**IMPLEMENTING SCALABLE INFRASTRUCTURE USING TERRAFORM**

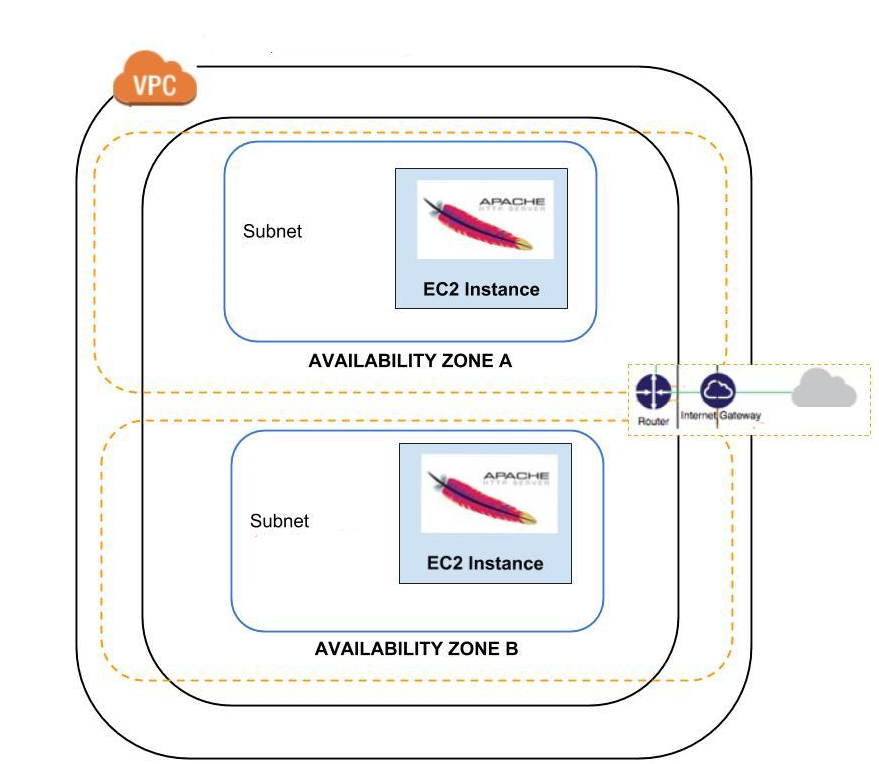
What is Terraform?

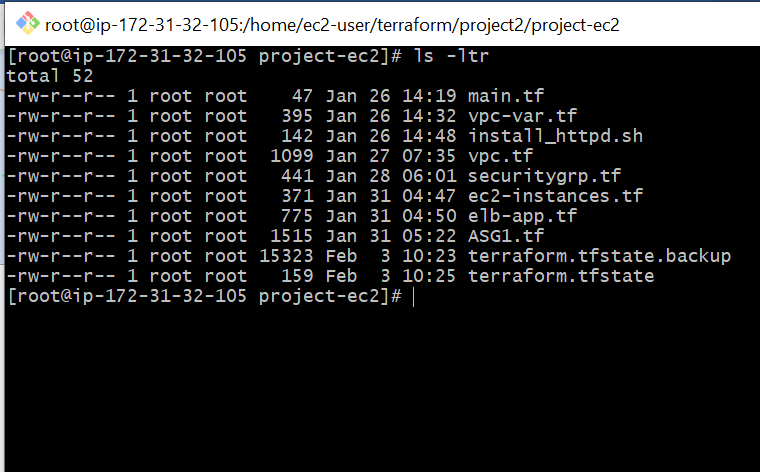
Terraform is an open sources infrastructure as a code **tool** and a product of HashiCorp. Which involves building, changing and versioning of infrastructure efficient and safely. Terraform's code is written in HashiCorp's proprietary language called **H**ashiCorp **C**onfiguration **L**anguage (**HCL**). HCL is a structured configuration language that is intended to be both machine friendly and human readable.

What are all the infrastructure’s providers supported by terraform?

1. Amazon webservices.
2. Microsoft Azure.
3. IBM Cloud(Bluemix).
4. Google cloud platform.
5. Oracle cloud infrastructure.
6. VMware’s OpenNebula.

**Architecture:**

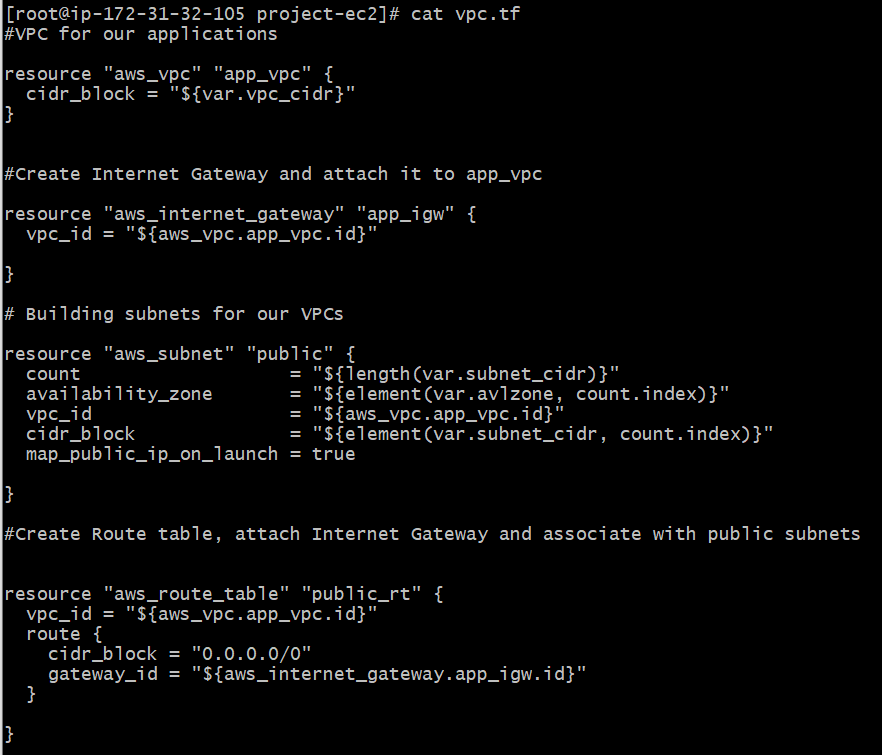
****

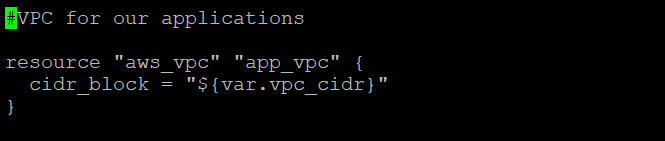


**VPC:**

Virtual private cloud (VPC) is virtual network dedicated to the aws.

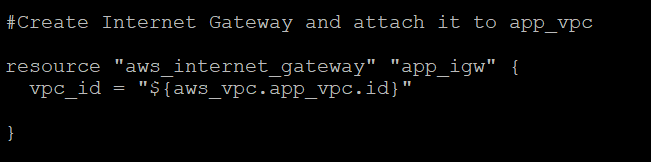
In terraform this is the way to declare the custom vpc and we must specify a range of IPv4 addresses for the VPC in the form of a classless Inter-Domain Routing (CIDR) block. Example: 10.0.0.0/16, this is the primary CIDR block for my VPC.





**INTERNET GATE WAY(IGW):**

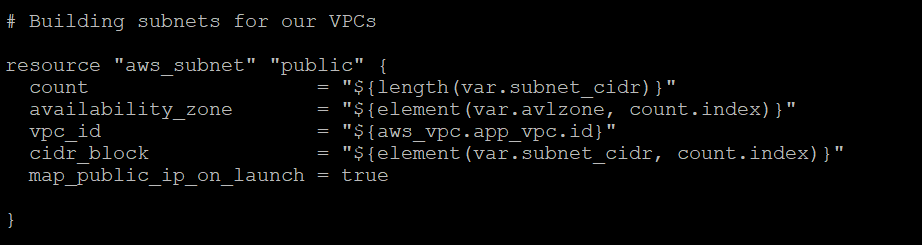
IGW is one of the VPC component that will allow the communication between instances in your VPC and the internet. We can have 5 IGW per REGION but one per VPC.



**SUBNETS:**

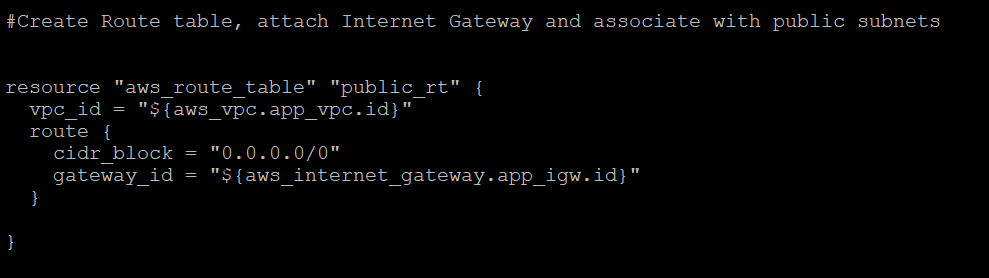
It’s like containers within your VPC that segment off a slice of the CIDR block you define in your VPC. Subnets allows us to give different access rules and place resources in different containers where those rules need to be applied.

There are two ways we can assign public\_IP one at the time of subnet creation and another at the time of instance creation but I have defined in subnet creation. While working with terraform we should make this “map\_public\_ip\_on\_launch” flag as true.



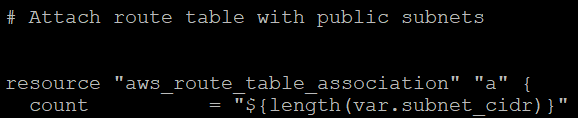
**ROUTE TABLE:**

A route table contains a set of rules, called routes, that are used to determine where network traffic from your subnet or gateway is directed. In this case I am directing to all.



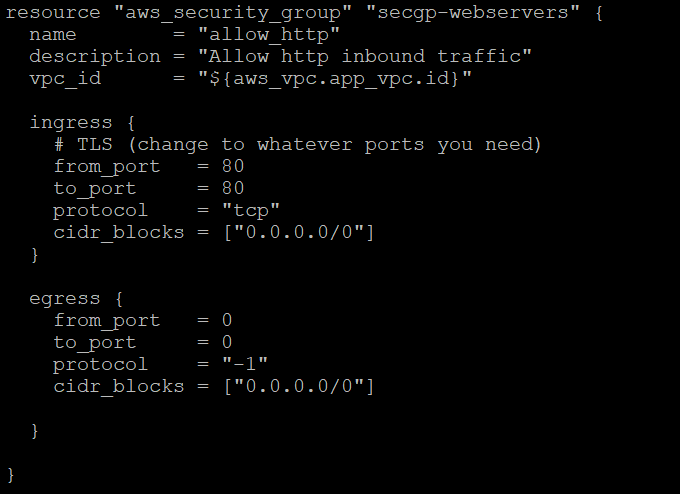
**ROUTE TABLE ASSICIATION WITH SUBNETS:**

Each subnet in our custom VPC must be associated with a route table, which controls the routing for the subnet. I must explicitly associate a subnet with a particular route table. Otherwise, the subnet is implicitly associated with the main route table that is the default route table created.



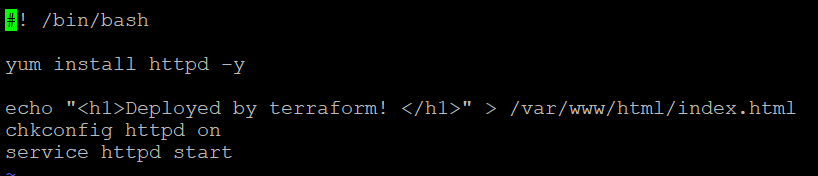
**SECURITY GROUP:**

AWS security groups (SGs) are associated with **EC2** instances and provide security at the protocol and port access level. Each security group works the same way as a firewall — contains a set of rules that filter traffic coming into and out of an **EC2** instance. In this screen shot I have opened port-80 which is inbound rule(ingress) and outbound rule(egress) allowing all the traffic.



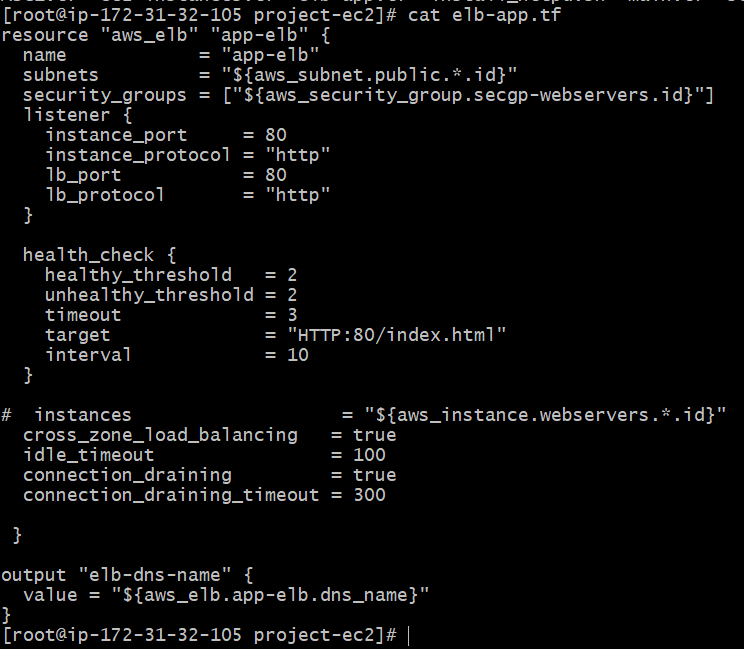
**SHELL SCRIPT:**

Its just a bootstrap script in which we are installing Apache service and it will act as a webserver. display “deployed by terraform” which is incorporated or called by the EC2 instance and installed in it.



**ELASTIC LOAD BALANCE:**

**Elastic Load Balancing** automatically distributes incoming application traffic across multiple targets, such as Amazon **EC2** instances, containers, IP addresses, and Lambda functions. It can handle the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones.



**Auto Scaling Group (ASG):**

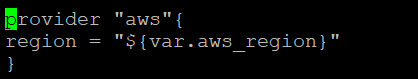
An Amazon EC2 Auto Scaling group (ASG) contains a collection of EC2 instances that share similar characteristics and are treated as a logical grouping for the purposes of fleet management and dynamic scaling. In the below screen shot I have defined min, max sizes its for number of EC2 instance required to run the application successfully. Along with that I have declared health\_check\_grace\_period, health\_check\_type and desired\_capacity there are all the parameter which are required to build the ASG.

TO ASG to build we must have AWS LAUNCH CONFIGURATION which is an instance configuration template that an Auto Scaling group uses to launch EC2 instances. This Include the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups.



**PROVIDER:**

Terraform supports multiple providers such as (Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack, Google cloud platform, GitHub etc.). But as we are working on AWS, so we are importing all the supporting files which will be used to run the terraform successfully or in other words provider is responsible for understanding API interactions and exposing resources.



**VARIABLE FILE:**

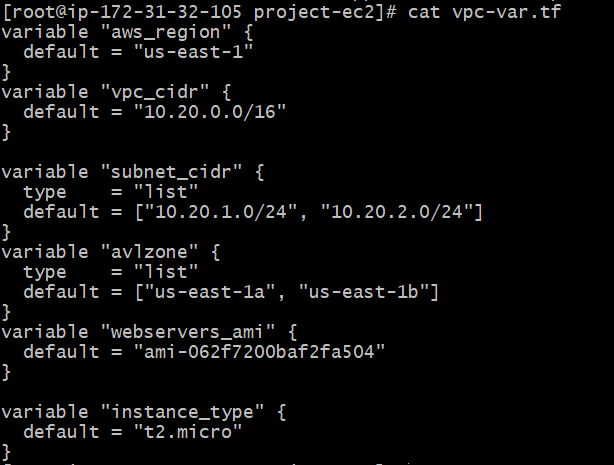
This file consists of variables which are defined and referred to different files.

I have declared primary cidr\_block with a range of IPv4 address.

I have declared secondary cidr\_blocks for subnets.

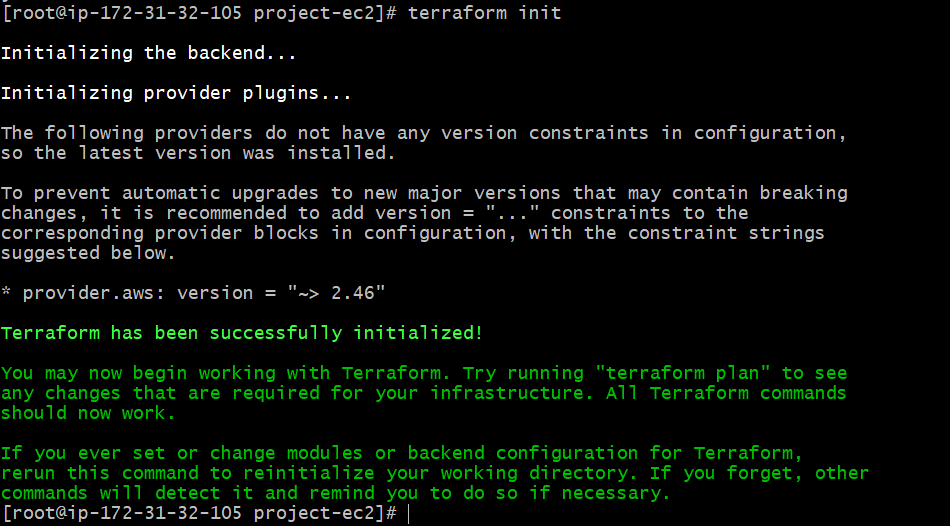
I have declared two different availability Zones but in same region.

I have declared Amazon Machine ID(AMI) and instance type which are used to create EC2 instances.



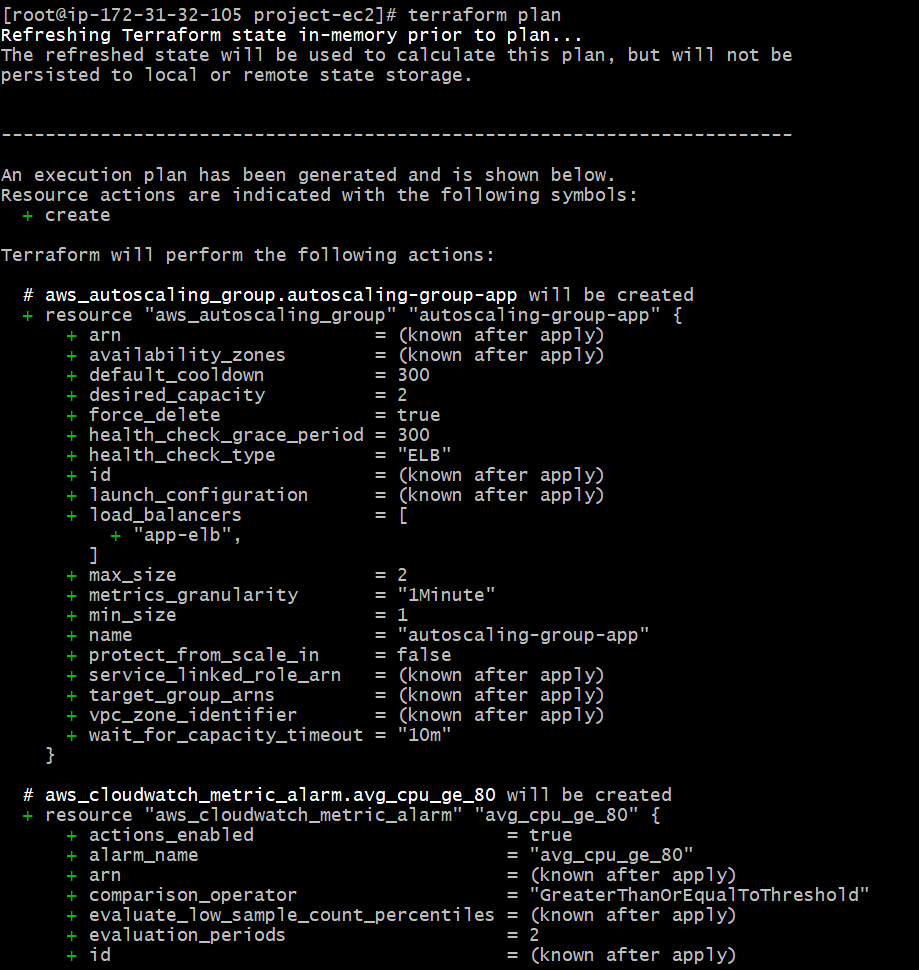
**TERRAFORM INIT:**

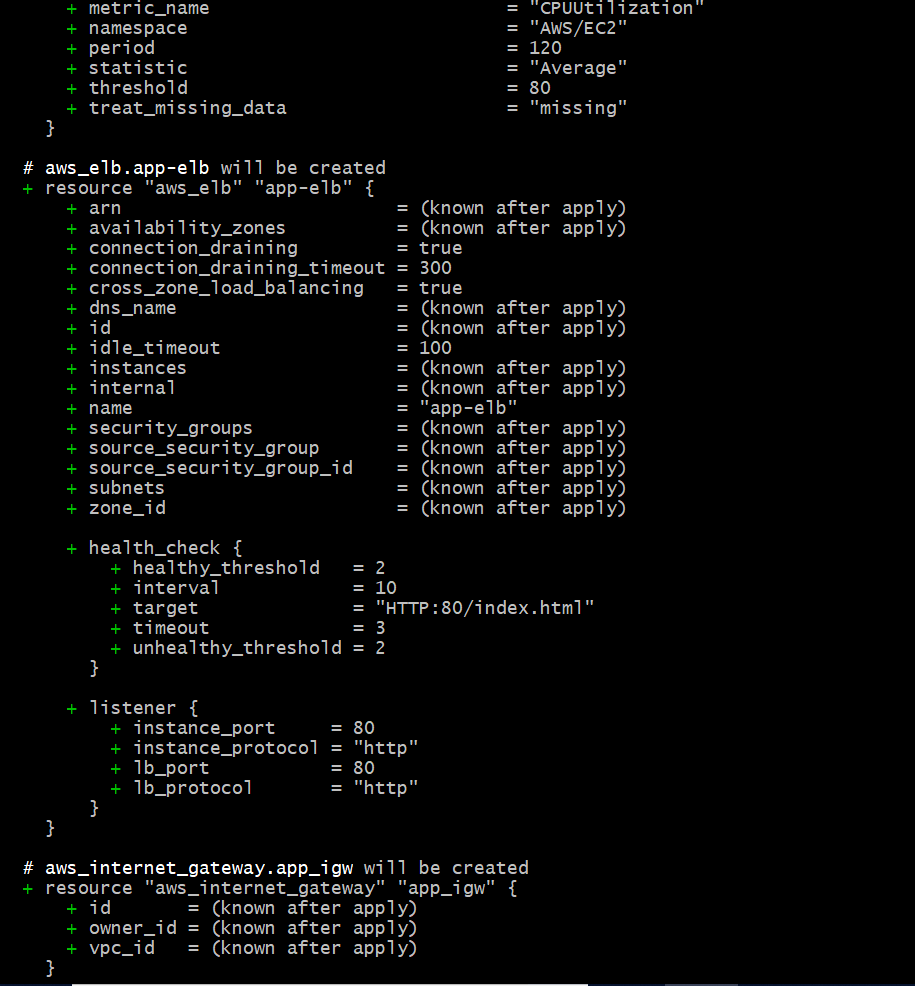
This will initialize all the terraform packages related to AWS and allowing all the terraform related commands.

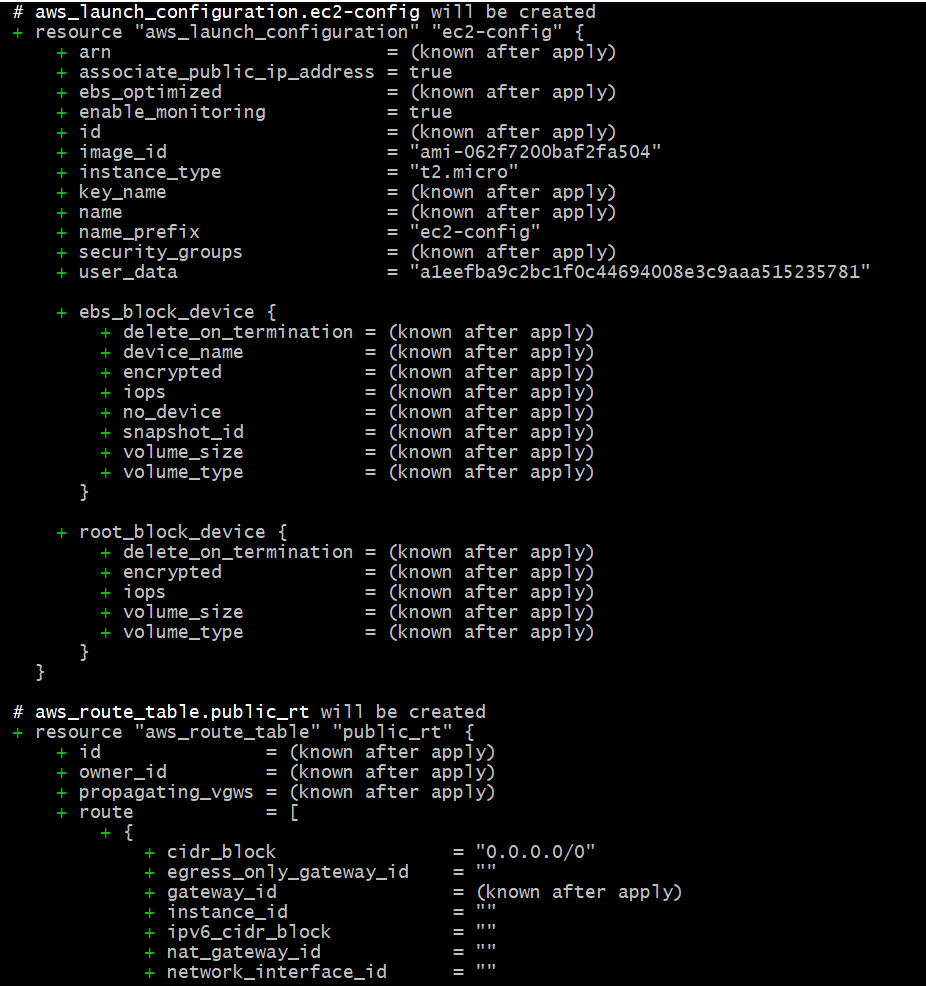


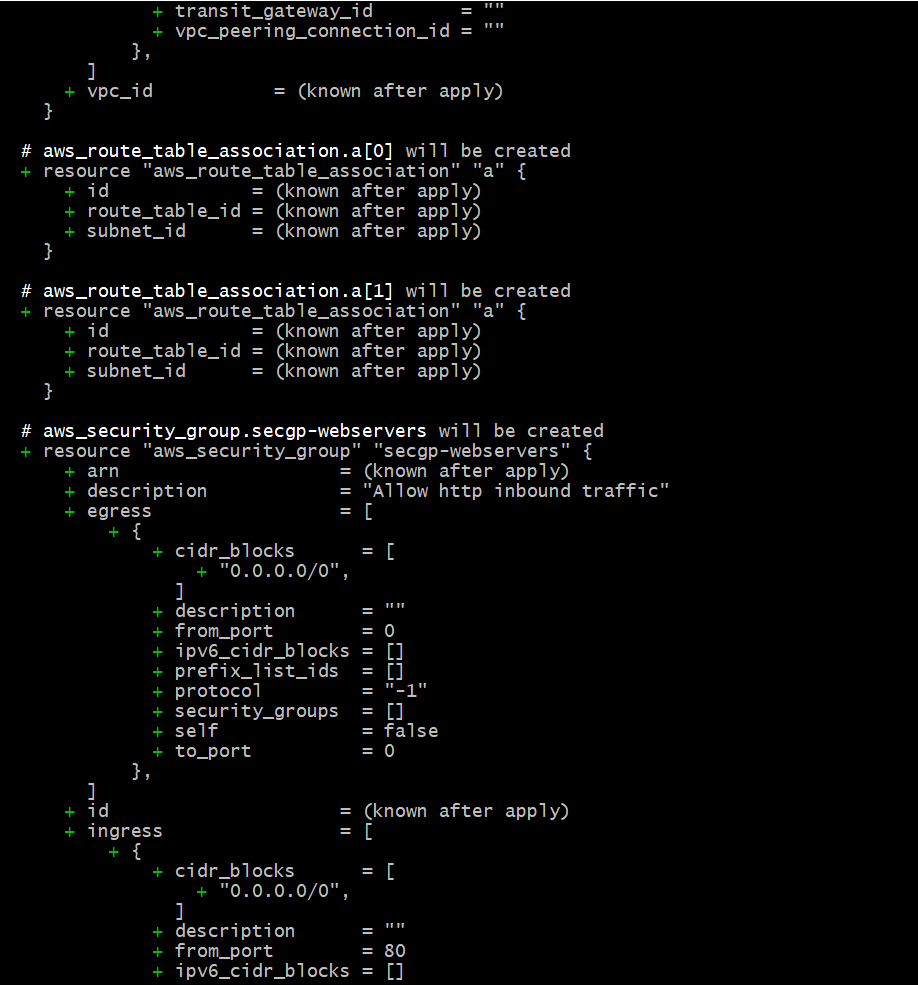
**TERRAFORM PLAN:**

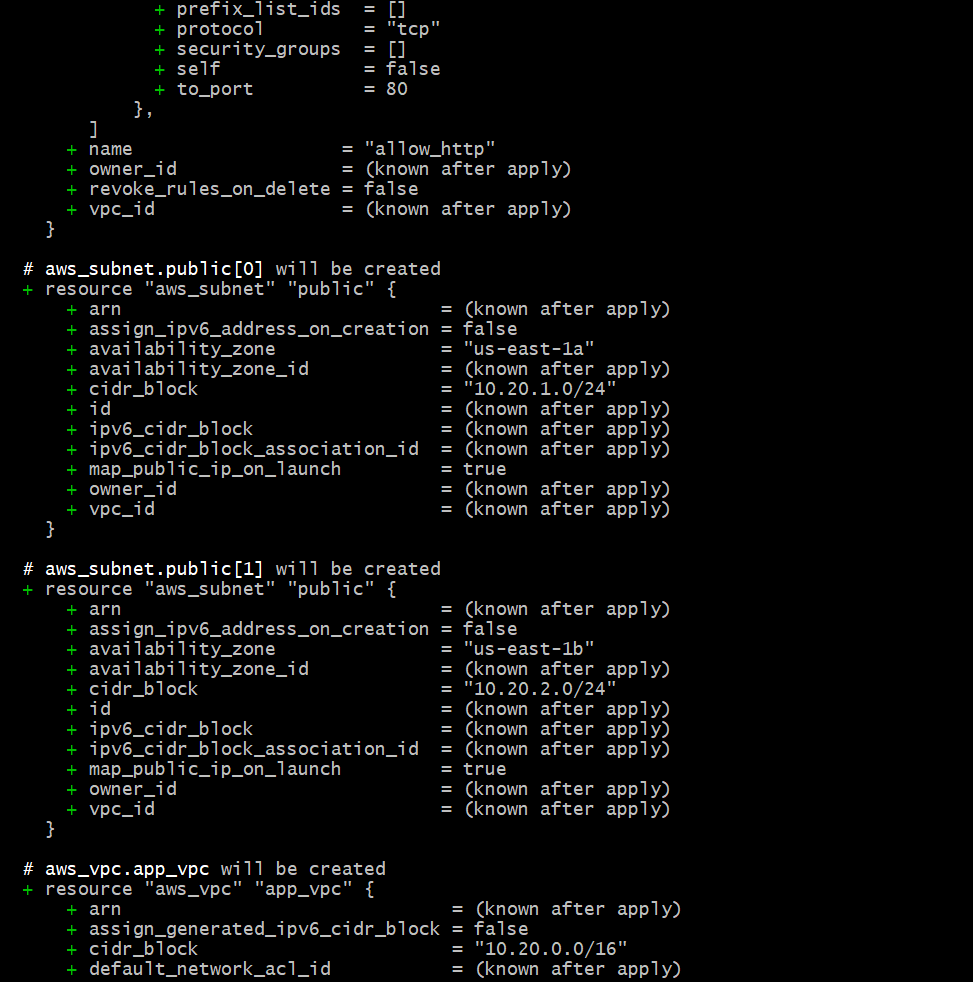
The terraform plan command is used to create an execution plan. Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files. Its an easy way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state.

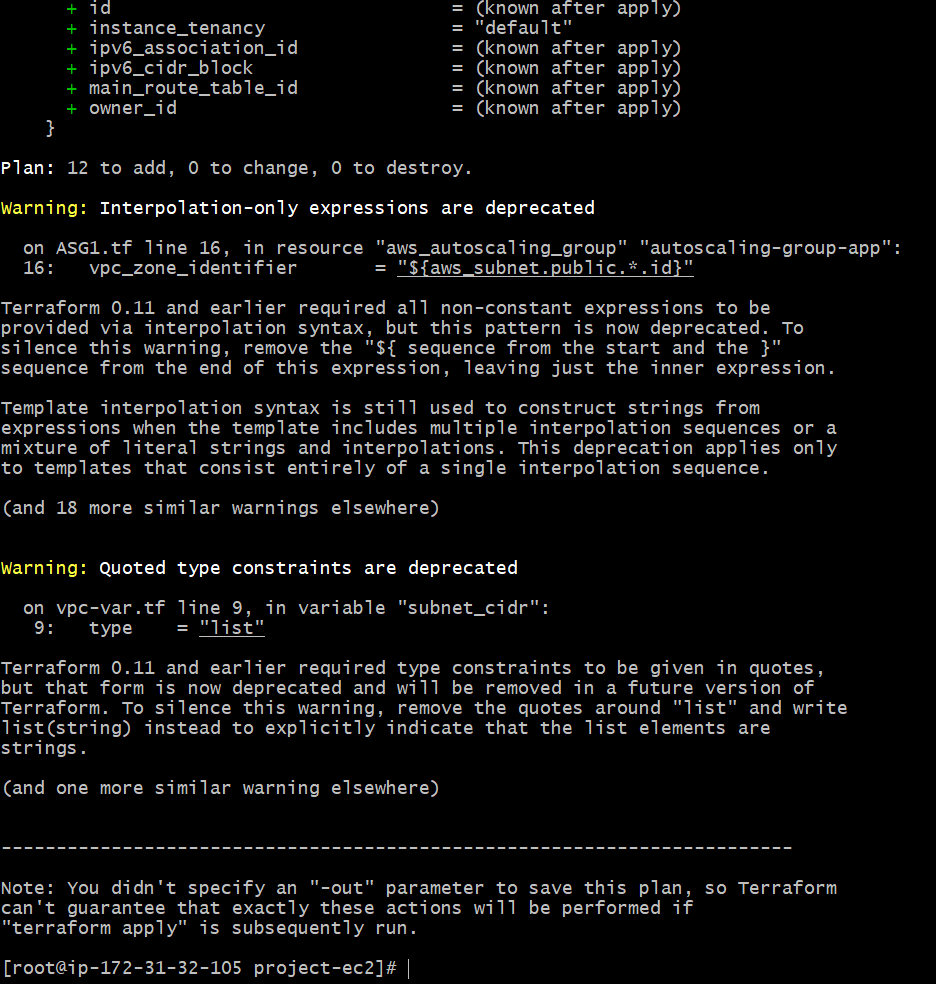






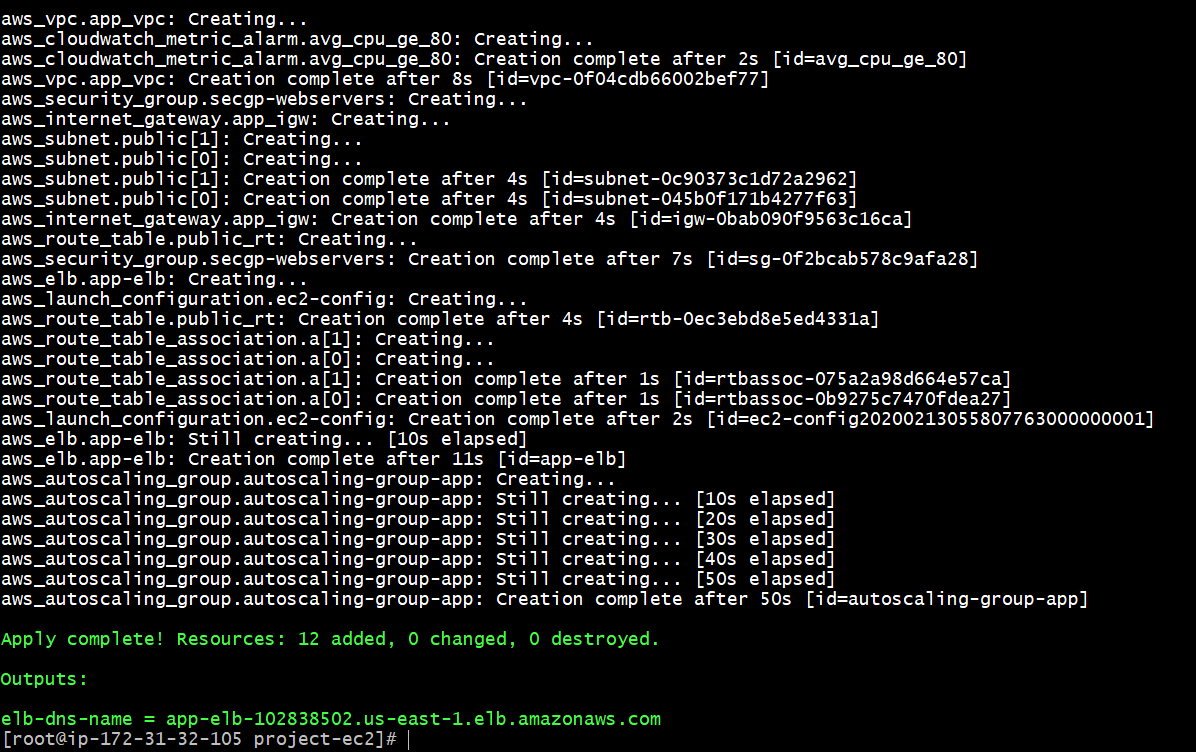


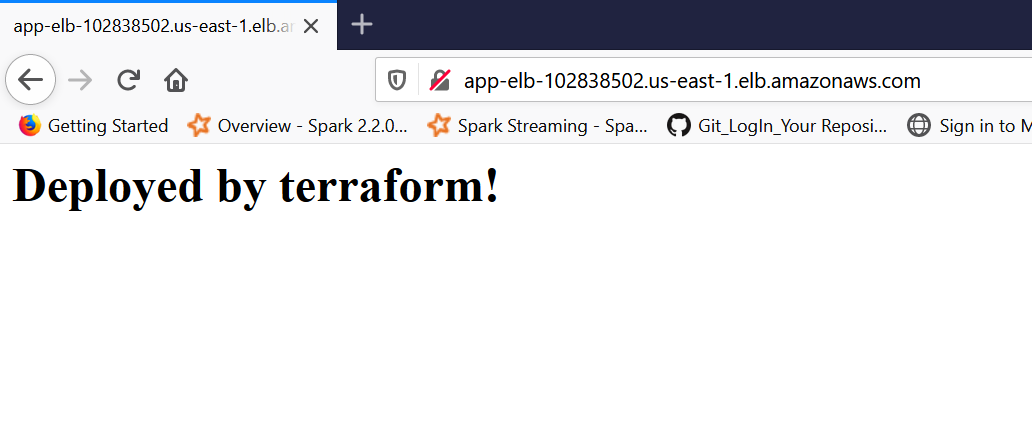




**TERRAFORM APPLY:**

To apply the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform plan execution plan.



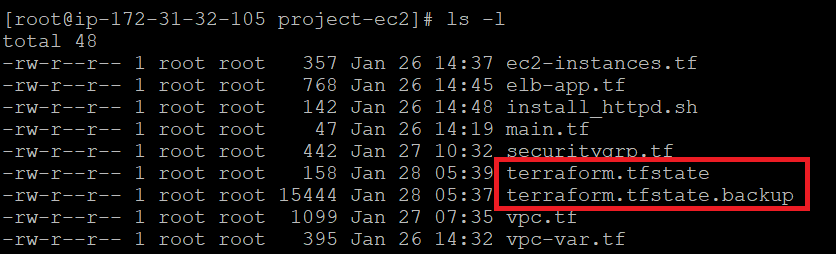


**TERRAFORM TFSTATE AND BACKUP FILES:**

When the infrastructure as a code executed successfully there are two files which are created these files are displayed below. This file will have all the information about the recent terraform build. It will have the info about the resources which are created, modified and destroyed.

If we rerun the code again it will compare terraform plan with the recent build and see if there are any changes and if in care no changes then terraform will not create any duplicate resources.

terraform.tfstate.backup is a copy of terraform.tfstate if in case the state file is lost or corrupted to simplify recovery.

****