Project Documentation- Course-1 End project

Project Title: **Automating Infrastructure using Terraform.**
Completion Date: 05th August 2023

Project Summary:

DevOps methodologies have multiple excellent tools for developers who want to deploy their applications quickly and easily to the cloud. It automates many steps involved in deployment, so you can focus on developing your application instead of worrying about the infrastructure.

Development Platform: Terraform/aws _instance

Developer: Vajinder Kumar
+91 9711819948
Portal.vajinder@gmail.com

Content

Abstract

Project Title: DevOps Certification training project-1

DevOps Certification training project-1 is a valuable tool for developers who want to deploy applications quickly, easily, and securely to the cloud. It automates many steps in the deployment process, is affordable, and can be scaled to meet the needs of growing businesses.

Here are some of the benefits of using DevOps Certification training project-1:

- It can help you save time by automating many of the tasks involved in deploying applications to the cloud.
- It is easy to use, even for developers who are not familiar with cloud computing.
- It is secure, with Private keys and Public keys.
- It is scalable, so you can easily add more applications or users as your business grows.

If you are looking for a tool to help you deploy applications to the cloud, DevOps Certification training project-1 is a great option.

Tools Used

1. Terraform
2. AWS Cli
3. Docker
4. Jenkins
5. Pyhthon

Problem Statement

Nowadays, developers need to know a lot of skills to deploy their applications on the cloud. This is because cloud computing is a complex and ever-changing field. Developers need to be able to understand and use a variety of cloud-based services, such as computing, storage, networking, and databases. They also need to be able to manage and monitor their applications in the cloud.

Here are some of the skills that developers need to know to deploy their applications on the cloud:

- Cloud computing fundamentals
- Cloud-based services
- Application management and monitoring
- Security and Compliance
- Networking and storage
- DevOps

Solution specification

This project automated essential steps by creating HCL scripts that automatically provision the server. These scripts also provide the necessary information to configure further, such as the default password for Jenkins and the public IP of the instance. Additionally, an Ansible script was created to automate deployment over Docker.

Development

AWS:

aws cli access is needed to create resources in aws cloud. Install aws cli and then get your access key and secret key from the aws console. And set up with the command in the screenshot below.

Terraform:

Run bellow scripts through command line with commands

- **notepad.demo.tf – open the script file**
- **terraform init – To in initiated provider**
- **terraform plan- to check plan**
- **terraform validate – to validate the script**
- **Terraform- apply – to apply the changes**
- **Terraform fmt – to set the format**
- **Terraform destroy- to delete all existing infrastructure.**
- **Terraform providers - to check initiated providers.**

```
porta@Spider MINGW64 /
$ cd /d

porta@Spider MINGW64 /d
$ cd terraform-demo

porta@Spider MINGW64 /d/terraform-demo
$ cd script

porta@Spider MINGW64 /d/terraform-demo/script
$ notepad demo.tf
```

```
porta@Spider MINGW64 /d
$ cd terraform-demo

porta@Spider MINGW64 /d/terraform-demo
$ cd script

porta@Spider MINGW64 /d/terraform-demo/script
$ notepad demo.tf

porta@Spider MINGW64 /d/terraform-demo/script
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/local from the dependency lock file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/local v2.4.0
- Using previously-installed hashicorp/tls v4.0.4
- Using previously-installed hashicorp/aws v5.11.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
```

```
porta@Spider MINGW64 /d/terraform-demo/script
$ terraform validate
Success! The configuration is valid.


porta@Spider MINGW64 /d/terraform-demo/script
$ terraform plan
tls_private_key.demo-private: Refreshing state... [id=c1cab06df1979092791074b801498e474dd9db19]
local_file.key-gen: Refreshing state... [id=958dfd49f9467b19c33e57db1ee940a408d600ad]
aws_key_pair.demo-key: Refreshing state... [id=demo-key]
aws_security_group.project-sg: Refreshing state... [id=sg-0e0d5f6fe2f1161a6]
aws_instance.project-ec2: Refreshing state... [id=i-0ac24d42f6d258fe4]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes
are needed.
```

$terraform init:

This command will download all the necessary providers which need to create resources mentioned in below scripts.

$terraform apply -auto-approve:

This command will run all below mentioned scripts and create resources mentioned in below scripts.

project-sg.tf:

This script will create a security group in aws.In which all the inbound and outbound rules are specified as per requirements.so that we can access the app from outside.The open ports are 22,80,8080.

demo-key.tf:

This script will create a pair of private and public keys. The ke needs to connect with the server through ssh.

project-ec2.tf :

This script will create an eac2 instance. Attach the key to the instance. And provision the Instance according to the shells scripts written in **installation-scripts.sh.**

```
porta@Spider MINGW64 /d/Terraform-demo/script
$ terraform apply
tls_private_key.demo-private: Refreshing state... [id=f0c4230e860f7d02d02ed538807d266f6e3e4a7b]
local_file.key-gen: Refreshing state... [id=e736f73d6a06c46b8a79fea65d78224b0fc20ce1]
aws_security_group.project-sg: Refreshing state... [id=sg-09f6521bab1b07285]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.project-ec2 will be created
  + resource "aws_instance" "project-ec2" {
      + ami                                  = "ami-0261755bbcb8c4a84"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = true
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "demo-key"
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = [
          + "project-sg-20230805113358930300000001",
        ]
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "project-ec2"
```

Provisioner:

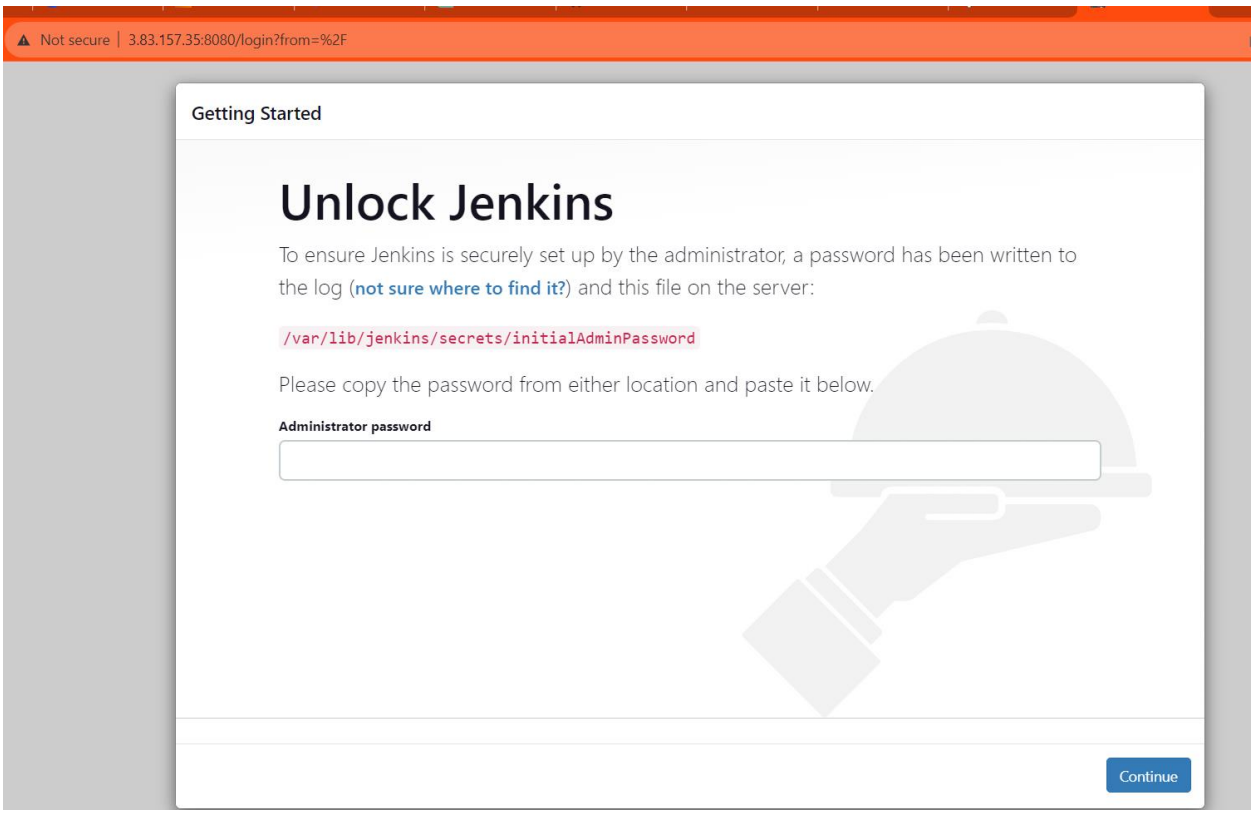
It is a block written in **dem-ec2.tf** which will help the resource created equipped with necessary tools. It will install **Jenkins**, **Docker** in the instance. As well as install the **Ansible plugin** into the jenkins.

For this I have set up the following files.

installation-jenkins.sh: This will install jenkins in the instance.

```
ubuntu@ip-172-31-83-188:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 99 not upgraded.
Need to get 95.9 MB of archives.
After this operation, 99.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

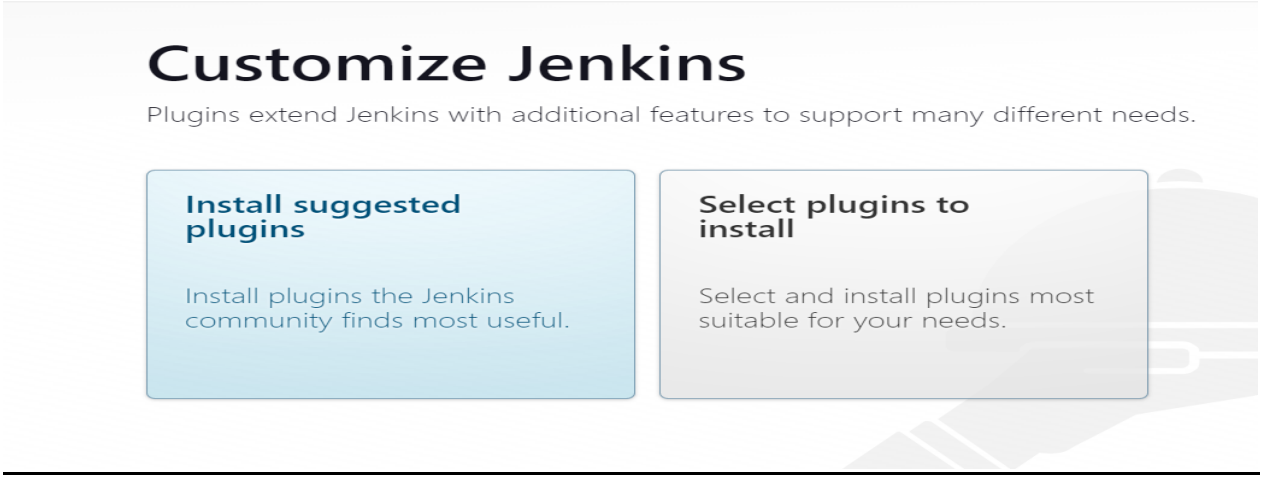
As jenkins works on 8080 which we need to enable for instance. Then we need to check in browser with URL- http://3.83.157.35:8080/login?from=%2F



Then take the password from the mentioned path by using below command & enter to login Jenkins window & finds jenkins

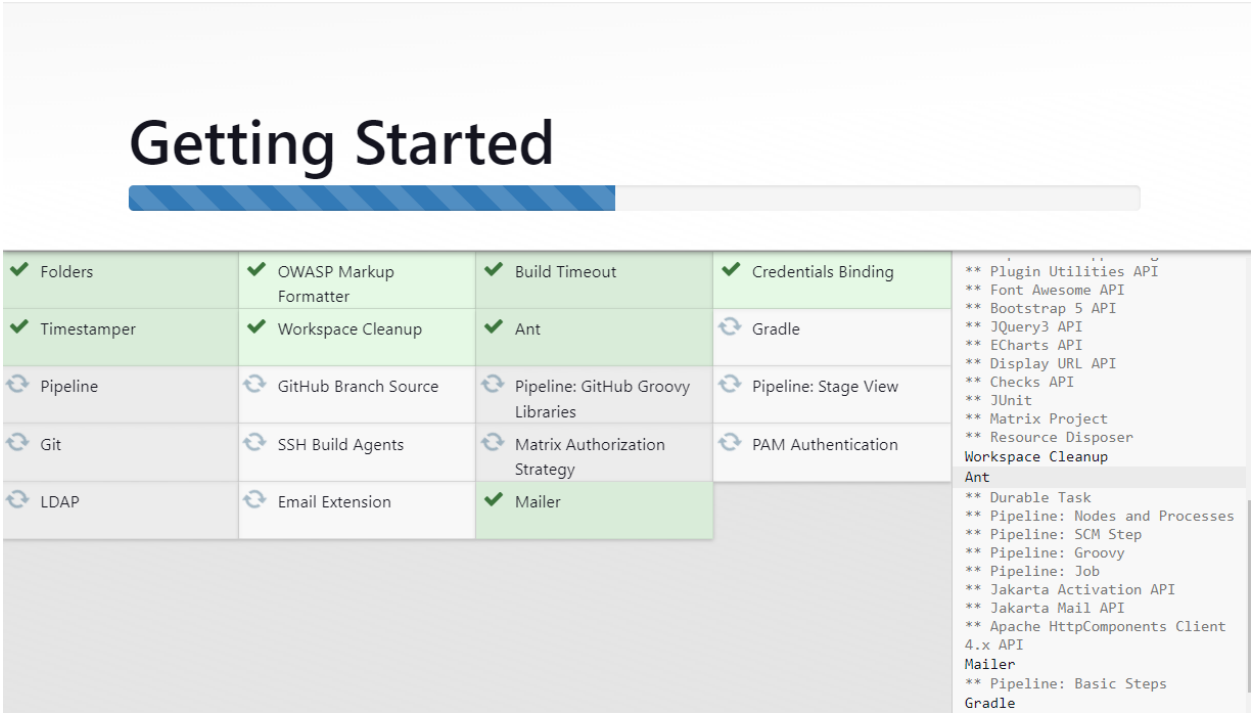
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Getting Started

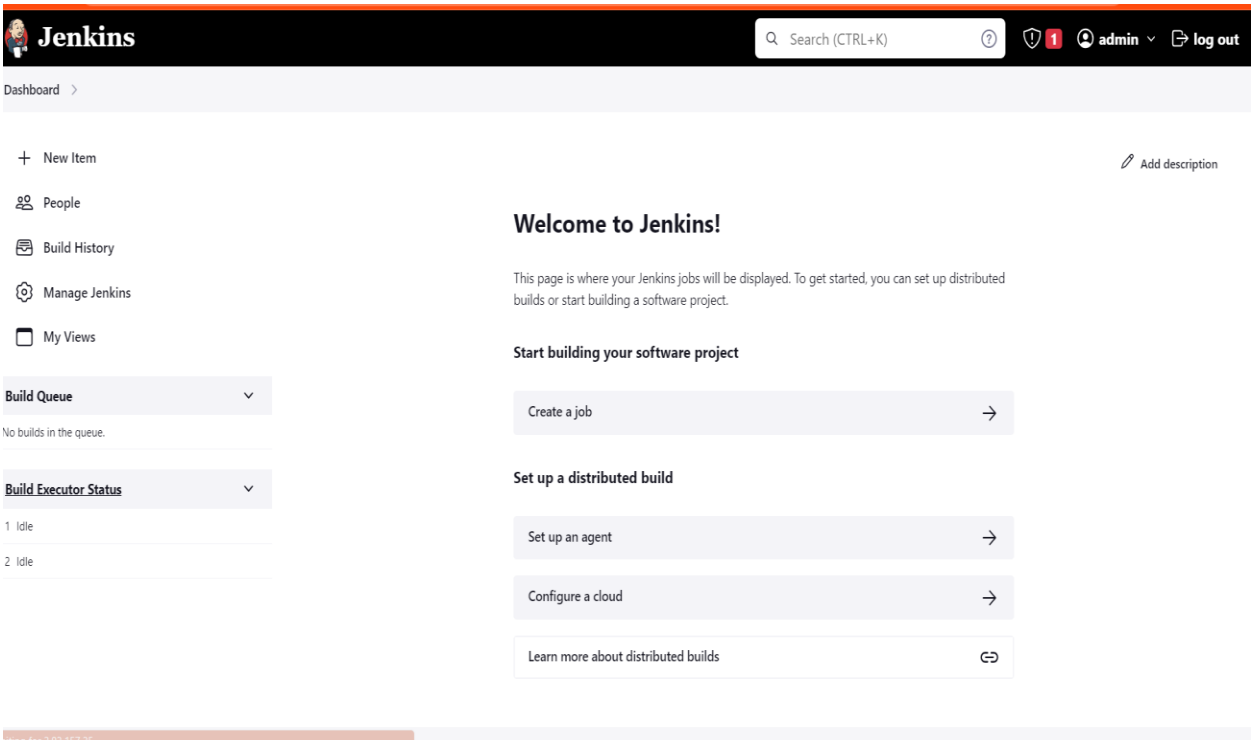


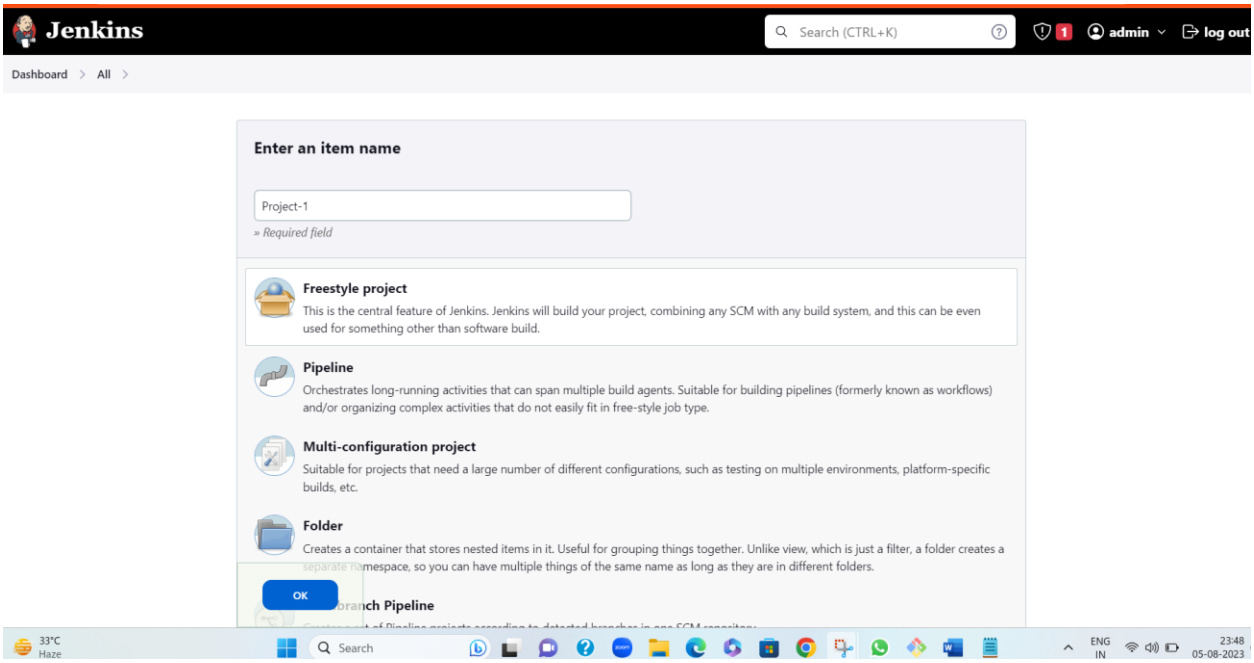
In next window need to install the suggested plugin or as per the requirement

Getting Started



Click on save and continue & Jenkins is ready to use for create project & ansible scripts, pipeline etc.





So in this way, we can create our own project & manage the resource in an effective way.

installation-docker.sh: This will install docker in the instance. As well as it will set the permissions for `/var/run/docker.sock` so that it won't throw an error while running the container from Ansible.
setting.sh: This script will install the **Ansible plugin** into Jenkins as well as it will show the default password so that we can unlock Jenkins from the browser.

Docker installation

- docker –version – to check docker version if its installed
- sudo apt install docker.io- Docker installation

ubuntu@ip-172-31-83-188:~$ docker --version
Docker version 20.10.25, build 20.10.25-0ubuntu1~20.04.1

```
ubuntu@ip-172-31-83-188:~$ whoami
ubuntu
ubuntu@ip-172-31-83-188:~$ touch test1
ubuntu@ip-172-31-83-188:~$ sudo docker version
sudo: docker: command not found
ubuntu@ip-172-31-83-188:~$ docker --version

Command 'docker' not found, but can be installed with:

sudo snap install docker     # version 20.10.24, or
sudo apt  install docker.io  # version 20.10.25-0ubuntu1~20.04.1

See 'snap info docker' for additional versions.

ubuntu@ip-172-31-83-188:~$
ubuntu@ip-172-31-83-188:~$ sudo apt  install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base libidn11 pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io libidn11 pigz runc ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 99 not upgraded.
Need to get 67.1 MB of archives.
After this operation, 294 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1 [57.4 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 bridge-utils amd64 1.6-2ubuntu1 [30.5 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 runc amd64 1.1.7-0ubuntu1~20.04.1 [3819 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 containerd amd64 1.7.2-0ubuntu1~20.04.1 [32.5 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 dns-root-data all 2019052802 [5300 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 libidn11 amd64 1.33-2.2ubuntu2 [46.2 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 dnsmasq-base amd64 2.80-1.1ubuntu1.7 [315 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/universe amd64 docker.io amd64 20.10.25-0ubuntu1~20.04.1 [30.3 MB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 ubuntu-fan all 0.12.13ubuntu0.1 [34.4 kB]
Fetched 67.1 MB in 2s (30.3 MB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 62457 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
Unpacking pigz (2.4-1)
```

Docker services / start/stop/run images

- **sudo docker run -it ubuntu /bun/bash – Docker run**
- **Sudo docker ps -a to know all services**
- **Sudo service Docker status – to know status of docker**
- **Sudo service Docker start – to start services**
- **Sudo service Docker stop – to stop the docker services**
- **sudo docker run -it centos /bin/bash**

```
ubuntu@ip-172-31-83-188:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-83-188:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS    PORTS     NAMES
3ee3fa93dc45   ubuntu    "/bun/bash"   About a minute ago   Created             pedantic_mayer
e321fa336263   ubuntu    "/bun/bash"   2 minutes ago        Created             wonderful_yonath
ubuntu@ip-172-31-83-188:~$
```

```
ubuntu@ip-172-31-83-188:~$ sudo docker run it centos /bin/bash
Unable to find image 'it:latest' locally
docker: Error response from daemon: pull access denied for it, repository does not exist or may require 'docker login
ce is denied.
See 'docker run --help'.
ubuntu@ip-172-31-83-188:~$ sudo docker run -it centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
[root@ef0af708fa80 /]#
```

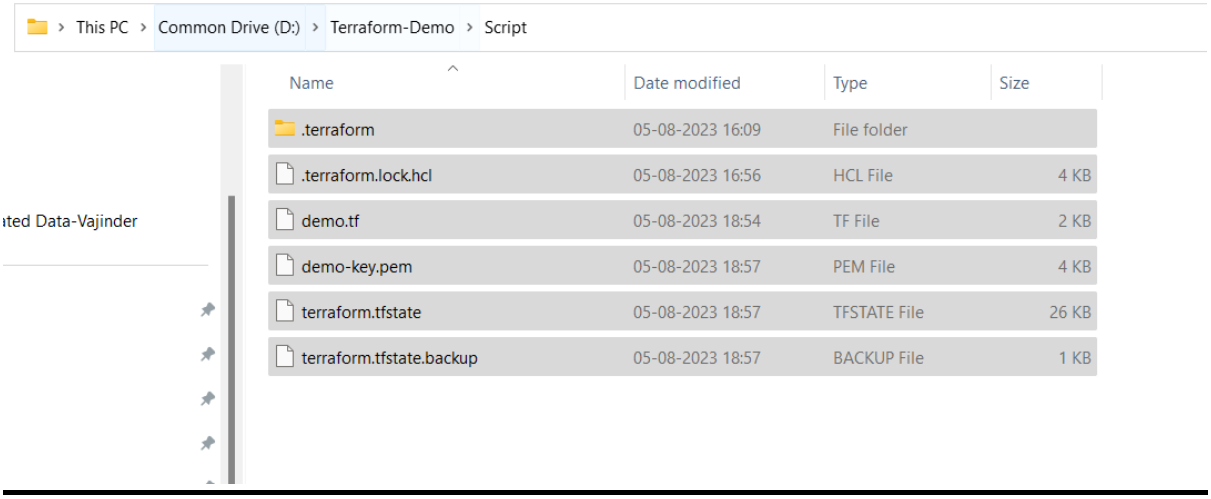
How to install Python on AWS instance

Sudo apt update- just to install latest update
sudo apt install python3- to install python on AWS instance
python3 –version – to check pyhthon version (also let know if already installed or not)

```
ubuntu@ip-172-31-83-188:~$ python3 --version
Python 3.8.10
ubuntu@ip-172-31-83-188:~$
```

```
ubuntu@ip-172-31-83-188:~$ sudo apt install python3
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.8.2-0ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 99 not upgraded.
ubuntu@ip-172-31-83-188:~$
```

Reference files:



Flow Diagram

Workflow: