to remote machine

- Run multiple commands parallelly with shell module.

Lab 4: Ansible Adhoc Commands

- Check the uptime using Ansible Ad-hoc execution with password-less authentication

  ansible client -m uptime

- Check OS release using Ansible Ad-hoc execution with password-less authentication

  ansible client -a "uname -a"

- Install a package named "telnet" on managed host

  ansible client -m package -a "name=telnet state=installed"

- Create a user named "yourname" with bash shell having user id of 9999 on managed host

  ansible client -m user -a "name=gagandeep uid=9999 state=present"

- Create a file named "/tmp/myfile" with permission 777 and user + owner as root on managed host

  ansible client -m file -a "path=/tmp/myfile mode=777 group=root owner=root state=touch"

- Copy a local file to remote machine

  ansible client -m copy -a "src=/tmp/myfile dest=/root mode=777"

- Run multiple commands parallelly with shell module

  ansible client -m shell -a "ls;uname -a"

56

Facts

Ansible uses "facts" to gather information about the host system (any host).

"ansible <client-name> -m setup"

"ansible <client-name> -m setup –ask-pass"

Command returns a list of key value pairs (specific to Ansible).

The returned key value pairs are "facts".   Example:

– [root@user20-master ~]# ansible user20-client -m setup | grep -i ansible_user_id

–   "ansible_user_id": "centos",

– [root@user20-master ~]# ansible user20-client -m setup -b | grep -i ansible_user_id

–   "ansible_user_id": "root",

– [root@user20-master ~]#

57

Ansible Playbooks

Ansible Playbooks

- YAML Structure

- Ansible Playbooks

- Playbooks Structure

- Playbooks Syntax

- Playbooks Pre-Execution

- Playbooks Smoke Test

- Playbooks Real Time Run

59

YAML Structure

Though YAML syntax may seem daunting and terse at first, there are only three very simple rules to remember when writing YAML for Playbooks.

- Rule One: **Indentation**

    – YAML uses a fixed indentation scheme to represent relationships between data layers. Ansible requires that the indentation for each level consists of exactly two spaces. **Do not use tabs.**

YAML Structure

Rule Two: **Colons**

– Dictionary keys are represented in YAML as strings terminated by a trailing colon. Values are represented by either a string following the colon, separated by a space:

my_key: my_value

– In Python, the above maps to:

{'my_key': 'my_value'}

– Alternatively, a value can be associated with a key through indentation.

my_key:
    my_value

YAML Structure

Rule Three: **Dashes**

- To represent lists of items, a single dash followed by a space is used. Multiple items are a part of the same list as a function of their having the same level of indentation.

  - list_value_one
  - list_value_two
  - list_value_three

- Lists can be the value of a key-value pair. This is quite common in Ansible:

  my_dictionary:
    - list_value_one
    - list_value_two
    - list_value_three

Ansible Playbooks

- **Ansible Playbooks are the file in YAML format with sequential instructions to perform operations on Managed Nodes.**

- Written in YAML (**YAML** Ain't Markup Language).

- In simple layman language Playbooks are simple YAML files containing implementation steps.

- Opposite to Ad-Hoc requests, no restrictions on running multiple operations on managed nodes in a single run.

- Very easy to write and understand than other configuration Management tools.

- Ansible Playbooks file extension is .yml or yaml.

63

Ansible Playbooks

- The Playbook are typically divided into three parts, with YAML format:

    - Start of a Play (Hosts, variables, connections and Users)

    - Tasks list

    - Handlers

Ansible Playbook

first_playbook.yml

```
---    Start of the Playbook
- hosts: all                           Generic Section          Playbook Sections

  tasks:
    - name: NTP OS Package Installation
      package: name=ntp state=present
    - name: NTP File Configurations
      copy: src=/etc/ntp.conf dest=/etc/ntp.conf
      notify:
      - restart ntp                                              Tasks Section
    - name: To start NTP services
      service: name=ntpd state=started enabled=yes

  handlers:
    - name: restart ntp
      service: name=ntpd state=restarted                        Handlers
```

Playbooks Structure

- Playbooks in Ansible are "collection of Plays".

- **Plays** are nothing but a "collection of attributes" and a "sequence of operations" to be performed on Managed Nodes.

- "Collection of Attributes" defines set of Managed Nodes, Communication Connection, Privilege escalations and different variable associated with playbooks.

- The "sequence of operations" is called "**Tasks**" in Ansible.

- Every Play is associated with one or more Ansible "Tasks".

Playbooks Structure

- The Tasks are created with the help of Ansible Modules to perform actions on Managed Nodes.

- Plays are associated with Managed Nodes and are called "hosts".

- Plays are associated with "Managed Nodes" to perform action on them.

- Playbooks are executed on Managed Nodes in sequence of contents written inside, so order of contents inside Playbook matters.

- Playbooks may have multiple plays associated with set of different "Managed Nodes".

Playbooks Structure

- **Start of a Play - Basic (Hosts and Users):**
- First part of Ansible Playbooks
- Provides the hosts/group of hosts to target
- Provides which user you want to perform the tasks
- For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks)

Examples:

- hosts: webservers

  remote_user: root

Playbooks Structure

Examples:

You can also login as you, and then become a user different than root:

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_user: postgres
```

You can also use other privilege escalation methods, like su:

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_method: su
```

Playbooks Structure

- **Tasks list:**

- 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. When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the rotation for the entire playbook. If things fail, simply correct the playbook file and rerun.

- The goal of each task is to execute a module, with very specific arguments

Playbooks Structure

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

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=started
```

Variables can be used in action lines. Suppose you defined a variable called vhost in the vars section, you could do this:

```
tasks:
  - name: create a virtual host file for {{ vhost }}
    template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

Playbooks Structure

- **Handlers:**

- In simple layman language Handlers are **Running Operations On Changes.**

- 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.

- The things listed in the notify section of a task are called handlers.

Demo
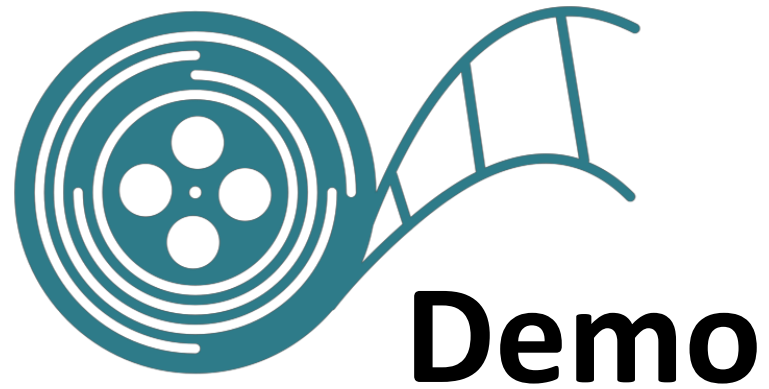
Playbooks Run

- Perform syntax check.

- Perform Dry Run Test.

- Perform Real time run.

- Check Results.

Playbooks Pre-execution

- Before running the Playbooks, lets explore some useful tips:
- Finding Modules and Attributes:
- ansible-doc –h
- ansible-doc –l
- ansible-doc <module-name>
- Ansible-doc –s <module-name>
- ansible-doc -v service

- To see what hosts would be affected by a playbook before you run it:
- ansible-playbook playbook.yml --list-hosts

- Command to see all FACTS:
- ansible all -m setup --ask-pass

Playbooks Syntax Checks

Syntax checks for Playbooks:

#ansible-playbook --**syntax-check** play1.yml

[root@ansible]#ansible-playbook --syntax-checkplay1.yml
ERROR! Syntax Error while loading YAML.

The error appears to have been in '/etc/ansible/play1.yml': line 3, column 8, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

0
- hosts: all
     ^ here

Playbooks Dry Run

Dry Runcheck for playbooks:

ansible-playbook <playbook-name> **--check**

[root@Ansible]#ansible-playbook play1.yml --check
SSH password:
PLAY [all] ***
TASK [setup] ***
ok: [192.168.74.51]
TASK [File Creation] **
changed: [192.168.74.51]
TASK [Directory Creation] **
changed: [192.168.74.51]
PLAY RECAP **
192.168.74.51   : ok=3   changed=2   unreachable=0   failed=0

Copyrights

Playbooks Real Time Run

Real time run for playbooks:

```
# ansible-playbook play1.yml
SSH password:
PLAY [all] ****************************************************************
TASK [setup] **************************************************************
ok: [192.168.74.51]
TASK [File Creation] ******************************************************
changed: [192.168.74.51]
TASK [Directory Creation] *************************************************
changed: [192.168.74.51]
PLAY RECAP ****************************************************************
192.168.74.51          : ok=3   changed=2   unreachable=0   failed=0
```

Step Run

Moving through task by task execution:

Special Case run Ansible playbooks with –step

Example:

```
[root@user20-master plays]# ansible-playbook ntp.yaml --step
PLAY [all] ************************************************************************************************************
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: n
Perform task: TASK: Second task (N)o/(y)es/(c)ontinue:
*******************************************************************
Perform task: TASK: Third task to start NTP Services (N)o/(y)es/(c)ontinue: y
TASK [Third task to start NTP Services] *********************************************************************************
ok: [user20-client]
PLAY RECAP
************************************************************************************************************
user20-client           : ok=1   changed=0   unreachable=0   failed=0
[root@user20-master plays]#
```

Lab5: Working with Playbook

- Let's do some Ansible Playbooks creations:

- Create two users – "your name" and "your name1" on next machine using playbook
- Create a group – "your name" on next machine

- Perform Syntax check
- Do Dry run of playbook
- Run the playbook

Lab5: Working with Playbook

- Creation of user:

[root@gagandeep]#cat user.yml

- hosts: all
  tasks:
   - name: Creating user
     user:
      name: gagandeep
      state: present
   - name: Creating another user
     user: name=gagandeep1 state=present shell=/bin/sh

Lab5: Working with Playbook

- Creation of group:

[root@ansible]#cat user.yml

- hosts: all

  tasks:

  - name: Group Creation

    group:

     name: group

     gid: 5555

  - name: Another Group Creation

    group: name=grp2 gid=5656

Lab5: Working with Playbook

- Playbook Run:

[root@ansible]#ansible-playbook --syntax-check user.yml

[root@ansible]#ansible-playbook --check user.yml

[root@ansible]#ansible-playbook user.yml

[root@ansible]#ansible-playbook user.yml --step

Lab6: Working with Playbook

- Create a Playbook for User and Group Creation with user name "usertest", shell bash, userid 6666 and pass the comments as "my first user". Group details will be name "grouptest" and group id 7777.

- Create a Playbook for files and directories:

  create a directory with root ownership; inside this directory, create one file with "test" with ownership of usertest (the user we have created in 1st example). Copy some content into the newly created file.

Handlers

- In simple layman language Handlers are **"Running Operations on Changes".**

- Handlers are list of tasks with unique names.

- "notify" is the keyword to trigger operations mentioned in Handlers.

- Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.

- If nothing changed in the tasks, Handlers will not run.

- Multiple "notify" with unique names are permitted.

- Handlers are always mentioned at last.

85

Handlers

Let's understand the importance of Handlers with below use cases:

- Playbook without Handlers

- Playbook with Handlers (with improper usage of handler)

- Playbook with Proper use of Handler

86

Lab - NTP

- Let's do a practical for NTP module including:

- Package
- Files
- Service

Lab - NTP

```
[root@Ansible]#cat ntp.yml
---
- hosts: all
  tasks:
    - name: NTP OS Package Installation
      package: name=ntp state=present
    - name: NTP File Configurations
      file: path=/etc/ntp.conf state=file
    - name: To start NTP services
      service: name=ntpd state=started enabled=yes
```

Lab - NTP

```
[root@gagandeep]#ansible-playbook ntp.yml --ask-pass
SSH password:

PLAY [all] ******************************************************************

TASK [setup] ****************************************************************
ok: [192.168.74.51]

TASK [NTP OS Package Installation] *****************************************
changed: [192.168.74.51]

TASK [NTP File Configurations] *********************************************
ok: [192.168.74.51]

TASK [To start NTP services] **********************************************
changed: [192.168.74.51]

PLAY RECAP ****************************************************************
192.168.74.51        : ok=4   changed=2   unreachable=0   failed=0
```

Copyrights

NTP without proper config

```
[root]#cat ntp.yml
---
- hosts: all
  tasks:
    - name: NTP OS Package Installation
      package: name=ntp state=present
    - name: NTP File Configurations
      file: path=/etc/ntp.conf state=file
      notify:
      - restart ntp
    - name: To start NTP services
      service: name=ntpd state=started enabled=yes
  handlers:
    - name: restart ntp
      service: name=ntpd state=restarted
```

Now run the Ansible Playbook and See nothing will change as no configuration is getting changed.

NTP with proper config

```
[root]#cat ntp.yml
---
- hosts: all
  tasks:
    - name: NTP OS Package Installation
      package: name=ntp state=present
    - name: NTP File Configurations
      copy: src=/etc/ntp.conf dest=/etc/ntp.conf
      notify:
      - restart ntp
   - name: To start NTP services
     service: name=ntpd state=started enabled=yes
  handlers:
   - name: restart ntp
     service: name=ntpd state=restarted
```

91

Lab 7: Working with handlers

- Create a Playbook for NTP configuration to run the operations on change of the configuration on /root/ntp.conf file change.
    - Task 1- to install ntp on next machine
    - Task 2 – to copy /root/ntp.conf file from master to client machine at /etc/ntp.conf
        - Add Notify section pointing to restart-ntp task of handler
    - Task 3: to start the service
    - Add handler restart-ntp to restart the ntp service

- Create one file /root/ntp.conf on master machine and run the playbook.

Ansible Variables

93

Ansible Variable overview

- Like any other Automation language Ansible also supports variables.

- In layman language variable is just an assigned value (string or characters).

- Variables can be used with any playbook, tasks, roles, templates and even with inventory files.

- There is a well-defined variable naming scheme in Ansible, it should always start with alphabet.

- Variables are first defined and then declared to be used in the Ansible.

- In simplest way variables are defined inside playbooks under "vars" section in the play.

Ansible Variable overview

- Variable Definition
- # Defining variable section in playbook using vars parameter
- vars:
-    user1: gagan

- Variable Declaration
- tasks:
-     - name: user creation
-       user: name={{ user1 }} state=present

95

Ansible Variable overview

You can define empty variables in playbook and later on provide variable values during runtime:

```
[root@user20-master plays]# cat variable-test1.yaml
---
- name: Starting a test play for Variable demo
  hosts: all
  vars:
    user1: gagan
    user2: ""
    uid1: 9876
    uid2: 8765
  tasks:
    - name: Creating a user1
      user: name={{user1}} uid={{uid1}} state=present
    - name: Creating user {{user2}}
      user: name={{user2}} uid={{uid2}} state=present
[root@user20-master plays]#
[root@user20-master plays]# ansible-playbook variable-test1.yaml -e user2="gagan"
```

96

Ansible Facts

- Facts are the special variables automatically discovered by Ansible.

- Dedicated "setup" module.

- Facts output is in JSON format.

- Facts provide information about hostname, kernel, network, OS and much more.

- The information from facts can be filtered using -a filter=<value> option to get specific information.

Ansible Facts

- [root@gagan-controller ~]# ansible [host] -m setup

- [root@gagan-controller ~]# ansible [host] -m setup | grep -i "ansible_"

- [root@gagan-controller ~]# ansible [host] -m setup -a filter="ansible_all_ipv4_addresses"

- [root@gagan-controller ~]# ansible [host] -m setup -a filter="ansible_date_time"

Ansible Facts

- Facts values are used as variables to create useful generic and dynamic playbooks.

- Facts outputs are JSON in format.

- Declared format in playbook is {{ <ansible_facts>}}

- Ansible automatically replace the Facts variable with values while Ansible run.

Ansible Facts

- First step is to find facts to be used in Playbook.

- [root@gagan-controller ~]# ansible centos-managed -m setup -a filter="ansible_hostname"

- [root@gagan-controller ~]# ansible "*" -m setup -a filter="ansible_eth0"

Note: "ansible_eth0"will be having detailed output in JSON format which you can even filter out for specific value. In our example we will filter out "interface address" as {{ ansible_eth0.ipv4.address }} which means check ansible_eth0 then go to ipv4 section and then return the address attribute value.

100

Ansible Facts

```
[root@]# cat factstest.yml
---
- name: This is my First Debug Play
  hosts: all
  tasks:
    - name: Testing Ansible Facts {{ ansible_hostname }}
# My task with outputs fetched from Facts
      debug: msg="Host {{ ansible_hostname }} is having IP address {{ ansible_eth0.ipv4.address}}"
...
```

Lab 8 : Working with Variables

- Install an httpd package in playbook. Package name should go as variable

- Configure hostname and IP address in webpage (using facts) and owner (through variable)

- Handler to be used for restarting httpd service

Lab 8 : Working with Variables

- Create a playbook for to install httpd(apache webserver)

```
---
- name: Apache webserver deployment
  hosts: client

  tasks:
  - name: Installing Apache packages
    yum: name=httpd state=present

  - name: Changing Listening port to 81
    lineinfile:
      path: /etc/httpd/conf/httpd.conf
      regexp: "Listen 80"
      line: Listen 81

  - name: Configuring Apache configuration files
    copy: dest=/var/www/html/index.html content="<h1>This is a demo for Apache Webserver!</h1>"

  - name: Starting Apache webservice
    service: name=httpd enabled=yes state=started
```

Ansible Verbose Run

Run Ansible in Verbose mode to show detailed execution logs

[root@user20-master plays]# ansible-playbook debug-register.yml **-vv**
ansible-playbook 2.4.2.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible-playbook
  python version = 2.7.5 (default, Oct 30 2018, 23:45:53) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
Using /etc/ansible/ansible.cfg as config file

[root@user20-master plays]# ansible-playbook debug-register.yml | wc -l
21
[root@user20-master plays]# ansible-playbook debug-register.yml -vvvvv | wc -l
233

Ansible Debug

Debug Module is used for debugging the outputs.

It prints statements during execution and can be useful for debugging variables or expressions without necessarily halting the playbook.

Parameter	Choices/Defaults	Comments
msg	Default: Hello world!	The customized message that is printed. If omitted, prints a generic message.
var		A variable name to debug. Mutually exclusive with the 'msg' option.
verbosity	Default: 0	A number that controls when the debug is run, if you set to 3 it will only run debug when -vvv or above

Ansible Debug

```
[root]# cat debug.yaml
---
- name: Debug Playbook
  hosts: all
  tasks:
    - debug:
        msg: First task to Print debug message

    - debug:
        msg: "System {{ inventory_hostname }} has uuid {{ ansible_product_uuid }}"
[root]#
```

Ansible Register

Ansible comes with a special variable name "Register" to capture command output run by Ansible run and save them into variable value.

It is mostly used with Debug Module.

```
[root]# cat debug-register.yaml
---
- name: Debug Playbook
  hosts: all
  tasks:
    - name: Talk package installation
      yum: name=telnet state=present
      register: output

    - name: Running debugger to show the output
      debug:
        var: output

[root]#
```

Register - Printing specific value

Specific value from Ansible Register output can also be printed:

```
---
- name: This is my First Debug Play
  hosts: all
  tasks:
    - name: install ntp
      yum: name=ntp state=installed
      register: ntp_out

    - name: printing complete output
      debug: var=ntp_out

    - name: printing specific reults for individual parameter
      debug: var=ntp_out.changed   // such parameter can be seen in output from previous task output
```

Lab 9 : Working with Registers

- Create a playbook with Debug and Register variable for below:
  - Install NTP package on next machine (you can use existing playbooks)
  - Capture the output of installation task into Register
  - Print the output with Debug Module
  - Run the playbook and check the output
  - Edit the debug level to 1 in playbook
  - Run the playbook and check the difference
  - Run the Playbook in verbose mode with –v and then check the difference.

Ansible Ignore Errors

By default if a task fails, Ansible stops running further playbook tasks and exits the program.

When you want to ignore failure for a task, which don't have any impact on pending tasks, you can ignore errors for that task.

```
[root]# cat ignore_error.yaml
---
- name: Ignore Error Playbook
  hosts: all
  tasks:
    - name: This will not be counted as failure
      command: /bin/date12
      ignore_errors: yes

    - name: Second task
      Service: name=ntpd state=restarted

[root]#
```

Working with Conditions

Conditional tasks can be performed using "When" Statement.

Ansible allows special statements with "AND" and "OR".

```
 ---
 - name: Conditional checks
   hosts: all
   tasks:
     - name: Installing web packages on all centos version 7 servers
       yum: name=httpd state=installed
       when: ansible_distribution == "CentOS" and ansible_distribution_major_version == "7"

   - name: "shut down CentOS 6 and Debian 7 systems"
     command: /sbin/shutdown -t now
     when: (ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_major_version'] == "6") or
           (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "7")
```

Working with Conditions

Working based on previous task output

```
    - name: Register a variable
      package: name=ntp state=installed
      register: ntp_out
      ignore_errors: true
     - name: debug
      debug:
        var: ntp_out
    - name: Use the variable in conditional statement
      shell: echo "motd contains the word ansible"
      when: ntp_out.rc == 0

    - name: Register a variable
      shell: cat /etc/motd
      register: motd_contents

    - name: Use the variable in conditional statement
      shell: echo "motd contains the word hi"
      when: motd_contents.stdout.find('hi') != -1
```

Lab 10 : Working with Registers

- Create a playbook with Debug and Register variable for below:
  - Install NTP package on next machine (you can use existing playbooks)
  - Create task 1 for running command "/dev/null"
  - Rest tasks of NTP and service run will stay as it is.
  - Run the playbook and observe playbook getting failed at task 1 itself
  - Put a condition for task one to ignore error if fails
  - Run the playbook and spot the difference
  - Add another task at last, to run only if tasks 1 fails or based on condition of previous tasks run (pass/fail)
  - Run the playbook and check the difference

Working with loops

Often, you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

Handling multiple similar tasks simultaneously can be done in Ansible to keep your code small and simple.

php   gcc   talk   vim   httpd

use **with_items or loop**

```
      - name: add several users
        user:
         name: "{{ item }}"
        state: present
        groups: "wheel"
        with_items:
          - testuser1
          - testuser2
```

Working with loops

e.g

```
  - file:
      src: '/tmp/{{ item.src }}'
      dest: '{{ item.dest }}'
    state: link
    with_items:
      - { src: 'x', dest: 'y' }
      - { src: 'z', dest: 'k' }
```

Use Loop instead of with_items

With_item is being replaced with loop now.

```
- name: add several users
  user:
    name: "{{ item }}"
  state: present
  groups: "wheel"
  loop:
      - testuser1
      - testuser2
```

Working with loop and condition

Re-iteration with condition:

```
- name: Run with items greater than 5
    command: echo {{ item }}
    loop: [ 0, 2, 4, 6, 8, 10 ]
    when: item > 5
```

Reading variables/item from files:

```
 - name: task to create multiple users
    user: name={{ item }} state=absent
    with_lines: cat /root/gagan/users.txt
```

Lab 11 : Working with Loops

Install below packages on CentOS version 7 machine only (condition of Centos 7):

php

gcc

talk

vim

httpd

Used conditions and Loops

Reboot Machines during a play

```
- name: Run with items greater than 5

  command: echo {{ item }}

  loop: [ 0, 2, 4, 6, 8, 10 ]

  when: item > 5


- name: Reboot a slow machine that might have lots of updates to apply

  reboot:

    reboot_timeout: 300


- name: second task for package

  package:

    name: telnet

    state: absent
```

Retries

We can retry failed task on a specific task:

- name: second task for package
  package:
    name: telnet
  state: absent
  retries: 2

Limiting playbook to limited servers can be done via:

ansible-playbook first.yml --**limit** client

Ansible Blocks

Blocks create logical groups of tasks. Blocks also offer ways to handle task errors, similar to exception handling in many programming languages.

```
tasks:
  - name: Install, configure, and start Apache
    block:
    - name: Install httpd and memcached
      yum:
        name:
        - httpd
        - memcached
        state: present

    - name: Apply the foo config template
        template:
        src: templates/src.j2
        dest: /etc/foo.conf

    - name: Start service bar and enable it
      service:
        name: bar
        state: started
        enabled: True
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
    ignore_errors: yes
```

Lab 12 : Working with Block

Modify previous loop & conditions playbook as below:

Instead of putting conditions at each tasks, put a condition at block level

Keep retries count to 3 for package installations

Best Practices

Logs

By Default, logging is disabled. Enable same for audit purpose and for storing the output for a later reuse.

Uncomment below logging configuration in ansible.cfg file:

log_path = /var/log/ansible.log

To disable tasks logs at master/client level, below parameters can be touched:

\# prevents logging of task data, off by default

\#no_log = True

\# prevents logging of tasks, but only on the targets, data is still logged on the master/controller

\#no_target_syslog = False

\#By default, logs goes as per syslog configuration on client machine.

Working with tags

Ansible tags are used to run only one or some specific tasks from a large playbook instead of running the whole playbook. We use "tags:" attribute to use Ansible tags.

It helps us to keep all the related tasks in one playbook and save the execution time as it provides the capability to run the specific task only

\# ansible-playbook first.yml –list-tags

\# ansible-playbook first.yml --tags <tag_name>

\# ansible-playbook first.yml --tags <tag_name1, tag-name2>

\# ansible-playbook first.yml --skip-tags <tag_name>

You can also use **--start-at-task** for running playbook from a specific task.

125

Working with tags - Example

```
---
- name: Ansible TAG example
  hosts: localhost
  tasks:
    - name: create multiple files from a location
      file:
       path: '/home/azureuser/{{ item }}'
       state: touch
      loop:
         - file1
         - file2
      tags:
         - create_file
    - name: delete multiple files from a locatoion
      file:
       path: '/home/azureuser/{{ item }}'
       state: absent
      loop:
         - file1
         - file2
      tags:
         - delete_file
```

Disable Gather_facts if not required

You can disable the facts in playbook to avoid network congestion due to Facts collection feature of Ansible before every node run:

```
---
- name: This is my First Debug Play
  hosts: all
  gather_facts: no

  tasks:
   - name: Testing Ansible Facts {{ ansible_hostname }}
     debug: msg="Host {{ ansible_hostname }} is having IP address {{ ansible_eth0.ipv4.address }}"
...
```

Note: This playbook will fail as we are using environment variables without fetching the facts

Import tasks from other playbooks

You can import tasks from other playbooks if required:

```
---
- hosts: gagan-client
  vars:
    user: gagandeepsingh
    uid: 10000
  tasks:
    - name: Creating {{user}}
      user: name={{user}} uid={{uid}} state=present
      register: outputs
    - name: Checking outputs
      debug: var=outputs
    - name: Import tasks from
      import_tasks: ./ntp.yml
...
```

Note: import_tasks playbook should have only tasks included.

You can also import complete playbook with import_playbook as new play.

Forks

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks. If you want to change this default behavior, you can use a different strategy plugin, change the number of forks, or apply one of several play-level keywords like serial.

Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing

--forks with commands

or editing the default in the configuration file.

Ansible.cfg:

Forks=10

The default is a very conservative five (5) forks, though if you have a lot of RAM + Good NW bandwidth, you can easily set this to a higher value for increased parallelism.

You can also set fork during execution with –f option.

12 9

Strategy

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks.

#ansible-doc -t strategy -l

debug   Executes tasks in interactive debug session

host_pinned Executes tasks on each host without interruption

linear   Executes tasks in a linear fashion

free   Executes tasks without waiting for all hosts

#

Linear: The default behavior described above is the **linear** strategy.

Free: Ansible will not wait for other hosts to finish the current task before queuing more tasks for other hosts. All hosts are still attempted for the current task, but it prevents blocking new tasks for hosts that have already finished.

You can change behavior in ansible.cfg file:

#strategy = free

Or in playbook too:

- hosts: all

  strategy: free

To do the practical, better have 8-10 tasks with 4-5 managed hosts and use yum lock on next machine.

13 0

Rolling Update Batch Size

By default, Ansible will try to manage all of the machines referenced in a play in parallel. For a rolling update use case, you can define how many hosts Ansible should manage at a single time by using the serial keyword:

```
---
- name: test play
  hosts: webservers
  serial: 2
  gather_facts: False

  tasks:
   - name: task one
     command: hostname
   - name: task two
     command: hostname
```

131

Rolling Update Batch Size

In the previous page example, if we had 4 hosts in the group 'webservers', 2 would complete the play completely before moving on to the next 2 hosts:

```
PLAY [webservers] *************************************

TASK [task one] *************************************
changed: [web2]
changed: [web1]

TASK [task two] *************************************
changed: [web1]
changed: [web2]

PLAY [webservers] *************************************

TASK [task one] *************************************
changed: [web3]
changed: [web4]

TASK [task two] *************************************
changed: [web3]
changed: [web4]

PLAY RECAP *************************************
web1    : ok=2   changed=2   unreachable=0   failed=0
web2    : ok=2   changed=2   unreachable=0   failed=0
web3    : ok=2   changed=2   unreachable=0   failed=0
web4    : ok=2   changed=2   unreachable=0   failed=0
```

Rolling Update Batch Size

The serial keyword can also be specified as a **percentage**, which will be applied to the total number of hosts in a play, in order to determine the number of hosts per pass:

```
---
- name: test play
  hosts: webservers
  serial: "30%"
```

Or can be defined as a **list**:

```
---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10
```

133

Max FailurePercentage

By default, Ansible will continue executing actions as long as there are hosts in the batch that have not yet failed. The batch size for a play is determined by the serial parameter. If serial is not set, then batch size is all the hosts specified in the hosts: field.

In some situations, such as with the rolling updates described above, it may be desirable to abort the play when a certain threshold of failures have been reached. To achieve this, you can set a maximum failure percentage on a play as follows:

- hosts: webservers
  max_fail_percentage: 30
  serial: 10

Lab 13 : Max-fail-percentage

- Install an httpd package in playbook

- Configure hostname and IP address in webpage (using facts) and owner (through variable)

- Handler to be used for restarting httpd service

- Use Register and Debug to print the output of installation of httpd package

- Use serial to run it on one host at a time

- Use Max-fail-percentage to 30%

- Encrypt inventory file using vault

- Run playbook and verify the outcome in your browser

- Disable gather facts and then check the runtime output

Lab 14 : Software Orchestration

Create a playbook for Tomcat end to end installation and extraction:

- Pre-requisities: java-1.8.0-openjdk package should be present
          wget should be present
          /opt/tomcat directory should exists

- Now download tomcat tar file from http://mirror.23media.de/apache/tomcat/tomcat-8/v8.5.29/bin/apache-tomcat-8.5.29.tar.gz and copy it under /opt/tomcat directory

- Extract the downloaded tomcat file
- Change the permissions of startup.sh shutdown.sh catalina.sh under /opt/tomcat/apache-tomcat-8.5.29/bin/ directory

- Now download the sample application from https://tomcat.apache.org/tomcat-8.0-doc/appdev/sample/sample.war and place it under /opt/tomcat/apache-tomcat-8.5.29/webapps/

- Wait for 60 secs

- Start the tomcat services /opt/tomcat/apache-tomcat-8.5.29/bin/catalina.sh start

- Verify your ans by checking the browser at client on port 8080

Lab 14 : Software Orchestration

```
---
- name: Tomcat web server deployment
  hosts: client

  tasks:

  - name: Pre-req's for Tomcat to ensure no sample war is present
    file: path=/opt/tomcat/apache-tomcat-8.5.29/webapps/sample.war state=absent

  - name: Install prerequisties
    yum: name=java-1.8.0-openjdk state=present

  - name: Installing wget
    yum: name=wget state=present

  - name: Creating Directory structure
    file: name=/opt/tomcat state=directory

  - name: Downloading Tomcat Binaries
    get_url: url=http://mirror.23media.de/apache/tomcat/tomcat-8/v8.5.29/bin/apache-tomcat-8.5.29.tar.gz dest=/opt/tomcat

  - name: Extracting Tomcat
    shell: cd /opt/tomcat; tar -xvzf apache-tomcat-8.5.29.tar.gz; cd apache-tomcat-8.5.29/bin/; chmod 777 startup.sh shutdown.sh catalina.sh

  - name: War deployment
    shell: cd /opt/tomcat/apache-tomcat-8.5.29/webapps/; wget https://tomcat.apache.org/tomcat-8.0-doc/appdev/sample/sample.war; cd /opt/tomcat/apache-tomcat-8.5.29/bin; sleep 60

  - name: Starting Tomcat
    command: /opt/tomcat/apache-tomcat-8.5.29/bin/catalina.sh start
```

Ansible Vault

Ansible Vault

Ansible Vault is used to store confidential data. It can encrypt or decrypt the playbooks/hosts files to protect sensitive information.

Dedicated CLI utility "ansible-vault".

Pre-requisite is to have cryptography package.

Ansible-vault positional arguments:

create	Create new vault encrypted file
decrypt	Decrypt vault encrypted file
edit	Edit vault encrypted file
view	View vault encrypted file
encrypt	Encrypt YAML file
encrypt_string	Encrypt a string
rekey	Re-key a vault encrypted file

Ansible Vault

[root@ansible-controller playbooks]# ansible-vault create vault.yml

New Vault password:

Confirm New Vault password:

[root@ansible-controller playbooks]# ansible-vault encrypt existingplaybook.yaml

New Vault password:

[root@ansible-controller playbooks]# ansible-vault decrypt existingplaybook.yaml

Enter Vault password:

[root@ansible-controller playbooks]# ansible-vault edit vault.yml

Enter Vault password:

Ansible Vault

[root@ansible-controller playbooks]# ansible-vault view vault.yml

Vault password:

- name: Installing telnet with Vault protected file

  hosts: all

  tasks:

   - name: Telnet Package

     yum: name=telnet state=present

Ansible Vault

[root@ansible-controller playbooks]# ansible-playbook --check vault.yml
ERROR! Decryption failed on /var/tmp/playbooks/ vault.yml

[root@ansible-controller playbooks]# ansible-playbook --check vault.yml --ask-vault-pass
Vault password:
PLAY [Installing telnet with Vault protected file]
****

TASK [Gathering Facts]
***ok:
[ansible-managed]
TASK [Telnet Package]
**changed:
[ansible-managed]
PLAY RECAP
**ansible-
managed   : ok=2   changed=1   unreachable=0   failed=0

Multiple Password Management

ansible-vault encrypt --vault-id **password1**@prompt /etc/ansible/hosts
ansible-vault encrypt --vault-id **password2**@prompt loop.yml

[root@gagan-ansible gagan]# ansible-playbook loop.yml --vault-id password1@prompt --vault-id
password2@prompt
Vault password (**password1**):
Vault password (**password2**):

PLAY [first play]
****
*

Lab 15 : Working with Vaults

Working with vaults:

Encrypt your playbook with ansible-vault command
Run playbook and spot the output
Provide password with --ask-vault-pass and then spot the outcome
Encrypt your inventory file as well with different password and vault-id
Now run playbook with providing vault-id and password for same.

Ansible Jinja2 Template

Ansible Jinja2

Jinja2 is a powerful and easy to use python-based templating engine that comes in handy in an IT environment with multiple servers where configurations vary every other time. Creating static configuration files for each of these nodes is tedious and may not be a viable option since it will consume more time and energy. And this is where templating comes in. Template files bear the **.j2** extension,

Template architecture

A Jinja2 template file is a text file that contains variables that get evaluated and replaced by actual values upon runtime or code execution. In a Jinja2 template file, you will find the following tags:

{{ }} : These double curly braces are the widely used tags in a template file and they are used for embedding variables and ultimately printing their value during code execution. For example, a simple syntax using the double curly braces is as shown: The {{ webserver }} is running on {{ nginx-version }}

{% %} : These are mostly used for control statements such as loops and if-else statements.

{# #} : These denote comments that describe a task.

Ansible Jinja2 – Example 1

```
---
- hosts: localhost
  vars:
    version_number: '2.4'
    server: 'centos'


  tasks:
   - name: jinja2 example
     template:
       src: example_template.j2
       dest: ~/jinja_result.txt
```

Create src file example_template.j2

Hey guys! Apache webserver {{ version_number }} is running on {{ server }} Enjoy!

Ansible Jinja2 – Example 2

```
---
- hosts: localhost
  tasks:
   - name: push new index.html file
     template:
        src: index.html.j2
        dest: /var/www/html/index.html
        mode: 0775
```

Create index.html.j2
```
<html>
    <center><h1> The Apache webserver is running on {{ ansible_hostname }} </h1>
    </center>
</html>
```

Ansible Jinja2 – Lab

Practice the above provided jinja 2 examples

Ansible Roles

Ansible Roles

Ansible Roles are the way to keep Ansible Playbooks well organized.

Role Directory structure allow to load tasks, handlers, variables, static files, templates.

Can be shared with external world with ease.

Best way to orchestrate application deployments.

Default configuration directory /etc/ansible/roles.

Dedicated Role creation utility: ansible-galaxy.

Ansible Roles

Lets create our first role with ansible-galaxy utility, execute this command in dedicated directory only.

ansible-galaxy init gaganrole.

Go inside the gaganrole directory and walk through the directory structure

Ansible Roles Structure

Default – This section is use to hold default variables by a role.

Handlers – This section is used to hold function associated with notify to trigger service-related operations.

Meta – This section is used to hold information like author of a role, description, company name, license and more importantly the **dependencies**.

Tasks – This is the section which will contain the actual code.

Tests – Test cases will be kept here.

Vars – This section is also for variables but with higher precedence than defaults section.

files - contains files which can be deployed via this role.

Ansible Roles - Demo

Lets create a simple role to understand how actual role works.

Create a Role Structure.

Define the role in the playbook directory, test and execute the playbook.

Deployment of Ansible Roles

[root@ansible-controller role]# cat defaults/main.yml

---

defaults variable file for file for gaganrole

user:

   gagandeep

uid:

   7654

Note: make sure, not to write "vars" on top.

Deployment of Ansible Roles

[root@ansible-controller role]# cat tasks/main.yml

---

tasks file for gaganrole

- name: Creating {{user}}

  user: name={{user}} uid={{uid}} state=present

  register: outputs

- name: Checking outputs

  debug: var=outputs

Note: make sure, not to write "tasks" on top.

Deployment of Ansible Roles

[root@ansible-controller playbooks]# cat gaganroletest.yml

- name: Testing role

  hosts: all

Define roles here

  roles:

   - gaganrole

Deployment of Ansible Roles

Perform Syntax checks on playbook

Dry run test

Real time run on the playbook containing role

Lab 16 : Working with Roles

Create a role for NTP Service with below

Create tasks for below:

User creation [yourname] with UID 1020

Package Installation

File Copying

Service Start

Use Variables for Username and UID

Handlers should be there to restart the NTP Service

Ansible Roles Import tasks

You can Import other tasks/playbooks in the main.yml file to ease your code and enhance the re-usability of code:

roles/example/tasks/main.yml
- name: added in 2.4, previously you used 'include'
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'
roles/example/tasks/redhat.yml
- yum:
    name: "httpd"
    state: present
roles/example/tasks/debian.yml
- apt:
    name: "apache2"
    state: present

Lab 17 : Working with import task

Use the existing ntp role and modify it for below:

Add a test tasks.yaml for below use case:

- o Tests tasks to be performed for checking "ntp –q" output.

- o redirect output on screen.

- o Ignore the failure of tests tasks.

Import the task inside main.yaml file

Ansible Nested Roles

For application orchestration we need to perform multiple deployments in sequence on managed nodes, Include and Dependency Management in roles allow us to achieve this in seconds.

Defining Roles dependency inside meta/main.yml

Define dependent role under dependencies section

Practical cases:
- Role with single main.yml file
- Role with multiple .yml file with include keyword
- Orchestrate role (i.e multiple roles with dependency)

Ansible Nested Roles

```
---
dependencies:
  - role: common
    vars:
      some_parameter: 3
  - role: apache
    vars:
    apache_port: 80
  - role: postgres
    vars:
      dbname: blarg
      other_parameter: 12
```

Lab 18 : Nested Roles

Create a role for Http Service with below

Create tasks for below:

      Package Installation

      File Copying

      Service Start

Require nested role NTProle (created during previous labs).

Ansible Galaxy

Ansible Galaxy

Ansible Galaxy is an official community site for users to share and use Ansible roles.

GUI: https://galaxy.ansible.com.

Anyone can search, share, download and use available modules.

Dedicated CLI utility "ansible-galaxy".

Search the roles, install them, modify them and use as per requirements.

Ansible Galaxy

ansible-galaxy search docker

ansible-galaxy search --author=admingagan

ansible-galaxy install admingagan.ansible_docker_role

ls -lrt /etc/ansible/roles/admingagan.ansible_docker_role

Amend the role inside a playbook

Run syntax check, followed by a Dry Run test

Run real time run

Ansible an Orchestrator

Ansible Orchestration Features

Ansible can be used as an orchestrator which can provision a server for you, Install the software on top of it and do the required configuration, without manual efforts.

You can not only provision a VM on-prem, you can provision same in cloud as well.

Ansible 2.7.4 supports AWS, Azure, GCP, Openstack and Vcenter through its modules.

Approx. 3300 Modules are available in Ansible 2.9 version

Redhat Openstack, Openshift and may such big softwares are getting installed through Ansible scripts. What can be a big example than this ☺

Ansible Orchestration Features

```
[root@ip-172-31-26-124 projects]# ansible --version
ansible 2.7.4
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible
  python version = 2.7.5 (default, Oct 30 2018, 23:45:53) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]

[root@ip-172-31-26-124 projects]# ansible-doc -l | wc -l
2080

[root@ip-172-31-26-124 projects]# ansible-doc -l | egrep -i "start or stop an instance in ec2|azure_rm_resource
|gcp_compute_instance |vmware_guest |os_server "
azure_rm_resource                       Create any Azure resource.
ec2                           create, terminate, start or stop an instance in ec2
gcp_compute_instance                    Creates a GCP Instance
os_server                       Create/Delete Compute Instances from OpenStack
vmware_guest                       Manages virtual machines in vCenter
[root@ip-172-31-26-124 projects]#
```

Ansible Orchestration – EC2

```
[root@client tmp]# cat newinstance.yml
---
- name: Creating a play for tower test EC2 instance creation
  hosts: localhost
  vars:
    ami: ami-0d69571bc7b7ca095
    subnet: subnet-bed02af3
    sg: kubernetes
  key:  hdbc
  tasks:
    - name: Provision an Instance directed by Gagandeep Singh
      ec2:
        key_name: "{{ key }}"
        region: us-east-2
        instance_type: t2.micro
        image: "{{ ami }}"
        wait: yes
        group: "{{ sg }}"
        count: 1
        vpc_subnet_id: "{{ subnet }}"
        assign_public_ip: yes
        instance_tags: '{"Name":"Ansible-gagan-ec2"}'
```
Note: Need boto to be installed and AWS AK/SK to be set in environment

Ansible For Windows

Ansible for windows

Working over WINRM port – 5985 and 5986

With Ansible's native Windows support, you can, out of the box:

- Gather facts on Windows hosts
- Install and uninstall MSIs
- Enable and disable Windows Features
- Start, stop, and manage Windows services
- Create and manage local users and groups
- Manage Windows packages via the [Chocolatey package manager](#)
- Manage and install Windows updates
- Fetch files from remote sites
- Push and execute any PowerShell scripts you write

Ansible for windows

Pre-requisites:

For Ansible Master:
Winrm module required. Install same via
Pip install pywinrm
pip install pywinrm --ignore-installed

For Client machines:

https://docs.ansible.com/ansible/devel/user_guide/windows_setup.html#common-winrm-issues

Ansible for windows

Windows Host:
https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html

Upgrading PowerShell and .NET Framework
=====

```
$url = "https://raw.githubusercontent.com/jborean93/ansible-windows/master/scripts/Upgrade-PowerShell.ps1"
$file = "$env:temp\Upgrade-PowerShell.ps1"
$username = "Administrator"
$password = "Password"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force

# Version can be 3.0, 4.0 or 5.1
&$file -Version 5.1 -Username $username -Password $password -Verbose
```
=======
WinRM Setup:

```
$url = "https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1"
$file = "$env:temp\ConfigureRemotingForAnsible.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)

powershell.exe -ExecutionPolicy ByPass -File $file
```

========
Check for Listener:

```
winrm enumerate winrm/config/Listener
```

175

Ansible for windows

```
[root@gagan-master ~]# cat /root/inventory
gagan-client

[win]
3.21.37.81

[win:vars]
ansible_user=Administrator
ansible_password="p"
ansible_connection=winrm
ansible_winrm_transport=basic
ansible_winrm_server_cert_validation=ignore

[root@gagan-master ~]#
```

Ansible for windows

```
---
- name: first play to install package
  hosts: all
  gather_facts: yes
  tasks:
   - name: block run
     block:
       - name: installaing telnet package
         yum: name=telnet state=installed
         register: result
       - name: print output
         debug: var=result
     when: ansible_distribution == "CentOS"
     ignore_errors: true
   - name: windows run
     block:
       - name: Install IIS (Web-Server and Web-Common-Http)
         win_feature:
           name:
             - Web-Server
             - Web-Common-Http
           state: present
       - name: default-website-index
         win_copy:
           dest: "C:\\inetpub\\wwwroot\\index.html"
           content: "Welcome to the windows instance. This server is configured through Ansible!"
     when: ansible_os_family == "Windows"
```

Ansible For Network

How things are different?

Unlike most Ansible modules, network modules do not run on the managed nodes.

Because the majority of network devices can not run Python, the Ansible network modules are executed on the Ansible control node, where ansible or ansible-playbook runs.

Network modules also use the control node as a destination for backup files, for those modules that offer a backup option.

Communication Protocol:

Value of ansible_connection	Protocol	Requires	Persistent?
network_cli	CLI over SSH	network_os setting	yes
netconf	XML over SSH	network_os setting	yes
httpapi	API over HTTP/HTTPS	network_os setting	yes
local	depends on provider	provider setting	no

How things are different?

As of Ansible 2.9, you can use the top-level Ansible parameter become: yes with become_method: enable to run a task, play, or playbook with escalated privileges on any network platform that supports privilege escalation. You must use either connection: network_cli or connection: httpapi with become: yes with become_method: enable.

```
ansible_connection: network_cli
ansible_network_os: ios
ansible_become: yes
ansible_become_method: enable
```

Ansible config - AWS VPC

```
- name: "Create and associate production DMZ network ACL with DMZ subnets"
  hosts: localhost
  tasks:
    - name: ec2 nacl
      ec2_vpc_nacl:
        vpc_id: "vpc-0dcf6b8288c282a6c"
#       nacl_id: "acl-0ae45dca0642e8786"
        name: ansible-nacl
        region: us-east-1
        subnets: ['subnet-0e0f1e4ce3a801440']
        tags:
          Description: production DMZ
        ingress:
          - [100, 'tcp', 'allow', '0.0.0.0/0', null, null, 22, 22]
          - [200, 'tcp', 'allow', '0.0.0.0/0', null, null, 80, 80]
          - [300, 'icmp', 'deny', '0.0.0.0/0', 0, 8]
          - [301, 'tcp', 'allow', '0.0.0.0/0', null, null, 3389, 3390]
        egress:
          - [100, 'all', 'allow', '0.0.0.0/0', null, null, null, null]
        state: 'present'
```

Note: Need boto to be installed on Ansible master and AWS AK/SK to be set in environment

Ansible For Cisco IOS

Ansible – ios modules

Create a user and password on ios for ansible user and add the entries in ansible inventory file:

Router:

conf t
username password
username privilege 15
line vty 0 4
transport input all
login local
ip domain-name .com
crypto key generate rsa
ip ssh version 2
transport input ssh
exit
wr

/etc/ansible/hosts:

[ios]
myrouter
#router2
#router3

[ios:vars]
ansible_user=gagan
ansible_password=gagan
ansible_network_os=ios
ansible_connection=network_cli
# If using an enable password add these
#ansible_become_pass=foo
#ansible_become=yes
#ansible_become_method=enable

Ansible – ios adhoc

Verify ssh connection manually and then run ansible adhoc commands:

```
ansible "*" -m raw -a "show version"
ansible "*" -m raw -a "show run"  | grep -i username
ansible "*" -m raw -a "show run"  | grep -i 'changed\|username'
ansible "*" -m raw -a "show mac address-table"
ansible "*" -m raw -a "show arp"


ansible "*" -c local -m ios_command -a "commands='show version'"
ansible "*" -c local -m ios_ping -a "dest='192.168.122.76'"
ansible "*" -c local -m ios_user -a "name=test nopassword=yes state=present"
ansible "*" -c local -m ios_command -a "commands='show version | in uptime'"  | grep uptime


ansible "*" -c local -m ios_banner -a "banner=login text='hello all'"
ansible "*" -c local -m ios_banner -a "banner=exec text='hello all again'"
ansible ios -c local -m ios_facts
```

Ansible ios - Playbook

Backup: Take backup and verify backup with router_config_date name under backup directory in your present working directory of playbook:

```
---
- name: Playbook to take backup of routers
  hosts: ios
  gather_facts: no


  tasks:
  - name: BACK UP CONFIG
    ios_config:
      backup: yes

# Backup will go under your pwd/backup directory with routername_config_date filename
```

Ansible ios_banner

Banner: You can setup both login and motd banner via direct text or files:

```
---
- hosts: ios
  gather_facts: no

  tasks:

  - name: CONFIGURE LOGIN BANNER FROM FILE
    ios_banner:
      banner: login
      text: "{{ lookup('file', './raw_banner.cfg') }}"
      state: present

  - name: CONFIGURE MOTD BANNER DIRECTLY
    ios_banner:
      banner: motd
      text: |
        ********************************************************************
        *                         TEST WARNING
        *       This is test MOTD banner
        *******************************************************************#
      state: present
```

186

Ansible IOS - Logging

Logging Server config: You can setup Logging server via below playbook

```
---
- hosts: ios
  gather_facts: no
  vars:
  log_servers:
      - logging 8.8.8.8
      - logging 9.9.9.9
  tasks:
  - name: set logging commands
    ios_config:
      commands: "{{ item }}"
    loop: "{{ log_servers }}"
    register: set_logging
  - name: get the new log server configs
    ios_command:
      commands:
        - show running-config full | include logging
    register: log1
  - debug: var=log1.stdout_lines
```

Ansible Tower

Ansible Tower Technical Introduction

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

In simple layman terms, Ansible Tower is a UI wrapper for Ansible Core Engine.

Ansible Tower is very Powerful way to deploy applications, provision and manage infrastructure .

Tower setup and learning curve is simple and effective.

Ansible Tower only support seup on RHEL 7, CentOS 7, or Ubuntu 14.04 LTS or 16.04 LTS, and requires Ansible Core 2.1.X or later.

Fully support by Redhat

Ansible Tower Technical Introduction

Ansible Engine vs Ansible Tower

Ansible Engine

- Simplified dashboard and stream line interface
- Free (Open-source)
- CLI based
- Need deep understanding
- Community Support Only
- All managed using CLI
- No RBAC (need to manage all with OS skills)

Ansible Tower

- Enterprise Edition(Paid)
- Simple Dashboard with robust UI
- Basic Understanding is enough to manage Ansible Engine
- Enterprise Vendor Support
- No CLI (though CLI-Tower Client is available for CLI lovers)
- Robust RBAC (Role Based Access) for permissions
- Job Scheduling
- Graphical Inventory Management
- REST API
- Multiplay book workflow
- Real time job status update
- External logging Integrations

Tower Functional Architecture
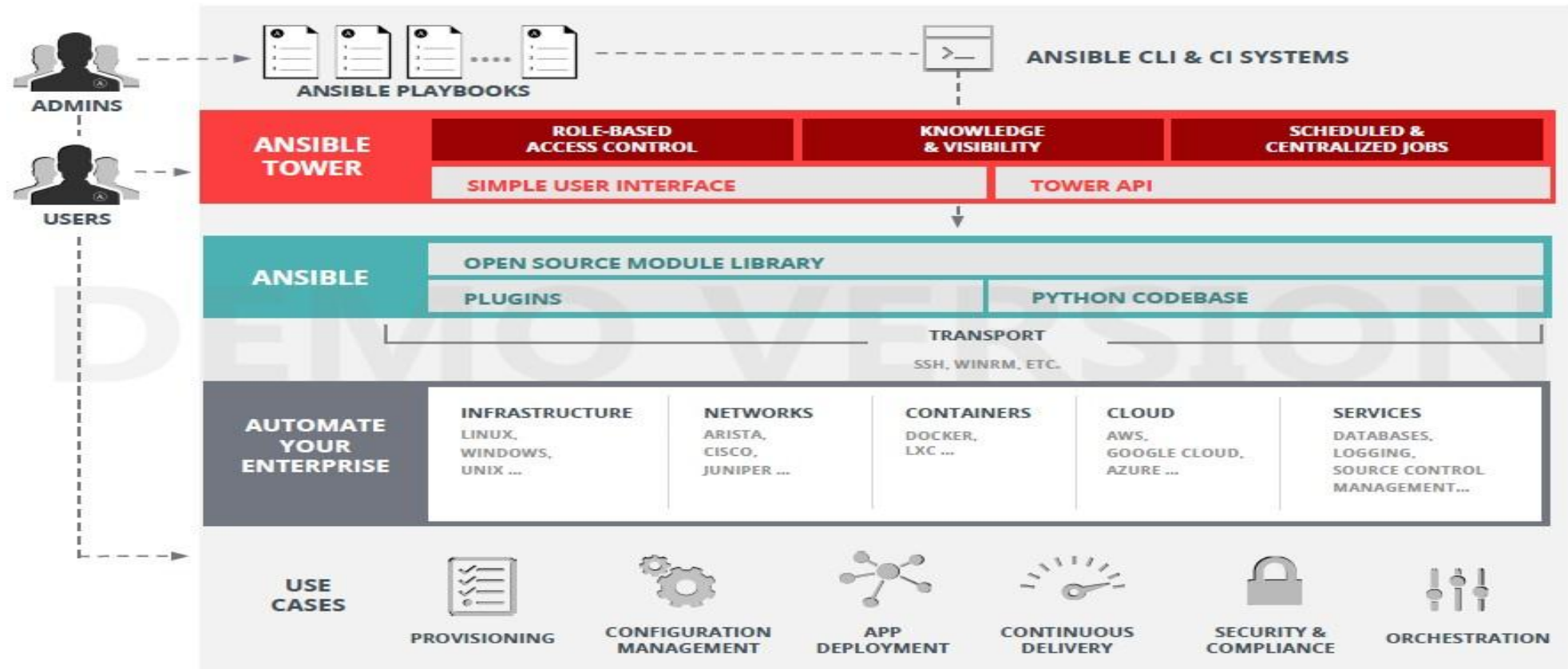
Ansible Tower

Ansible Tower (AWX for opensource) is a Web-based solution that makes Ansible even more easy to use for IT teams of all kinds. It's designed to be the hub for all of your automation tasks.

More information is available on GitHub: https://www.ansible.com/tower-trial

-Inventory can be graphically managed or synced with a wide variety of cloud sources.

-Command line tools are available for easy integration with Jenkins as well.

-Provisioning callbacks provide great support for autoscaling topologies.

Ansible Tower

-Amazing browsable REST API.

-Tower allows you to control access to who can access what, even allowing sharing of SSH credentials without someone being able to transfer those credentials.

-Tower is free for usage for up to 10 nodes, and comes bundled with amazing support from Ansible, Inc. As you would expect, Tower is installed using Ansible playbooks!

Ansible Tower - Prerequisites

Supported Operating Systems

- – Red Hat Enterprise Linux 6 64-bit
- – Red Hat Enterprise Linux 7 64-bit
- – CentOS 6 64-bit
- – CentOS 7 64-bit
- – Ubuntu 12.04 LTS 64-bit
- – Ubuntu 14.04 LTS 64-bit

Memory

- - 2 GB RAM minimum (4+ GB RAM recommended)
- – 2 GB RAM (minimum and recommended for Vagrant trial installations)
- – 4 GB RAM is recommended per 100 forks

Ansible Tower - Prerequisites

Harddrive Space

   – 20 GB of dedicated hard disk space
   – 10 GB of the 20 GB requirement must be dedicated to /var/, where Tower stores its files and working directories
   (dedicating less space will cause the installation to fail)

64-bit support required (kernel and runtime)

Package Download and Extraction

Inventory file changes (for passwords)

Note: Redis passwords must not contain spaces or any of the following characters: @, :, -, \, /, #For example, Passw0rd is acceptable, but P@ssword is not.

Why Ansible Tower

User friendly Ansible Tower Dashboard

Real time job (Playbooks) status updates

Multi-Playbook workflow for robust CI/CD pipeline

All automation jobs get logged, who ran what

Scale capacity with Ansible Tower clusters

Scheduling of the jobs

Manage and track inventories

Remote command execution

Installation of Tower

Download Ansible Tower "ansible-tower-setup-latest.tar.gz" file from RedHat/Ansible site.

Practical's of Tower Installation.

Installation of Tower

wget https://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.tar.gz

tar -xvf ansible-tower-setup-latest.tar.gz

Set the passwords in inventory file, extracted during tower zip file.

Once done run the setup.sh for the tower setup.

Login to the console using Public IP

As soon as you login, Tower ask for License. You have to request for a License till 10 free nodes or incase you have existing license you can use that. Once done accept the agreement and submit.

Request for a License (for free trail version or Enterprise for more than 10 nodes), you will get an email with License and details. Browse the license file, accept the agreement and Submit.
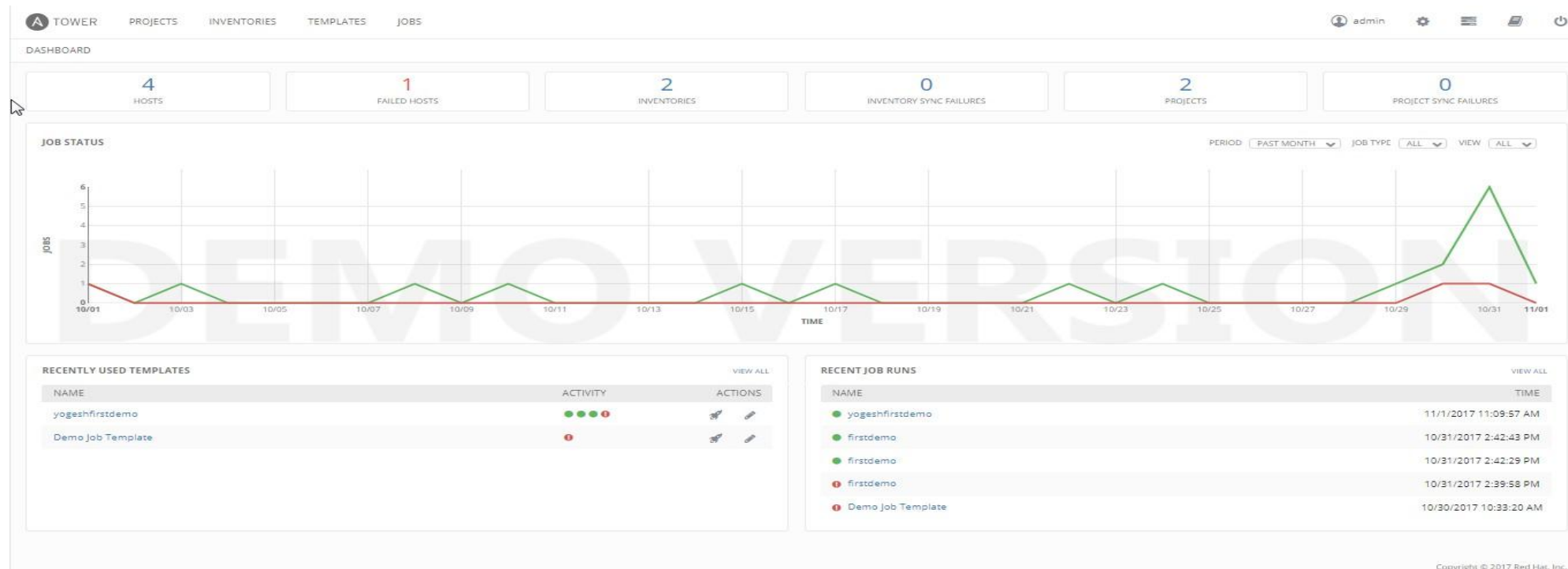
Note: Check the logs in the OS log file (/var/log/messages)

Note: Set password in inventory file.   Redis passwords must not contain spaces or any of the following characters: @, :, -, \, /, #For example, Passw0rd is acceptable, but P@ssword is not.

Ansible Tower Key Interface

Simplified dashboard and streamline interface.

Ansible Tower Setup

Remote command executions for ad-hoc requests.

Create "machine" credentials

Create a new project, do to so you need a new project directory at /var/lib/aws/projects to have your playbooks.

Create a new inventory now

Add new host inside newly created inventory

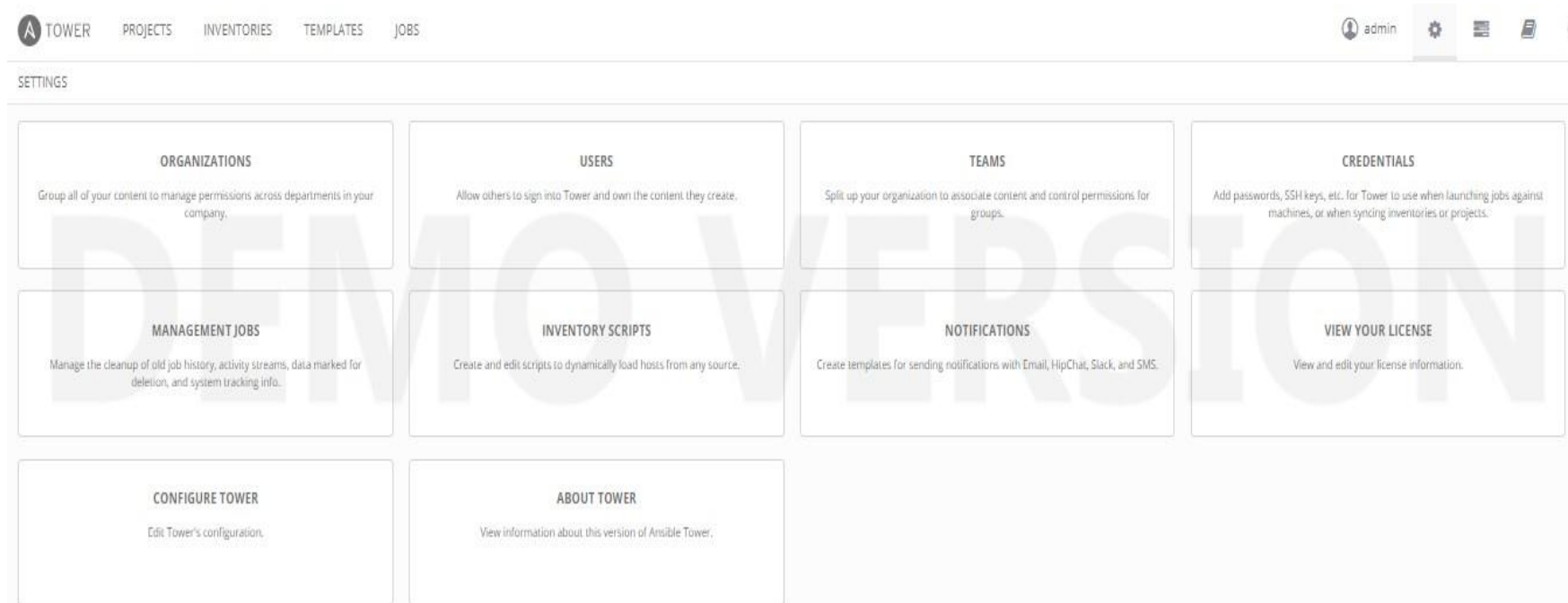Now comes the real challenge to give the permissions to the awx user to run jobs via other users

Try running a adhoc request (you will get an error for local host)

Add the public key in the local host inside authorized keys to run the jobs.
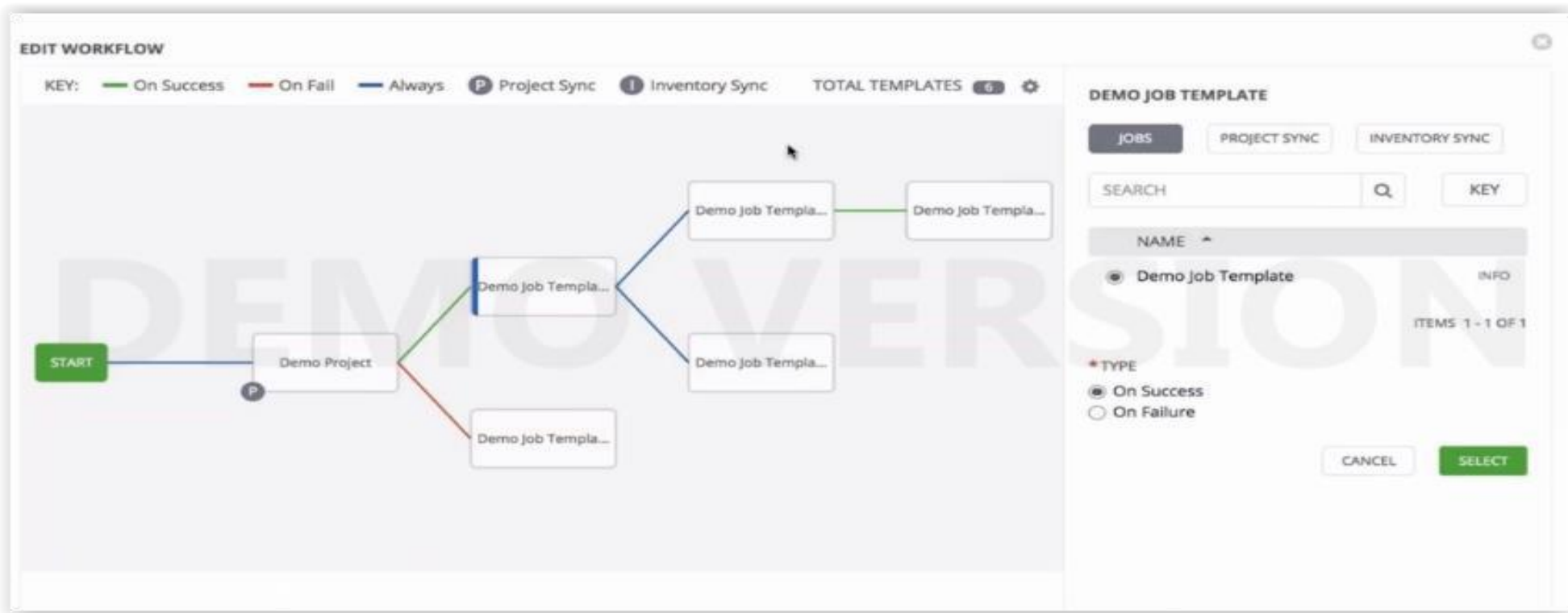
Try running the jobs it will be a success now

Ansible Tower RBAC
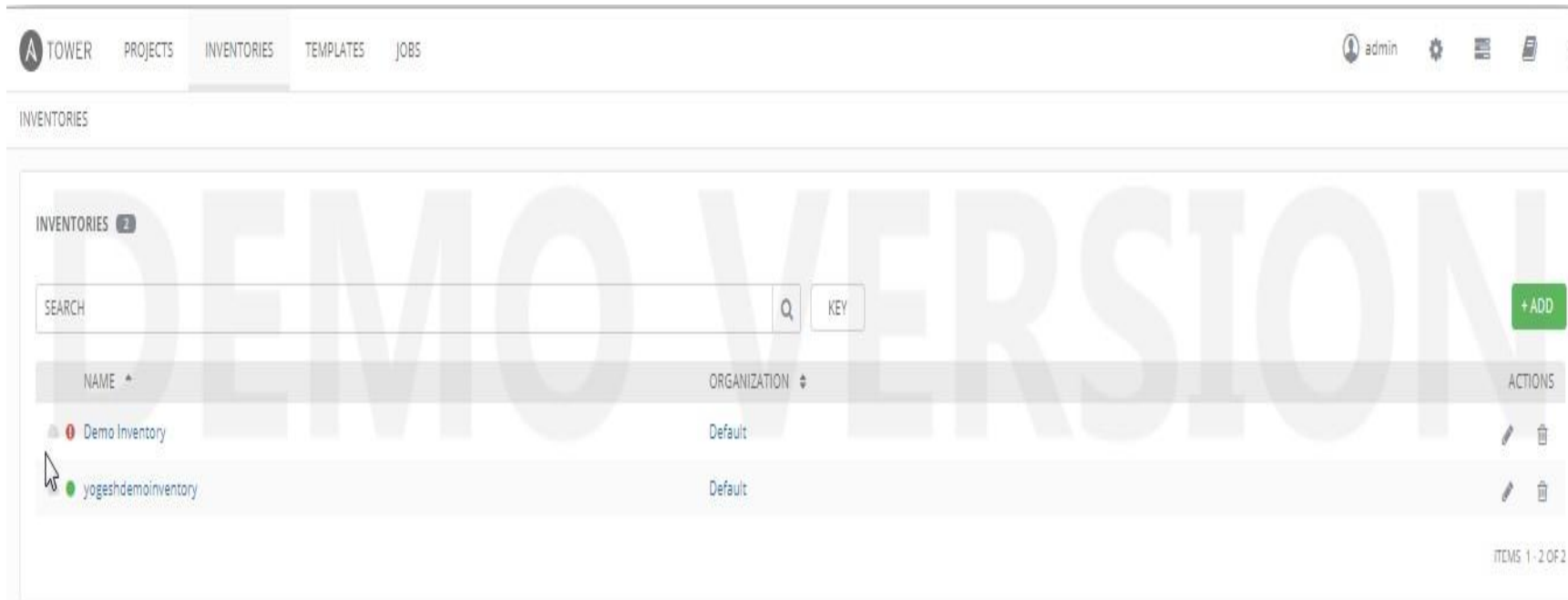
RBAC "role based access control".

Ansible Tower Workflow

Multi Playbook workflow for orchestration.

Ansible Tower Inventory

Inventory Management.

Ansible Tower Key Features

- Simplified dashboard and stream line interface
  using UI option of Tower

- Remote Command Execution
  ad hoc requests for faster execution on complete Ansible inventory

- Integrated notifications
  such as email, messages, hip chat etc

- RBAC "role based access control"
  Permission based on user/ team roles

- Push button deployment for application stack
  run playbooks with button click on UI

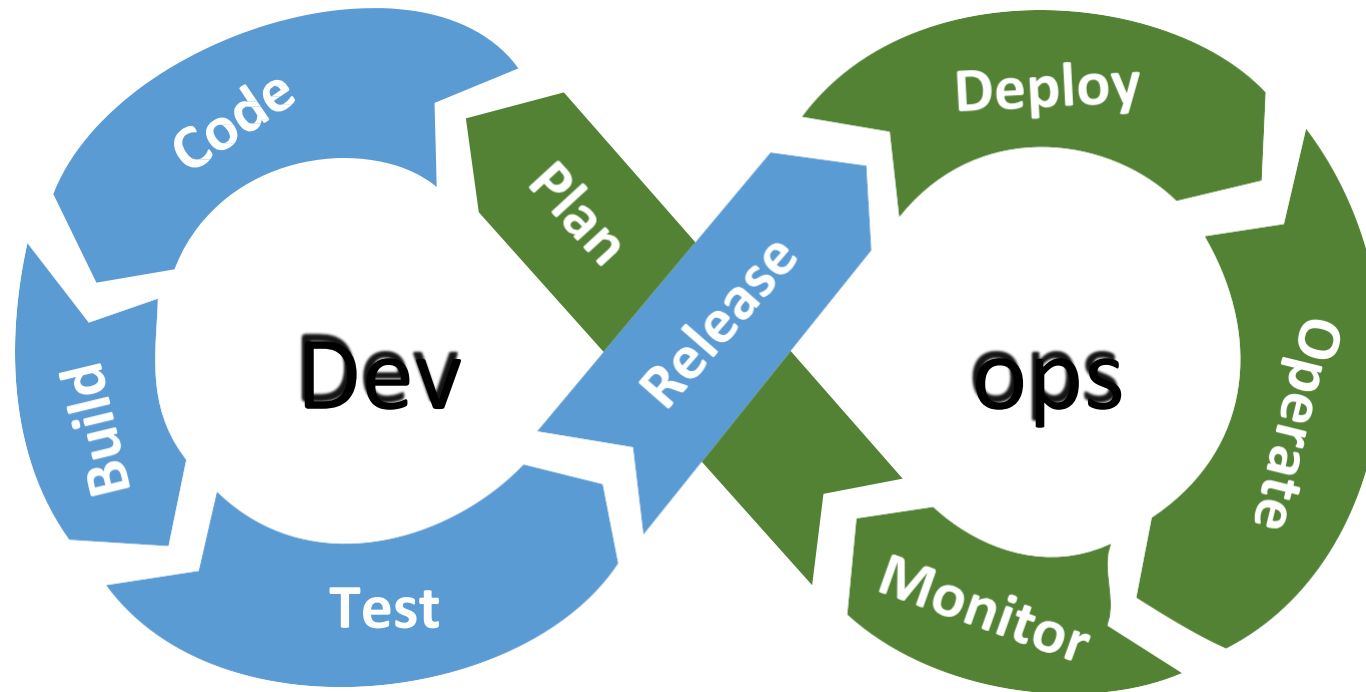- Live job status updates
  running job status on UI

Ansible Tower Key Features

- Job templates to launch playbooks

  Inputs on run time (variables, inventory files etc)

- Rest API to embed tower with other tools

  Integration with existing tools.

- Auditing and logging

  Integration with 3rd party auditing & logging tools.

- Clustering for Scalability

  flexible for tower infra extension

- Redhat support for quick fixes

  vendor support

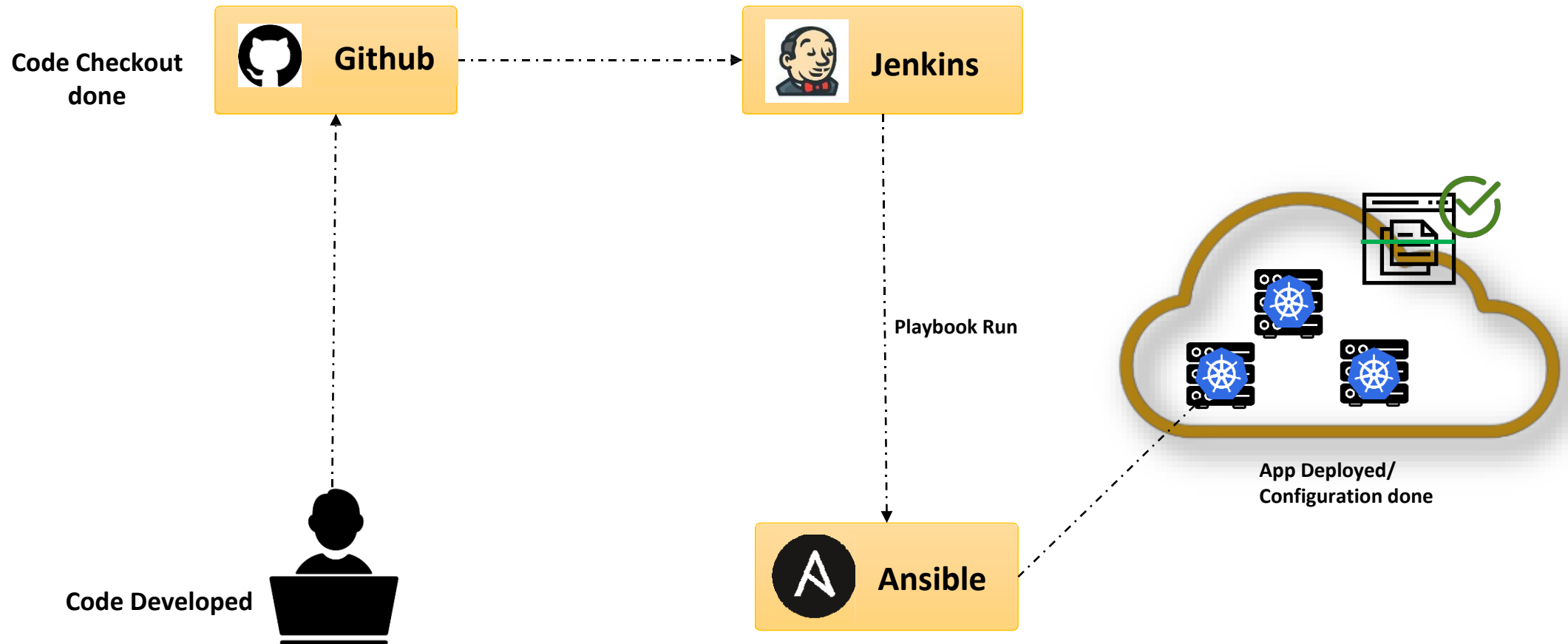# Ansible in DevOps

Ansible in DevOps

Ansible in DevOps

Continuous Integration

| Code | Auto | Unit Test | Auto | Integration Test | Auto | Acceptance Test | Manual | Deployment |

Continuous Deployment

| Code | Auto | Unit Test | Auto | Integration Test | Auto | Acceptance Test | Auto | Deployment |

Ansible in DevOps

Code Checkout done

Github

Jenkins

Playbook Run

Code Developed

Ansible

App Deployed/ Configuration done

210

Questions & Answers