## Day 1 - Iac - Terraform Introduction:

we created our resources manually. So in real time. You don't create one in ec2 or one s3 bucket. So your cloud engineers or the Devops engineer what they have to do they have to create multiple. They have to launch multiple Ec 2 and they have to launch multiple S3 buckets. And you have to configure everything in the same way. You have to configure hundreds of Ec. 2 and you have to follow the same configuration. You have to be very consistent about your configuration.

In that case, when your infrastructure goes big, you have to create multiple resources that is, maintaining everything manually become difficult. So what we do, we we will be writing few lines of code to automate the resource creation process. So terraform is one of the important .

you maintain us proper documentation via code. So everything is automated via code here. So if there is any change, or anybody is creating any infrastructure or resources in inside your aws account it will automatically detect the change.

using terraform with the single command, you can delete your complete infrastructure that you have created.

- IAAC | Automate Infrastructure
- Define Infrastructure State
- Ansible, puppet or chef automates mostly OS related tasks.
  - Defines machines state
- Terraform automates infra itself
  - Like AWS, GCP, Azure, digital ocean etc

configure anything into your resource. So suppose you are creating Ec. 2. with your terraform, and you want to install some software or tools inside your Ec 2 that could also be **automated via tools like ansible puppet chef**. So these 3 tools are mainly used for configuration management.

# Introduction

- Terraform works with automation softwares like ansible after infra is setup and ready.
- No Programming, its own syntax similar to JSON.

**Terraform syntax:**

# Summarizing

instance.tf

```
provider "aws" {
  access_key = "ACCESS_KEY"
  secret_key = "SECRET_KEY"
  region = "ap-south-1"
}

resource "aws_instance" "intro" {
  ami = "ami-009110a2bf8d7dd0a"
  instance_type = "t2.micro"
}
```

You can give any name for your terraform code. I've given us instance. You can give any name to your terraform. We call it as terraform file or terraform template you can give any name, but the extension should be extension of .tf,.So terraform file everything.

We have it in the form of blocks. Okay, so you have a block of code, a chunk of code to perform a certain action.

initially, we have multiple blocks. Okay, we call it as provider block and resource block. We have other blocks also. But the standard block to create a minimum resource. You at least need a **provider block and a resource block**. Without these 2 block you cannot launch any resources in a cloud environment.

**provider block:**

you start with the provider block because you have to tell terraform. As I told you, terraform works with any cloud provider you have to tell terraform in which cloud provider we want to create the resources.

then this access key and secret access key you can either give. This is nothing but your IM user. You can either give it inside your provider block or before starting this before creating this terraform.

you have to tell terraform in which region we have to create the resources either in the Mumbai region or North Virginia.

**Resource block:**

these are the keywords [aws instance]. So here, this is the name of your block [resource].

Block always starts with the keyword resource and the type of the resource that you're creating, whether you're creating an S3 bucket or an Ec2. If it is an Ec2 then you have to give aws_instance. for S3 bucket, it would be different.

You have to give the mandatory value, the default value for creating the ec2. The **mandatory parameters are the ami and the instance type**. Without the Ami you cannot launch a Ec. 2 and you have to also mention the instance type. So these 2 are mandatory parameter that you have to pass it inside your Ec2 creation.

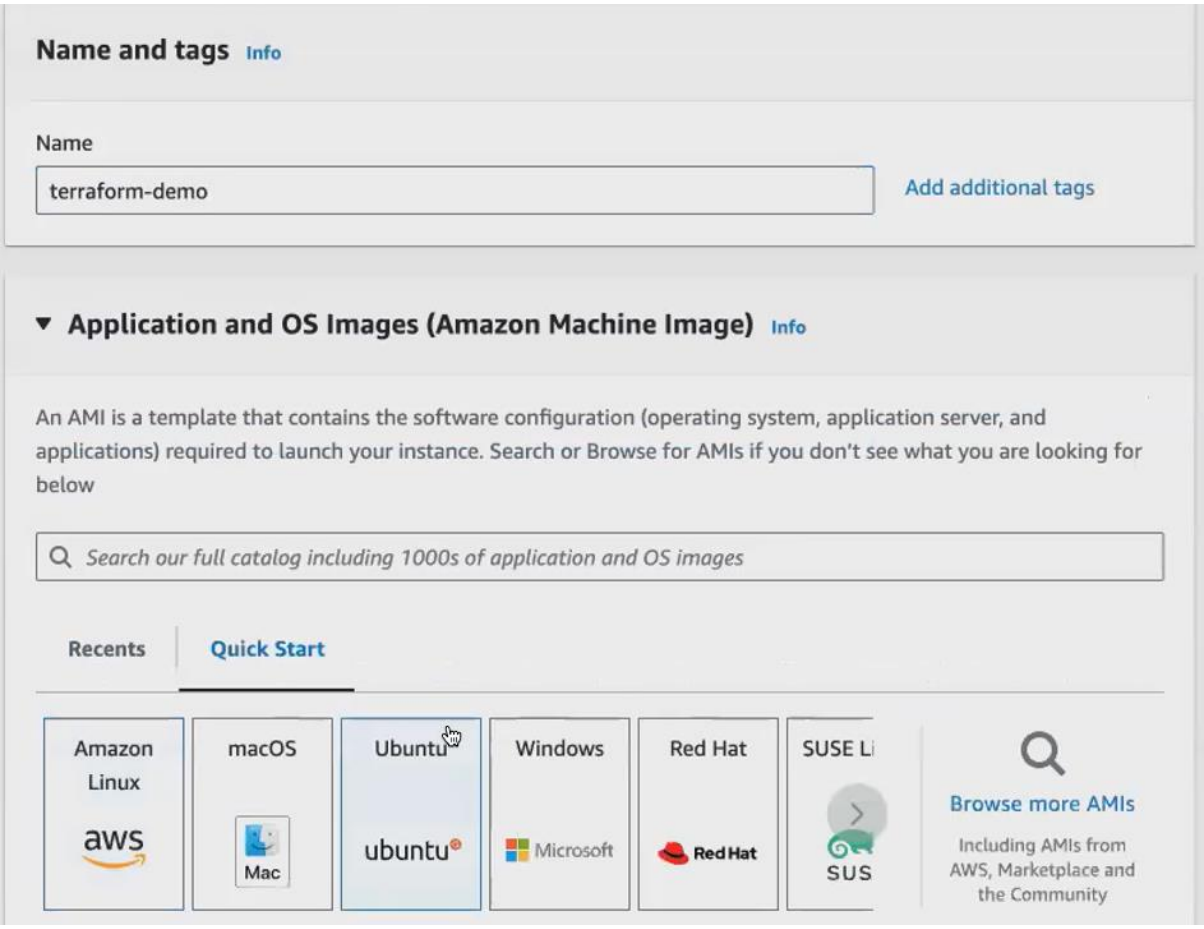
**apart from this, you can give user data. You can give security group to whatever you configured in the manual way that is in the console. You can. Same thing. You can do it here, but in the form of code.**

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

**https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli**

**how to install terraform docs**

**create a new instance :**




Connect the instance .use the below command ans install terraform .that doc link is above

Ubuntu/Debian CentOS/RHEL Fedora Amazon Linux

Ensure that your system is up to date and you have installed the `gnupg`, `software-properties-common`, and `curl` packages installed. You will use these packages to verify HashiCorp's GPG signature and install HashiCorp's Debian package repository.

```
$ sudo apt-get update && sudo apt-get install -y gnupg software-properties
```

Install the HashiCorp GPG key.

```
$ wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
```

Verify the key's fingerprint.

```
$ gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
```

Check the terraform installed or not [all cmds in above doc]

what is the main purpose of the terraform is to create your resources. So terraform can create resources even in your local, not only in the cloud, in the local. Also, you can create resources using terraform. So in your local machine, if you want to launch a file, okay, you want to create a simple file in your local machine using terraform.

Befor creating t.f we create directory

```
ubuntu@ip-172-31-33-105:~$ mkdir localfile
ubuntu@ip-172-31-33-105:~$ cd localfile/
ubuntu@ip-172-31-33-105:~/localfile$ ls
ubuntu@ip-172-31-33-105:~/localfile$ vi file.tf
```

I'm going to create a simple file using terraform in my local Ec2

**I'm going to create a resource.**

**What type of resource I'm going to create. I'm going to create a local file.**

**his is the keyword to create a local file local_5.**

**Local_file [resource type] .we create a file so give this as resource type**

**Demo [block name]**

**Inside the block give {**

- **Filename**
- **Where that stored [/home/ubuntu/demo.txt**
- **Add content in this file [“hello devops] }**

```
resource "local_file" "demo" {
        filename = "/home/ubuntu/demo.txt"
        content = "Hello Devops"
}
```

**Terraform initi –** inti necessary plugins and all

```
ubuntu@ip-172-31-33-105:~/localfile$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.5.1...
- Installed hashicorp/local v2.5.1 (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.
ubuntu@ip-172-31-33-105:~/localfile$
```

**Terraform validate-** any syntax error or not sure about file syntax. Give that

```
ubuntu@ip-172-31-33-105:~/localfile$ terraform validate
Success! The configuration is valid.

ubuntu@ip-172-31-33-105:~/localfile$
```

**Terraform plan-** give you an overview of the resource creation. It will not do anything, it will not make any changes in your machine. Instead, it will show you that this is what the terraform is going to do, it will just give you a overview.

**Terraform apply** - when you give terraform apply whatever changes you have mentioned will be applied. So now you can see. Apply complete resource. One, it is added.

```
ubuntu@ip-172-31-33-105:~/localfile$ ls
file.tf  terraform.tfstate
ubuntu@ip-172-31-33-105:~/localfile$ cd
ubuntu@ip-172-31-33-105:~$ ls
demo.txt  localfile
ubuntu@ip-172-31-33-105:~$ ls -lrt
total 8
-rwxrwxr-x 1 ubuntu ubuntu   12 Jul 10 11:40 demo.txt
drwxrwxr-x 3 ubuntu ubuntu 4096 Jul 10 11:40 localfile
ubuntu@ip-172-31-33-105:~$ cd localfile/
ubuntu@ip-172-31-33-105:~/localfile$ ls
file.tf  terraform.tfstate
ubuntu@ip-172-31-33-105:~/localfile$ vi terraform.tfstate
ubuntu@ip-172-31-33-105:~/localfile$
```

**Terraform.tfstate [state file]**-it is like a record for your telephone. It keeps track of the resources that it's creating, using the terraform file. So state file is very important. If, in case if you want to update. update your infrastructure without the state file, you cannot update your infrastructure.

```
ubuntu@ip-172-31-33-105:~/localfile$ vi terraform.tfstate
ubuntu@ip-172-31-33-105:~/localfile$ cd
ubuntu@ip-172-31-33-105:~$ mkdir ec2
ubuntu@ip-172-31-33-105:~$ cd ec2/
ubuntu@ip-172-31-33-105:~/ec2$ ls
ubuntu@ip-172-31-33-105:~/ec2$ vi ec2.tf
```

**EC2.tf:**

now, where I'm going to launch. I'm going to launch in the cloud environment. Okay, it's not necessary that you have to write this code in the Ec 2. You can also do it in your local mission as well. So if I want now using this **EC2.tf**, file using this terraform template, I'm going to launch my Ec2. Permission. So I have to give. I have to tell terraform, in which provider which provider I want to launch mine. which provider. I want to launch my Ec. 2.

so you always start with the **provided block. The name of the provider is Aws.**

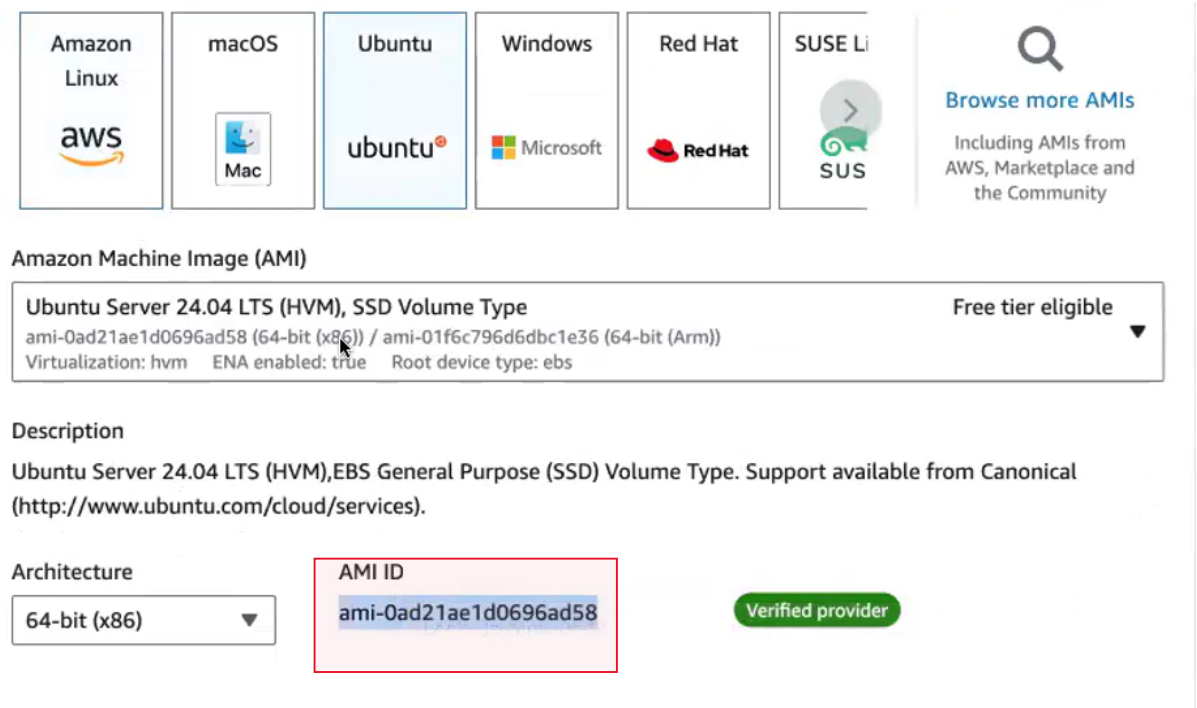
So you mentioned provider, you give the name of your provider and the region. So this is very important. So region is the default thing you have to give next what I'm going to create,

To create a **resource block**. So what type of resource I'm going to create? I'm going to create a aws resource aws Ec2 .so you can use the same thing. So aws underscore instance .aws_instance is keyword.

Then you can give the **name of your resource as a demo**. This can be any anything you can give any name.

So what is the base mandatory parameter. You have to pass the **ami**. So the region. And so, as I told you, ami id are region specific. The region that I am using is Mumbai region. I will go and pick the Mumbai regions ami .

click one eC, 2. Go to launch, instance, because every region the Ami will be different. Okay, so ubuntu 24 will be your terraform will create an ec2 with ubuntu 24 running in it. So go to the ami, go to your resource, block, paste it.




after the ami you have to mention the **instance type**. without the instance type you cannot create, you have to mention the configuration. So we go with **T. 2 micro** itself. Since we are in the free tire, we should always stick with the T. 2. Micro.

you have to name your instances. So how do you name your Ec 2 .You have to **give a tag.** You give the instance type and tag tag is optional. But just to identify, just to name your Ec. 2 mission, or even without the tag also your Ec. 2 mission will be

created, but it will get created without the name you have to give the name after creation. Okay, you have to manually give the name.

```
provider "aws" {
region = "ap-south-1"
}

resource "aws_instance" "demo" {
        ami = "ami-0ad21ae1d0696ad58"
        instance_type = "t2.micro"
        tags = {
                Name = "myterraform-instance"
        }
}
```

Save this file.

Your terra account has to talk with your aws account to create your Ec2 comunicate, to your terra account cannot go and randomly talk with any of your aws account. It should have some authentication. You have to **1st authenticate your terraform**, so that your terraform will get all the permission to create your ec2 for that. What you need you need **Aws Cli.**

**So let me 1st install aws clients install.**

```
ubuntu@ip-172-31-33-105:~/ec2$ sudo apt install awscli
Reading package lists... Done
Building dependency tree... 50%
```

so to the root account, using the root account you cannot create, you cannot use the terraform you. You should have an Iam account with the IAM account how you do aws configure same thing. They are going to do it here we will be doing aws configure after doing the aws configure, that is, after logging. In using your proper iam account only, then you can run, execute this terraform.

we give **aws configur**. so it'll ask you for the keys. If you already have the key, you can make use of your old key itself else will have to go and generate it. Okay, now, I have configured my IAM account.

```
ubuntu@ip-172-31-33-105:~/ec2$ aws configure
AWS Access Key ID [None]: AKIASGIII4IGBNATXP47
AWS Secret Access Key [None]: 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
Default region name [None]: ap-south-1
Default output format [None]: json
ubuntu@ip-172-31-33-105:~/ec2$ ls
ec2.tf
ubuntu@ip-172-31-33-105:~/ec2$
```

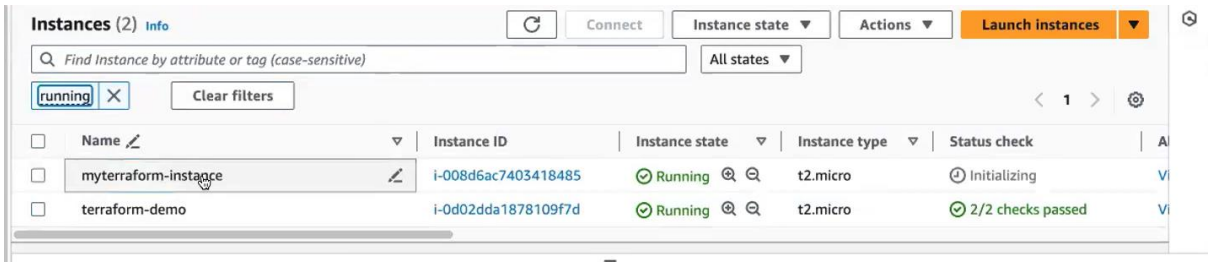
You can see now I haven't executed. **My terraform file will not find the state file, only after you execute the terraform file. you can find the State file.**

you have to initiate. So you have to initiate this directory. This is a new directory. You have to initiate with terraform. So this terraform init command will, it will set up the environment for you to create the resources in the Aws. So terraform initialization is very important.

1. **Terraform initi**
2. **Terraform validate**
3. **Terraform plan**
4. **Terraform apply.** So only this terraform apply command will create an Ec2 Instance for you. After that terraform state file created.
   **Terraform.tfstate** **[state file created].** created after give terraform apply

```
ubuntu@ip-172-31-33-105:~/ec2$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.57.0...
- Installed hashicorp/aws v5.57.0 (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.
ubuntu@ip-172-31-33-105:~/ec2$ terraform validate
Success! The configuration is valid.

ubuntu@ip-172-31-33-105:~/ec2$ terraform plan
```

```
ubuntu@ip-172-31-33-105:~/ec2$ terraform apply
```




**Ec2 instance created [ myterraform_instance ].state file also created .**same way you can launch multiple Ec, 2 machines with a single terraform file, you can launch multiple Ec 2 .

**the terraform state file**. Now, if you want to make any modification to your instance, in case if you want to launch a different instance with a different instance type, you want to change it, you can do it. You can change it what you can do you can. You can go and give instead of T2micro. If you want to launch T2medium. it will edit it. It will edit the Configuration. Your terraform will edit the Configuration, and it will recreate the instance with this instance block. So how your terraform will be created because it does have the state file with the help of state file. It will edit or it will update the instance type. You don't have the state file. You wanted to know altogether. It will create a new instance for you.

**I just create in another directory. Okay, inside this directory. What I will do. I will create a Terra account file.**

```
ubuntu@ip-172-31-33-105:~/ec2$ vi terraform.tfstate
ubuntu@ip-172-31-33-105:~/ec2$ vi ec2.tf
ubuntu@ip-172-31-33-105:~/ec2$ cd
ubuntu@ip-172-31-33-105:~$ mkdir samle
ubuntu@ip-172-31-33-105:~$ cd
ubuntu@ip-172-31-33-105:~$ cd S
```

so your main terraform file, where you're creating all your resources should be named as main.tf, though you can give any name, but it does not suggest the proper naming. Standard is you always give the main resource file where you're creating all your resources. That file should be named as main.tf.

**I will be creating 2 I will be creating 2 instances in a different region [Mumbai & ohio]**

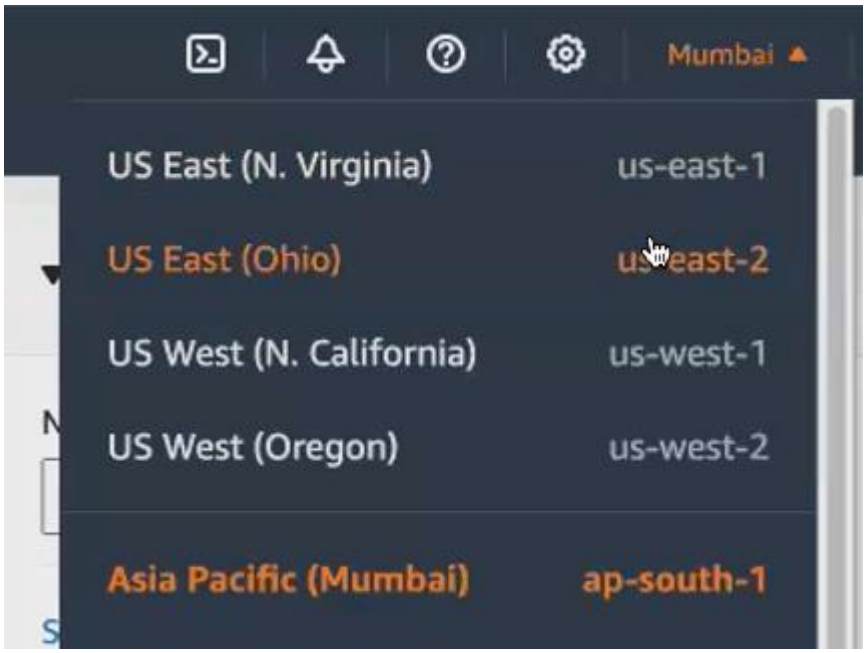
1. Give provider block .inside mention **alias and region name**

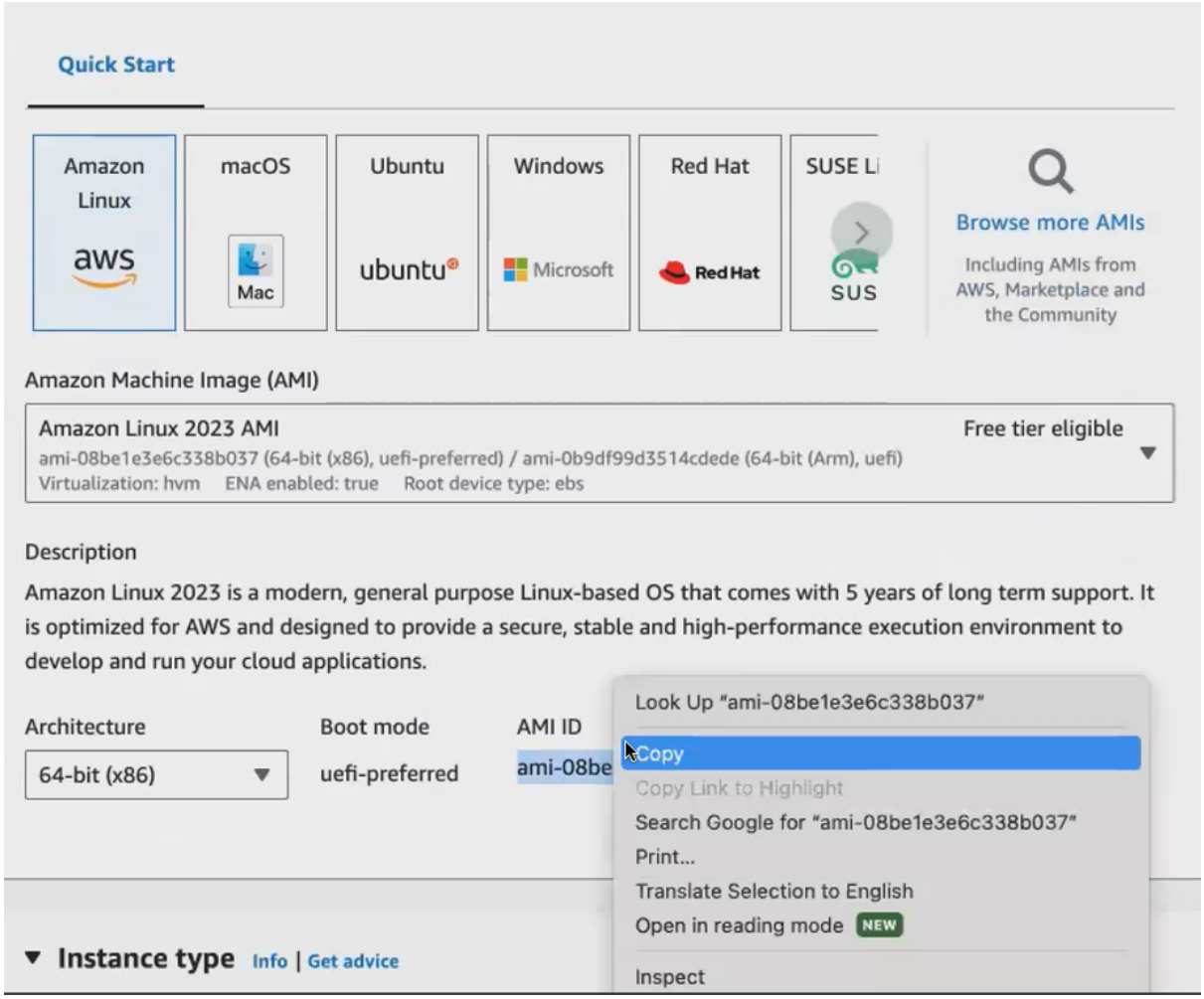
Alias means alias what it does it will allow you. It will help you to specify which provider using which provider of which region you want to create the easy to mission. So mainly, if you're configuring multiple instances with multiple region, then you can go for this. Als.

2. **we will give the resource block**

- **aws_instance name**
- **provider [give alias region name]**
- **ami id [take this from already created instance(myterraform-instance) in Mumbai region ,instance type**
- **user data**

3. here under advanced detail. You will also **give user data**. So we will see how to give this user data in your terraform template. Okay, so I will. Now, I will use a user data. I will give a user data. inside the user data, you give only your shell script same way. You can give this a shell script here.
4. Go and select another region (ohio) copy ami id and change the region. Remaining was same.





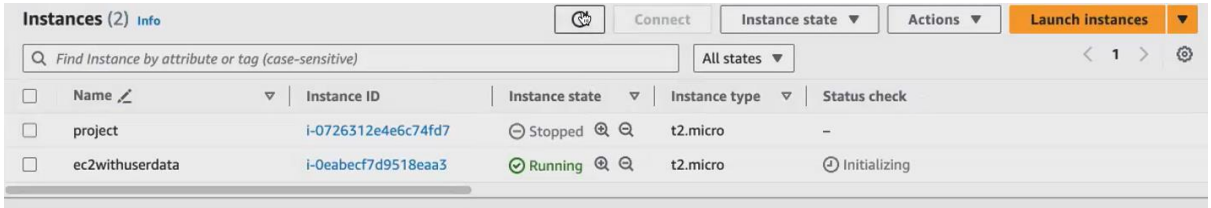

```
provider "aws" {
        alias = "ap_south_1"
        region = "ap-south-1"
}
provider "aws" {
        alias = "us_east_2"
        region = "us-east-2"
}

resource "aws_instance" "demo1" {
        provider = aws.ap_south_1

        ami = "ami-0ad21ae1d0696ad58"
        instance_type = "t2.micro"
        user_data = <<--EOF
                    #!/bin/bash
                    echo "Hello from ap-south-1" > /home/ubuntu/hello.txt
                    EOF

        tags = {
                Name = "ec2withuserdata"
        }
}
```

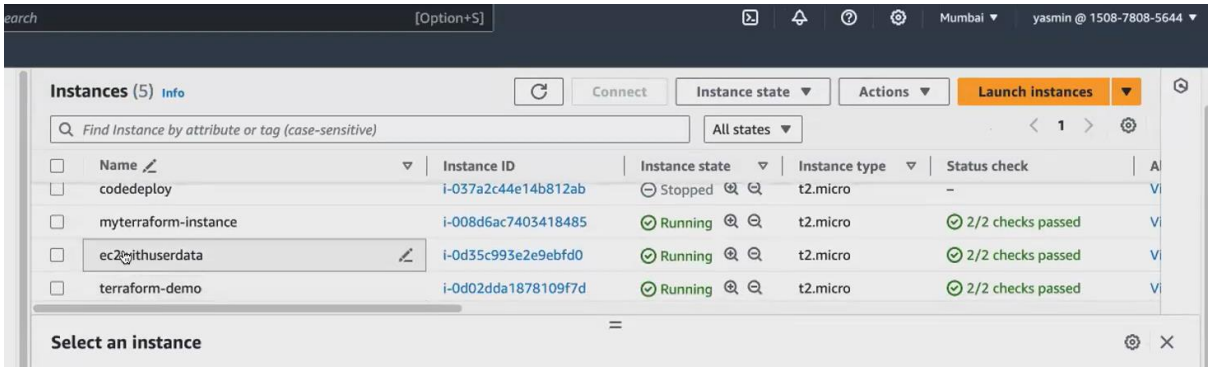
```
resource "aws_instance" "demo1" {
        provider = aws.ap_south_1

        ami = "ami-0ad21ae1d0696ad58"
        instance_type = "t2.micro"
        user_data = <<-EOF
                    #!/bin/bash
                    echo "Hello from ap-south-1" > /home/ubuntu/hello.txt
                    EOF

        tags = {
                Name = "ec2withuserdata"
        }
}
resource "aws_instance" "demo2" {
        provider = aws.us_east_2

        ami = "ami-08be1e3e6c338b037"
        instance_type = "t2.micro"
        user_data = <<-EOF
                    #!/bin/bash
                    echo "Hello from us-east-2" > /home/ec2-user/hello.txt
                    EOF

        tags = {
                Name = "ec2withuserdata"
        }
}
:wq
```

Give all commands one by one.

1. **Terraform initi**
2. **Terraform validate**
3. **Terraform plan**
4. **Terraform apply.** So only this terraform apply command will create an Ec2 Instance for you. After that terraform state file created.
   **Terraform.tfstate** created after give terraform apply

```
  Enter a value: yes

aws_instance.demo1: Creating...
aws_instance.demo2: Creating...
```



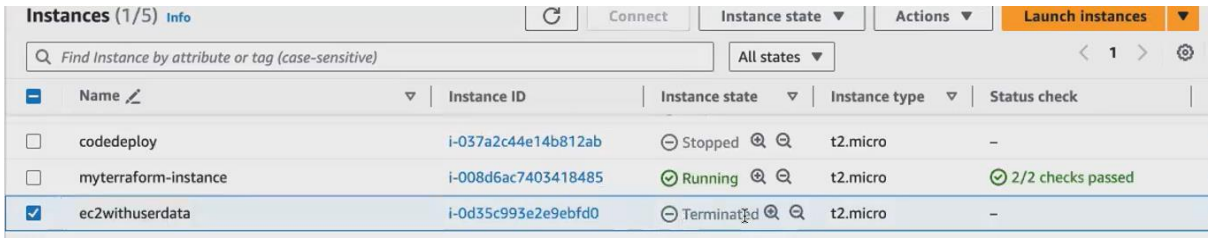



## How to delete that instance:

1. **Terraform destroy** [using the the terraform destroy will only destroy the resources that we have created inside the current directory ].in 1 directory we have to create only one terraform file

```
ubuntu@ip-172-31-33-105:~/samle$ terraform destroy
aws_instance.demo1: Refreshing state... [id=i-0d35c993e2e9ebfd0]
```

```
aws_instance.demo1: Destroying... [id=i-0d35c993e2e9ebfd0]
aws_instance.demo2: Destroying... [id=i-0eabecf7d9518eaa3]
aws_instance.demo1: Still destroying... [id=i-0d35c993e2e9ebfd0, 10s elapsed]
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 10s elapsed]
aws_instance.demo1: Still destroying... [id=i-0d35c993e2e9ebfd0, 20s elapsed]
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 20s elapsed]
aws_instance.demo1: Still destroying... [id=i-0d35c993e2e9ebfd0, 30s elapsed]
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 30s elapsed]
aws_instance.demo1: Still destroying... [id=i-0d35c993e2e9ebfd0, 40s elapsed]
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 40s elapsed]
aws_instance.demo1: Still destroying... [id=i-0d35c993e2e9ebfd0, 50s elapsed]
aws_instance.demo1: Destruction complete after 50s
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 50s elapsed]
aws_instance.demo2: Still destroying... [id=i-0eabecf7d9518eaa3, 1m0s elapsed]
aws_instance.demo2: Destruction complete after 1m3s

Destroy complete! Resources: 2 destroyed.
ubuntu@ip-172-31-33-105:~/samle$
```




## Terraform code:

```
provider "aws" {
alias = "ap_south_1"
region = "ap-south-1"
}
provider "aws" {
```

```
alias = "us_east_2"
region = "us-east-2"
}
resource "aws_instance" "demo1" {
provider = aws.ap_south_1
ami = "ami-0ad21ae1d0696ad58"
instance_type = "t2.micro"
user_data = <<-EOF
#!/bin/bash
echo "Hello from ap-south-1" > /home/ubuntu/hello.txt
EOF
tags = {
Name = "ec2withuserdata"
}
}
resource "aws_instance" "demo2" {
provider = aws.us_east_2
ami = "ami-08be1e3e6c338b037"
instance_type = "t2.micro"
user_data = <<-EOF
#!/bin/bash
echo "Hello from us-east-2" > /home/ec2-user/hello.txt
tags = {
Name = "ec2withuserdata"
}
}
```

**Terraform commands:**

```
terraform init
terraform validate
terraform plan
terraform apply
```

**Task for this class: Launch Linux EC2 instances in two regions using a single Terraform file.**