**Infrastructure As Code(IAC)**

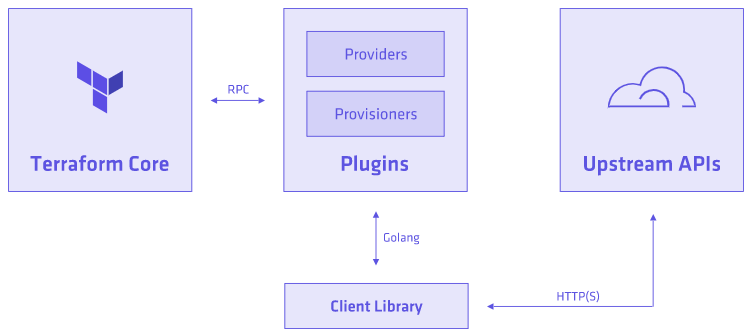
IAC is the concept of managing and provisioning the infrastructure using the configuration files which contains the instructions usually we called it as a code.

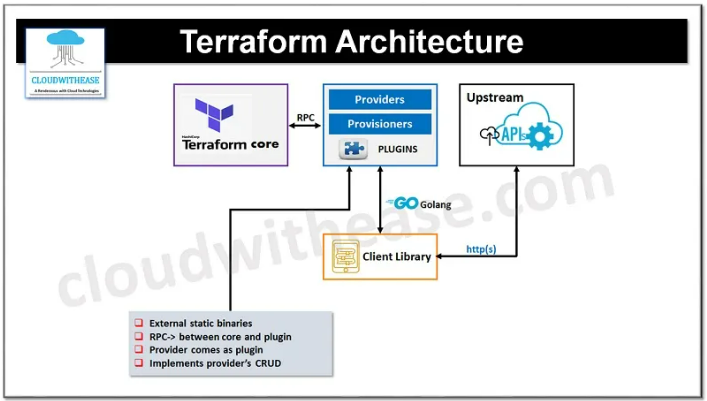
Infrastructure provisioning, is the process of setting up the resources like VM’s, LB’s, VPC etc and also make them available for the users and other systems who wants to use them.

Terraform is an **infrastructure-as-code** software tool that enables us to build and ,manage the infrastructure. Terraform is an open source tool developed by HashiCorp using the “Go” programming language. These Go language files are compiled into a binary file called “terraform” which makes the requests to the cloud providers’ using the Application Programming Interface(API).

Terraform makes theses requests based on user written infrastructure templates. These templates are simple text files and wriiten in json and saved with an extension “.tf”.

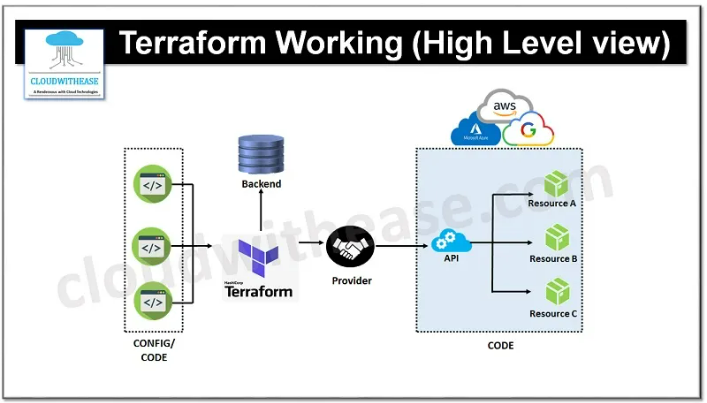
**Terraform Architecture**





**Terraform Core:**

Terraform’s Core is another terraform project, which will reads the terraform configuration files and manages the dependencies of the resources mentioned in the configuration files with an ordering.

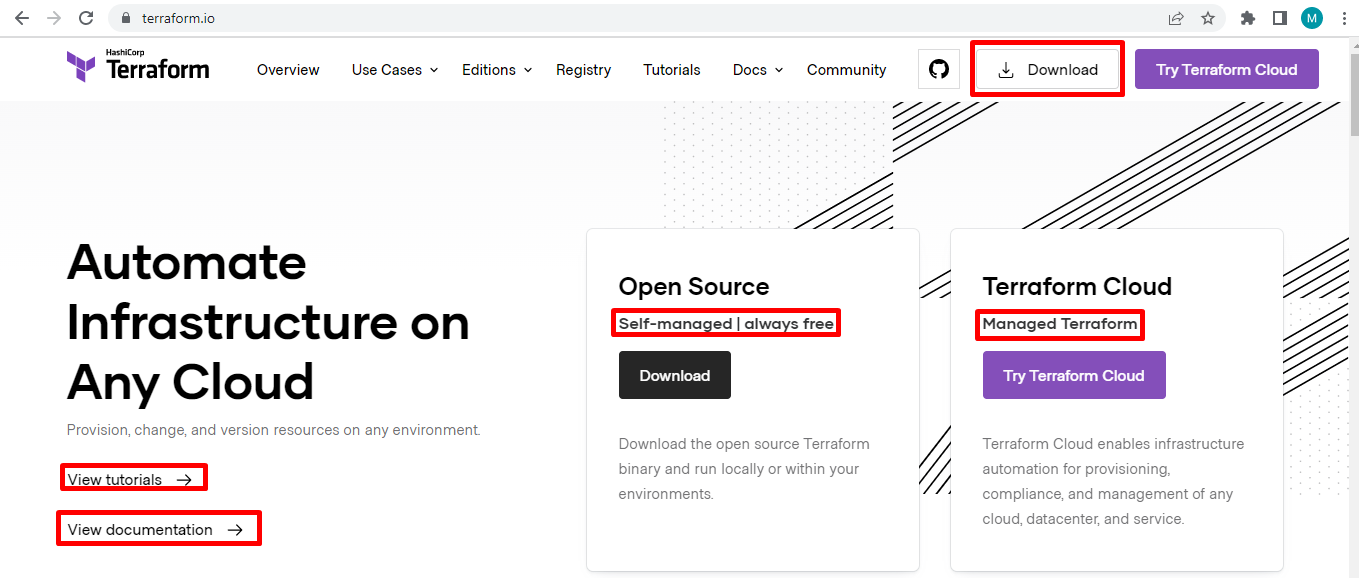


**Terraform Provider**

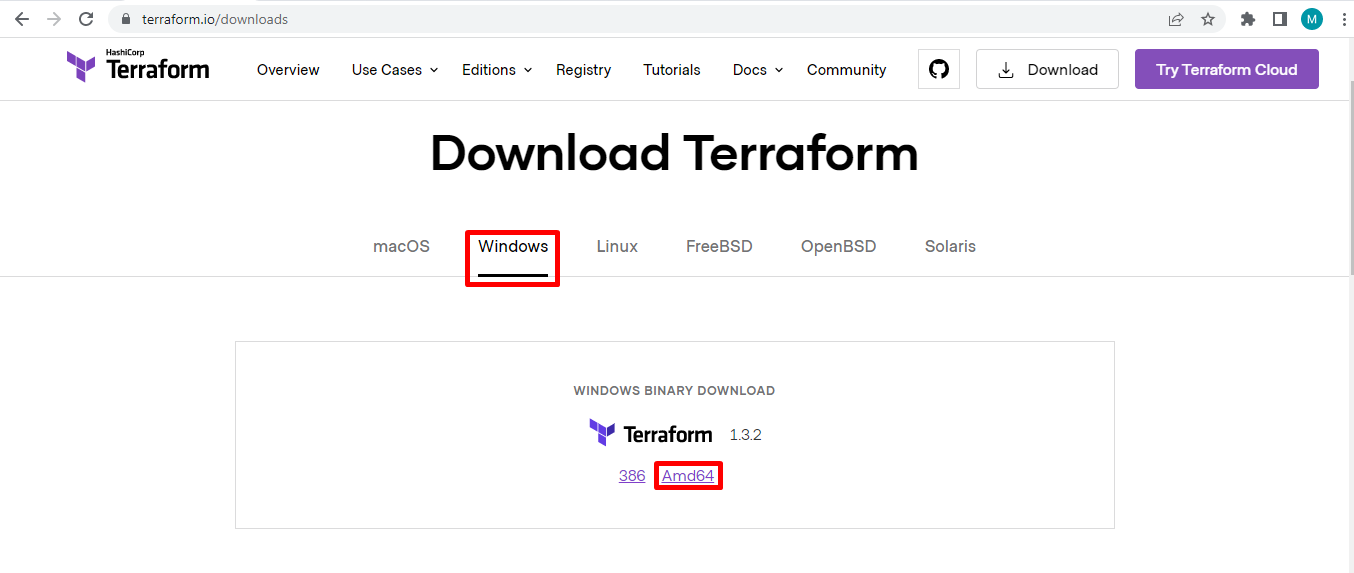
Terraform provider will used to connect to the providers of type cloud and on-premise. Terraform is supporting 34 official cloud providers. When we mentioned the provider details in the configuration file, then terraform provider will use the plugins and go language client library to connect to the providers API like AWS, AZURE, GCP etc using their up stream API.

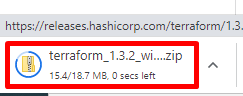
**Terraform Setup**

Step-1: go to terraform official web site and click on the down load button as shown below:

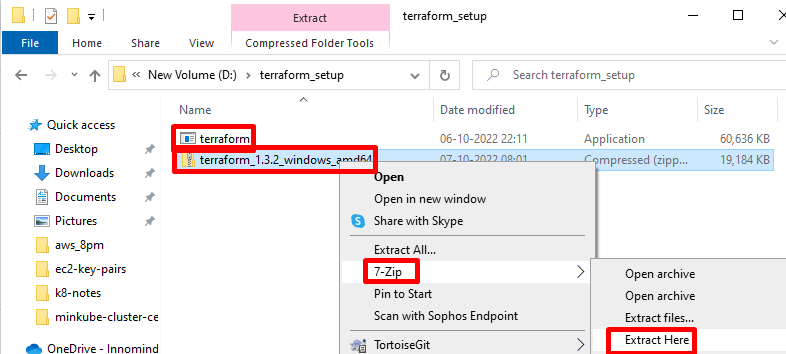


Step-2: once we click on the above download button, the following page will be opened to down load the terraform binary for the various Operating Systems. Choose the windows 64 bit OS binary to download as shown below:

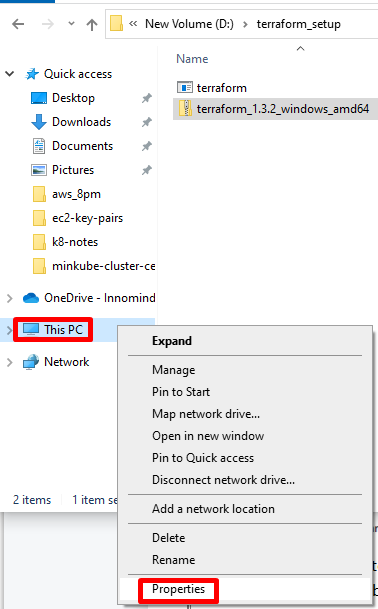


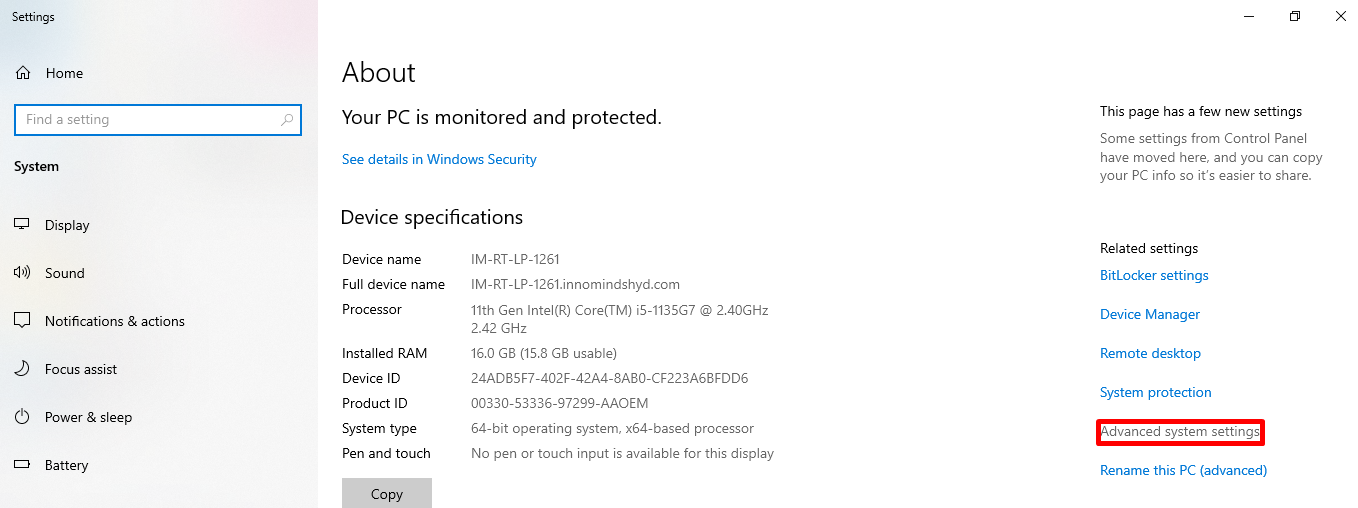


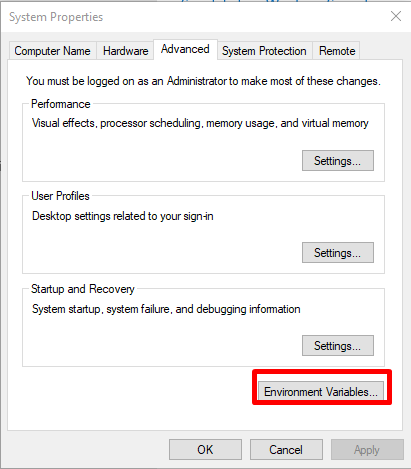
Step-3: create any of the folder and copy the downloaded binary to the folder and extract it as shown below:

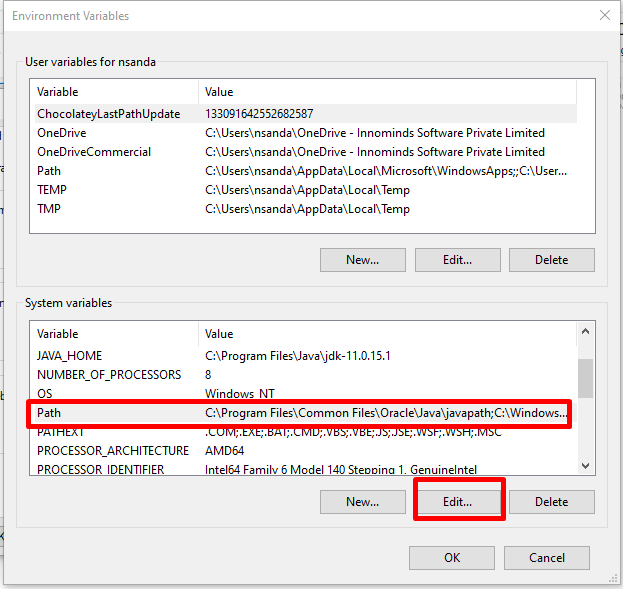


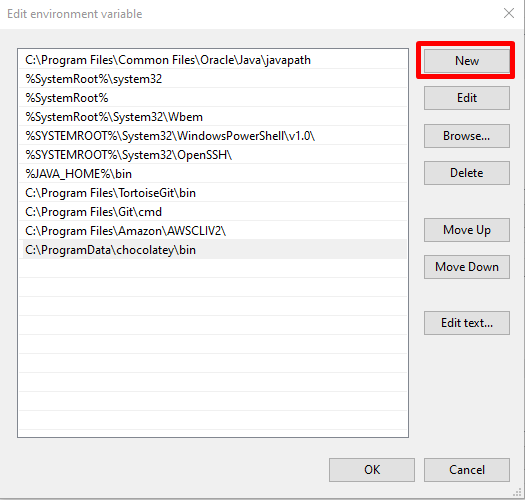
Step-4: once after the terraform binary extracted, set the binary path in system variables path as shown below to available it from any of the windows folder

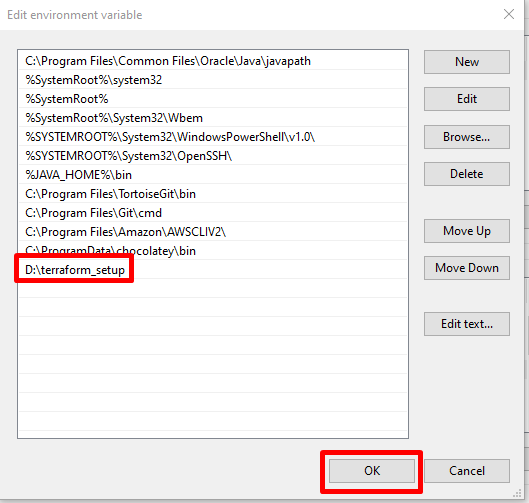




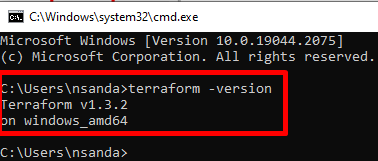






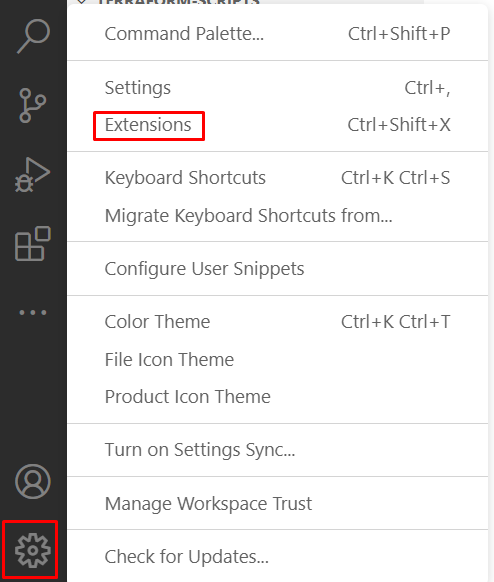


Once the path setup has been done open the command prompt and then check the terraform version as shown below:

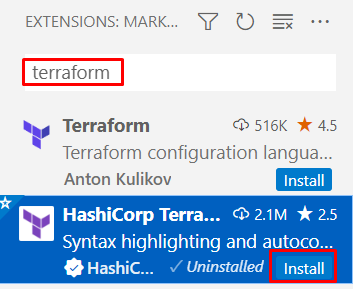


**Setup Visual Studio Code IDE**

Step-1: open visual studio code IDE and go to manage settings, choose the extensions as shown below:



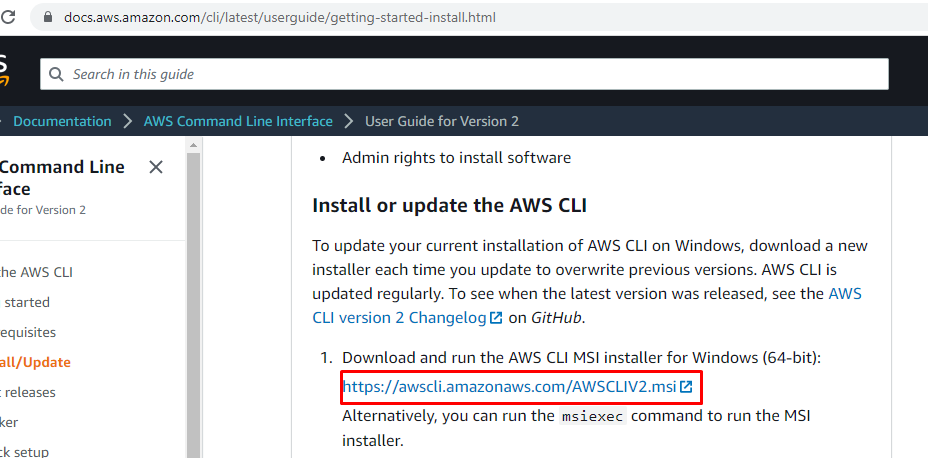
Step-2: once the extensions window open, search for terraform extension, and install the Hashicorp Terraform extension as shown below:



**Install and Setup AWS CLI**

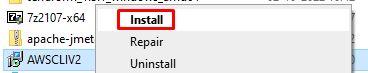
Step-1: go to the url <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

And install the AWS Cli for windows as shown below:

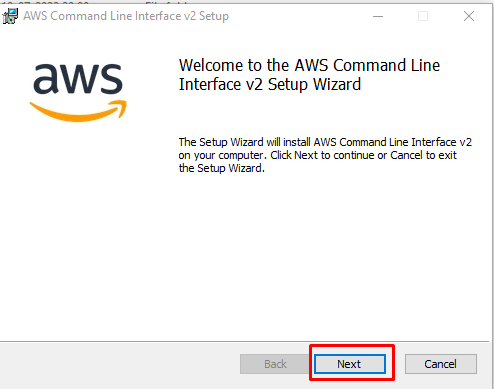


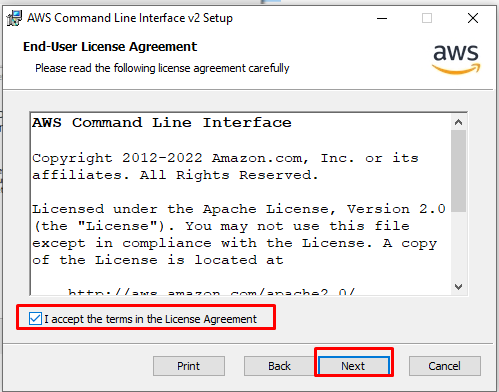


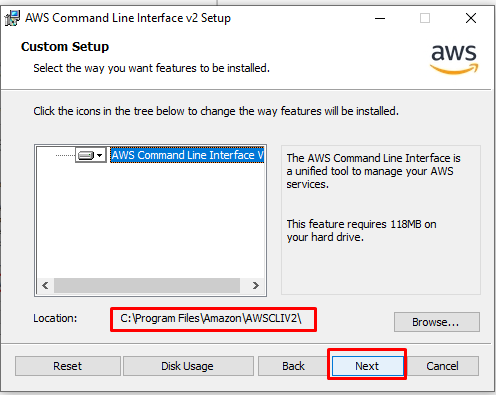
Step-2: once AWS Cli installed, right click on it and click install as shown below:

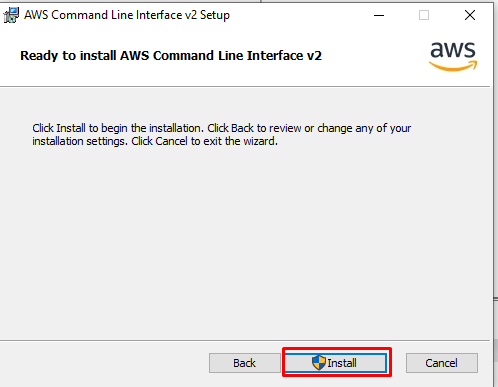


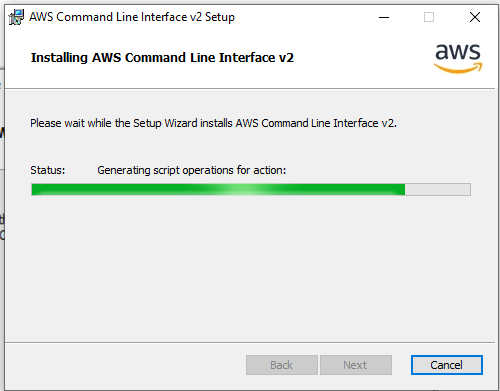
Step-3:

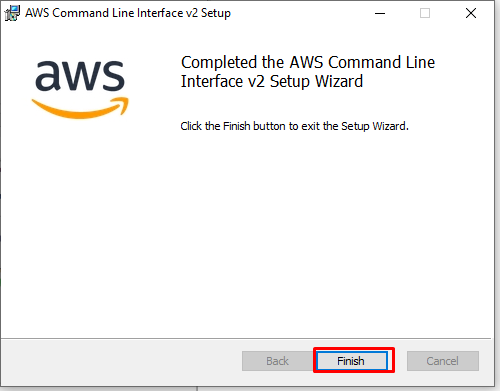




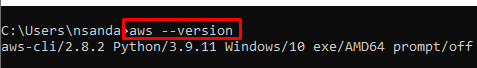




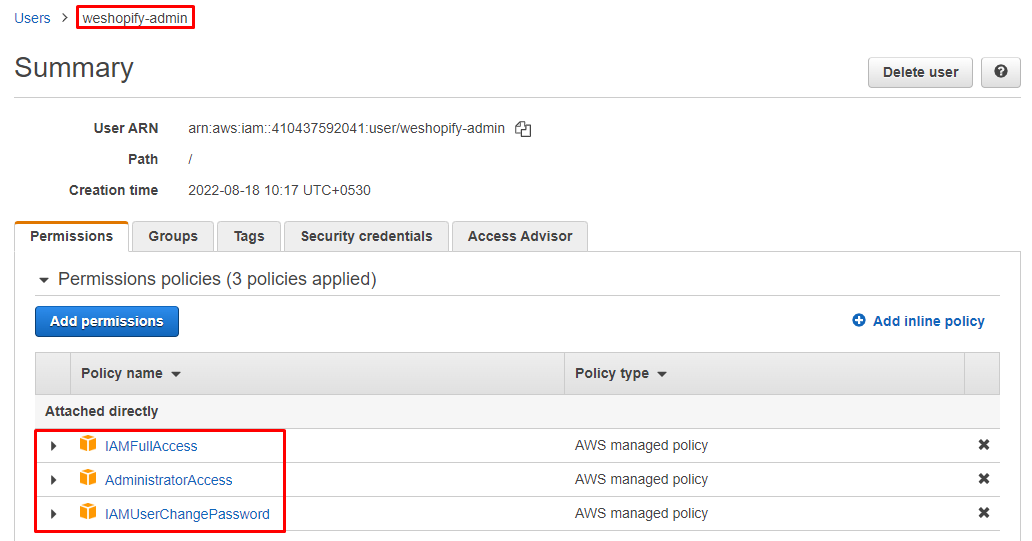




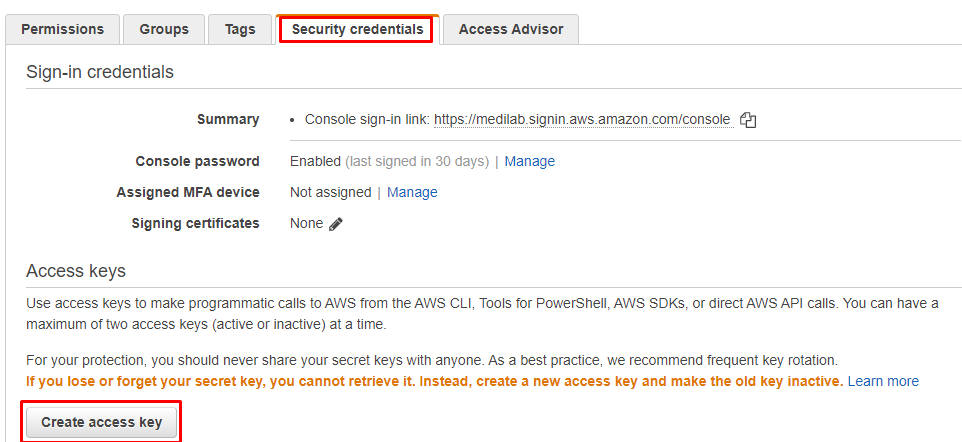
Step-3: to check the AWS CLI installation done successfully open command prompt and type AWS



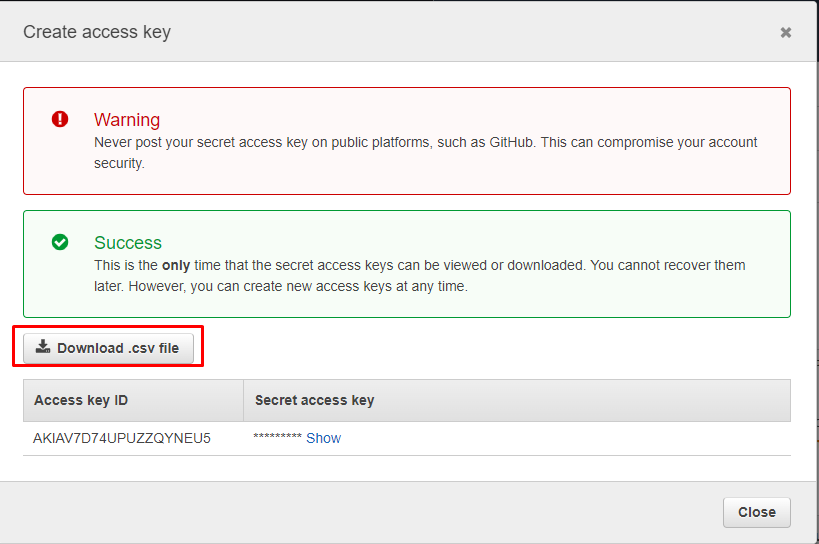
Step-4: create an IAM User with the administrative access to connect to the AWS Cloud from the AWS CLI.



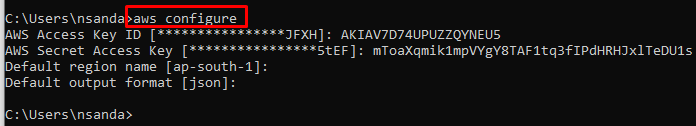
Step-5: create access key for the user



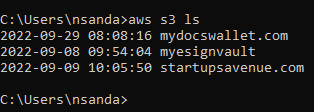
Step-6: once the access key generated download the csv file as shown below:



Step-7: connect to the AWS cloud from the AWS CLI using the below command



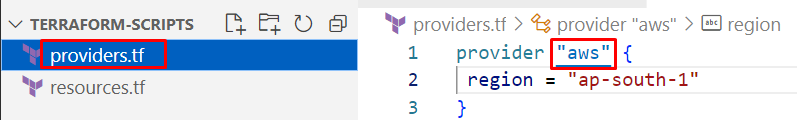
Step-8: to confirm whether we have connected successfully or not , retrive the resources like s3 buckets, ec2 instances, vpc etc as shown below:



**Writing the Terraform Scripts**

While writing the terraform script, always the first script we will write is provides.tf where we have to mentioned, to which cloud provider we want to connect and provision the infrastructure.

Step-1: create **providers.tf** file

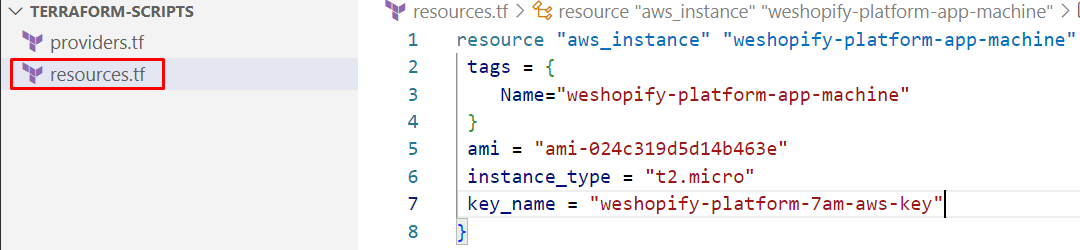


Step-2: create **resources.tf** file

Lets create ec2 instance, to do this we need the following details:

* Tagging
* AMI
* Instance Type
* KeyPair
* Network settings: VPC, Security Group

So the same information we also have to provide in the terraform script to create an ec2 instance.



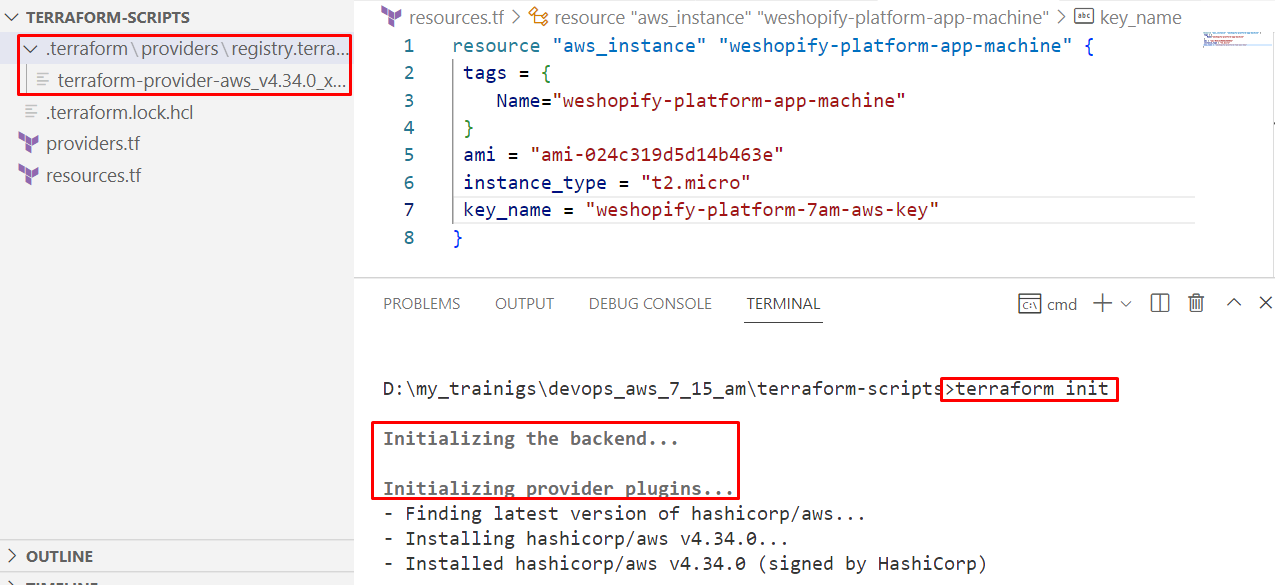
**Note:** in the above screen shot, we missed the security group which is required to connect to the ec2 machine. Hence please refer the below screen shot. Also the security group id should be match with the vpc in which we are creating the ec2 instance.



**Deploying the terraform scripts**

**Terraform initialization:**

When we type **terraform init** at the location where we have the provider.tf file, then the provider mentioned in the file along with its plugins will get download as shown below. This mandatory step we need to do before deploying any script and it should be a one time activity.



**Terraform scripts deployment**

To deploy the terraform scripts we will use the terraform work flow by initialization the terraform provider.

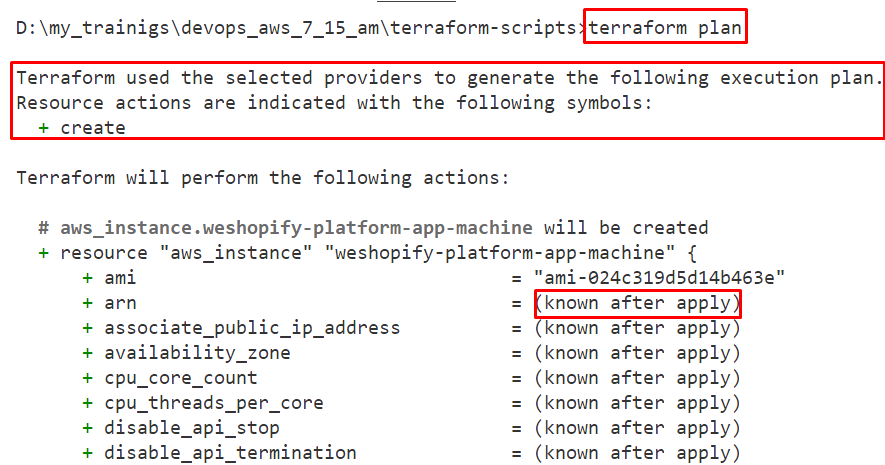
**The core Terraform workflow has three steps:**

1. Write - Author infrastructure as code.

This step we already have done in the above steps like by writing the terraform scripts for the ec2 resource provisioning.

1. Plan - Preview changes before applying.

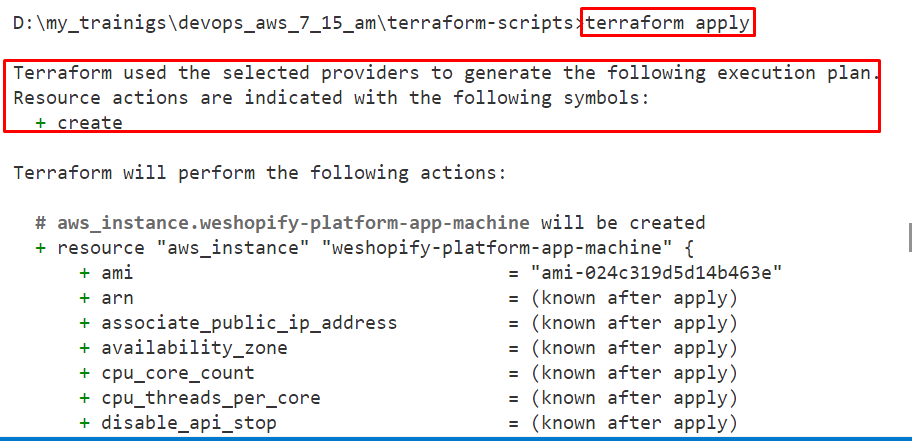
Once the script writing is done, the Plan phase of work flow will be used to ensure that the script written with correct syntax and is valid. if valid then it will create a plan for deployment as shown below:

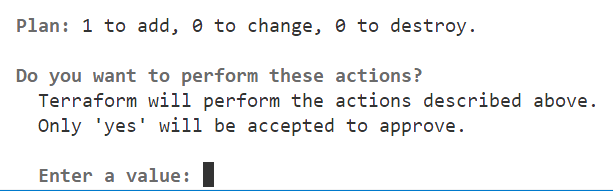


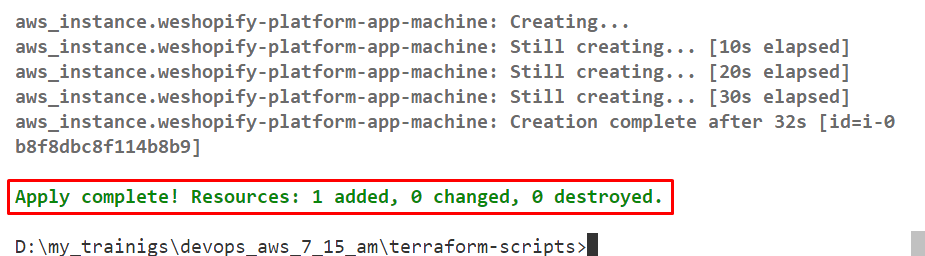
Here “known after apply” means the next phase followed by the plan is the “apply” so the value for the resource attributes for which we have not mentioned, the terraform will insert the default values when the script will be deployed by using the “apply”

1. Apply - Provision reproducible infrastructure.

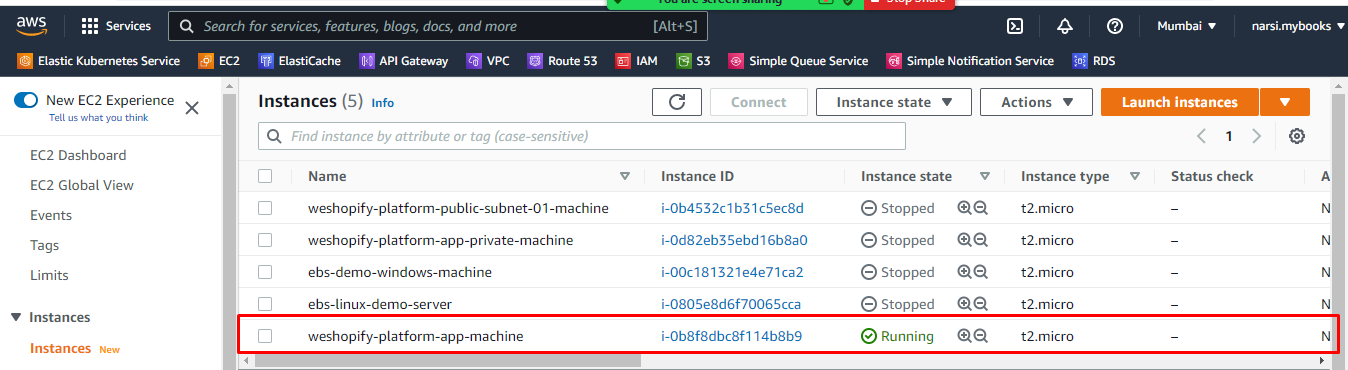
As soon as the plan (a deployment template) has been prepared with the resources values we have given in our resources.tf file, we have to apply the plan for the targeted cloud I.e. AWS in our case.





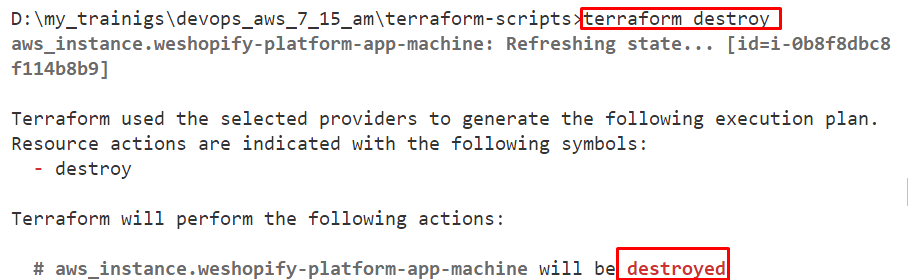


We can see now the ec2 instance created and running in the aws management console as shown below:



**Destroying/Deleting the resources created**

Terraform destroy will delete the recently deployed plan and delete the resources as shown below:





**Production Architecture Setup using Terraform**

Our Production Architecture contains a VPC in one of the AWS region lets say Mumbai I.e. ap-south-1 and with two subnets I.e. public-cloud-subnet and private-cloud-subnet.

These two subnets again will be in two availability zones I.e. ap-south-1a and sp-south-1b.

The AWS resources like ec2, RDS, k8s resources etc should be in private-cloud-subnet with the NAT gateway.

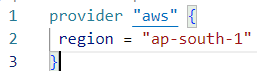
The public-cloud-subnet contains the resources like Load Balancer, Gateway etc with the Internet gateway.

To map these NAT Gateway and Internet gateway to the subnets, we have to take two route tables by attaching NAT gateway and Internet gateway to them.

|  |  |  |  |
| --- | --- | --- | --- |
| **AWS Resource** | **AZ** | **RESOURCE Attached To** | **Region** |
| Public route table | ap-south-1a, ap-south-1b | Public route table | Ap-south-1 |
| Private route table | ap-south-1a, ap-south-1b | Private route table | ap-south-1 |
| VPC(10.0.0.0/16) | ap-south-1a, ap-south-1b | Public route table | ap-south-1 |
| Public-cloud-subnet | ap-south-1a((10.0.10.0/24), ap-south-1b((10.0.20.0/24) | Public route table | ap-south-1 |
| Private-cloud-subnet | ap-south-1a((10.0.30.0/24), ap-south-1b((10.0.40.0/24) | Private route table | ap-south-1 |
| NAT Gateway | ap-south-1a, ap-south-1b | Private route table | ap-south-1 |
| Internet Gateway | ap-south-1a, ap-south-1b | Public route table | ap-south-1 |

**Providers Script**

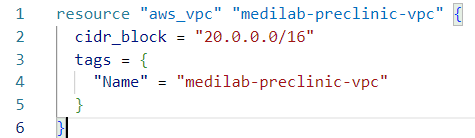
Create a file with any name like **providers.tf** with an AWS provider to connect to the aws cloud perticular region



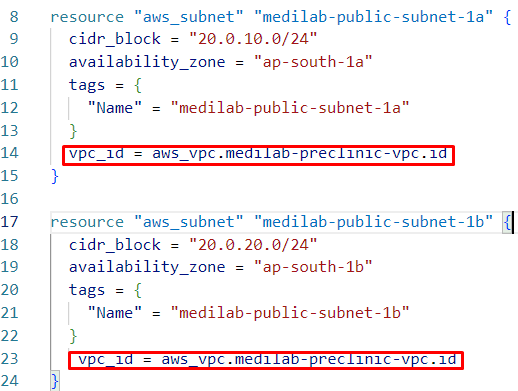
**AWS VPC Script**

**Script-1: Custom VPC Creation**

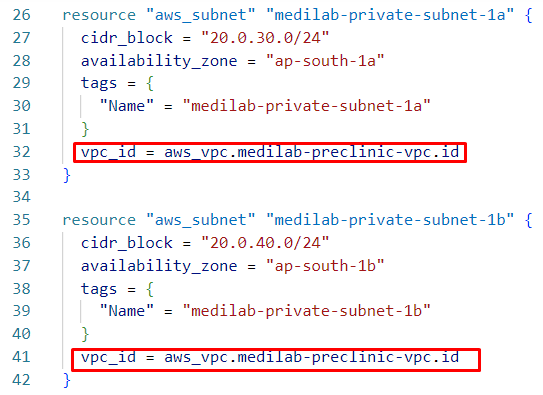
Step-1: Create a file with any name like **vpc\_resources.tf** and add the below script.



Step-2: Create the Public subnets in the same file I.e. **vpc\_resources.tf**  and attach it to the VPC created above:

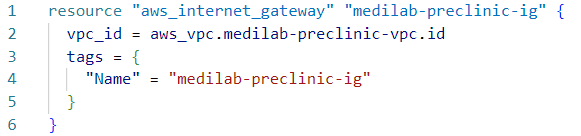


Step-3: Create Private Subnets in the same file I.e. vpc\_resources and attach it to the VPC

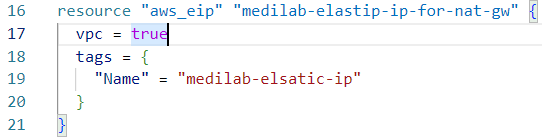


**Gateway Resources Script**

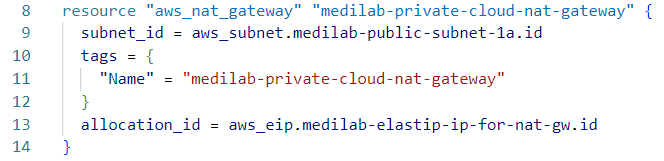
Step-1: Create the internet gateway as shown below:



Step-2: Create Elastic IP because it requires to attach it to the nat-gateway as shown below:

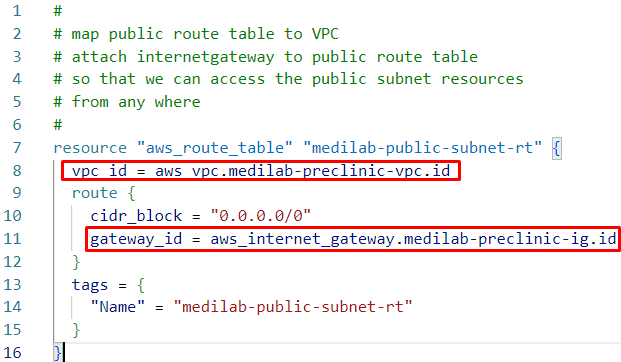


Step-3: Create Nat-Gateway as shown below:

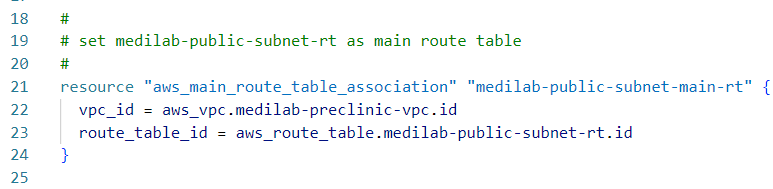


**Public Route Table Script**

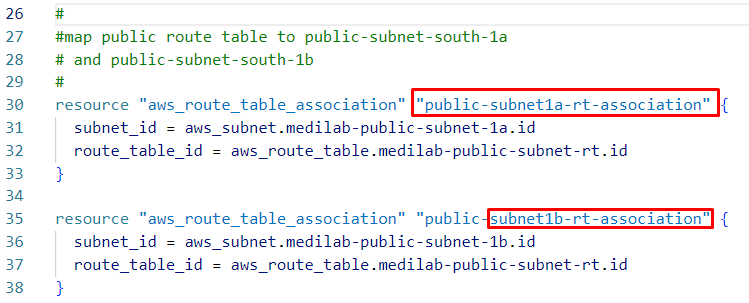
Step-1: Create a file with any name like **public\_route\_table.tf** and add the following configuration to create the route table by attaching the internet gateway to this route table.



Step-2: Set public Route Table created above as the main route table.

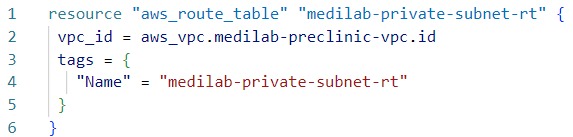


Step-3: Associate the route table with the public subnets.

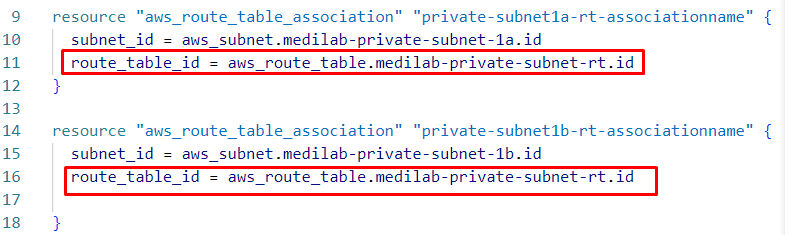


**Private Route Table Script**

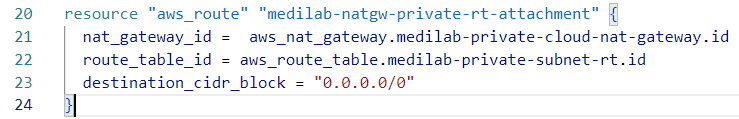
Step-1: Create a file with the any name like **private\_route\_table.tf** and add the following configuration to create the route table



Step-2: Associate the route table to the private subnets.



Step-3: Attach the nat gateway to the route table created above to make it as a private route table.



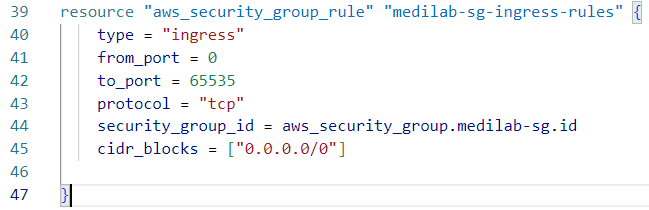
**Security Group Script**

Step-1: Create a file with any name like “security\_group.tf” and add the below script to create the security group which we will use fof the ec2 machines we will create in the custom VPC



Here egress rules means the outbound rules where the ec2 instances required to connect to external network like when updating packages using “apt-get update”.

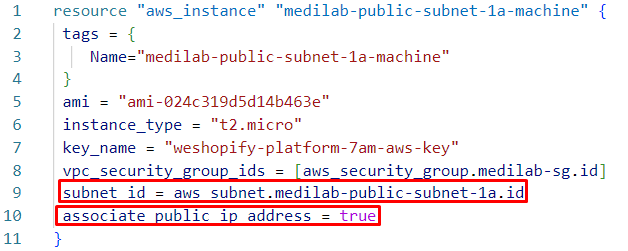
Step-2: Configure the inbound rules



Here inbound/ingress rules means, from the external network we can access the applications inside the ec2 instances.

**Ec2 Resources Script**

Step-1: Create an ec2 machine in public subnet by **enabing the public ip assignment.**



Step-2: Create an ec2 machine in private subnet, here don’t assign any public ip

