Ansible

Configuration Management tool

Introduction 3

Prerequisites 3

Install Ansible 4

Install on Mac OS: 4

Install on CentOS, RHEL, or Scientific Linux (YUM/RPM based machine): 4

Install on Ubuntu, Debian (APT based machines) 4

Ansible General Terms 5

Control Machine: 5

Inventory: 5

Playbooks: 5

Play: 5

Tasks: 5

Module: 5

Facts: 6

Handlers: 6

Variables: 6

Role: 7

Using AWS EC2 instance as node 9

Launch AWS EC2 instance: 13

# Introduction

Ansible is a configuration management tool. Configuration management tools are to keep the servers in a desired state. These tools are used to check if the server is in a desired state or not, if not then configuration management tool brings that server in the desired state i.e. correct packages are installed, configuration files contain the expected values, files are having expected, right services are running, and so on.

Along with the Ansible, there are other famous configuration management tools as well in this category Chef, Puppet, and Salt. With the most of the tools you will have to install client on the node/server you want to manage. Ansible is agent less tool, which means there is no need to install agent or client on the node. It is ssh based tool and you just need ssh connectivity from your machine to the nodes. Hence it is less complex than other tools. It is simple, doesn't mean that is just “for loop” over ssh. There are companies which are managing thousands of nodes using Ansible. Ansible connects to nodes in parallel and do the task. It uses ssh multiplexing for enhanced performance. Generally, Ansible is used to manage Linux nodes. Ansible 1.7 and greater also supports windows. But we will cover only Linux OS.

To manage the nodes, we need to write scripts in Ansible and these scripts are called playbooks. Playbooks are written in YAML format. YAML is a format like XML, JSON.

# Prerequisites

Ansible is python based tools but it is not necessary that you need to know python to use Ansible. You do not need to know python until you want to write your own modules. Modules, are the programs written to perform the tasks i.e. install package, check service, copy files, change permission and many more. In most of the cases you do not need to right the modules because there are more than 200 modules available which can fulfill your requirements.

As mentioned earlier that it is ssh based tool and used to manage Linux/Unix nodes so you should have basic idea about below:

* different Linux distros i.e. Ubuntu, CentOs, RedHat
* Package managers apt, yum
* ssh connectivity
* bash command line
* use sudo
* Services status, stop, start
* install package

# Install Ansible

Here, we will discuss that how to install Ansible on different OS.

## Install on Mac OS:

To install on MAC OS, you will need pip (Python Package Index) install on your machine. As, it is mentioned earlier that Ansible is a python based so you can install it using pip. If pip is not installed on your MAC then you can install using below command:

|  |
| --- |
| $ sudo easy\_install pip |

Once pip is install then you can install Ansible using below command:

|  |
| --- |
| $ sudo pip install ansible |

\* sudo is used to run a command with root privileges

## Install on CentOS, RHEL, or Scientific Linux (YUM/RPM based machine):

To install on the Linux OS which are YUM/RPM based we need to run below command as Ansible is already available in the official YUM/RMS repository.

|  |
| --- |
| $ sudo yum install ansible |

## Install on Ubuntu, Debian (APT based machines)

To install on APT based Linux distributions, we will have to add apt repository which contains Ansible package to apt package manager.

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

Then update the package manager index to reflect the changes we made.

|  |
| --- |
| $ sudo apt-get update |

Then Ansible can be installed using below command.

|  |
| --- |
| $ sudo apt-get install ansible |

You can check the installation using:

|  |
| --- |
| $ ansible –version |

# Ansible General Terms

There are lots of terms which are used while using or talking about Ansible. I will describe most of the terms in simple words.

## Control Machine:

Control machine is a machine/server from where you will run the Ansible commands. It is the machine which controls all other nodes/machines. You need to install Ansible on this machine.

## Inventory:

Inventory is the tem used for the machines/nodes which you want to manage. It is a file which contains the list of the nodes you want to control through Ansible. Generally, this file is called hosts files, but you can give any name and provide the name of the inventory file during the Ansible commands. You can also create different group of hosts in the inventory file. So in simple terms, it is a list of host names which you want to manage.

## Playbooks:

This is the file which do the all magic in Ansible. You write the script in YAML format and these scripts are called playbooks and are stored as .yml file. Playbook is a combination of plays, modules, tasks, handlers etc. Most of the time you will spend in writing the playbooks as it is the core of Ansible and everything you want to do with Ansible is written in playbooks.

## Play:

Play is a part of playbook. It is a combination of different modules and tasks. A play is defined to perform some common task on a host or on a group of hosts. There can be more than one play in a playbook. You can provide any name to paly but is it recommended to provide some useful name i.e. config web server

## Tasks:

A play can perform different tasks on the node/s. Tasks parameter is used to define different tasks in playbook. Each task contains a name and uses module to perform the desired task i.e. install package, restart service etc.

## Module:

Module is a program written is python. There are more than 200 modules available. These modules are used to perform a task. These modules are used with arguments in the playbook. And to use the module with Ansible command line you can specify module using –m parameter and argument using –a parameter.

## Facts:

When you will run a playbook or Ansible command on a node then first Ansible collects all the facts about that node i.e. CPU architecture, type of operating system, IP address, memory usage, disk usage and many more. This information is stored in the variables and those variables are called facts. These variables can be used in the playbook like any other variable.

## Handlers:

Handler is conditional forms that is used in Ansible. It is similar to task and perform some action but this task is called based on the condition. Handler is called from a task if that task has changed the state of the node. Task calls a handle by passing handler name as argument to to notify module.

## Variables:

You can define the variables in the playbook or in a separate file. These variables can be used in playbook. To define variable in a playbook you need to use “vars:” keyword. To include the variables defined in a file, you can include that file in the playbook using “vars\_files:” keyword. To access a variable in playbook, you should use pair of curly braces {{ variable\_name }}

Here is the example of a playbook which is having most of the terms, explained above:



## Role:

Roles are used to reuse the code you have written. It is like include statement of any programming language. Instead of writing the code again in your playbook you can use roles to include the things which you have already written. For example: you want to use a play which is written in another playbook then instead of writing the whole code again you can use role to include that play from another playbook to your current playbook. Similarly, you can include tasks, handlers, variables, and so on using roles. Roles are used to organize the structure. Instead of writing large playbooks, you should use roles to keep the playbook more structured and clean. So in simple terms, it is a process to break a playbook into multiple files.

To define a role, you will have to create a directory called *roles* to the same directory where your playbook is present (we can also define roles at global level). For example, to create a role my\_role, you should create a directory inside roles directory.

roles/my\_role directory should contain following files and directories.

*roles/my\_role/tasks/main.yml*

Tasks

roles/my\_role/files/

Holds files to be uploaded to hosts

*roles/my\_role/templates/*

Holds Jinja2 template files

*roles/my\_role/handlers/main.yml*

Handlers

*roles/my\_role/vars/main.yml*

Variables that shouldn’t be overridden

*roles/my\_role /defaults/main.yml*

Default variables that can be overridden

*roles/my\_role /meta/main.yml*

Dependency information about a role

These files are optional. For example, if your role doesn't contain any handlersthen there is no need to have handlers/main.yml file.

You can define roles in a playbook like below:

1. If role is not using any variable:



|  |
| --- |
| ---  - name: deploy my\_role on host\_group1  hosts: host\_group1  roles:  - role: my\_role |

1. If role is using variables:



|  |
| --- |
| ---  - name: deploy my\_role on host\_group1  hosts: host\_group1  vars\_files:  - secrets.yml  roles:  - role: my\_role  user: "{{ user\_name }}"  password: "{{ password }}" |

# Using AWS EC2 instance as node

There are different ways to define AWS EC2 instance as a node in your Ansible inventory file. To access a EC2 instance from command line through ssh we need below details:

1. User name to connect with EC2 instance
2. Host name of the instance (a public IP or public DNS name)
3. Identity file

You can get the username and hostname from the AWS management console. Identity file, you need to download while creating AWS EC2 instance. Now we can configure inventory file with these details and can control EC2 instance from Ansible.

There are 2 most commonly used ways.

1. Provide all the details in inventory file

You can create an inventory file i.e. hosts and put the details into that file as below

*<alias> ansible\_ssh\_host=<public\_ip> ansible\_ssh\_user=<user> ansible\_ssh\_private\_key\_file=<identity\_key\_file\_path>*

Then you can run command to check if you have established the connectivity or not.

For example: There are 2 AWS EC2 instances, one is using Ubuntu and another one is using redhat OS.



|  |
| --- |
| BANL141cc14d1:playbooks ngupta9$ cat hosts  ubuntu1404 ansible\_ssh\_host=50.112.173.252 ansible\_ssh\_user=ubuntu ansible\_ssh\_private\_key\_file=~/.ssh/ngupta9\_instance1.pem  redhat7.3 ansible\_ssh\_host=35.162.109.135 ansible\_ssh\_user=ec2-user ansible\_ssh\_private\_key\_file=~/.ssh/ngupta9\_instance1.pem  BANL141cc14d1:playbooks ngupta9$  BANL141cc14d1:playbooks ngupta9$ ansible ubuntu1404 -i hosts -m ping  ubuntu1404 | SUCCESS => {  "changed": false,  "ping": "pong"  }  BANL141cc14d1:playbooks ngupta9$  BANL141cc14d1:playbooks ngupta9$ ansible all -i hosts -m ping  ubuntu1404 | SUCCESS => {  "changed": false,  "ping": "pong"  }  redhat7.3 | SUCCESS => {  "changed": false,  "ping": "pong"  } |

You can make the entry like above and run the commands.

1. Keep the details in the ssh config files and use the host used in ssh config file into your inventory file. See screenshot:



|  |
| --- |
| 1. BANL141cc14d1:playbooks ngupta9$ cat hosts 2. ubuntu1404 3. redhat7.3 4. BANL141cc14d1:playbooks ngupta9$ 5. BANL141cc14d1:playbooks ngupta9$ ansible ubuntu1404 -i hosts -m ping 6. ubuntu1404 | SUCCESS => { 7. "changed": false, 8. "ping": "pong" 9. } 10. BANL141cc14d1:playbooks ngupta9$ ansible all -i hosts -m ping 11. ubuntu1404 | SUCCESS => { 12. "changed": false, 13. "ping": "pong" 14. } 15. redhat7.3 | SUCCESS => { 16. "changed": false, 17. "ping": "pong" 18. } 19. BANL141cc14d1:playbooks ngupta9$ |

# Launch AWS EC2 instance:

AWS EC2 instance can be launched through Ansible playbook. Below are the prerequisites for that:

* Ansible
* Python
* Python pip
* Python boto library

We are already having Ansible and python installed on the machine. We will install boto library.

|  |
| --- |
| $ sudo pip install --upgrade  Requirement already up-to-date: pip in /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg  $  $ sudo pip install boto  Collecting boto  Downloading boto-2.45.0-py2.py3-none-any.whl (1.3MB)  100% |################################| 1.4MB 416kB/s  Installing collected packages: boto  Successfully installed boto-2.45.0  $ |

To create EC2 instance we will need AWS access key and secret key. We can create a new user in AWS or can use a user already created user. You can get the keys using below steps:

* Go to AWS console
* Go to IAM (Identity and Access Management) inside services
* Click on users then you can create a new user or use an existing user
* Then create environment variable on your machine to use these keys



Now create a host file with below contents:

|  |
| --- |
| $ pwd  /Users/ngupta9/launch\_aws\_ec2/playbooks  $ cat hosts  [local]  localhost  [webserver]  $ |

Then create playbook *launch\_ec2\_aws.yml* (you can give any name of your choice), which will be used to launch AWS EC2 instance

|  |
| --- |
| ---  - name: Provision an EC2 Instance  hosts: local  connection: local  gather\_facts: False  tags: provisioning  # Necessary Variables for creating/provisioning the EC2 Instance  vars:  instance\_type: t2.micro  security\_group: webserver # Change the security group name here  image: ami-1633f379 # Change the AMI, from which you want to launch the server  region: eu-central-1 # Change the Region,  keypair: FMA # Change the keypair name, this is the key which I am already having in AWS  count: 1  # Task that will be used to Launch/Create an EC2 Instance  tasks:  #create a security group  - name: Create a security group  local\_action:  module: ec2\_group  name: "{{ security\_group }}"  description: Security Group for webserver Servers  region: "{{ region }}"  rules:  - proto: tcp  from\_port: 22  to\_port: 22  cidr\_ip: 0.0.0.0/0  - proto: tcp  from\_port: 80  to\_port: 80  cidr\_ip: 0.0.0.0/0  - proto: tcp  from\_port: 443  to\_port: 443  cidr\_ip: 0.0.0.0/0  rules\_egress:  - proto: all  cidr\_ip: 0.0.0.0/0  register: basic\_firewall  #Create EC2 instance  - name: Launch the new EC2 Instance  local\_action: ec2  group={{ security\_group }}  instance\_type={{ instance\_type}}  image={{ image }}  wait=true  region={{ region }}  keypair={{ keypair }}  count={{count}}  register: ec2  # To add the created EC2 instance in inventory file  - name: Add the newly created EC2 instance(s) to the local host group (located inside the directory)  local\_action: lineinfile  dest="./hosts"  regexp={{ item.public\_ip }}  insertafter="[webserver]" line={{ item.public\_ip }}  with\_items: "{{ ec2.instances }}"  - name: Wait for SSH to come up  local\_action: wait\_for  host={{ item.public\_ip }}  port=22  state=started  with\_items: "{{ ec2.instances }}"  - name: Add tag to Instance(s)  local\_action: ec2\_tag resource={{ item.id }} region={{ region }} state=present  with\_items: "{{ ec2.instances }}"  args:  tags:  Name: webserver |

Some of the things which you should replace are the variables defined inside *vars:* section based on your requirements.

Then define the access and secret key variables. (Please replace the values with your key values)

|  |
| --- |
| $ export AWS\_ACCESS\_KEY\_ID="AKIAIZCDSEASFAFRFT2VQ"  $ export AWS\_SECRET\_ACCESS\_KEY="2Fw7QzC9QcVQg+HgvNoMuYxpKXlf4v6dKSX7njeb" |

Then run the playbook:

|  |
| --- |
| $ ansible-playbook -i hosts launch\_ec2\_aws.yml |

You will get output like below:



|  |
| --- |
| BANL141cc14d1:playbooks ngupta9$ ansible-playbook -i hosts launch\_ec2\_aws.yml  PLAY [Provision an EC2 Instance] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  TASK [Create a security group] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ok: [localhost -> localhost]  TASK [Launch the new EC2 Instance] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  changed: [localhost -> localhost]  [local]  TASK [Add the newly created EC2 instance(s) to the local host group (located inside the directory)] \*\*\*  changed: [localhost -> localhost] => (item={u'kernel': None, u'root\_device\_type': u'ebs', u'private\_dns\_name': u'ip-172-31-13-65.eu-central-1.compute.internal', u'public\_ip': u'35.157.157.99', u'private\_ip': u'172.31.13.65', u'id': u'i-0b96469c013f3ed60', u'ebs\_optimized': False, u'state': u'running', u'virtualization\_type': u'hvm', u'architecture': u'x86\_64', u'ramdisk': None, u'block\_device\_mapping': {u'/dev/sda1': {u'status': u'attached', u'delete\_on\_termination': True, u'volume\_id': u'vol-011ecae66451545b4'}}, u'key\_name': u'FMA', u'image\_id': u'ami-1633f379', u'tenancy': u'default', u'groups': {u'sg-88e70ee3': u'webserver'}, u'public\_dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'state\_code': 16, u'tags': {}, u'placement': u'eu-central-1b', u'ami\_launch\_index': u'0', u'dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'region': u'eu-central-1', u'launch\_time': u'2017-02-14T10:15:55.000Z', u'instance\_type': u't2.micro', u'root\_device\_name': u'/dev/sda1', u'hypervisor': u'xen'})  TASK [Wait for SSH to come up] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ok: [localhost -> localhost] => (item={u'kernel': None, u'root\_device\_type': u'ebs', u'private\_dns\_name': u'ip-172-31-13-65.eu-central-1.compute.internal', u'public\_ip': u'35.157.157.99', u'private\_ip': u'172.31.13.65', u'id': u'i-0b96469c013f3ed60', u'ebs\_optimized': False, u'state': u'running', u'virtualization\_type': u'hvm', u'architecture': u'x86\_64', u'ramdisk': None, u'block\_device\_mapping': {u'/dev/sda1': {u'status': u'attached', u'delete\_on\_termination': True, u'volume\_id': u'vol-011ecae66451545b4'}}, u'key\_name': u'FMA', u'image\_id': u'ami-1633f379', u'tenancy': u'default', u'groups': {u'sg-88e70ee3': u'webserver'}, u'public\_dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'state\_code': 16, u'tags': {}, u'placement': u'eu-central-1b', u'ami\_launch\_index': u'0', u'dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'region': u'eu-central-1', u'launch\_time': u'2017-02-14T10:15:55.000Z', u'instance\_type': u't2.micro', u'root\_device\_name': u'/dev/sda1', u'hypervisor': u'xen'})  TASK [Add tag to Instance(s)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  changed: [localhost -> localhost] => (item={u'kernel': None, u'root\_device\_type': u'ebs', u'private\_dns\_name': u'ip-172-31-13-65.eu-central-1.compute.internal', u'public\_ip': u'35.157.157.99', u'private\_ip': u'172.31.13.65', u'id': u'i-0b96469c013f3ed60', u'ebs\_optimized': False, u'state': u'running', u'virtualization\_type': u'hvm', u'architecture': u'x86\_64', u'ramdisk': None, u'block\_device\_mapping': {u'/dev/sda1': {u'status': u'attached', u'delete\_on\_termination': True, u'volume\_id': u'vol-011ecae66451545b4'}}, u'key\_name': u'FMA', u'image\_id': u'ami-1633f379', u'tenancy': u'default', u'groups': {u'sg-88e70ee3': u'webserver'}, u'public\_dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'state\_code': 16, u'tags': {}, u'placement': u'eu-central-1b', u'ami\_launch\_index': u'0', u'dns\_name': u'ec2-35-157-157-99.eu-central-1.compute.amazonaws.com', u'region': u'eu-central-1', u'launch\_time': u'2017-02-14T10:15:55.000Z', u'instance\_type': u't2.micro', u'root\_device\_name': u'/dev/sda1', u'hypervisor': u'xen'})  PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  localhost : ok=5 changed=3 unreachable=0 failed=0  BANL141cc14d1:playbooks ngupta9$ |

You can go to AWS console and verify that new AWS EC2 instance got created.

