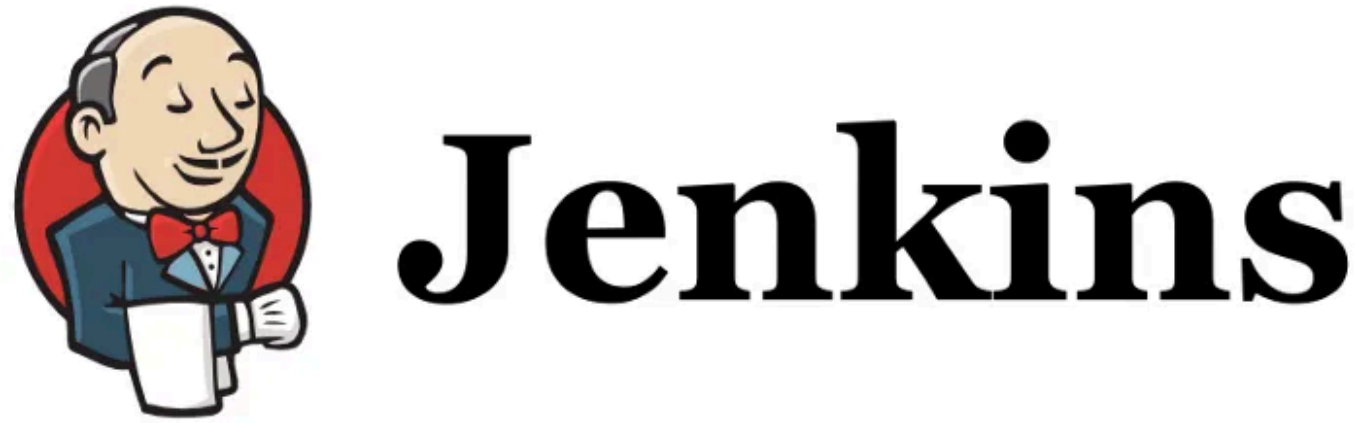
How to run an ansible playbook using Jenkins?

Dhruvin Soni · Follow

Published in DevopSquare · 10 min read · Aug 11, 2021

What is Jenkins?

Jenkins is a free and open-source **CI/CD automation server**. It helps to automate the parts of the software development lifecycle i.e building, testing, and deploying the code to various servers.



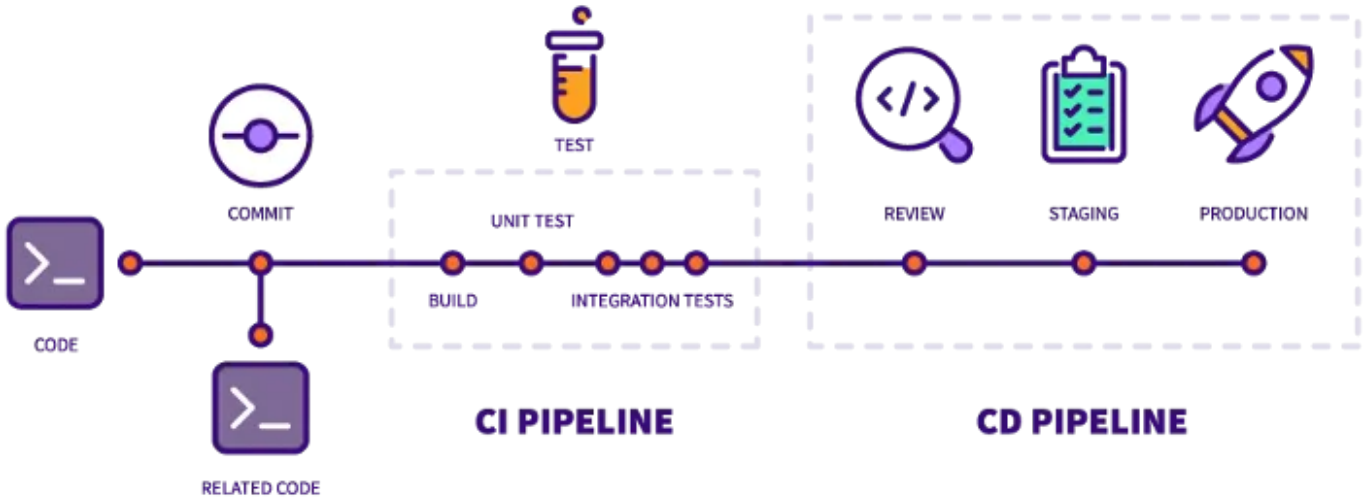
Jenkins

CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment.

Continuous Integration works by pushing small code chunks to your application's codebase hosted in a Git repository, and to every push, run a pipeline of scripts to build, test, and validate the code changes before merging them into the main branch.

Continuous Delivery and **Deployment** consist of a step further CI, deploying your application to production at every push to the default branch of the repository.

Read more about Jenkins from **here**



CI/CD Overview

What is Ansible?

Ansible is an open-source software provisioning, configuration management, and deployment tool. It runs on many Unix-like systems and can configure both Unix-like systems as well as Microsoft Windows. Ansible uses SSH protocol in order to configure the remote servers. Ansible follows the push-based mechanism to configure the remote servers.

Ansible

Prerequisites:

- A web-server that has **Jenkins** & **Ansible** installed in it
- Basic understanding of **Ansible** & **Jenkins**
- Basic understanding of **AWS**
- AWS **Access Key** & **Secret Key**

What is Ansible Vault?

Ansible Vault is a feature of ansible that allows you to keep **sensitive data** such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. You can create the new file as encrypted or can also modify the existing file as encrypted.

> *To create a new encrypted file using ansible vault you need to run the below command. It will ask for a password so, please give a password that you can easily remember*

```
ansible-vault create cred.yml
```

> *So, now add the below content in the newly created* `cred.yml` *file.*

```
aws_access_key: <your aws access key>
aws_secret_key: <your aws secret key>
```

After that, we need to configure the **ansible.cfg** file. But before configuring the file please make sure the below thing

> *Make sure in your directory you have the AWS key-pair in "pem" format.*
>
> *Edit the ansible.cfg and add below content in it*

```
[defaults]
host_key_checking = false
remote_user = ec2-user
ask_pass = false
private_key_file = <pem file's path>

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Now, Let's understand the code.

- For logging in to the EC2 instance we need the user which is defined by **remote_user.** In our case we are going to launch Amazon Linux 2 Instance so, our user will be **ec2-user**
- We don't need any password to log in so, that's why we set **ask_pass** to **false**
- **private_key_file** defines the path of the private key for SSH connection
- In the **privilege_escalation** section, we defined certain parameters which give ec2-user the sudo privileges

So, now let's start configuring the playbook. For this tutorial, I will break down the playbook into small sections. So, create a file with the **.yml** extension in your directory.

Step 1:- Start your playbook by defining below code

```
---
- hosts: localhost
  connection: local
  gather_facts: no
  vars_files:
  - cred.yml
```

- Here I have selected the **hosts** as the **local host** because I am running ansible on the local machine.
- I have defined the **connection** as **local** because I am running ansible on the local machine.
- I have defined **gather_facts** as **no** because I don't want to gather facts for my local machine
- **vars_file** contains the name of the variable file which we have created earlier that stores our AWS credentials

Step 2:- Define the environment variables

```
vars:
  title: "Demo"
  vpc_name: "{{ title }} VPC"
  igw_name: "{{ title }} IGW"
  subnet_name: "{{ title }} Subnet"
  acl_name: "{{ title }} ACL"
  instance_name: "{{ title }} Instance"
  security_group_name: "{{ title }} Security Group"
  route_table_name: "{{ title }} Route Table"

vpcCidrBlock: "10.0.0.0/16"
  subNetCidrBlock: "10.0.1.0/24"
  portCidrBlock: "0.0.0.0/0"
  destinationCidrBlock: "0.0.0.0/0"
```

```
state: "present"
  zone: "us-west-1a"
  region: "us-west-1"
```

- I have defined some environment variables which we are going to use in the next steps
- For this tutorial, I have used the **us-west-1** region you can use another region if you want.

Step 3:- Create the **VPC**

```
tasks:
- name: Create VPC
    ec2_vpc_net:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      name: "{{ vpc_name }}"
      cidr_block: "{{ vpcCidrBlock }}"
      region: "{{ region }}"
      dns_support: "yes"
      dns_hostnames: "yes"
      tenancy: "default"
      state: "{{ state }}"
      resource_tags:
        Name: "{{ vpc_name }}"
    register: vpc_result
```

- We can define the list of tasks after `tasks:` parameter
- `ec2_vpc_net` is a module for **AWS VPC**
- For `aws_access_key, aws_access_key, name, cidr_block, region, state,` I have used the value from the environment variables
- By setting `dns_support` & `dns_hostnames` to **yes** it will assign the DNS

- You can choose either **default** or **dedicated** `tenancy`
- `register` will store the output in the `vpc_result` variable

Step 4:- Create an Internet Gateway

```
- name: Create Internet Gateway
    ec2_vpc_igw:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      region: "{{ region }}"
      state: "{{ state }}"
      tags:
        Name: "{{ igw_name }}"
    register: igw_result
```

- `ec2_vpc_igw` is a module for **AWS Internet Gateway**
- For `aws_access_key, aws_access_key, region, state` I have used the value from the environment variables
- `vpc_result.vpc.id` will give the ID of the newly created VPC
- `register` will store the output in the `igw_result` variable

Step 5:- Create the Subnet

```
- name: Create Subnet
    ec2_vpc_subnet:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      region: "{{ region }}"
      state: "{{ state }}"
      az: "{{ zone }}"
```

```
      cidr: "{{ subNetCidrBlock }}"
      map_public: "yes"
      resource_tags:
        Name: "{{ subnet_name }}"
    register: subnet_result
```

- `ec2_vpc_subnet` is the module for **AWS Subnet**
- For `aws_access_key, aws_access_key, region, state, zone, cidr` I have used the value from the environment variables
- `vpc_result.vpc.id` will give the ID of the newly created VPC
- `register` will store the output in the `subnet_result` variable
- By defining `map_public` to **yes** it will automatically assign the **public IP** to the instances which are going to launch in this subnet

Step 6:- Create the Security Group

```
- name: Create Security Group
    ec2_group:
      name: "{{ security_group_name }}"
      description: "{{ security_group_name }}"
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      region: "{{ region }}"
      state: "{{ state }}"
      tags:
        Name: "{{ security_group_name  }}"
      rules:
        - proto: tcp
          from_port: 80
          to_port: 80
          cidr_ip: "{{ portCidrBlock }}"
        - proto: tcp
          from_port: 443
          to_port: 443
          cidr_ip: "{{ portCidrBlock }}"
        - proto: tcp
```

```
          from_port: 22
          to_port: 22
          cidr_ip: "{{ portCidrBlock }}"

  rules_egress:
        - proto: tcp
          from_port: 80
          to_port: 80
          cidr_ip: "{{ portCidrBlock }}"
        - proto: tcp
          from_port: 443
          to_port: 443
          cidr_ip: "{{ portCidrBlock }}"
        - proto: tcp
          from_port: 22
          to_port: 22
          cidr_ip: "{{ portCidrBlock }}"
    register: security_group_result
```

- `ec2_group` is the module for AWS Security Group
- For `aws_access_key, aws_access_key, region, state, cidr & name` I have used the value from the environment variables
- `vpc_result.vpc.id` will give the ID of the newly created VPC
- `rules` will create the inbound rules
- `rules_egress` will create the outbound rules
- `register` will store the output in the `security_group_result` variable
- I have allowed the inbound & outbound connection for port `22, 443 & 80`

Step 7:- Create the NACLs

```
- name: Create Network ACLs
    ec2_vpc_nacl:
      name: "{[ acl_name }}"
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
```

```
      vpc_id: "{{ vpc_result.vpc.id }}"
      region: "{{ region }}"
      state: "{{ state }}"
      subnets: [ "{{ subnet_result.subnet.id }}" ]
      tags:
        Name: "{{ acl_name }}"
      ingress:
      # rule no, protocol, allow/deny, cidr, icmp_type, icmp_code,
port from, port to
        - [100, 'tcp', 'allow', '0.0.0.0/0', null, null, 0, 65535]

      # rule no, protocol, allow/deny, cidr, icmp_type, icmp_code,
port from, port to
      egress:
        - [100, 'all', 'allow', '0.0.0.0/0', null, null, 0, 65535]
```

- `ec2_vpc_nacl` is the module for the **AWS NACLs**
- For `aws_access_key, aws_access_key, region, state, cidr & name` I have used the value from the environment variables

- `[ "{{ subnet_result.subnet.id }}" ]` will give the IDs of subnet on which it will assign this NACL
- I have allowed all the inbound & outbound connection

Step 8:- Create the Route Table

```
- name: Create Route Table
    ec2_vpc_route_table:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      region: "{{ region }}"
      state: "{{ state }}"
      tags:
        Name: "{{ route_table_name }}"
      subnets: [ "{{ subnet_result.subnet.id }}" ]
      routes:
```

```
        - dest: "{{ destinationCidrBlock }}"
          gateway_id: "{{ igw_result.gateway_id }}"
    register: public_route_table
```

- `ec2_vpc_route_table` is the module for **AWS Route Table**
- For `aws_access_key, aws_access_key, region, state, cidr & name` I have used the value from the environment variables
- `vpc_result.vpc.id` will give the ID of the newly created VPC
- `[ “{{ subnet_result.subnet.id }}” ]` will give the ID of the newly created Subnet
- `igw_result.gateway_id` will give the ID of the newly created Internet Gateway
- `register` will store the output in the `public_route_table` variable

Step 9:- Creating the **EC2 instance**

```
# Creating EC2 Instance
  - name: Create EC2 Instance
    ec2:
      aws_access_key: "{{ aws_access_key }}"
      aws_secret_key: "{{ aws_secret_key }}"
      instance_type: t2.micro
      image: ami-04b6c97b14c54de18
      wait: yes
      region: "{{ region }}"
      group: "{{ security_group_name }}"
      key_name: tests
      count: 1
      vpc_subnet_id:  "{{ subnet_result.subnet.id }}"
      assign_public_ip: yes
      instance_tags:
        Name: "{{ instance_name }}"
    register: ec2_result
```

- `ec2` is the module for **AWS EC2 Instance**
- For `aws_access_key, aws_access_key, region` I have used the value from the environment variables
- `image` is for the AMI. Here I have used Amazon Linux2 AMI
- By defining `wait` to **yes** it will wait for the EC2 instance to fully come up
- `key_name` is the SSH key that we are going to use. Make sure that you already have the SSH key in your **AWS** account
- `count` is the number of EC2 instances that we are going to launch
- `security_group_name` will give attach the created Security Group to the instance
- `subnet_result.subnet.id` will launch the EC2 instance in the newly created subnet
- By defining `assign_public_ip` to **yes** it will assign the public IP to the EC2 instance
- `register` will store the output in the `ec2_result` variable

Step 10:- Add newly created instance to the **host group**

- Now that we have successfully created the EC2 instance so, in order to configure it we need to add that instance into the **host group**

```
- name: Add new instance to host group
    add_host:
      hostname: "{{ item.public_ip  }}"
      groupname: webserver
    with_items: "{{ ec2_result.instances }}"
```

- `add_host` is the module for adding EC2 instance to the **host group**
- `item.public_ip` will give the public IP of the newly created instance
- `webserver` is the group name
- If we have created multiple EC2 instances then `with_items: “{{ ec2_result.instances }}”` will give the IP address of all the EC2 instances

Step 11:- Wait for the SSH port to be available

```
- name: Wait for SSH to come up
    wait_for:
      host: "{{ item.public_ip }}"
      port: 22
      delay: 10
      state: started
    with_items: "{{ ec2_result.instances }}"
```

- `wait_for` is the module which will wait for condition before continuing
- In the above module, we are waiting for port 22 to fully come up
- `item.public_ip` will give the public IP of the host
- `port` is 22
- `delay` will wait for 10 seconds to continue
- `state` When checking a port started will ensure the port is open
- If we have created multiple EC2 instances then `with_items: “{{ ec2_result.instances }}”` will give the IP address of all the EC2 instances

Step 12:- Configuring the **server**

```
# Deploying Apache Webserver
- hosts: webserver
  remote_user: ec2-user
  become: yes
  gather_facts: no
  pre_tasks:
    - name: 'install python'
      raw: 'sudo apt-get -y install python'
  tasks:
    - name: Install Apache
      yum:
        name: httpd
        state: present
    - service:
        name: httpd
        state: started
        enabled: yes
```

- In the above code, we are installing python and apache webserver in the newly created server
- `pre_tasks` will execute before the `tasks` module. In the code, I have installed the **python package** before installing the **apache webserver.** We need to ensure that Python gets installed the first thing before any other task
- `raw` will allow running any command in the server
- In `tasks` section I have installed **apache webserver**

Step 13:- Create a **variable** file

- Run the below command to create a variable file for storing the AWS Access key & Secret Key

```
ansible-vault create cred.yml
```

- The above command will ask for a password so, give the password and add the below content to it

```
aws_access_key= <Your-Access-Key>
aws_secret_key= <Your-Secret-Key>
```

Step 14:- Create a password file

- Create a file called `password.txt` and add your ansible vault password in it

Step 15:- Store all the files in **GitHub**

- Create a Repository on GitHub and push all the files in that Repository

Now, Let's create a pipeline in **Jenkins**

Step 1:- Install **Ansible** plugin

- Go to **Manage Plugins -> Search for Ansible -> Install Ansible Plugin -> Restart Jenkins**

Step 2:- Add **Ansible's** path in **Jenkins**

- Go to **Global tool configuration -> Search for Ansible -> Add path of ansible -> Click on Save**

Step 3:- Create **Pipeline**

- Create a new Pipeline Job
- Under the pipeline, section add the below content in it & Click on Save

```
pipeline{
  agent any
  stages{

      stage("Git Checkout"){
        steps{
          sh 'git clone <Your-Repository-URL>'
        }
      }

      stage("Run an ansible playbook"){
        steps{
          sh 'ansible-playook <playbook-name.yml> --vault-password-
file <password file>'
        }
      }
   }
}
```

- The above code will create **2 stage**
- First stage will fetch the code from **GitHub**
- Second stage will run the **ansible playbook**
- After the Jenkins **JOB** is completed successfully, You can check the below resources in your AWS account.

1. **VPC**
2. **Public Subnet**
3. **Security Group**
4. **Internet Gateway**

5. **Route Table**
6. **EC2 Instance**
7. **NACLs**

- You can verify the apache webserver's output by navigating `http://ec2-ip` you should see the default apache web server's output.

That's it now, you have learned how to create various resources in AWS using Ansible. You can further explore ansible from **here**.

You can find the entire code in **GitHub** account also

Feel free to check out my other repositories also.

If you found this guide helpful then do click on 👏 the button and also feel free to drop a comment.

Follow for more stories like this 😊

Jenkins | Ansible | Ci Cd Pipeline | DevOps