Components of terraform:

Terraform Executable

Terraform Files

Terraform Plugins/Providers

Terraform State

**Providers:** providers are collection of resources and data sources

Terraform is used to create, manage, and update infrastructure resources such as physical machines, VMs, network switches, containers, and more. Almost any infrastructure type can be represented as a resource in Terraform.

A provider is responsible for understanding API interactions and exposing resources. Providers generally are an IaaS (e.g. Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Cloud, DNSimple, Cloudflare).

Terraform init----it will check the configuration files in the working directory and providers, if the provider plugin is not available, it will download the provider plugin.

Terraform plan---it will check the configuration files and also check with variables if there are any.

It is a step before executing apply, it is always suggested to save the planoutput to any file. So that

The plan can be verified before it actually applyied.

Terranform plan -out aws.tfplan---the executed plan will store the information to aws.tfplan file

It looks at the existing configuration and it will show if anything is missing and help us to know what will going to do, when it actually runs. Once we get the terraform plan.

Terraform apply---these will take terraform plan output file and apply to configure the resources.

Once we run the terraform plan output file and complete the execution. **By default It will create a “state file” in our current working directory, if do not specify any particular path to store the file.**

Terraform destroy---looks at the state file and check all the resources which got created and delete all the resources.

Terraform state ---it is json file that contains the states of the resources configured,

When we run apply command tfstate file will be created and tfstate.lock file will be created, tfstate.lock will be available until the apply command is completed. Once it is done it will disappear, we cannot do any changes to tfstate file, until the tfsate.lock file is there

<https://www.katacoda.com/courses/terraform/playground#!>.

State files contains :

Version No: 4

Terraform Version No: “0.12.5”

Serial No: 29

Lineage:

Outputs : {}

Resources: []

Provisioners: Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction. Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.

The primary responsibility of provisioner plugins is the execution of commands or scripts on a specific resource after creation or upon its destruction.

The **file** provisioner is used to copy files or directories from the machine executing Terraform to the newly created resource. The file provisioner supports both ssh and winrm type [connections](https://www.terraform.io/docs/provisioners/connection.html).

Build in provisioners types

Chef provisioner

File provisioner

Habitat provisioner

Local-exec provisioner

Remote-exec provisioner

Puppet provisioner

Salt-masterless provisioner

Local-exec provisioner: The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource.

Remote-exec provisioner: The remote-exec provisioner invokes a script on a remote resource after it is created. This can be used to run a configuration management tool, bootstrap into a cluster, etc. To invoke a local process, see the local-exec [provisioner](https://www.terraform.io/docs/provisioners/local-exec.html) instead. The remote-exec provisioner supports both ssh and winrm type [connections](https://www.terraform.io/docs/provisioners/connection.html).

Functions:

Min ( )

Lower( )

Merge ( )

File ( )

Cidrsubnet ( )

Timestamp ( )

Modules: A **module** is a container for multiple resources that are used together. Every **Terraform** configuration has at least one **module**, known as its root **module**, which consists of the resources **defined** in the . tf files in the main working directory <https://www.terraform.io/docs/configuration/modules.html>

An **instance profile** is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.

**Remote State**: By default, Terraform stores state locally in a file named terraform.tfstate. When working with Terraform in a team, use of a local file makes Terraform usage complicated because each user must make sure they always have the latest state data before running Terraform and make sure that nobody else runs Terraform at the same time.

Remote state is stored in a backend.

With remote state, Terraform writes the state data to a remote data store, which can then be shared between all members of a team. Terraform supports storing state in [Terraform Cloud](https://www.hashicorp.com/products/terraform/), [HashiCorp Consul](https://www.consul.io/), Amazon S3, Alibaba Cloud OSS, and more.

Remote state is a feature of [backends](https://www.terraform.io/docs/backends). Configuring and using remote backends is easy and you can get started with remote state quickly. If you then want to migrate back to using local state, backends make that easy as well.

“With **remote state**, **Terraform** writes the **state** data to a **remote** data store, which can then be shared between all members of a team”.

**Backends:** A "backend" in Terraform determines how state is loaded and how an operation such as apply is executed. This abstraction enables non-local file state storage, remote execution, etc. types of backed S3,Consul,AzureRM, Alibaba OSS

By default, Terraform uses the "local" backend, which is the normal behavior of Terraform you're used to. This is the backend that was being invoked throughout the [introduction](https://www.terraform.io/intro/index.html).

Here are some of the benefits of backends:

* **Working in a team**: Backends can store their state remotely and protect that state with locks to prevent corruption. Some backends such as Terraform Cloud even automatically store a history of all state revisions.
* **Keeping sensitive information off disk**: State is retrieved from backends on demand and only stored in memory. If you're using a backend such as Amazon S3, the only location the state ever is persisted is in S3.
* **Remote operations**: For larger infrastructures or certain changes, terraform apply can take a long, long time. Some backends support remote operations which enable the operation to execute remotely. You can then turn off your computer and your operation will still complete. Paired with remote state storage and locking above, this also helps in team environments.

What are the parameter defined as part of .tf file

Variables

Providers

Data Source

Resources

Provisioners---these will be defined as the last one in the tf file under resources.

Output

Adding a vpc in terraform

resource “aws\_vpc” “vpc” {}

resource “aws\_internet\_gateway” “igw” {}

resource “aws\_subnet” “subnet1” {}

resource “aws\_route\_table” “rtb” {}

resource “aws\_route\_table\_association” “rtb-subnet1” {}

when we use update the configuration by using the same tf file, if we ran apply command terraform.backup file will be created.

.

**Terrform Installation:**

**Download the terrform software from terrform site using wget command. Unzip the file and copy to /usr/local/bin (on Ubuntu)**

**Step 1: sudo apt-get update**

**Step 2: sudo apt-get install unzip**

**Step 3: wget** [**https://releases.hashicorp.com/terraform/0.12.24/terraform\_0.12.24\_linux\_amd64.zip**](https://releases.hashicorp.com/terraform/0.12.24/terraform_0.12.24_linux_amd64.zip)

**Step 4: uzip terraform\_0.12.24\_linux\_amd64.zip**

**Step 5: mv terraform /usr/local/bin**

**Setp 6: check terraform version (terraform –version)**

**We need to create IAM user with programmatic access, with access key and secret key.**

**root@ip-172-31-13-158:~/terraform# aws configure**

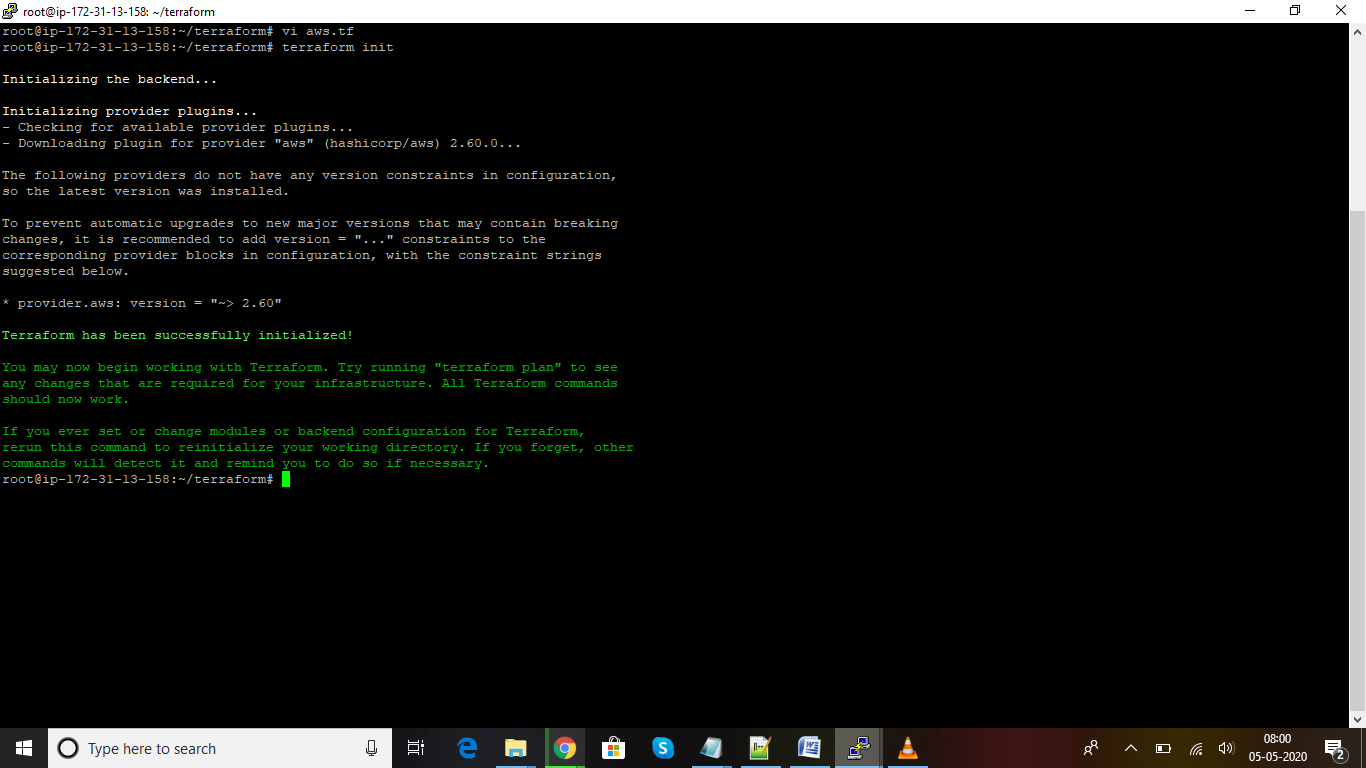
**AWS Access Key ID [None]: your key**

**AWS Secret Access Key [None]: your key**

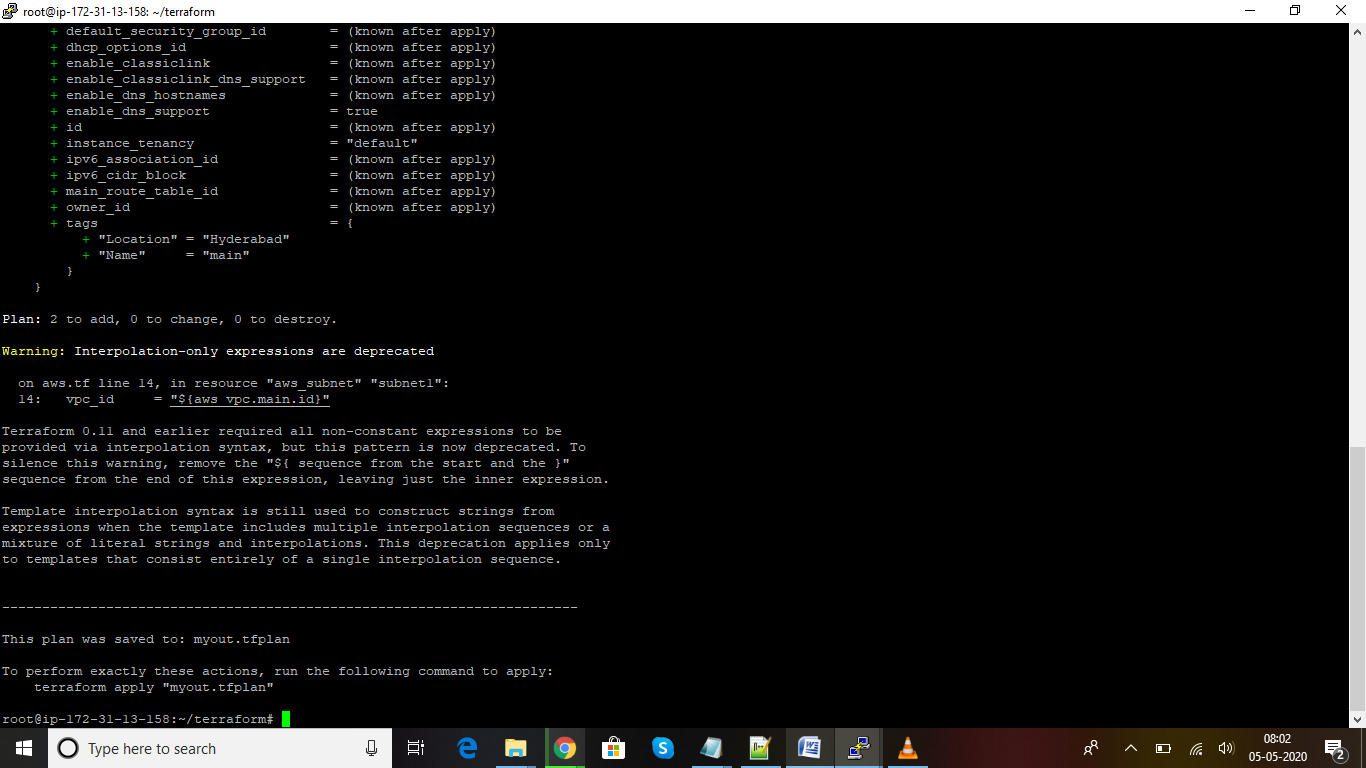
**Default region name [None]: your region**

**Default output format [None]:**

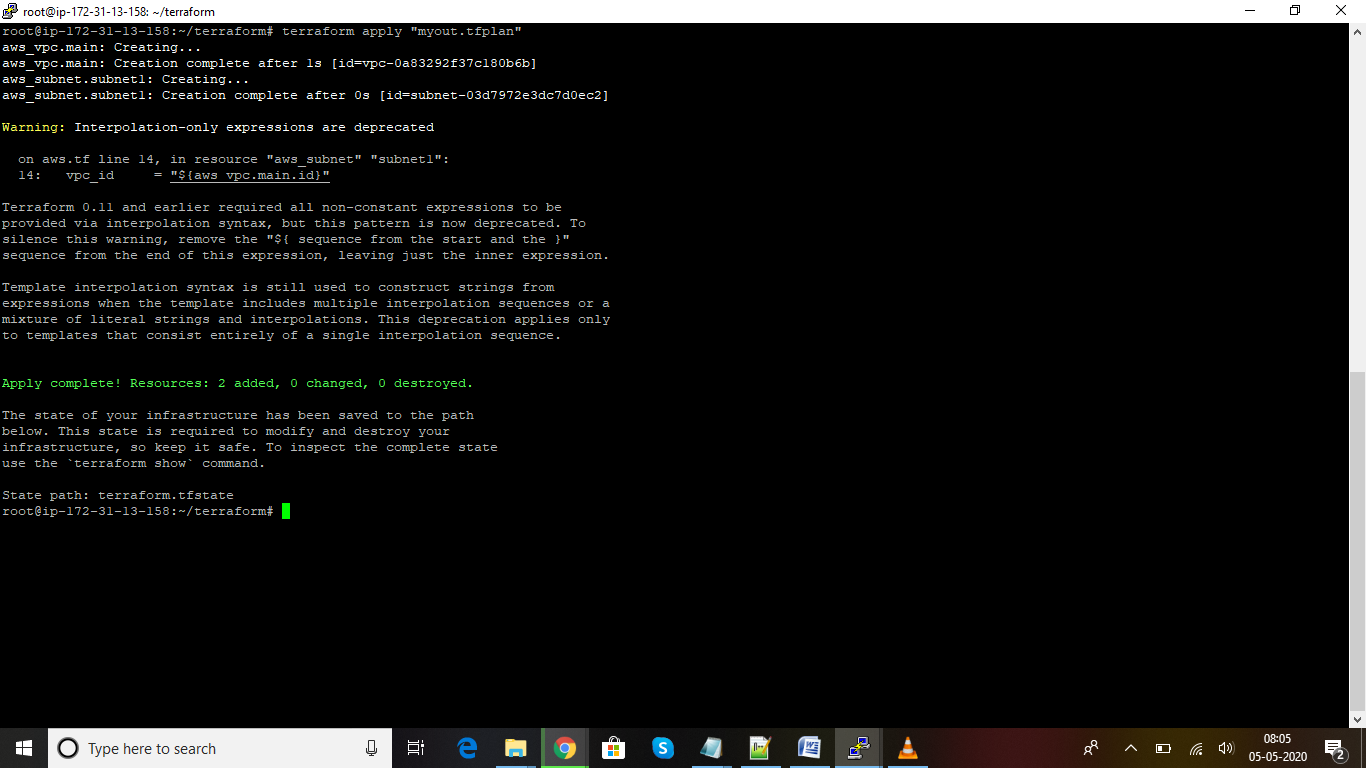
**Terraform init**

****

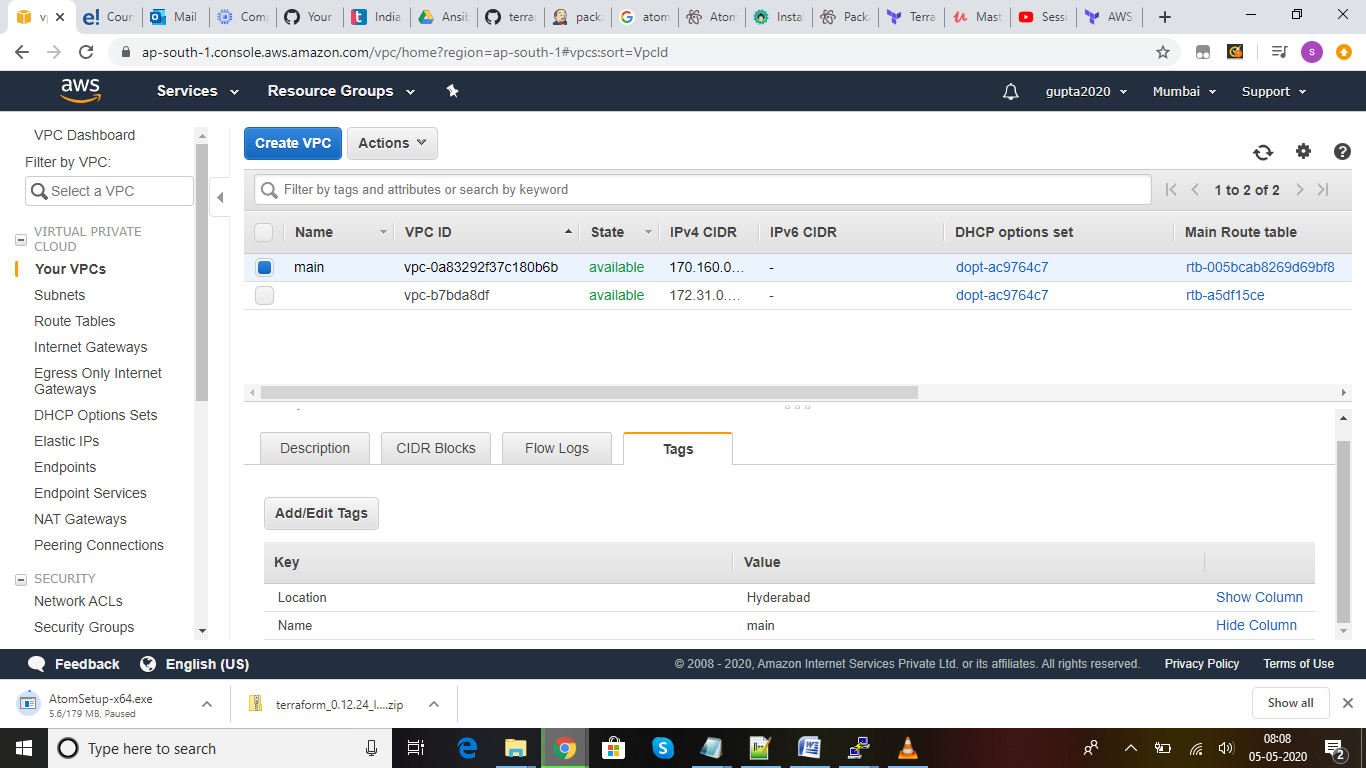
**Terraform plan: if we execute plan command it will show, it will show what are the resources created as part of these .tf file, these is nothing but a preview before creating the actual one. We can output the plan to any file and can be used at the time of applying.**

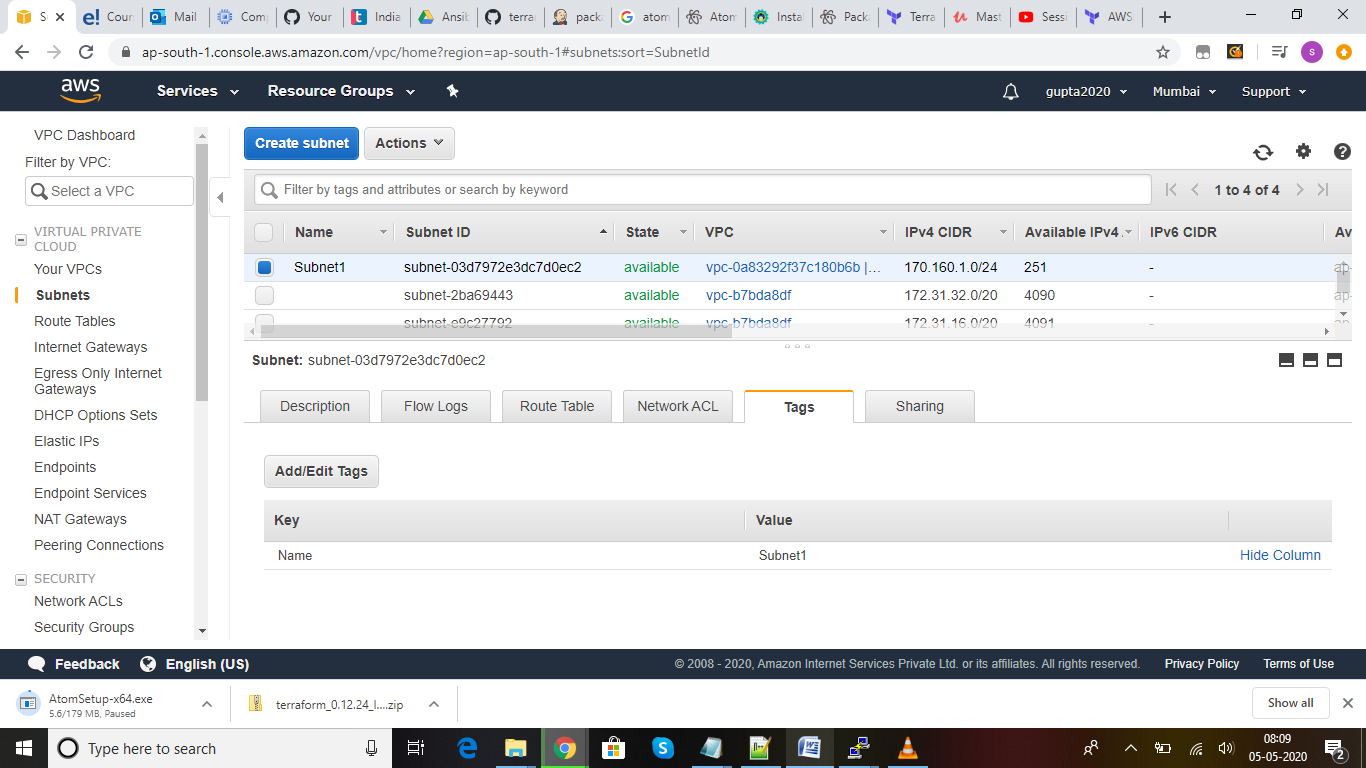
****

**Terraform Apply: These command will create the resources, which are listed in the .tf file, by default it will search for .tf files in the current working directory to execute the files.**

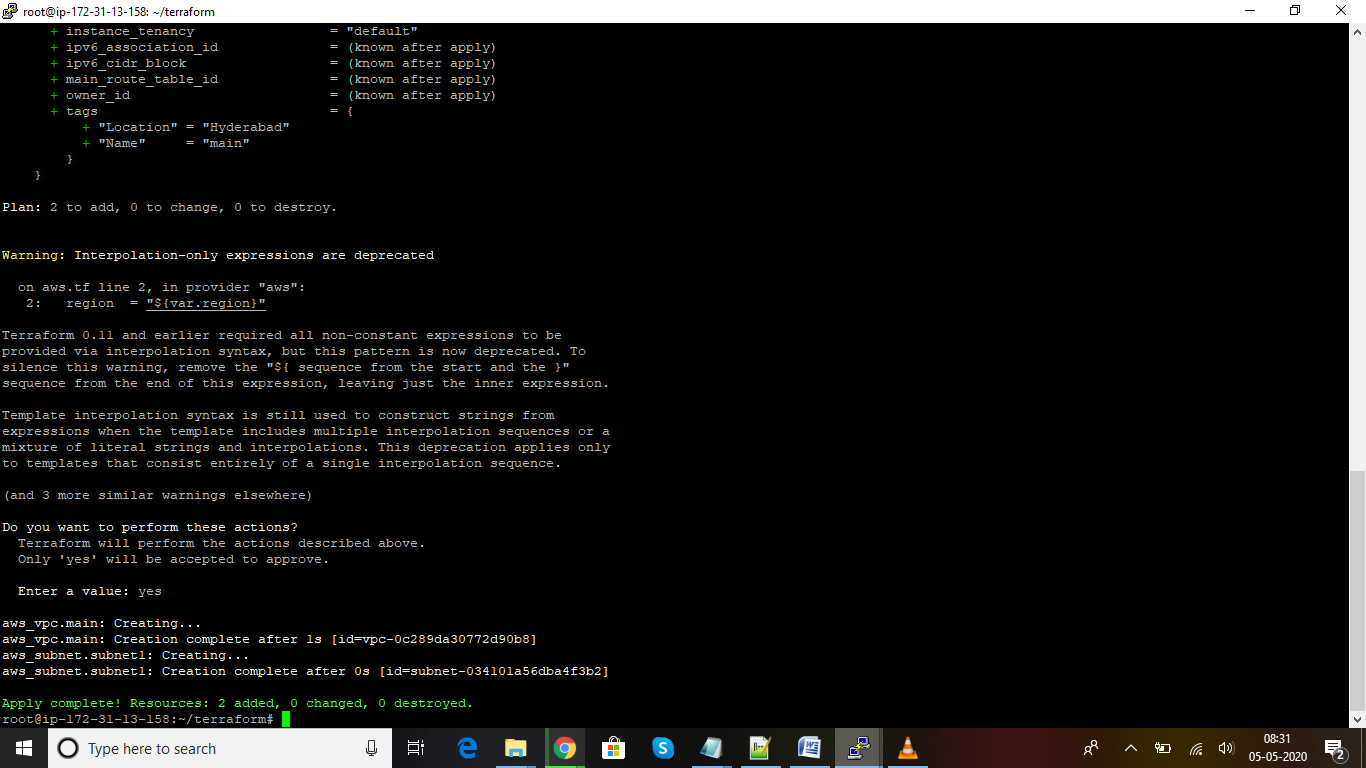
****

**VPC Created from tf file**

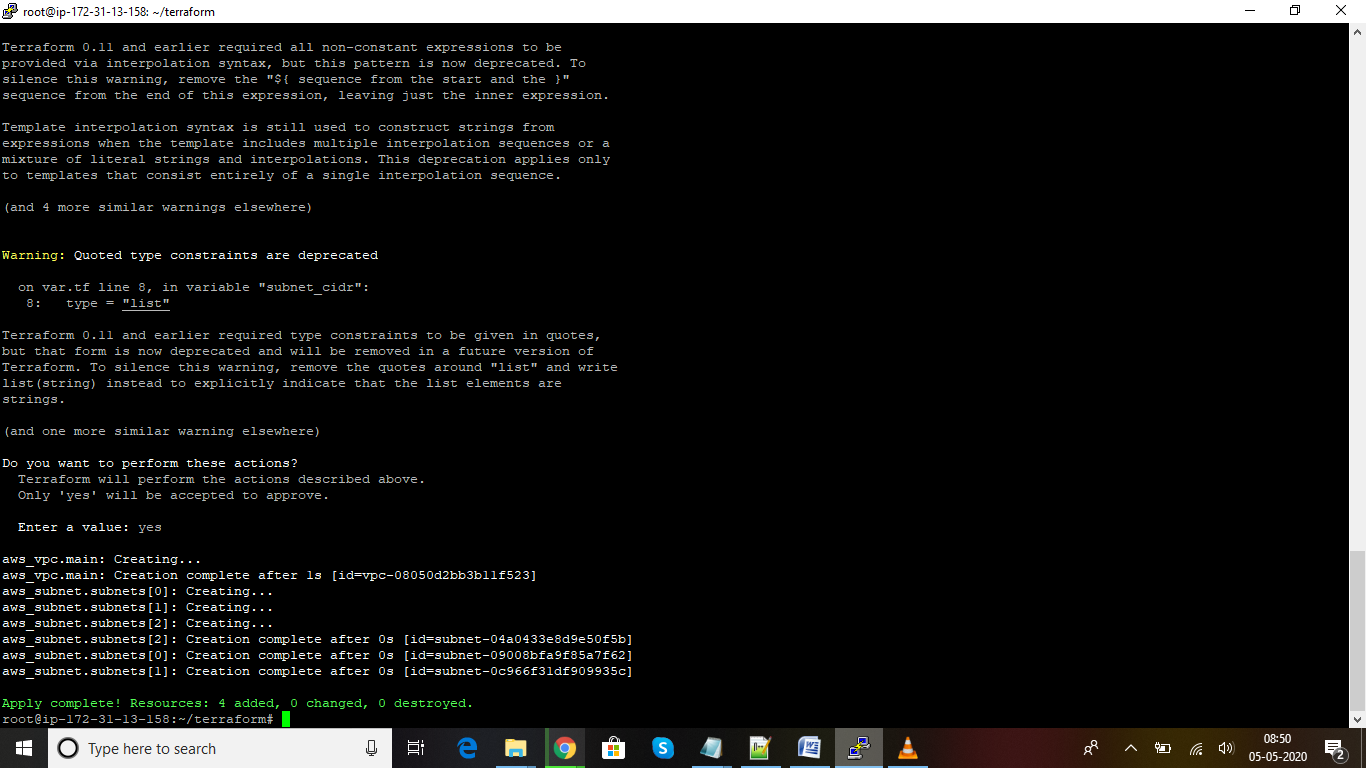
****

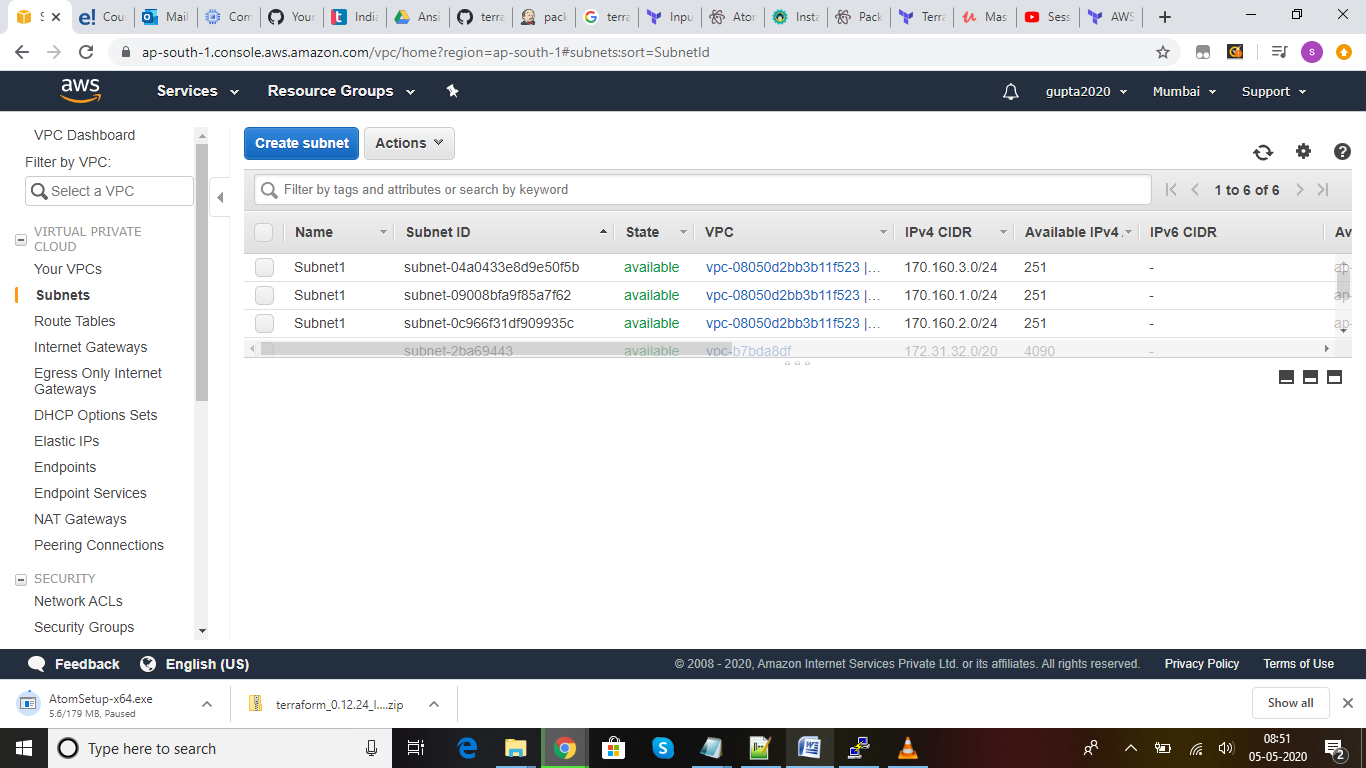
****

**Using Terraform Variables, Data Sources and Terraform Loops. Created a var.tf files and created the variables and passed it as reference**

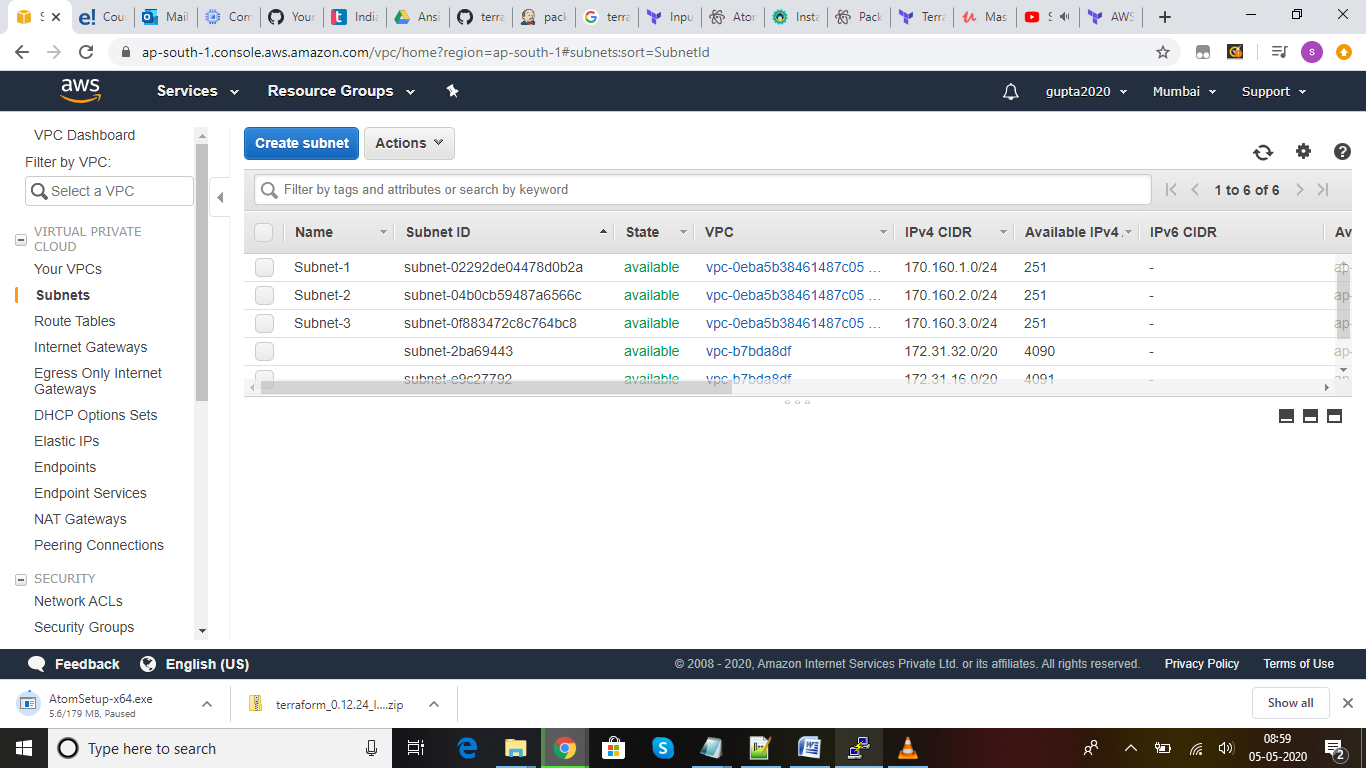
****

**Using interpolation and the lists in the variables files, created 3 subnets**

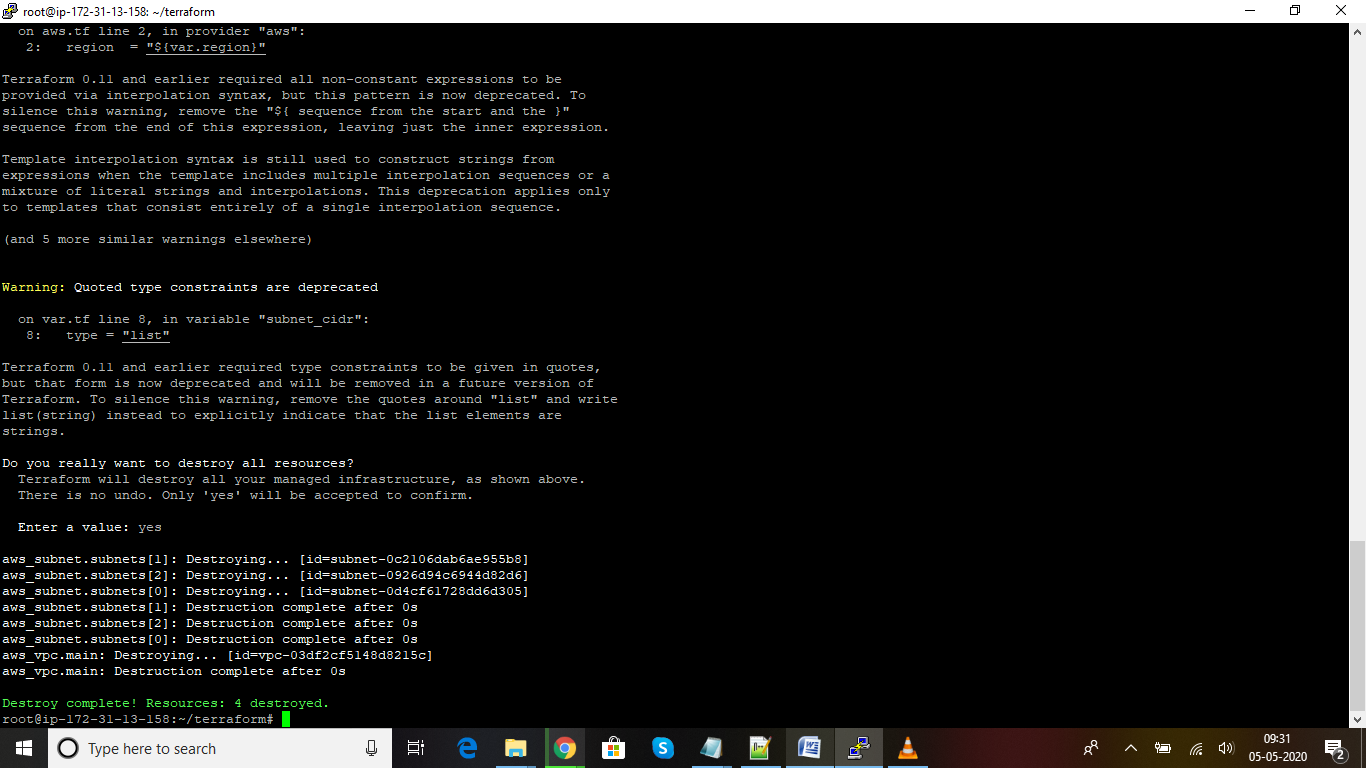
****

****

**Now changed the tags by using count.index+1**

****

**Terraform Destroy: Since we are using free tier AWS Account, once we are done with our practice, we can delete the resources which are created using .tf file.**

****