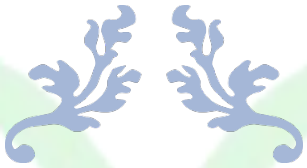


ANSIBLE DOCUMENTATION

Mithun Technologies, +91 99809 23226, devopstrainingblr@gmail.com

MITHUN SOFTWARE SOLUTIONS
Bangalore

Ansible

- Introduction
- What is Ansible?
- Configuration Management
- Ansible Features
- Ansible Architecture
- Host Inventory file
- Ansible Installation
- Ansible Commands
- Develop the Ansible Play Books
- Tags
- Handlers
- Group Variables and Host Variables
- Loops
- Conditional Statements
    - when
- Ansible Vault
- Ansible Roles
- Working With Dynamic Inventory

Addon Topics

- Run Ansible play book in Windows Machines
- Ansible Tower (https://youtu.be/O3giQN_r_io)

Introduction

Ansible is an open source, a Configuration Management Tool and Deployment tool, maintained by Redhat.

The main components of Ansible are playbooks, configuration management, deployment.

Ansible uses playbooks to deploy, manage, build, test and configure anything from full server environments to custom compiled source code for applications.

Ansible was written in Python.

Configuration can be any task which we can perform on servers (hosts).
It can be
1)Install/Update/Uninstalling software (packages)
2) Copy files. Changing the permissions on the files or directories.
3) Start/Stop/ Restart the services

4) Creating / deleting the users.

Various Configuration Management Tools

a) Ansible
b) Chef
c) Puppet
d) Salt Stack

Ansible Features

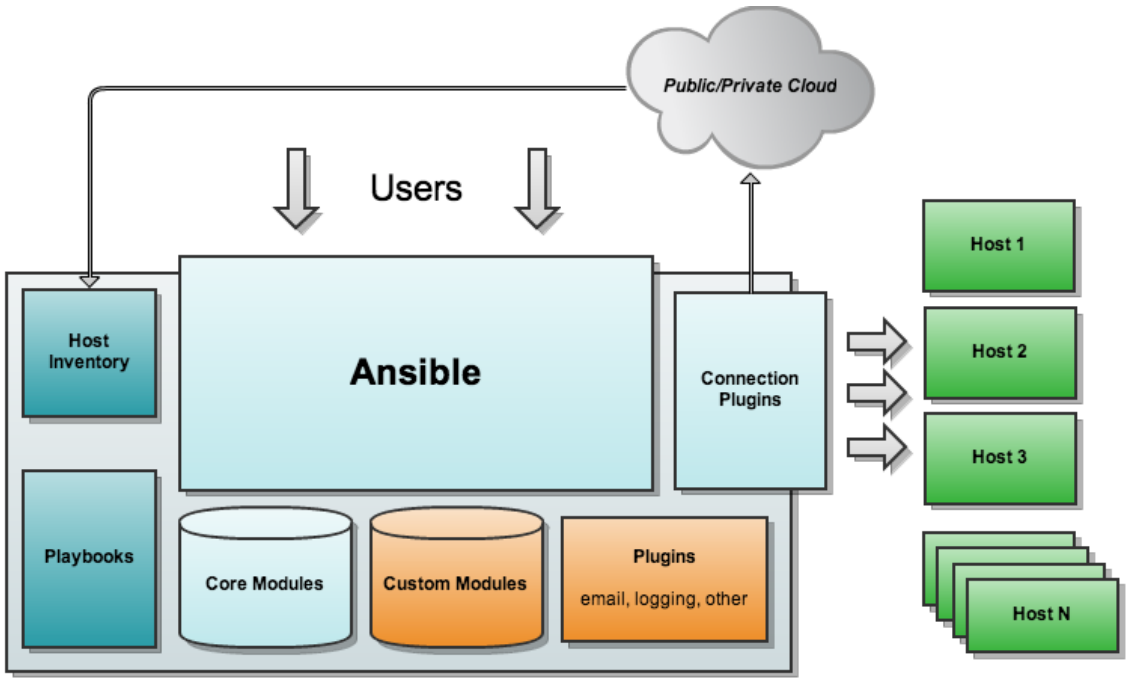
Ansible configure machines in an agent-less manner using SSH.
Built on top of Python and hence provides a lot of Python's functionality.
YAML-Based Playbooks
Uses SSH for secure connections.
Follows Push based architecture for sending configurations.

Push Based Vs Pull Based

Tools like Puppet and Chef are pull based.
Agents on the server periodically checks for the configuration information from central server (Master).

Ansible is push based.
Central server pushes the configuration information on target servers.
You control when the changes are made on the servers.

Ansible Architecture



Inventory file

Ansible's inventory hosts file is used to list and group your servers. Its default location is /etc/ansible/hosts.

See the contents in hosts file as follows.

cat /etc/ansible/hosts (default inventory file path)
#192.168.122.1 ---> This is one of the nodes IP
192.168.122.2

mithuntechnologies.dev.com

In Inventory file you can mention IP address or Hostnames also.

Some important points in Inventory file.
- Comments begin with the '#' character
- Blank lines are ignored
- Groups of hosts are delimited by [header] elements
- You can enter hostnames or ip addresses
- A hostname/ip can be a member of multiple groups
- Ungrouped hosts are specifying before any group headers, like below

Sample Inventory file

We can use '#' for comments in inventory file.

#Blank line are ignored.

#Ungrouped hosts are specifying before any group headers, like below
192.168.122.1
192.168.122.2
mithun-technolopgies.dev.com

[webservers]
#192.168.122.1
192.168.122.2
192.168.122.3

[dbservers]
#mithun-techno.db1.com
#mithun-techno.db2.com
#mithun-techno.db3.com
mithun-techno.db[1:3].com
mithun-techno.db5.com

192.168.122.4

192.168.122.5
192.168.122.6

appservers ansible_host=mithun-techno.appserver1.com ansible_connection=ssh ansible_port=5555

mailservers ansible_host=mithun-techno.mailserver.com ansible_connection=winrm

databseservers ansible_host=mithun-techno.db.com ansible_connection=ssh

Inventory Parameters

ansible_connection=ssh/winrm/localhost
ansible_port=22/5986
ansible_user=root/administrator
ansible_ssh_pass=<<Password for node>>

for localhost

localhost ansible_connection=localhost

If you want to have your Ansible hosts file in another location, then you can set this environment variable:

export ANSIBLE_HOSTS=/root/custom_ansible_hosts

Or you can specify the Ansible hosts location when running commands with the --inventory-file= (or -i) flag:

ansible all --inventory-file=/root/ansible_hosts -m ping

Reference URL: http://docs.ansible.com/ansible/latest/intro_inventory.html

---

Ansible Installation in Redhat Server

Create ansible user in all machines (Ansible server & Host Servers(1,2,3,…N))

1) Create the user ansible and set the password on all hosts:
   # sudo useradd ansible
   # sudo passwd ansible

2) Make the necessary entry in sudoers file /etc/ sudoersfor ansible user for password-less sudo access:

```
# visudo
ansible ALL=(ALL) NOPASSWD: ALL
```

3) Make the necessary changes in sshd_config file /etc/ssh/sshd_ config to enable password based authentication:
   Un comment PasswordAuthentication yes and comment PasswordAuthentication no.
   And save the file .
   Then restart sshd service.

```
# vi /etc/ssh/sshd_config
# sudo service sshd restart
```

Install Ansible in Red hat (Ansible Server)

1) SSH to Redhat System & Switch to ansible user

   $ sudo su ansible

2) Install python
   # sudo yum install python3 -y

3) Update python alaternatives
   # sudo alternatives --set python /usr/bin/python3

4) Verify Python Version
   # python --version

5) Install EPEL Repository
   # sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm -y

6) Install ansible using Yum
   # sudo yum -y install ansible

7) Verify Ansible version
   # ansible –version

Generate SSH Key, Copy SSH Key(Ansible Server)

1) Now generate SSH key in Ansible Server:
   $ sudo su ansible
   $ ssh-keygen

2) Copy it to Host servers as ansible user:
Repeat below command by updating HOST IP for all the HOST Servers.

#ssh-copy-id ansible@<HostIP>

Update Host Inventory in Ansible Server to add host servers' details.

1) Add Host Server details

vi /etc/ansible/hosts

Connect using username and password
#192.168.1.105 ansible_user=ansible ansible_password=password

Connect using username and pem(Make Sure Have pem file at given path)
#172.31.35.23 ansible_user=ec2-user ansible_ssh_private_key_file=~/aws.pem

If ssh keys are copied
172.31.35.23

2) Use ping module to test Ansible and after successful run you can see the below output.

```
$ ansible all -m ping
 172.31.35.23 | SUCCESS => {
 "changed": false,
  "ping": "pong"
  }
```

3) Install sshpass in Ansible server if you get below error .

"to use the 'ssh' connection type with passwords, you must install the sshpass program
$ sudo yum install -y http://mirror.centos.org/centos/7/extras/x86_64/Packages/sshpass-1.06-2.el7.x86_64.rpm

---

Ansible AD-HOC Commands

To run any ansible command we will follow below syntax.

ansible [group Name|HostName] -m <<Module Name>> -a <<Command Name>>

Here -m is the module name and -a is the arguments to module.

Example:
ansible all -m shell -a date: It will display date from all host machines.

There are two default groups, **all** and **ungrouped**. **all** contains every host. **ungrouped** contains all hosts that don't have another group

ansible-doc -l: It will display the all the modules available in Ansible.
ansible-doc yum: It will display more information about yum module along with examples.

Ping Module
---
ansible all -m ping: It will ping all the servers which you have mentioned in inventory file (/etc/ansible/hosts).
ansible all -m ping -o: It will display the output in single line.

```
[root@localhost ~]# ansible all -m ping
192.168.122.1 | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
[root@localhost ~]# ansible all -m ping -o
192.168.122.1 | SUCCESS => {"changed": false, "failed": false, "ping": "pong"}
[root@localhost ~]#
```

Shell Module
---
ansible all -m shell -a 'uptime' : Uptime of all the machines.
Here m means module and -a means argument.
(OR)
ansible all -a 'uptime'

```
[root@localhost ~]# ansible all -m shell -a 'uptime'
192.168.122.1 | SUCCESS | rc=0 >>
 17:15:46 up 22:23,  6 users,  load average: 0.35, 0.22, 0.15

[root@localhost ~]#
```

```
[root@localhost ~]# ansible all -a 'uptime'
192.168.122.1 | SUCCESS | rc=0 >>
 17:17:56 up 22:25,  6 users,  load average: 0.36, 0.21, 0.15

[root@localhost ~]#
```

ansible all -m shell -a 'date' : Date of all machines

```
[root@localhost ~]# ansible all -m shell -a 'date'
192.168.122.1 | SUCCESS | rc=0 >>
Sat Nov 11 17:13:51 IST 2017

[root@localhost ~]#
```

ansible all -m shell -a 'cat /etc/*release' : Redhat release of all the machines.
ansible all -m shell -a 'mount' : Kind of mount on all the machines
ansible all -b -m shell -a 'service sshd status' : Check the service status on all the machines.
ansible all -m shell -a "uname -a" -v

```
[root@localhost ~]# ansible all -m shell -a 'uname -a' -v
Using /etc/ansible/ansible.cfg as config file
192.168.122.1 | SUCCESS | rc=0 >>
Linux localhost.localdomain 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC
 2017 x86_64 x86_64 x86_64 GNU/Linux

[root@localhost ~]#
```

ansible dbservers -a "df -h" : Here it will check the disk space use for all the nodes which are from dbservers group.

ansible webservers -a "free -m":

ansible webservers -a "date":

Yum Module

---

ansible all -b -m yum -a "name=vim" : It will install vim package in all node machine which you have menyioned in host inventory file.
ansible localhost -b -m yum -a "name=git" : To install git package in localhost.
ansible all -b -m yum -a "name=httpd state=present" : To install httpd package in all node machines.

Note: Here state=present, is not a mandatory, it is by default.

ansible all -b -m yum -a "name=httpd state=latest" : To update httpd package in all node machines.
ansible all -b -m yum -a "name=httpd state=absent" : To remove httpd package in all node machines.

Service Module

---

ansible all -b -m service -a "name=httpd state=started"
ansible all -b -m service -a "name=httpd state=restarted"

ansible all -b -m service -a "name=httpd state=stopped"

ansible all -s -m service -a 'name=httpd state=started' :

Note: Here -b or -s either option we can use. But -s is deprecated and going to remove in 2.9 version.

rpm -qa | grep httpd : It will check weather httpd package is installed or not.

Uninstall Apache HTTP server using Linux command

---

yum erase httpd httpd-tools apr apr-util -y ---> Execute this command as a root user.

Copy Module

---

ansible all -b -m copy -a "src=mithuntechnologies.txt dest=/tmp/mithuntechnologies.txt"

If any access issue, need to give the sudo access to ansible in all hostmachines(nodes) as follows.

visudo (OR) vim /etc/sudoers --> Execute as a root user. And add below line in sudoers file.

ansible ALL=(ALL) NOPASSWD: ALL

Setup Module

This module is automatically called by playbooks to gather useful variables about remote hosts that can be used in playbooks.
In Playbooks we will use **gather_facts: no** to disable.

ansible all -m setup: It will give all facts about remote hosts.

ansible all -m setup -a 'filter=facter_*': It will give the all the facts which are started with "facter_"

YAML Ain't Markup Language (OR) Yet Another Markup Language

```
YAML: YAML Ain't Markup Language or Yet Another
Markup Language
```

What It Is YAML:

YAML is a human-friendly data serialization language for all programming languages.

YAML is a data serialization language for storing information in a human-readable form.

It is similar to XML and JSON files but uses a more minimalist syntax **YAML is commonly used** to create **configuration files**. Widely used in development & **In DevOps tools like Ansible, Docker, Kubernetes, Gitlab, Azure DevOps** etc.

Advantages ofYAML

- Lightweight
- It is very easy and simple for represent complex mapping
- Human friendly readable and writable
- Simple to modify with any text editor
- Suitable for configuration settings
- Support for major programming languages

Readable syntax

YAML files use an indentation to show the structure of your data. You're required to **use spaces** to create **indentation** rather than **tabs** to avoid confusion **or errors.**

No executable commands

As a data-representation format, YAML does not contain executables. It's therefore very safe to exchange YAML files with external parties.

YAML must be integrated or use with other languages, like Python, Go or Java, to add executables.

Built-in commenting

YAML allows you to add comments to files using the hash symbol (#)

YAML Syntax

YAML has a few basic concepts that make up(represent or define) the majority of data.

Key-value pairs

```
<key>: <value>
```

In general, most things in a YAML file are a form of key-value pair where the key represents the pair's name and the value represents the data linked to that name. Key-value pairs are the basis for all other YAML constructions.

Scalars and mapping

Scalars represent a single stored value. Scalars are assigned to key names using mapping. You define a mapping with a name, colon, and space, then a value for it to hold.

YAML supports common types like integer and floating-point numeric values, as well as non-numeric types Boolean and String.

```
name: "Mithun"       # Sting data
age: 35              # Integer data
salary: 152000.65    # Float values
isWorking:  true.    # Boolean values true or false yes or no
```

Numbers

YAML supports integers, floating numbers, and exponential floating numbers.

```
integer: 123
float: 123.123
```

Booleans

In YAML, we can represent the boolean value **True** and **False** in three different ways. Look at

them.

- The values **True, On,** and **Yes** are considered as **True** in YAML.
- The values **False, Off,** and **No** are considered as **False** in YAML.

```
employed: true
contractor: false
```

String

Strings are a collection of characters that represent a sentence or phrase. You either use | to print each string as a new line or > to print it as a paragraph.

Strings in YAML can be represented with or without quotes. Both are similar.. But, if we need to use the escape sequences, then we must use double-quotes

```
str: Hello World
data: |
  These
  Newlines
  Are broken up
data: >
  This text is
  wrapped and is a
  single paragraph
```

Sequence(lists or arrays)

Sequences are data structures similar to a **list or array that hold multiple values under the same key.** They're defined using a block or inline flow style.
Block style uses spaces to structure the document. It's easier to read but is less compact compared to flow style.
In Block Style Array elements in separate lines preceded hyphen (-)

```
---
# Shopping List Sequence in Block Style
shopping:
- milk
- eggs
- juice
```

Flow style allows you to write sequences inline using square brackets, similar to an array declaration in a programming language like Python or JavaScript. Flow style is more compact but harder to read at a glance.

```
---
# Shopping List Sequence in Flow Style
shopping: [milk, eggs, juice]
```

Dictionaries

Dictionaries are collections of key-value pairs. We can have any valid data type as a value to the key. YAML even supports the nested dictionaries.

```
# An employee record
employee:
  name:  Mithun
  job:   Architect
  team: DevOps
```

Dictionaries can contain more complex structures as well, such as sequences(Array or List of Dictionaries). Nesting sequences is a good trick to represent complex relational data.

```
# List of employee(dictionaries) records with complex data.
employess:
- name: Bhaskar
  job: Developer
  skills:
  - python
  - perl
  - ruby
- name: Balaji
  job: Developer
  skills:
  - java
  - go
```

Multi-document support

You can have multiple YAML documents in a single YAML file to make file organization or data parsing easier.

The separation between each document is marked by three dashes (---)

```
---
player: playerOne
action: attack (miss)
---
player: playerTwo
action: attack (hit)
```

Sample yaml files in ansible, docker & Kubernetes Ansible Playbook

```
- hosts: webservers
  become: yes
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: ensure apache is running
    service:
      name: httpd
      state: started
```

Docker Compose file

```
version: '3.1'

services:
  springboot:
    image: dockerhandson/spring-boot-mongo:latest
    environment:
    - MONGO_DB_HOSTNAME=mongo
    - MONGO_DB_USERNAME=devdb
    - MONGO_DB_PASSWORD=devdb123
    ports:
    - 8080:8080
    depends_on:
      - mongo
    networks:
    - springappnetwork
  mongo:
    image: mongo
    environment:
    - MONGO_INITDB_ROOT_USERNAME=devdb
    - MONGO_INITDB_ROOT_PASSWORD=devdb123
    volumes:
      - mongodb:/data/db
    restart: always
    networks:
    - springappnetwork

volumes:
  mongodb:
    external: true

networks:
  springappnetwork:
```

Kubernetes Manifest file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: javawebappdeployment
  name: javawebapp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: javawebapp
  template:
    metadata:
      labels:
        name: javawebapp
    spec:
      containers:
      - image: dockerhandson/java-web-app:1
        name: javawebapp
        ports:
        - containerPort: 8080
---
# Node Port Service
apiVersion: v1
kind: Service
metadata:
  labels:
    name: javawebapp
  name: javawebapp
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    name: javawebapp
```

YAML vs JSON vs XML

YAML (.yml or .yaml):

- Human-readable code
- Minimalist syntax
- Solely designed for data
- Similar inline style to JSON (is a superset of JSON)
- Allows comments
- Strings without quotation marks
- Considered the "cleaner" JSON
- Advanced features (extensible data types, relational anchors, and mapping types preserving key order)

Use Case: YAML is best for data-heavy apps that use DevOps pipelines or configurations. It's also helpful for when other developers on your team will work with this data often and therefore need it to be more readable.

JSON(.json)

- Harder to read
- Explicit, strict syntax requirements
- Similar inline style to YAML (some YAML parsers can read JSON files)
- No comments
- Strings require double quotes

Use Case: JSON is favoured in web development as it's best for serialization formats and transmitting data over HTTP connections.

XML(.xml)

- Harder to read
- More verbose
- Acts as a markup language, while YAML is for data formatting
- Contains more features than YAML, like tag attributes
- More rigidly defined document schema

Use Case: XML is best for complex projects that requiree fine control over validation, schema, and

namespace. XML is not human-readable and requires more bandwidth and storage capacity, but offers unparalleled control.

To Comments we will use # in YAML.

Yaml file extension is .yml or yaml

Key Value Pair

Fruit: Apple
Vegetable: Carrot
Liquid: Water
Meet: Chicken

Note: Need to give the space between ':' and value.

Array/List

Fruits:
- Orange
- Apple
- Banana
- Guava

Vegetables:
- Carrot
- Cauliflower
- Tomoto

Here - dash indicate the element of any array.

Playbooks

Playbook is a single YAML file, containing one or more 'plays' in a list.

Plays are ordered sets of tasks to execute against host servers from your inventory file.

Play defines a set of activities (tasks) to be run on hosts.

Task is an action to be perform on the host.
Examples are a) Execute a command
b) Run a shell script
c) Install a package
d) Shutdown/Restart the hosts.

Playbooks start with the YAML three dashes (---) and end with …

Each play has first hosts, variables, and tasks

FileName: pingServers.yml

```
---
- hosts: all
  gather_facts: no
  remote_user: ansible
  tasks:
    - name: Test connection
      ping:
      remote_user: ansible
...
```

#hosts: The tasks will be executing in specified group of servers.
#name: which is the task name that will appear in your terminal when you run the playbook.

#remote_user: This parameter was formerly called just user. It was renamed in Ansible 1.4 to make it more distinguishable from the user module (used to create users on remote systems).

Remote users can also be defined per task.

Run the playbook Using below command

ansible-playbook <<Playbbok file name>>:
ansible-playbook samplePlayBook.yml: It will run the samplePlayBook.yml playbook.

ansible-playbook samplePlayBook.yml -v: It will run the samplePlayBook.yml playbook in verbose.

ansible-playbook samplePlayBook.yml -vv: It will run the samplePlayBook.yml playbook in double verbose (It will give some more information).

ansible-playbook samplePlayBook.yml -vvv: It will run the samplePlayBook.yml playbook in more verbose.

Note: If any error while running playbook, use -v, -vv or -vvv option to debug the playbook.

ansible-playbook --help: It will provide help on **ansible_playbook** command.

ansible-playbook playbook.yml --syntax-check: It will check the syntax of a playbook.

ansible-playbook playbook.yml --check: It will do in dry run.

ansible-playbook playbook.yml --list-hosts: It will display the which hosts would be affected by a playbook before you run it.

ansible-playbook playbook.yml --step: It execute one-step-at-a-time, confirm each task before running with (N)o/(y)es/(c)ontinue .

FileName: createFilePlaybook.yml

```
---
- hosts: all
  become: true
  tasks:
   - name: Creating a file
     file:
      path: /tmp/mithuntecnoroot.sh
      owner: root
      mode: 0777
      state: touch
...
```

FileName: installHTTPServer.yml

#This playbbok will install HTTP server and will not start the server.

```
---
- hosts: all
  become: true
  tasks:
    - name: Install Apache HTTP server
      yum: name=httpd update_cache=yes state=latest
```

...

FileName: installHTTPServerandStart.yml

```
#This playbbok will install HTTP server and start the server.
---
- hosts: all
  become: true
  tasks:
    - name: Install Apache HTTP server
      yum: name=httpd update_cache=yes state=latest
    - name: Start HTTP Server
      service: name=httpd enabled=yes state=started
...
```

FileName: installHTTPServerWithHandlers.yml

```
#This play book will explains handlers as well.
---
- hosts: all
  become: true
  tasks:
    - name: Install Apache HTTP Server
      yum: name=httpd update_cache=yes state=latest
      notify:
        - Start HTTP Server
  handlers:
    - name: Start HTTP Server
      service:
          name=httpd
          state=restarted
...
```

#become: true: Is used to run commands with privileges, like if we're executing them with sudo.

#name which is the task name that will appear in your terminal when you run the playbook.
In this case, we called it "Install Apache HTTP server"

Note: You can also use become on a particular task instead of the whole play.

Uninstall Apache HTTP server using below command
----
sudo yum erase httpd httpd-tools apr apr-util -y

Nginx HTTP server installation

```
---
- hosts: localhost
  tasks:
       - name: Install nginx server
         yum: name=nginx state=present
         become: true
       - name: Start nginx server
         service: name=nginx enabled=yes state=started
         become: true
...
```

Items that begin with a - are considered list items.

FileName: playbook1.yml

```
---
   -   hosts: appservers
       tasks :
          - name: Execute the 'date' command
            command: date

         - name: Execute script on server
           script: sample_script.sh

         - name: Install httpd service
           yum:
             name: httpd
             state: present

         - name: Start web server
             service:
             name: httpd
             state: started
```

Handlers

Handlers are special task that run at the end of a play if notified by another task.
If a configuration file gets changed notify a service restart task it needs to run.

```
- hosts: localhost
  become: true
  tasks:
    - name: install httpd
      yum: name=httpd update_cache=yes state=latest
      notify:
        - start httpd
  handlers:
    - name: start httpd
      service:
          name=httpd
          state=restarted
...
```

Loops

#This playbbok will install HTTP server,wget and vim package using loops.

```
---
- hosts: localhost
  become: true
  tasks:
    - name: Install list of packages
      yum: name='{{item}}' state=present
      with_items:
       - httpd
       - wget
       - vim
       - zip
       - unzip
...
```

Note: Loop feature will be removed in version 2.11

Instead of using a loop to supply multiple items and specifying like **name: "{{ item }}"**, use **name: ['httpd', 'wget', 'vim', 'zip, 'unzip']** as follows.

```
---
- hosts: localhost
  become: true
  tasks:
   - name: Install list of packages
     yum:
      name: ['httpd', 'wget', 'vim', 'zip', 'unzip']
...
```

Variables

There are different ways in which you can define variables in Ansible. The simplest way is by using the **vars** section of a playbook. The below example defines a variable name called **package** and it is using in a task called **Install a Package**.

FileName: variables-playbook.yaml

```
---
- hosts: localhost
  become: true
  vars:
     package: vim
  tasks:
     - name: Install a Package
       yum:
         name: "{{package}}"
         state: latest
...
```

Group Variables and Host Variables

You can define custom variables for each group and host that you define in host inventory.
These variables are known as group_vars for groups and host_vars for hosts. Any variables that you define for a host or a group can be used in both playbooks and templates.

Both group_vars and host_vars are defined in their own folders, ‘groups_vars’ and ‘host_vars’, respectively. For group_vars, the file must be named exactly the same as the group.
For host_vars, the file has to be named exactly the same as the host.

Say you have a group ‘appServers’ and you want to define a variable for all of them. Create an empty file first in the group_vars folder in the root of your ansible directory (where you put you playbooks or in ansible home(installation) directory):

group_vars/appServers

Then add a variable to the file:

package=httpd

This makes the variable ‘package’ available to all playbooks that run on this group. I’ll show you how to use these in your playbook in a bit.

Say you have one group appServer you want to execute a certain action in a playbook by using varible defined in group_vars. The following is an action you could use, using a group_var definition:

```
---
- hosts: appServers
  become: true
  tasks:
    - name: Install a Package
      yum:
        name: "{{package}}"
        state: latest
```

You can also do this with host_vars:

host_vars are similar to this. Create a file first:

host_vars/localhost

And add a variable:

package=git

this makes the variable ‘package’ available to all playbooks that run on this host.

```
---
- hosts: localhost
  become: true
  tasks:
    - name: Install a Package
      yum:
        name: "{{package}}"
        state: latest
```

If group_vars,host_vars has same variable with different values.While play is executed in that host variable value from host_vars will take precedence.

Conditional Statements

The When Statement

Sometimes you will want to skip a particular step on a particular host. This could be something as simple as not installing a certain package if the operating system is a

particular version, or it could be something like performing some cleanup steps if a filesystem is getting full.

This is easy to do in Ansible with the *when* clause, which contains a raw Jinja2 expression without double curly braces

```
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_facts['os_family'] == "Debian"
    # note that all variables can be directly in conditionals without double curly braces
```

You can also use parentheses to group conditions:

```
tasks:
  - 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")
```

Multiple conditions that all need to be true (a logical ‘and’) can also be specified as a list:

```
tasks:
  - name: "shut down CentOS 6 systems"
    command: /sbin/shutdown -t now
    when:
      - ansible_facts['distribution'] == "CentOS"
      - ansible_facts['distribution_major_version'] == "6"
```

Tags

Tags are useful to be able to run a specific without running the whole playbook.

We can use tags on play and task level.

```
---
- hosts: localhost
  become: yes
```

```
tasks:
- name: Install Apache HTTP server on RedHat Server
  tags:
    - install
  yum:
    name: httpd
    state: present
  when: ansible_os_family == "RedHat"
- name: Install Apache HTTP server on Ubuntu server
  tags:
    - install
    - start
  apt:
    name: apache2
    state: present
  when: ansible_os_family == "Debian"
- name: Install Apache HTTP server on CentOS server
  yum:
    name: httpd
    state: present
  when:
    - ansible_facts['distribution'] == "CentOS"
    - ansible_facts['distribution_major_version'] == "7"
- name: Print the Ansible free memory
  debug:
    msg: "free memory is {{ansible_memory_mb.real.free}}"
```

ansible-playbook sampleplaybook.yaml --list-tags: It will display all available tags in specified playbook.

ansible-playbook sampleplaybook.yml --tags "install,start" : This command will run the tags install and start.

ansible-playbook sampleplaybook.yml --skip-tags "install": This command will skip the tags specified tags, install.

https://docs.ansible.com/ansible/2.4/playbooks_tags.html

Ansible Vault

A typical Ansible setup will involve needing some sort of secret to fully setup a server or application.Common types of "secret" include passwords, SSH keys, SSL certificates, API tokens and anything else you don't want the public to see.

Since it's common to store Ansible configurations in version control, we need a way to store secrets securely.
Ansible Vault is the answer to this. Ansible Vault can encrypt anything inside of a YAML file, using a password of your choice.

Using Ansible Vault

A typical use of Ansible Vault is to encrypt variable files. Vault can encrypt any YAML file, but the most common files to encrypt are:

1. Files within the group_vars directory.
2. A role's defaults/main.yml file
3. A role's vars/main.yml file.
4. Any other file used to store variables.

Let's see how to use Ansible Vault with some variable files.
Let's take below host inventory file we have defined host details along with username & password to connect to the host. If you observe password is visible to everyone. As per standards we should not expose passwords to everyone.

```
[ansible@ip-172-31-21-155 project]$ cat hosts
172.31.29.109 ansible_user=ansible ansible_ssh_pass=DevOps@2018
```

We have two options to here.

- Encrypt complete host inventory file using ansible vault. But this is not suggestable as we are encrypting complete file to encrypt password.
- Create group variables or host variables file and encrypt using ansible vault. And refer variable in host inventory from group/host variables file.

Step1: Create group variables for all groups.

```
[ansible@ip-172-31-21-155 project]$ mkdir group_vars
[ansible@ip-172-31-21-155 project]$ vi group_vars/all.yml
```

Add your password (Key value pair) in group variables yml

```
[ansible@ip-172-31-21-155 project]$ cat group_vars/all.yml
ansible_pwd: DevOps@2018
[ansible@ip-172-31-21-155 project]$
```

Step 2: Encrypt existing group variables yml file using ansible vault.

The typical use case is to have a normal, plaintext variable file that we want to encrypt. Using
ansible-vault, we can encrypt this and define the password needed to later decrypt it:

```
[ansible@ip-172-31-21-155 project]$ ansible-vault encrypt group_vars/all.yml
New Vault password:
Confirm New Vault password:
Encryption successful
[ansible@ip-172-31-21-155 project]$
```

If see the content the complete file is encrypted. Content is not in human readable format.

```
[ansible@ip-172-31-21-155 project]$ cat group_vars/all.yml
$ANSIBLE_VAULT;1.1;AES256
36306339363662323338336335313861633862653937386538343938306238636531383364616363
66636235363535363034616331363561663030623864646 60a39303437366362316564393261656 2
66613364623731656665306163626432303762643833663961306661356134333831643231613163
37313864653565316 50a32386334616438326133636430323461653134333339663239306131373 3
65323466633261653261646664353161653233643633303136376463393634633361
[ansible@ip-172-31-21-155 project]$
```

Step 3:
Update your host inventory to refer password from group_vars/all.yml file make sure you use the same key name which you defined in group_vars/all.yml.

```
[ansible@ip-172-31-21-155 project]$ cat hosts
172.31.29.109 ansible_user=ansible ansible_ssh_pass={{ansible_pwd}}
[ansible@ip-172-31-21-155 project]$
```

Step 4: Execute ansible adhoc command ping to test the connectivity.

```
2. ansible@ip-172-31-21-155:~/projec
[ansible@ip-172-31-21-155 project]$ ansible -i hosts -m ping all
ERROR! Attempting to decrypt but no vault secrets found
[ansible@ip-172-31-21-155 project]$
```

We will get error since we are referring password from group_vars/all.yml which is encrypted using vault. So, while executing play book or adhoc commands we must pass ansible vault password what ever
You have typed in while encrypting.

Use below command to execute.

```
[ansible@ip-172-31-21-155 project]$ ansible -i hosts -m ping all --ask-vault-pass
Vault password:
172.31.29.109 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[ansible@ip-172-31-21-155 project]$
```

Go through below to know more about other ansible vault commands or options.

Encrypting an Existing File

The typical use case is to have a normal, plaintext variable file that we want to encrypt. Using
ansible-vault, we can encrypt this and define the password needed to later decrypt it:

```
# Encrypt a role's defaults/main.yml file
ansible-vault encrypt defaults/main.yml

        New Vault password:
> Confirm New Vault password:
> Encryption successful
```

The ansible-vault command will prompt you for a password twice (a second time to confirm the first). Once that's done, the file will be encrypted! If you edit the file directly, you'll just see encrypted text. It looks something like this:

```
$ANSIBLE_VAULT;1.1;AES256
65326233363731663631646134306563353236653338646433343838373437373430376
464616339
33333832333734653531313232376365383633613164313 80a643336643862663739623
631616530
35356361626434653066316661373863313362396162646365343166646231653165303
431636139
623036616436313 8340a35663163393032303265346662653138326161353963336536 6
631623238
32396637623866633135363231346664303730353230623439633666386662346432363
164393438
33653666373064326233373337383934316335303862313838383966623134646230346
330303136
66333232363062303837333533303130386238323165623632346239383538343437663
437373730
35666532333065383439
```

Creating an Encrypted File

If you are creating a new file instead of encrypting an existing one, you can use the create
command:

```
ansible-vault create defaults/secrets.yml
> New Vault password:
> Confirm New Vault password:
```

Editing a File

Once you encrypt a file, you can only edit the file by using ansible-vault again (unless there's an editor out there that can integrate with Vault to let you edit in the IDE?!). Here's how to edit that file after it's been encrypted:

```
ansible-vault edit defaults/main.yml
> Vault password:
```

This will ask for the password used to encrypt the file.

> You'll lose your data if you lose your password!

Since we're running these commands in the CLI, this will open the file in a terminal-based app. This usually means your default editor as defined the EDITOR environment variable, if that is set:

```
echo $EDITOR
> vim
```

My EDITOR environment variable isn't set, but my default is Vim. To use Nano instead of Vim, you can set that variable while running Vault:

```
EDITOR=nano ansible-vault edit defaults/main.yml
```

Decrypting a File

You can decrypt a file to get it back to plaintext as well:

```
ansible-vault decrypt defaults/main.yml
> Vault password:
```

Encrypting Specific Variables

You don't have to encrypt a whole file! This is nice to track changes better in git, where you don't have an entire file changing for just a small change (even just opening an encrypted file will change the encrypted hash).

ansible-vault encrypt_string <secretvalume> --name <secrentname>

The most basic use case is to just run it interactively on the CLI to get the properly formatted YAML as output:

```
ansible-vault encrypt_string
> New Vault password:
> Confirm New Vault password:
> Reading plaintext input from stdin. (ctrl-d to end input)
> this is a plaintext string
> !vault |
>          $ANSIBLE_VAULT;1.1;AES256
> 393937666637616533373864366364663965313532613832376135313565313439306631336238 39
> 343661383430326461303862343230383739326166323364 0a36363334333762306561316630 6363
> 373361323634623861383435353462643330616561346366313261646430353134333938316161 31
> 363561356537393931 0a316132313764356432333366396533663965333316233653866343232333 34
```

That string could be used in a variable file like so (as variable some string):

```
---
some_string: !vault |
    $ANSIBLE_VAULT;1.1;AES256

393937666637616533373864366364663965313532613832376135313565313439306631336238
39

343661383430326461303862343230383739326166323364 0a3636333433376230656131663063
63

373361323634623861383435353462643330616561346366313261646430353134333933831616
131
```

You can do this in one line also:

```
ansible-vault encrypt_string 'this is a plaintext string' --name
'some_string'
> New Vault password:
> Confirm New Vault password:
> some_string: !vault |
>           $ANSIBLE_VAULT;1.1;AES256
> 343962326431333230346663353139396338653565343030643962386439393433376263306666164

> 623130306137366632626438653866656437376266333231 0a32393862623936376334363835326
4
> 646462666633613866333863316561633534386230336266333666643035363961363538343364
36
> 636330353230326564 0a396264616562663963653034376462613035383333373437653362616566
>           3531
> Encryption successful
```

The output can be copied/pasted or appended into an existing YAML file!

Running Ansible with Encrypted Variables

If your roles or playbooks reference encrypted variables, you need to have given Ansible the password to decrypt them. You can do this in two ways:
Ask for Vault Password

Have Ansible ask for the vault password as a prompt:

```
ansible-playbook --ask-vault-pass -i inventory_file some_playabook.yml
```

Using the --ask-vault-pass flag will instruct Ansible to ask for the vault password so it can decrypt the variable files correctly.
Use a Vault File

Another handy thing you can do is store a vault password in a file and use that. This is handy for automation.
To do so, first create a file with a password:

```
echo "secret_password" > vault_password
```

Then you can reference that file with the --vault-password-file flag. This flag can be used with any ansible-playbook or ansible-vault command to pre-define the password, so you do not get a prompt:

```
# When creating/editing/encrypting/decrypting/rekeying a file:
ansible-vault --vault-password-file=vault_password create
defaults/foo.yml
ansible-vault --vault-password-file=vault_password edit
defaults/foo.yml
```

```
# When running ansible playbooks
ansible-playbook --vault-password-file=vault_password -i inventory_file
some_playabook.yml
```

Issues

There are only a few issues to really worry about:

- If you lose your password, you lose your data. One common way to make this more manageable is to use a shared password manager.
- If you're tracking your Ansible roles in git, you'll notice that even opening an encrypted file (and not changing it) will change the file; Merely opening a file means a new git commit. This is an annoying result of how the encryption is done when opening (decrypting) and closing (re-encrypting) a file for editing. This can be mitigated by encrypting only specific variables within a file as shown above.

Ansible Roles

With more complexity in functionality, it becomes difficult to manage everything in one ansible playbook file. Sharing code among teams become difficult. Ansible Role helps solve these problems. Ansible role is an independent component which allows reuse of common configuration steps. Ansible role has to be used within playbook. Ansible role is a set of tasks to configure a host to serve a certain purpose like configuring a service. Roles are defined using YAML files with a predefined directory structure.

What is Ansible roles?

1. Ansible roles are consists of many playbooks, which is similar to modules in puppet and cook books in chef. We term the same in ansible as roles.
2. Roles are a way to group multiple tasks together into one container to do the automation in very effective manner with clean directory structures.
3. Roles are set of tasks and additional files for a certain role which allow you to break up the configurations.
4. It can be easily reuse the codes by anyone if the role is suitable to someone.
5. It can be easily modify and will reduce the syntax errors.

Below is a sample playbook codes to deploy Apache web server. Let's convert this playbook codes into Ansible roles.

```
---
- hosts: all
  become: true
  tasks:
```

```
 - name: Install httpd Package
   yum: name=httpd update_cache=yes  state=latest
 - name: Copy httpd configuration file
   copy: src=httpd.conf dest=/etc/httpd/conf/httpd.conf
 - name: Copy index.html file
   copy: src=index.html dest=/var/www/html
   notify:
   - restart apache
 - name: Start and Enable httpd service
   service: name=httpd state=restarted enabled=yes
 handlers:
 - name: restart apache
   service: name=httpd state=restarted
```

How do we create Ansible Roles?

To create a Ansible roles, use ansible-galaxy command which has the templates to create it. This will default directories and do the modifications else we need to create each directories and files manually.

Let's take an example to create a role for Apache Web server.

```
# mkdir /etc/ansible/rolesDemo
# ansible-galaxy init /etc/ansible/rolesDemo/apache

    -    apache was created successfully
[ansible@ip-172-13-17-90 ~]#
```

where, ansible-glaxy is the command to create the roles using the templates.

init is to initiliaze the role.

apache is the name of role.

List out the directory created under /etc/ansible/rolesDemo.

Note : if tree command is not working install tree package using package manager.

```
[ansible@ip-172-13-17-90 ~]# tree /etc/ansible/rolesDemo/apache/
```

```
/etc/ansible/rolesDemo/apache/

|-- README.md

|-- defaults

|  `-- main.yml

|-- files

|-- handlers

|  `-- main.yml

|-- meta

|  `-- main.yml

|-- tasks

|  `-- main.yml

|-- templates

|-- tests

|  |-- inventory

|  `-- test.yml

`-- vars

`-- main.yml

8 directories, 8 files

[ansible@ip-172-13-17-90 ~]#
```

We have got the clean directory structure with the ansible-galaxy command. Each directory must contain a main.yml file, which contains the relevant content.

Directory Structure:

A role directory structure contains directories: defaults, vars, tasks, files, templates, meta, handlers. Each directory must contain a main.yml file which contains relevant content. Let's look little closer to each directory.

1. defaults: contains default variables for the role. Variables in default have the lowest priority so they are easy to override.
2. vars: contains variables for the role. Variables in vars have higher priority than variables in defaults directory.
3. tasks: contains the main list of steps to be executed by the role.
4. files: contains files which we want to be copied to the remote host. We don't need to specify a path of resources stored in this directory.
5. templates: contains file template which supports modifications from the role. We use the Jinja2 templating language for creating templates.
6. meta: contains metadata of role like an author, support platforms, dependencies.
7. handlers: contains handlers which can be invoked by "notify" directives and are associated with service.

First, move on to the Ansible roles directory and start editing the yml files.

cd /etc/ansible/rolesDemo/apache

1. defaults

Edit main.yml available in the defaults folder to define the default variables.

base_httpd_listen_port: 80

2. Tasks

Edit main.yml available in the tasks folder to define the tasks to be executed.

vi tasks/main.yml

```
---
-   name: Install httpd Package

yum: name=httpd state=latest

-   name: Copy httpd configuration file

 template: src=httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf

- name: Copy index.html file
```

copy: src=index.html dest=/var/www/html

notify:

- restart apache

3. Templates

Copy the required files (httpd.conf.j2) to the templates directory.

4. Files

Copy the required files (index.html) to the files directory.

5. Handlers

Edit handlers main.yml to restart the server when there is a change. Because we have already defined it in the tasks with notify option. Use the same name "restart apache" within the main.yml file as below.

- name: restart apache

  service: name=httpd state=restarted

6. Meta

Edit meta main.yml to add the information about the roles like author, descriptions, license, platforms supported.

[ansible@ip-172-13-17-90]# cat meta/main.yml

galaxy_info:
author: MithunTechnologies.net
description: Apache Webserver Role
company: MithunTechnologies.net

We have got all the required files for Apache roles. Let's apply this role into the ansible playbook "site.yml" as below to deploy it on the client nodes.

cat /etc/ansible/rolesDemo/site.yml ---

- hosts: appServers

roles:
- apache

sWe have defined this changes should be run only on appServers, you can also use "all" if need. Specify the role name as "apache", also if you have created multiple roles, you can use the below format to add it.

- apache
- common

Let's verify for syntax errors:

[ansible@ip-172-13-17-90]# ansible-playbook /etc/ansible/rolesDemo/site.yml --syntax-check

playbook: /etc/ansible/rolesDemo/site.yml

If No errors found. Let move on to deploy the roles.

[ansible@ip-172-13-17-90]# ansible-playbook /etc/ansible/rolesDemo/site.yml

Source Code can be downloaded from git hub here

Ansible Modules

setup module
file
pause
yum and apt
service
copy
ping

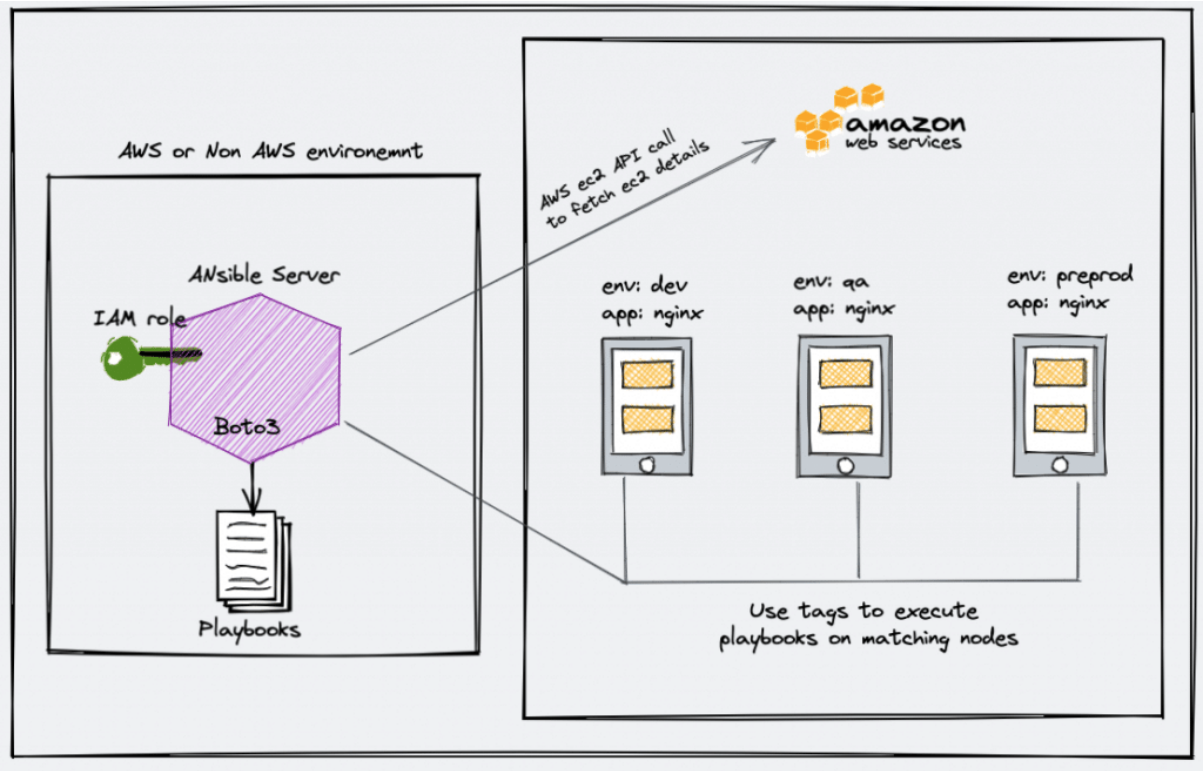
service
command
debug
template
uri
user
assert

System
Commands
Database
Cloud
Windows

Working With Dynamic Inventory

To working with AWS dynamic inventory, we need **boto3** and **botocore** python modules.

https://docs.ansible.com/ansible/latest/collections/amazon/aws/aws_ec2_inventory.html



- Get inventory hosts from Amazon Web Services EC2.
- Uses a YAML **configuration file that ends with** `aws_ec2.(yml|yaml)`.

Requirements

The below requirements are needed on the local controller node that executes this inventory.

- boto3
- botocore

First, install **python3** if you haven't installed it yet.

```
$ sudo yum install -y python3
```

Install "**boto3**"

```
$ pip3 install --user boto3
$ pip3 install --upgrade requests --user
```

Create a file named inventory_aws_ec2.yml in the project directory.

Note: The file name needs to be ended with **aws_ec2.yaml/yml**.

```
$ vi inventory_aws_ec2.yml
```

Paste the content below into the **inventory_aws_ec2.yml** file. As you see that this file begins with defined the plugin: **aws_ec2**.

Note: In this example, I added one tag to the target nodes (via AWS Console) "Name" to groups them. And use filter to see only **running** instances.

```
plugin: aws_ec2

regions:
  - ap-south-1

filters:
  instance-state-name : running

keyed_groups:
  - key: tags.Name
    prefix: ""
```

```
  separator: ""

hostnames:
  - private-ip-address

compose:
  ansible_host: private_ip_address
```

But at this point, the Control node needs authentication to access the AWS resources. ***If you want, you can add your AWS access key and secret to the config file.***

But I think it is not a safer way, and I prefer to use the **IAM role** instead. So Ansible will automatically use this role to make the AWS API calls.

Step 3: Add An IAM Role And Attached It To Control Node

At **AWS Console**, go to **Identity and Access Management (IAM)** service and click the "**Create role**" button and then create a role with "**AmazonEC2ReadOnlyAccess**".

After that, we need to attach this role with the Control node.

- Go to **EC2 Dashboard**, and select the control-node instance
- Select "**actions"** → "**security"** → "**modify IAM role"**
- Select the role that has "**AmazonEC2ReadOnlyAccess**" and **save** it.

Step 4: Pinging The Target Nodes With Dynamic Inventory

First, check the inventory.

Note: We will use the "**-i"** flag to refer to the **inventory_aws_ec2.yml** file because we haven't changed the inventory variable in the config file yet.

```
$ ansible-inventory --graph -i inventory_aws_ec2.yml
```

```
[ansible@ip-172-31-43-253 ~]$ ansible-inventory -i inventory_aws_ec2.yml --graph
@all:
  |--@Ansible_Server:
  |  |--172.31.38.238
  |--@HostOne:
  |  |--172.31.32.212
  |--@HostTwo:
  |  |--172.31.43.6
  |--@Kubernetes_Master:
  |  |--172.31.10.88
  |--@Kubernetes_Worker:
  |  |--172.31.34.238
  |  |--172.31.40.93
  |--@TestAnsible:
  |  |--172.31.43.253
  |--@TestServer:
  |  |--172.31.32.25
  |--@aws_ec2:
  |  |--172.31.10.88
  |  |--172.31.32.212
  |  |--172.31.32.25
  |  |--172.31.34.238
  |  |--172.31.38.238
  |  |--172.31.40.93
  |  |--172.31.43.253
  |  |--172.31.43.6
  |--@ungrouped:
```

Using Dynamic Inventory Inside Playbook

If you want to use dynamic inventory inside the playbook, you just need to mention the group name in the hosts variable as shown below.

```
- hosts: HostOne
  gather_facts: false
  tasks:
    - name: Run Shell Command
      command: echo "Hello World"
```

and execute like

```
$ ansible-playbook -i inventory_aws_ec2.yml <playbookName>.yaml
```

```
$ ansible-playbook -i inventory_aws_ec2.yml <playbookName>.yaml -u= <username> --private-key=<PemFilePath>.pem
```

```
$ ansible-playbook -i inventory_aws_ec2.yml <playbookName>.yaml -u=<username> --private-key=<PemFilePath>.pem -l <groupName>
```

Addon Topics

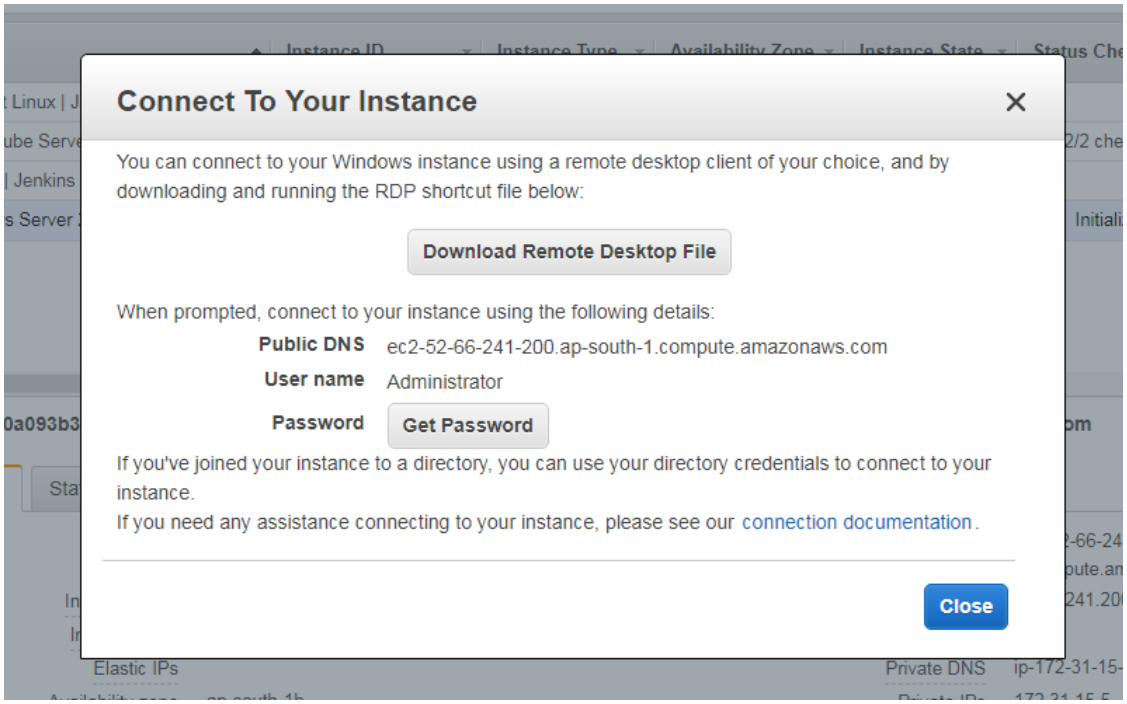
Ansible Playbook (Linux – Windows) using Winrm

1. Install winrm in Linux server

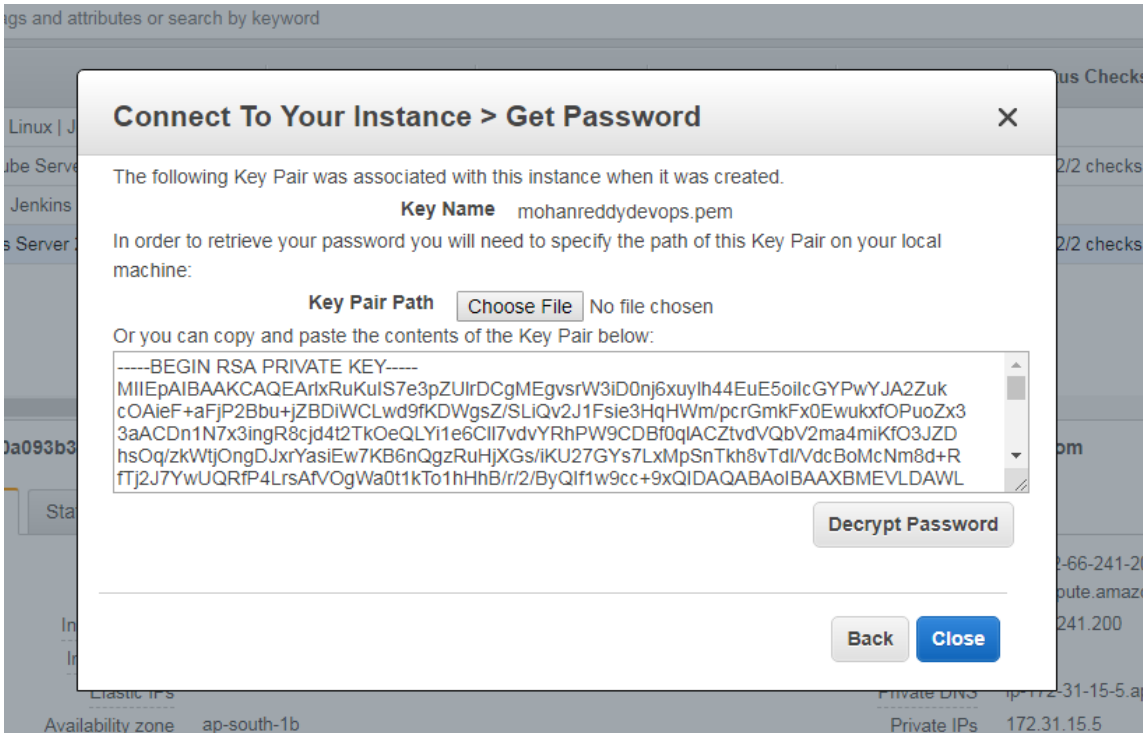
   sudo pip3 install pywinrm
2. Create a Windows Server 2019 Instance in AWS
3. Open Winrm-https port in security groups

4. Login to Windows Server using Remote Desktop tool

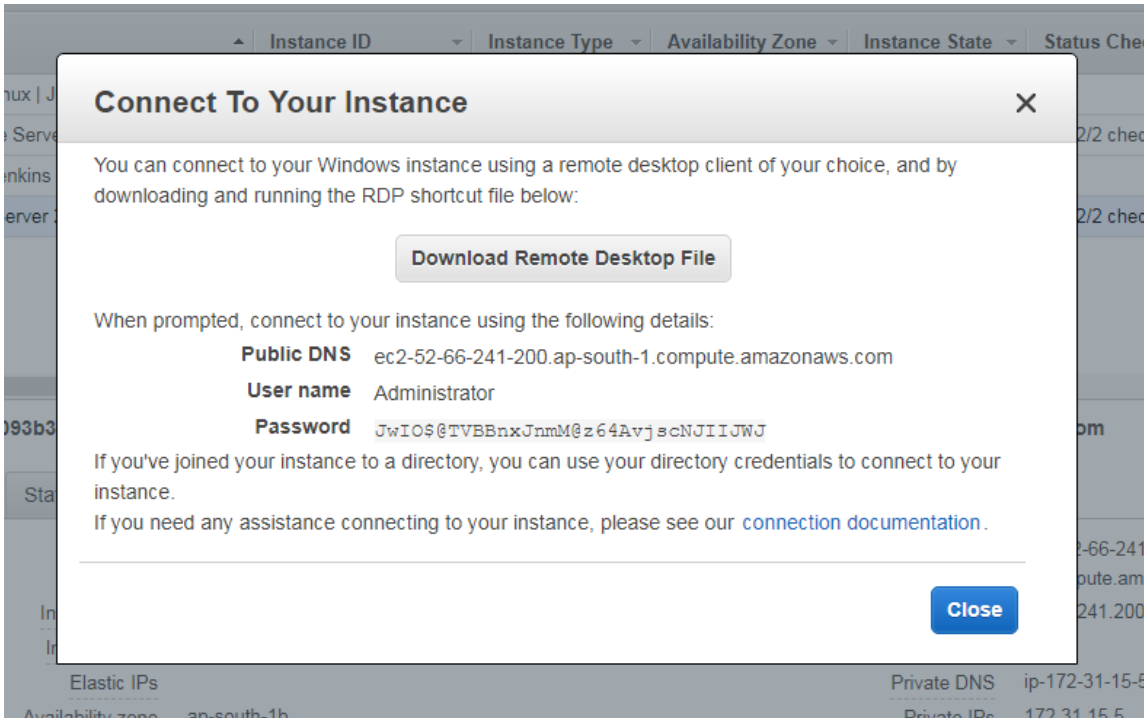
   Connect -> Click Get Password



Click -> Choose File & select your AWS pem file & Click Decrypt Password

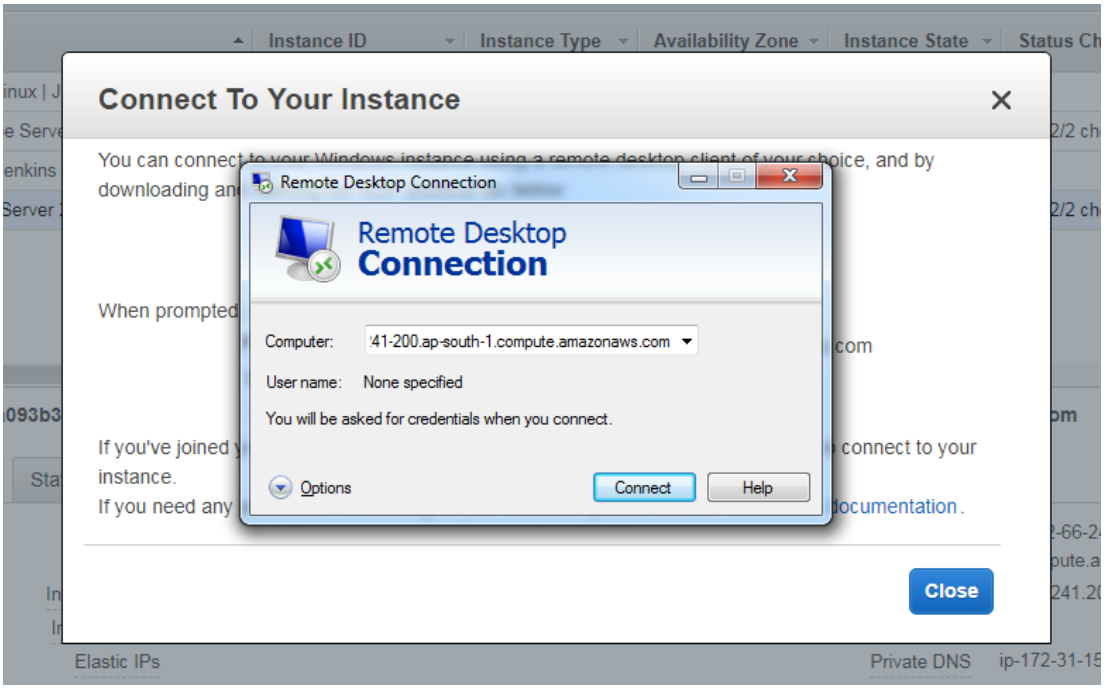


Copy -> Public DNS, User Name & Password



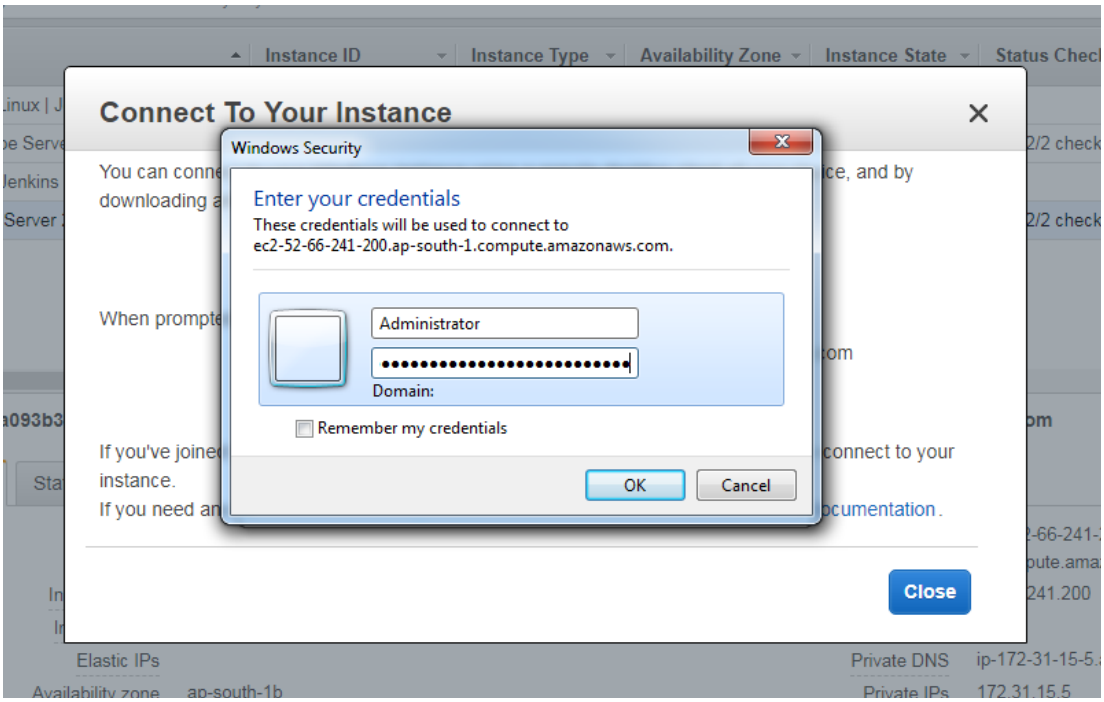
Open Remote Desktop Connection –> Paste Public DNS in place of Computer

& Click -> Connect



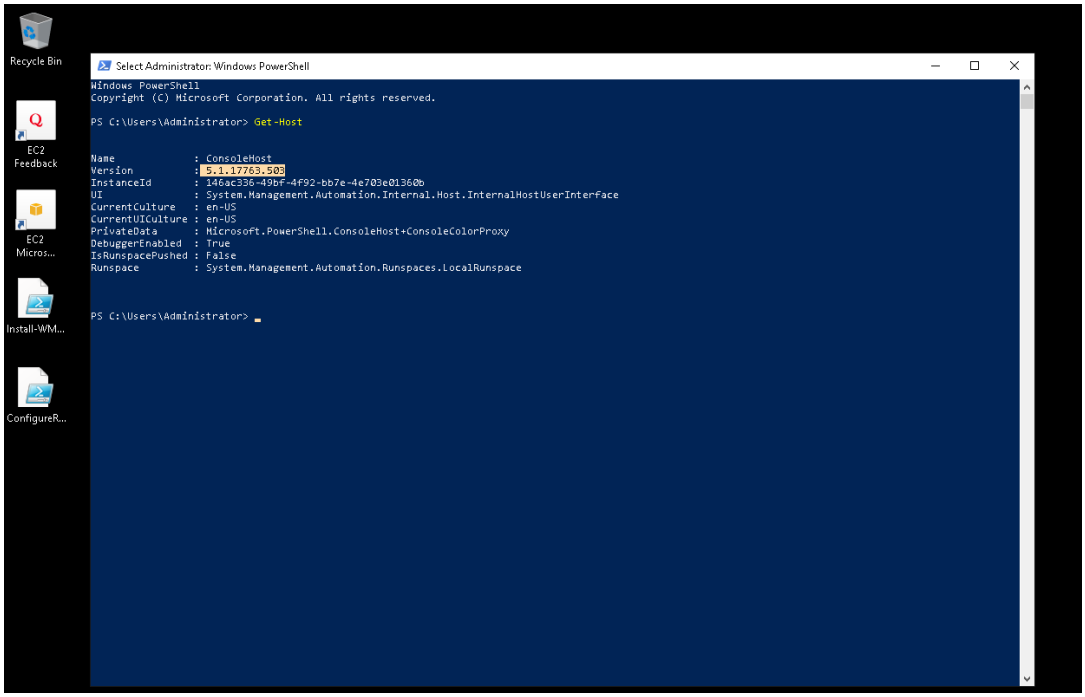
Provide Username & Password and Click -> OK

Now Remote Desktop will connect to the Windows Server 2019



Windows Server 2019 Connected

5. Open Windows Powershell and Check for version. It is above 3.0, so we can configure winrm now (use Get-Host command to check version)



6. Configure Winrm

6.1 Winrm Memory Hotfix

Goto this link

https://github.com/jborean93/ansible-windows/blob/master/scripts/Install-WMF3Hotfix.ps1

Copy the code and save it in Windows Server Machine Desktop as Install-WMF3Hotfix.ps1

Open Windows Powershell and go to Desktop (cd ~/.Desktop)

Execute the following command

```
powershell.exe -ExecutionPolicy ByPass -File Install-WMF3Hotfix.ps1 –Verbose
```

6.2 Winrm Setup

Got to this link

https://github.com/ansible/ansible/blob/devel/examples/scripts/ConfigureRemotingForAnsible.ps1

Copy the code and save it in Windows Server Machine Desktop as ConfigureRemotingForAnsible.ps1

Open Windows Powershell and go to Desktop (cd ~/.Desktop)

Execute the following command

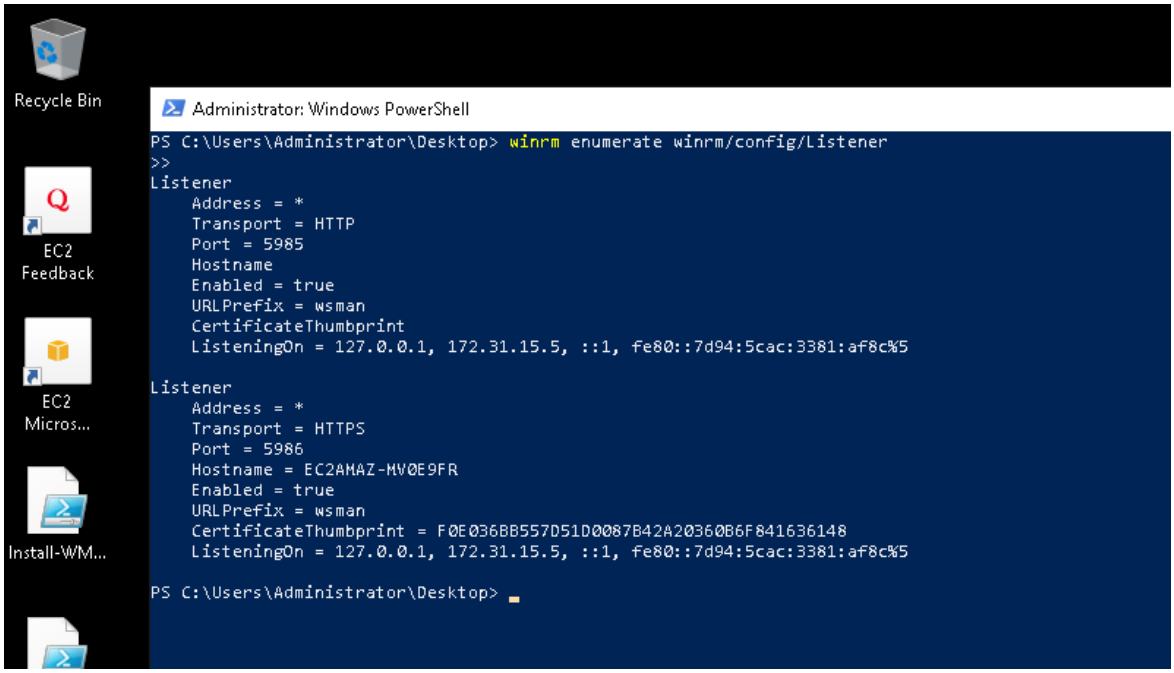
```
powershell.exe -ExecutionPolicy ByPass -File ConfigureRemotingForAnsible.ps1
```

6.3 Winrm Listener

Execute this command in Powershell to know about the winrm listener port details

winrm enumerate winrm/config/Listener

Winrm uses Port 5986 for HTTPS & 5985 for HTTP



7. Configure Windows Host in Ansible Server in /etc/ansible/hosts

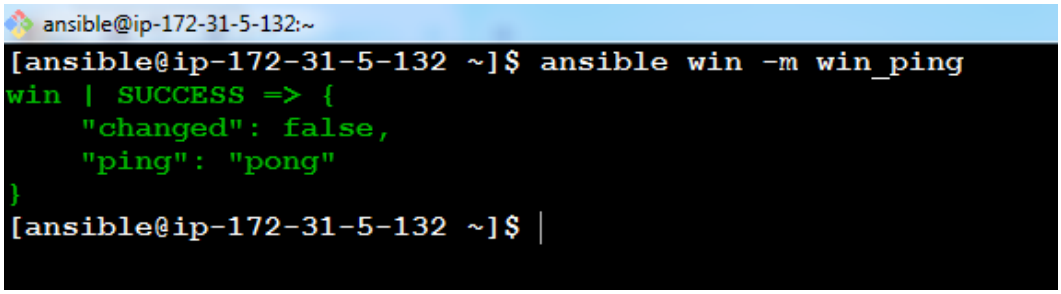
   [windowsServers]

   PRIVATE IP ansible_password=PASSWORD ansible_connection=winrm ansible_port=5986 ansible_user=Administrator ansible_winrm_server_cert_validation=ignore

   (Note: Replace PRIVATEIP & PASSWORD with your Windows Server 2019 Instance Credentials)

8. Ping Windows Server using the command

   ansible **windowsServers** –m win_ping

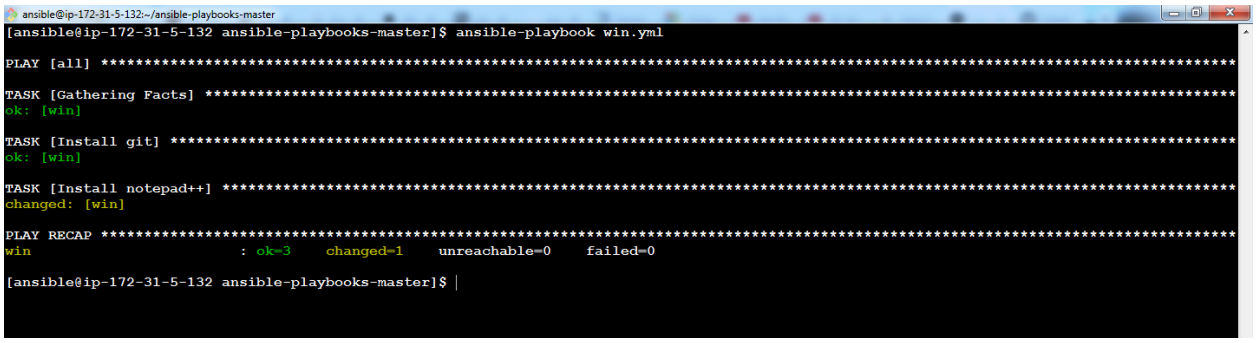


9. Write a playbook to Install Git & Notepad++ in Windows Server 2019

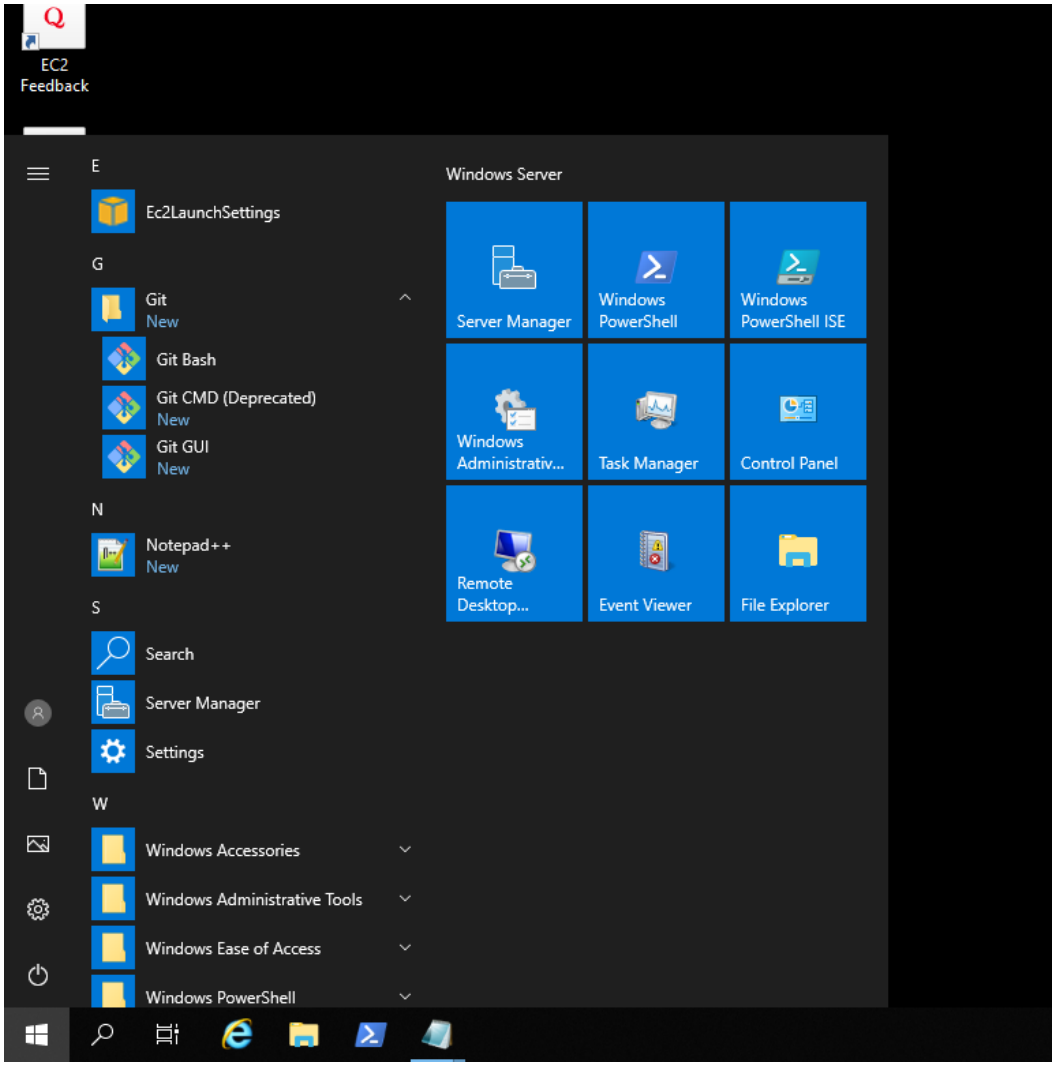
```
- hosts: windowsServers
  tasks:
   - name: Install git
     win_chocolatey:
      name: git
      state: present

   - name: Install notepad++
     win_chocolatey:
      name: notepadplusplus
      state: present
```

10. Execute Playbook to install Git and Notepad++ in Windows Server 2019

ansible-playbook win.yml



11 . Check the Installed Git & Notepad++ in Windows Server 2019



Ansible Tower

Ansible Tower (Red Hat® Ansible® Tower) is web based GUI tool with which you can managing your infrastructure centrally and control most aspects of infrastructure Configurations.

Ansible Tower is Ansible at a more enterprise level. Centralize your Ansible infrastructure from a modern UI, featuring role-based access control, job scheduling, and graphical inventory management.

Main feature that make Ansible Tower :

- Clean dashboard
- Manage inventory dynamically
- Real time Job status
- Job scheduling
- Integrate internal notification
- Role base access control (RABAC)
- Audit job and Tower resource
- Adhok remote command access (built in module)
- Store credential safely for different tool
- REST API to take it further
- Self- service UI

Prerequisites To Install Ansible Tower

The following are the pre-requisites to install Tower:
Ansible Tower is supported by the following operating systems:

- Red Hat Enterprise Linux 6 64-bit
- Red Hat Enterprise Linux 7 64-bit
- Red Hat Enterprise Linux 8 64-bit
- CentOS 6 64-bit
- CentOS 7 64-bit
- Ubuntu 12.04 LTS 64-bit
- Ubuntu 14.04 LTS 64-bit
- Ubuntu 16.04 LTS 64 bit

You should have the latest stable release of Ansible.
64-bit support required (kernel and runtime) and **20 GB hard disk**
For Amazon EC2: Instance size of m4.large or larger is required

Step 1: Update system and add EPEL repository

Update Package Manager

$ sudo yum -y update

Install EPEL Repository on RHEL

$ sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm -y

Confirm EPEL installation

$ sudo dnf repolist epel

Ansible Tower uses Ansible playbook to deploy itself so we also need Ansible installed.

$ sudo yum -y install ansible vim curl

Step 2: Download Ansible Tower archive

Download the latest Ansible Tower release.

$ mkdir /tmp/tower && cd /tmp/tower

$ curl -k -O https://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.tar.gz

Extract downloaded archive.

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

Step 3: Install Ansible Tower

Navigate to the created directory.

$ cd ansible-tower-setup*/

Edit inventory file to set required credentials(Admin Password & PG Password).

$ vim inventory

```
[tower]

localhost ansible_connection=local

[database]

[all:vars]

admin_password='admin123'

pg_host=''

pg_port=''

pg_database='awx'

pg_username='awx'

pg_password='db@123'

pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

# Isolated Tower nodes automatically generate an RSA key for authentication;

# To disable this behavior, set this value to false

# isolated_key_generation=true

# SSL-related variables

# If set, this will install a custom CA certificate to the system trust store.
```

Installation of Ansible Tower.

$ sudo ./setup.sh

Start Ansible Tower

$ sudo ansible-tower-service start

Step 4: Configure Ansible Tower

We will use the Web UI since this is the most preferred method by most new Ansible Tower users. Open your favorite browser point to your Ansible Tower **server IP or hostname via *https* protocol**.

Login as admin user and password set in the inventory file.
Once you are logged in, you need to configure Ansible Tower license. Browse to the license file and accept the terms. **If you don't have a license, get trial one** here.

Resources
http://www.yamllint.com/
https://valdhaus.co/writings/ansible-mac-osx/
http://binarynature.blogspot.in/2016/01/install-ansible-on-os-x-el-capitan_30.html

Follow below URLs for DevOps and Cloud Technologies Videos and Webinar URLs.

YouTube Channel
https://www.youtube.com/channel/UC-Jr307MbEREy8bG6McwD6w

Instagram
https://www.instagram.com/mithun_technologies_mt/

Facebook
https://www.facebook.com/mithuntechnologies

LinkedIn
https://www.linkedin.com/in/mithuntechnologies

Twitter
https://twitter.com/mithuntechno

Mithun Technologies Blog
http://mithuntechnologies-devops.blogspot.com/

Regards,

Mithun Technologies Admin Team,
+91-99809 23226