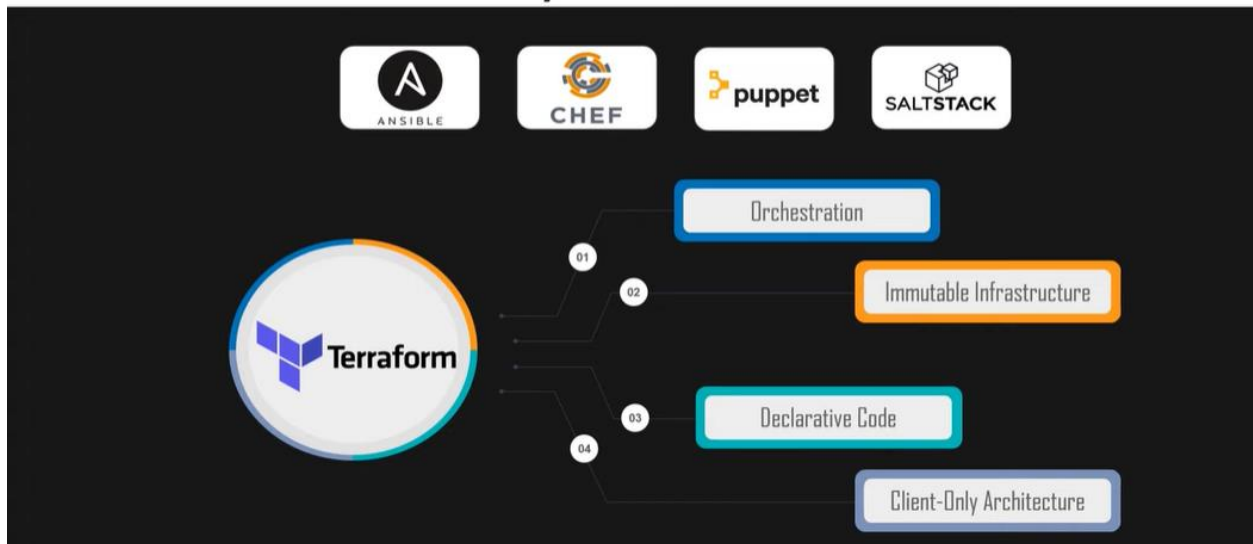


## Terraform DEMO

# Why Terraform?



Ansible, Chef, Puppet and Saltstack are great automation and configuration tools.

However, they are not the same as terraform because they only configure and deploy and already existing server.

Configuration means the ability to install the updates and required programs and at the same time manage the server. However, the server that is being managed and installed the updates (Configuration) must already be existing.

However, terraform is unique from the other configuration tools because its strength is server provisioning/infrastructure provisioning or orchestration/composition.

It ensures that the infrastructure that is required for the Ansible, Chef, Puppet and Saltstack to automate, manage and configure is made available.

It usually creates immutable infrastructure. Unlike the configuration tools, the server, or the infrastructure they are working on must be

updated frequently and this usually can read to the bugs if not correctly and frequently updated.

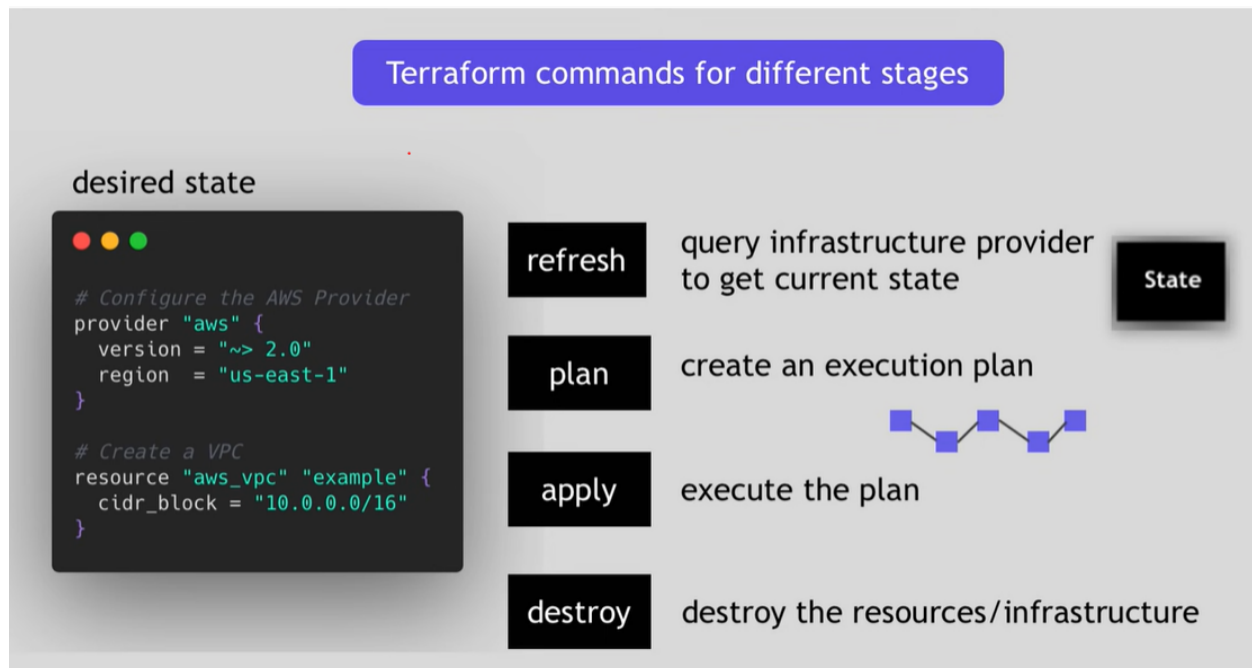
However, terraform does not have this problem because it is always connected to the provider and every time you declare new commands, it will always refresh, meaning checking the status of the setup, compare with your declarative file (tf-config file) which contains the desired end results and come up with a plan to get the results required.

It deals with the current snapshot of the setup that the provider has. No bugs issue.

Imperative/procedural commands – It means you must carefully sit down and plan the steps to achieve the desired end results. However, in terraform, you do not need to do all this. You just need to declare what you end results should look like and terraform will compare your requirements with the current state of the infrastructure and execute a plan with all the steps to give you the desired results.

This is very useful especially when managing and changing your infrastructure. In the imperative/procedural commands, you must go to your server, count the number of servers you want to add or remove, plan the steps, and execute. However, in terraform, you just need to declare what the end results should look like and terraform will do the rest of the job.

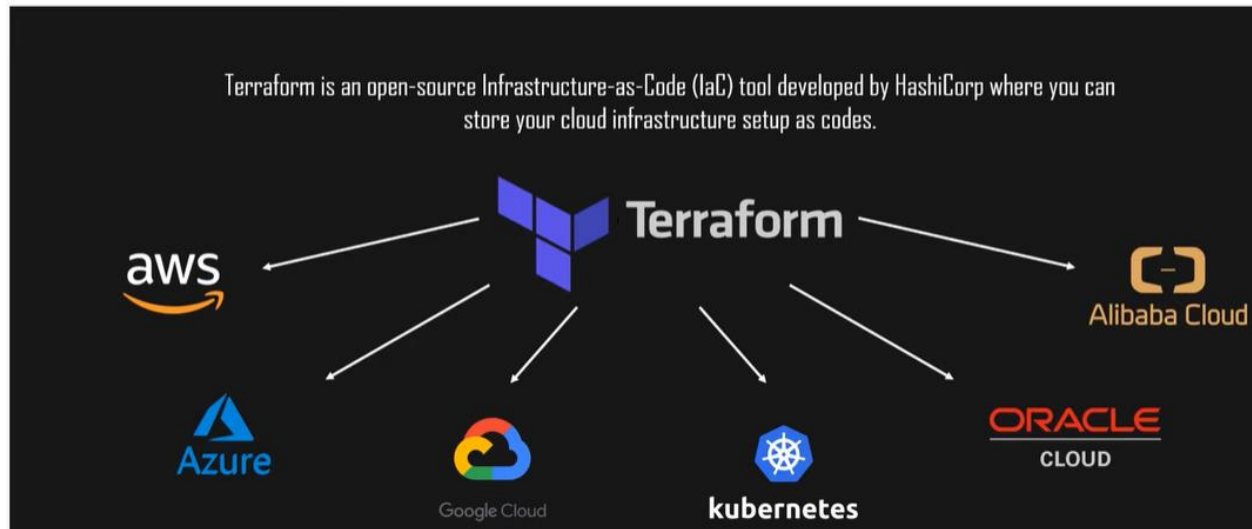
Look at these declarative commands – You just stating what the end results should be and terraforms is executing them.



Terraform provides the best security possible because it connects with the end provider of the resources. For instance, it executes the plan using the AWS resources and on the AWS console which is very secure. No need to worry about security and therefore many companies love to terraform.

Other configuration tools are very limited because you can only access the provider platform through SSH. For instance, Ansible can only access the instances on AWS through SSH protocol, which makes it to be limited. However, terraform has the AWS provider as part of its component and has access to all the resources on AWS. Same case to other resource providers.

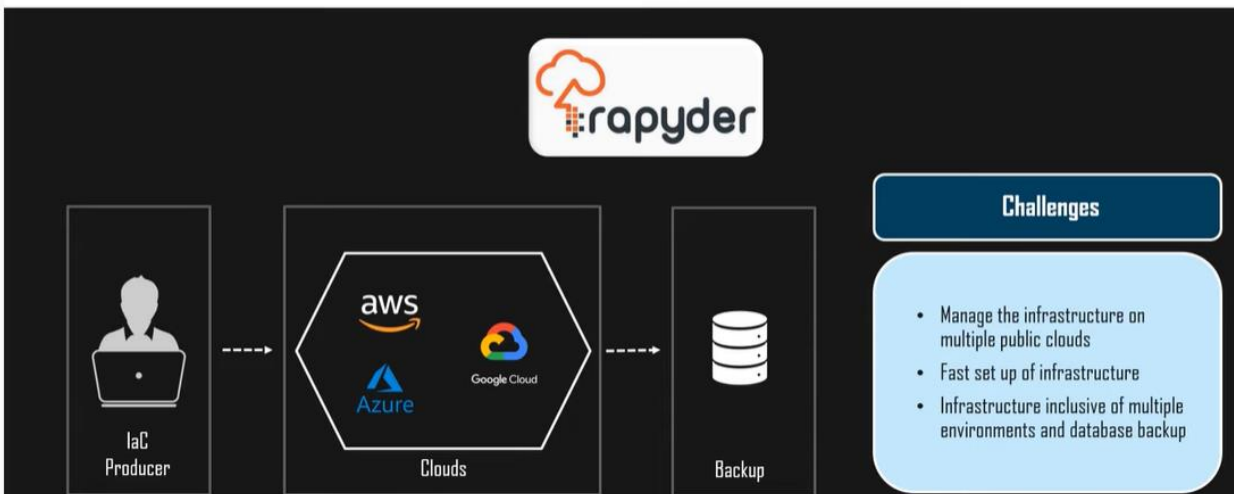
# What is Terraform?



Terraform supports many resources providers.

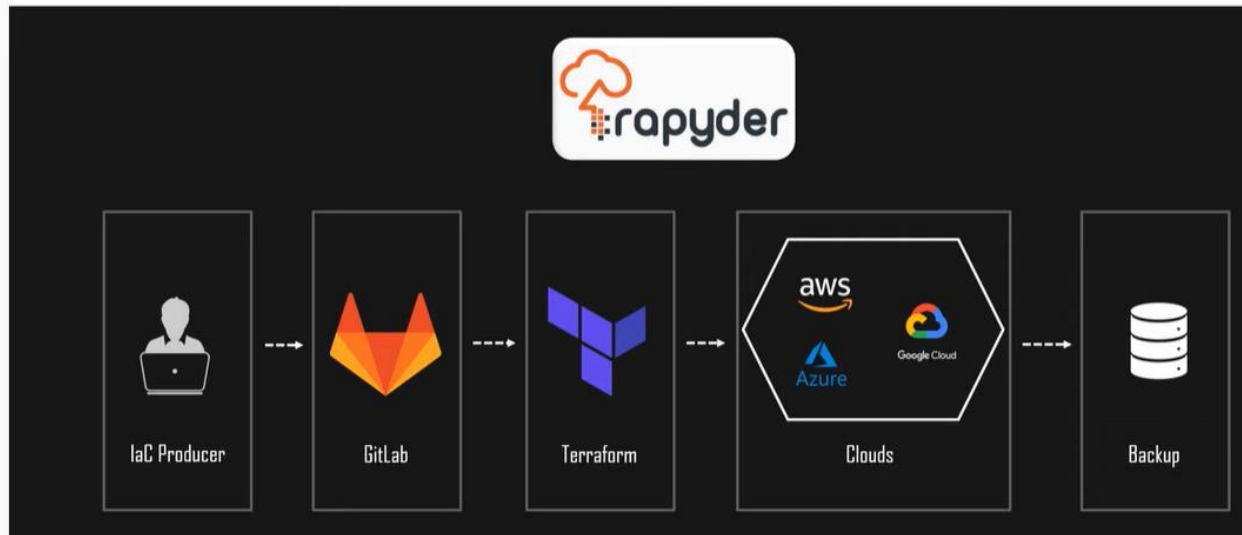
Why terraform is important

## Terraform Use Case



Fast setup and managing infrastructure from different clouds within a matter of a few days.

# Terraform Use Case



Terraform was able to provision this infrastructure by pulling the declarative codes from GITLAB, comparing it that tf-config repository with the current set ups in different cloud platforms, develop steps and plan on how to achieve the desired end results and execute the plan.

In case of any error, the plan can be destroyed in a matter of a minute.

This infrastructure was made ready within a short period of time and was ready to be deployed in production which can also be automated by Ansible and now the client is happy to test the provisioned infrastructure in a matter of a minute and deploy to end user which can be done with Ansible.

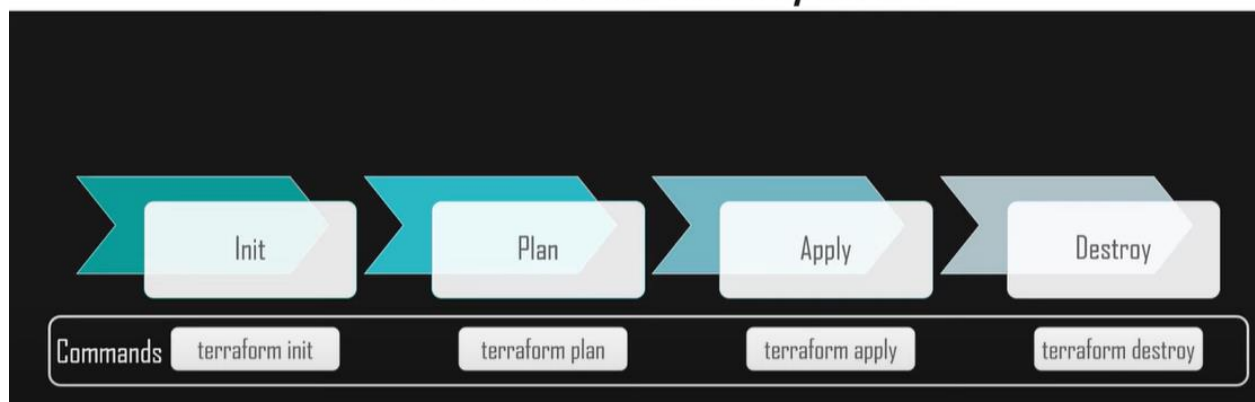
# Terraform Use Case



It is also easy for the end client to replicate, make the necessary changes and manage the provisioned infrastructure with ease because of the declarative commands.

Use terraforms ensures that you do not need to do any updates and you will always have the most updates resources especially on the data section.

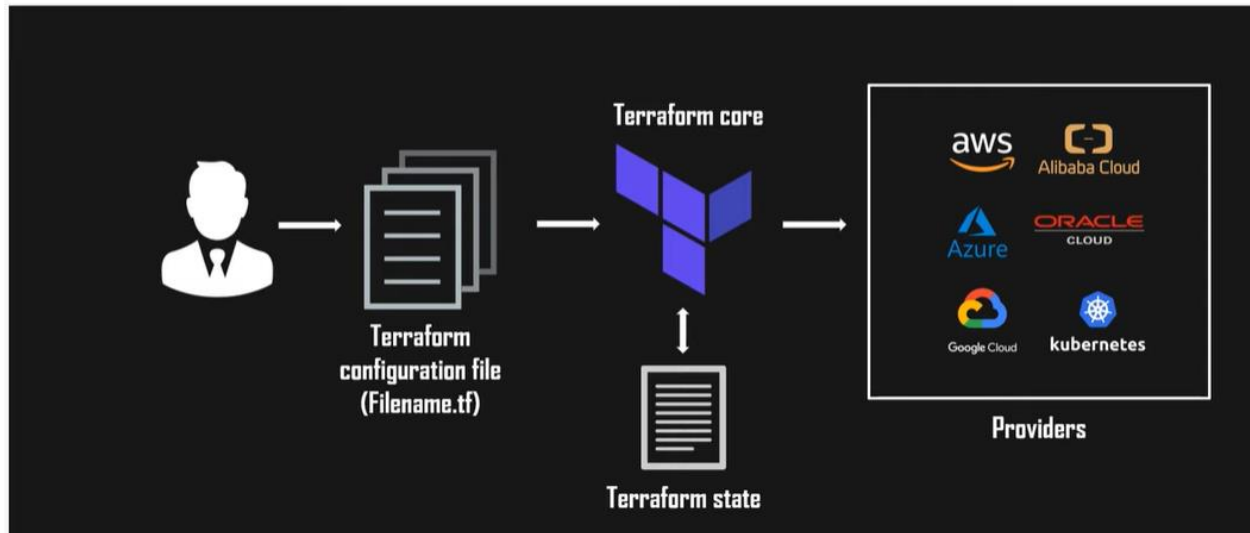
# Terraform Lifecycle



The lifecycle of terraform include 4 stages. Initialization (terraform is checking for the current state of the infrastructure and comparing with the declared results in the tf-config files and preparing them. When in this step, it deletes, add, updates and combine codes as needed),

planning (this is where terraform will plan on that will enable it to give you the desired end results. The plan will include all the steps needed. You do not need to bother about the steps to reach the desired state/goal), Applying or executing the plan, and destroying the applied plan if needed.

## How Does Terraform Work?



You have two inputs through the terraform CORE

Tf-config is the files with the end user desire state

State is the terraform capability to keep the status of how the infrastructure is set up/look like. that you

Terraform just take the set up and check the desired state in the configuration file and come with the plan to get the end results as requested in the configuration file.

Then it executes the plan through the second component of the terraform which is the provider, where all the services/resources you need is located. It supports many resources providers.

Step 1

Installing terraform

1. Open ubuntu
2. SSH to your code editor
3. Go to official website of terraform

<https://www.terraform.io/downloads.html>



**macOS**

64-bit | Arm64



**FreeBSD**

32-bit | 64-bit | Arm



**Linux**

32-bit | 64-bit | Arm | Arm64



**OpenBSD**

32-bit | 64-bit

Copy the address for the 64 -bit



**macOS**

64-bit | Arm64



**FreeBSD**

32-bit | 64-bit | Arm



**Linux**

32-bit | 64-bit | Arm | Arm64




**OpenBSD**

32-bit | 64-bit

Go to code editor and run the command

# wget the copied link location of 64 bits



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE  ssh
root@osboxes:/home/osboxes# wget https://releases.hashicorp.com/terraform/1.0.8/terraform_1.0.8_linux_amd64.zip
```

Enter

```
root@osboxes:/home/osboxes# wget https://releases.hashicorp.com/terraform/1.0.8/terraform_1.0.8_linux_amd64.zip
--2021-10-09 07:49:12-- https://releases.hashicorp.com/terraform/1.0.8/terraform_1.0.8_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 2a04:4e42:31::439, 151.101.209.183
Connecting to releases.hashicorp.com (releases.hashicorp.com)|2a04:4e42:31::439|:443... failed: Connection timed out.
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.209.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32681118 (31M) [application/zip]
Saving to: 'terraform_1.0.8_linux_amd64.zip'

terraform_1.0.8_linux_amd64.zip  100%[=====>] 31.17M  8.43MB/s  in 4.2s

2021-10-09 07:51:26 (7.43 MB/s) - 'terraform_1.0.8_linux_amd64.zip' saved [32681118/32681118]
```

A zipped file of terraform is now downloaded.

Let us unzip it

```
root@osboxes:/home/osboxes# wget https://releases.hashicorp.com/terraform/1.0.8/terraform_1.0.8_linux_amd64.zip
--2021-10-09 07:49:12-- https://releases.hashicorp.com/terraform/1.0.8/terraform_1.0.8_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 2a04:4e42:31::439, 151.101.209.183
Connecting to releases.hashicorp.com (releases.hashicorp.com)|2a04:4e42:31::439|:443... failed: Connection timed out.
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.209.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32681118 (31M) [application/zip]
Saving to: 'terraform_1.0.8_linux_amd64.zip' → zip file

terraform_1.0.8_linux_amd64.zip  100%[=====>] 31.17M  8.43MB/s  in 4.2s

2021-10-09 07:51:26 (7.43 MB/s) - 'terraform_1.0.8_linux_amd64.zip' saved [32681118/32681118]
```

Run the command # unzip and the file name above

```
root@osboxes:/home/osboxes# unzip terraform_1.0.8_linux_amd64.zip
Archive:  terraform_1.0.8_linux_amd64.zip
  inflating: terraform
root@osboxes:/home/osboxes# mv terraform /usr/local/bin/
root@osboxes:/home/osboxes# terraform -v
Terraform v1.0.8
on linux_amd64
root@osboxes:/home/osboxes#
```

Run # mv terraform /usr/local/bin - moving the unzipped file to the local bin in your computer/server/machine

Run # terraform -v and you should see the version installed in your system

Run terraform and you can see all the command supported by terraform

Let create a directory to have a new working environment

Run # mkdir terraform-demo

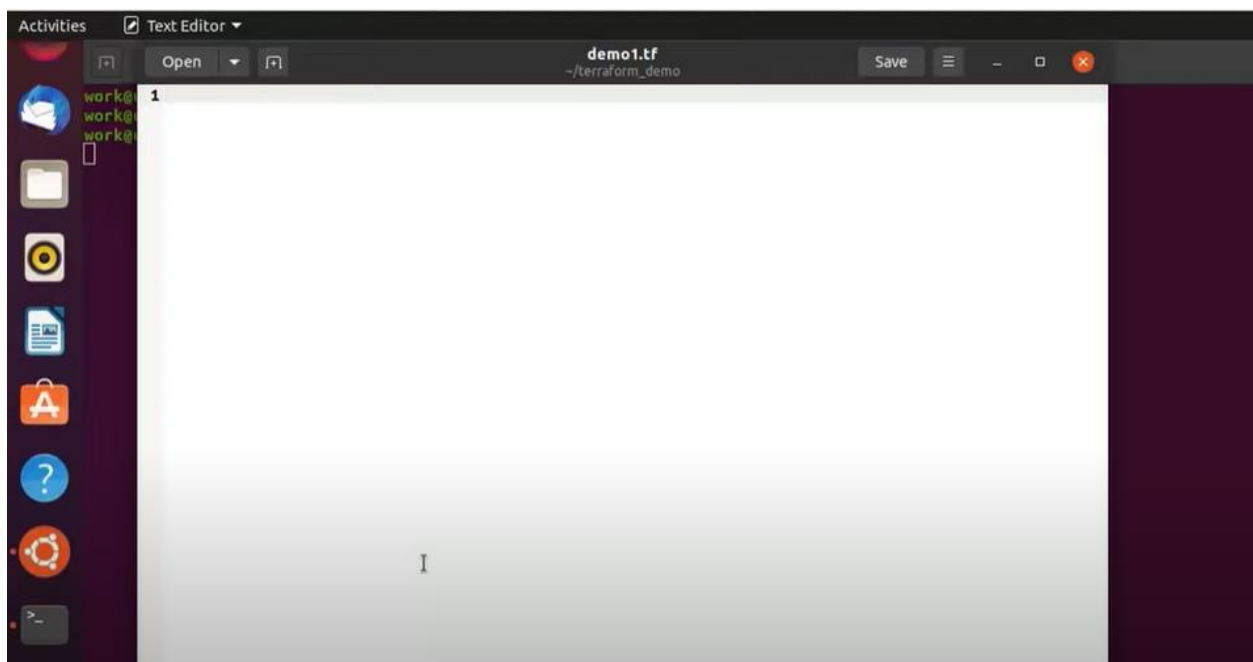
CD to the new directory

Let create a configuration file inside the directory

Run # gedit demo1.tf [ you muss add the extension .tf just like in docker we were using the .yaml extension]

Enter and a blank white editor should appear

I had to switch back to ubuntu console because vsc could not connect with the gedit server



This is where we will put all the configuration declarative commands.

It is important to remember that terraform provides you with every support for your code online.

Nothing to memorize

Let create a simple instance on AWS and destroy it afterwards.

Go on web browser and type:

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

You must define usage first before you can use the aws service. We can get this from AWS

```
provider "aws" {  
    region    = "us-west-2"  
    access_key = "my-access-key"  
    secret_key = "my-secret-key"  
}
```

Add user

1 2 3 4 5

✓ Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://232110768834.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key
▶ ✓	terraform-deme1	AKIATMCXBILBBM2WDARA	***** <a href="#">Show</a>

Download the CSV file

A1	✕	✓	f <sub>x</sub>	User name															
	A	B	C	D	E	F	G	H	I	J									
1	User name	Password	Access key ID	Secret access key	Console login link														
2	terraform-deme1		AKIATMCXBILBBM2WDARA	pmdvXytPPtF9gZkCAoqxr3mbQ2JCKfO5Y/srUmtld	<a href="https://232110768834.signin.aws.amazon.com/console">https://232110768834.signin.aws.amazon.com/console</a>														
3																			
4																			
5																			

Copy and paste on gedit.tf configuration file

Access key

Secret Key

We are done with creating usage for East

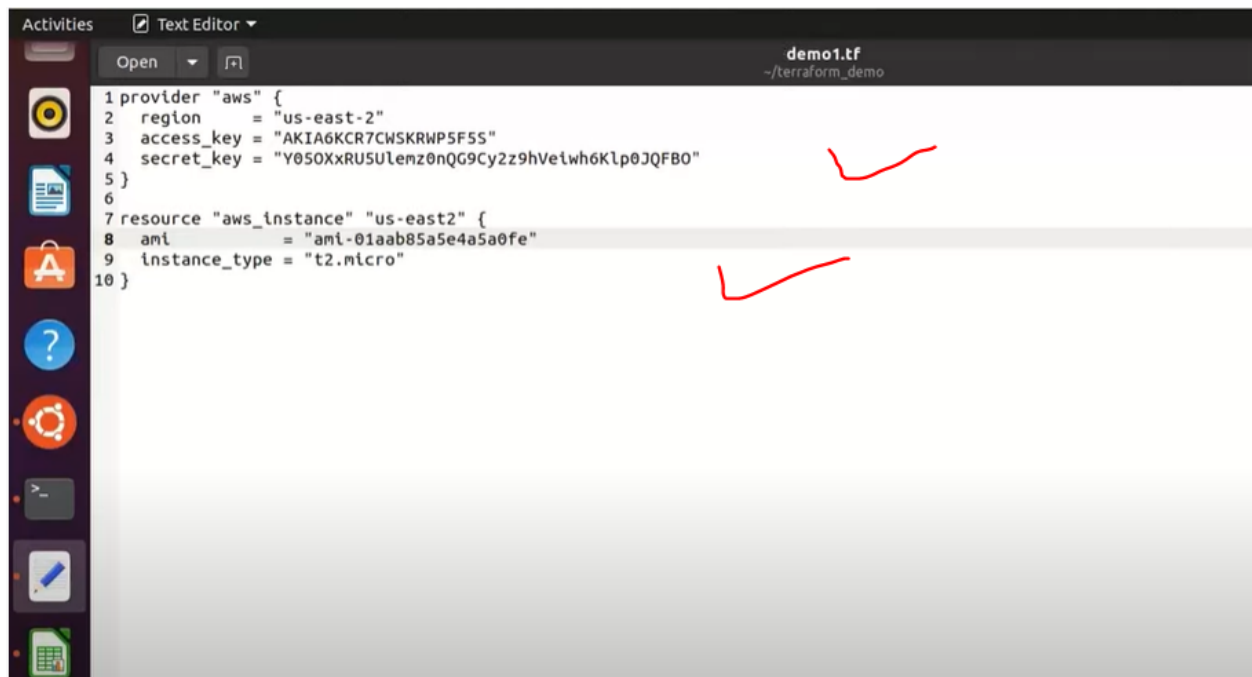
Then we move to the first resource we want to create

Let create the EC2 instance

Do not memorize anything

Use the link: ec2 instance terraform and all the declarative codes will be there

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>

A screenshot of a Linux desktop environment with a text editor open. The text editor has a dark theme and shows a Terraform configuration file named 'demo1.tf'. The code defines an AWS provider and an EC2 instance resource. Two red checkmarks are drawn on the right side of the code, one next to the provider block and one next to the resource block. The desktop background is dark, and a sidebar with application icons is visible on the left.

```
1 provider "aws" {  
2   region = "us-east-2"  
3   access_key = "AKIA6KCR7CWSKRWP5F5S"  
4   secret_key = "Y050XxRUSUlemz0nQG9Cy2z9hVeIwh6Klp0JJQFB0"  
5 }  
6  
7 resource "aws_instance" "us-east2" {  
8   ami = "ami-01aab85a5e4a5a0fe"  
9   instance_type = "t2.micro"  
10 }
```

Let us save the file

First step

Initialize the file

```

work@ubuntu:~/terraform_demo$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.26.0...
- Installed hashicorp/aws v3.26.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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
commands will detect it and remind you to do so if necessary.

```

# terraform init

Next command

# terraform plan

```

+ http_put_response_hop_limit = (known after apply)
+ http_tokens                  = (known after apply)
}

+ network_interface {
+ delete_on_termination = (known after apply)
+ device_index          = (known after apply)
+ network_interface_id  = (known after apply)
}

+ root_block_device {
+ delete_on_termination = (known after apply)
+ device_name           = (known after apply)
+ encrypted              = (known after apply)
+ iops                   = (known after apply)
+ kms_key_id            = (known after apply)
+ tags                  = (known after apply)
+ throughput            = (known after apply)
+ volume_id             = (known after apply)
+ volume_size           = (known after apply)
+ volume_type           = (known after apply)
}
}

Plan: 1 to add, 0 to change, 0 to destroy.

-----
Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.

```

Next command

# terraform apply

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.us-east2: Creating...
aws_instance.us-east2: Still creating... [10s elapsed]
aws_instance.us-east2: Still creating... [20s elapsed]
aws_instance.us-east2: Still creating... [30s elapsed]
aws_instance.us-east2: Creation complete after 33s [id=i-06351f8dc8e53c512]
```

Enter value: yes

Go back to your aws console and you will find your created instance

Next command

#terraform destroy

The instance will be destroyed

Source to use

<https://aws.plainenglish.io/terraform-deploying-a-three-tier-architecture-in-aws-4c8ecce40790>

[https://www.youtube.com/watch?v=SLB\\_c\\_ayRMo&t=265s](https://www.youtube.com/watch?v=SLB_c_ayRMo&t=265s)