**AWS APPLICATION LOAD BALANCER USING TERRAFORM**

In contemporary cloud infrastructure setups, managing and distributing incoming traffic effectively across various instances is central to ensuring the high accessibility and scalability of applications. Among the bunch of services given by AWS (Amazon Web Services), Elastic Load Balancing (ELB) stands apart as a basic part of working with this task. ELB automatically circulates approaching application traffic across a fleet of targets, for example, EC2 instances, containers, IP addresses, or Lambda functions, across different availability zones, ensuring adaptation to non-critical failure and high accessibility.

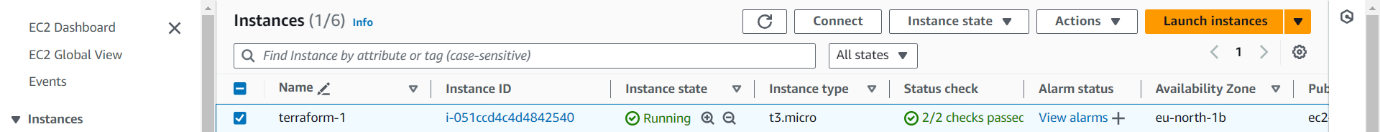
**Primary Terminologies**

* [**AWS Elastic Load Balancer**](https://www.geeksforgeeks.org/elastic-load-balancer-in-aws/)**(ELB):** Flexible Burden Balancer is an overseen administration given by Amazon Web Administrations (AWS) that naturally disseminates approaching application traffic across different targets, for example, EC2 occasions, holders, IP locations, or Lambda capabilities, in various accessibility zones. It guarantees high accessibility, adaptation to internal failure, and adaptability of utilizations by uniformly appropriating the responsibility and rerouting traffic away from undesirable targets.
* **Terraform:** Terraform is an open-source Infrastructure as Code (IaC) tool created by HashiCorp. It allows users to define and provision data center infrastructure utilizing a declarative configuration language called HashiCorp Configuration Language (HCL). Terraform empowers clients to manage and automate the deployment of infrastructure resources across different cloud providers, including AWS, Azure, Google Cloud Platform, and others.
* **Terraform Configuration:**Terraform configuration refers to a set of records containing infrastructure code written in HCL, determining the ideal condition of the infrastructure. These configuration file characterize the resources, their properties, conditions, and connections expected to provision and manage with the infrastructure. Terraform utilizes these configuration files to make an execution plan and apply changes to the infrastructure.
* **Load Balancer Listener:** A [Load balancer](https://www.geeksforgeeks.org/load-balancing-in-cloud-computing/) listener is a part that listens in for connection requests from clients and advances them to the proper objective in light of predefined rules. It works on a particular port and convention characterized by the client and courses incoming traffic to the related objective group or backend service. Listeners play assume a critical part in characterizing how traffic is distributed and managed by the load balancer.
* **Target Groups:** Target Groups are sensible groupings of targets, (for example, EC2 instances, containers, IP addresses, or Lambda functions) enlisted with a load balancer to get approaching traffic. Each target group is related to at least one audience member and characterizes the standards for routing targets to the objectives, for example, health checks, protocols, ports, and conditions. Target groups consider adaptable directing and load adjusload-adjustinging systems in light of the qualities of the targets.

**Step-by-step process to create AWS load balancer using Terraform**

**Step 1: Launch An Instance**

* Launch an EC2 Instance according to your specifications like ami, storage type, and security groups.



* Now connect with the git bash terminal or other terminals like putty, command prompt, PowerShell, etc.

**Step 2: Install Terraform**

* Now install unzip switching to the root user, because aws cli will be in zip file, so to remove zip to unzip we install unzip by using

“sudo apt/yum install unzip -y”

* Now install aws cli from the aws documentation website

“curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install”

* Now install terraform packages from the official site of hashicorp or follow the below commands

**“**wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

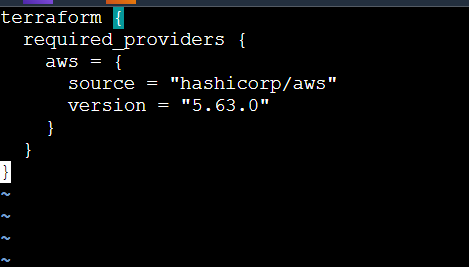
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform**”**

**Step 3: Create A File And Write Terraform Script for to create AWS load balancer**

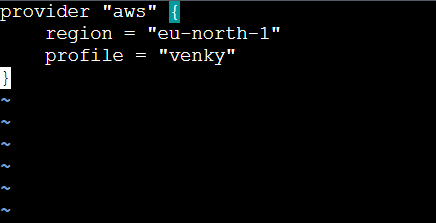
**Terraform block**

In this block, we will provide provider details and its version.



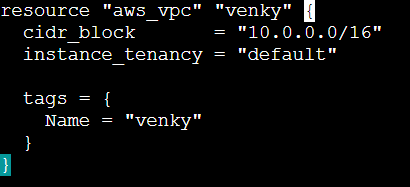
**Provider Block**

In this section, we will provide details of the provider and also we will pass aws “aws\_access\_key\_id” and “secret\_access\_key” in the profile which is already configured.



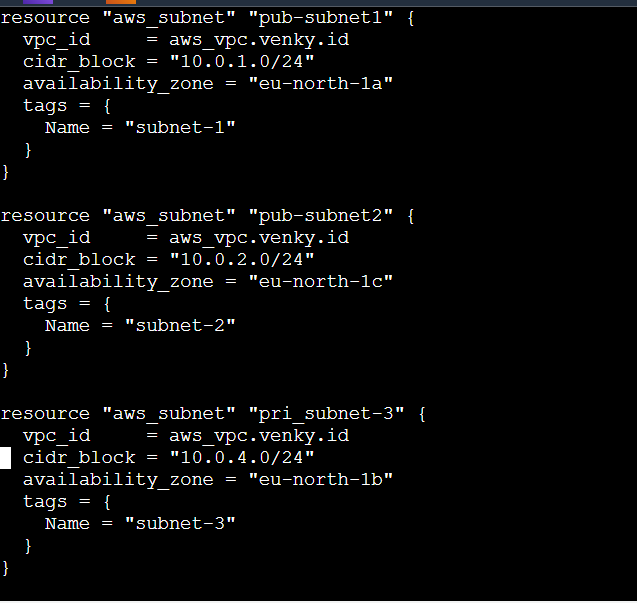
**Creation of VPC**

In this section, we will create vpc for our EC2 instance



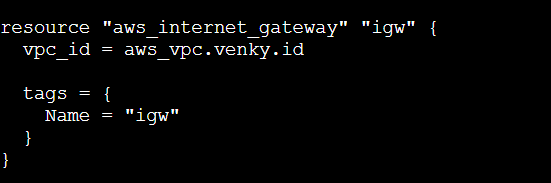
**Creation of subnets**

In this section, we will create two public subnets and one private subnet to VPC



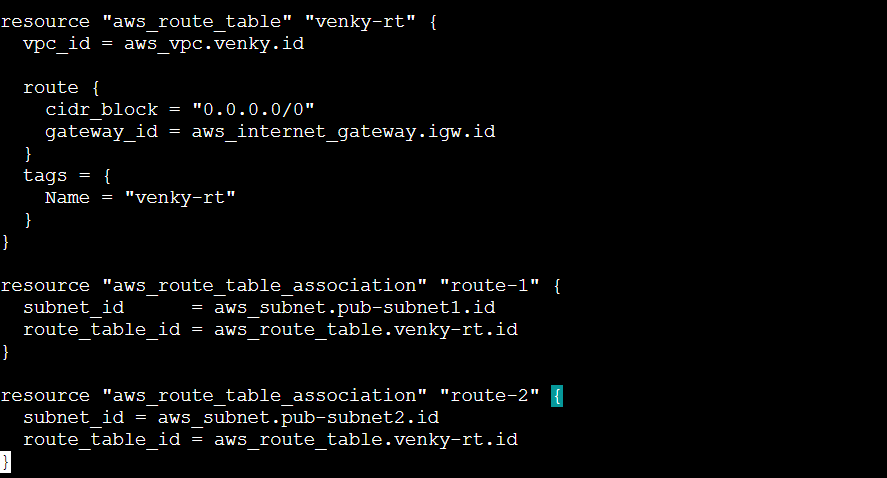
**Internet Gateway Creation**

In this section, we will create an internet gateway to VPC to allow internet traffic



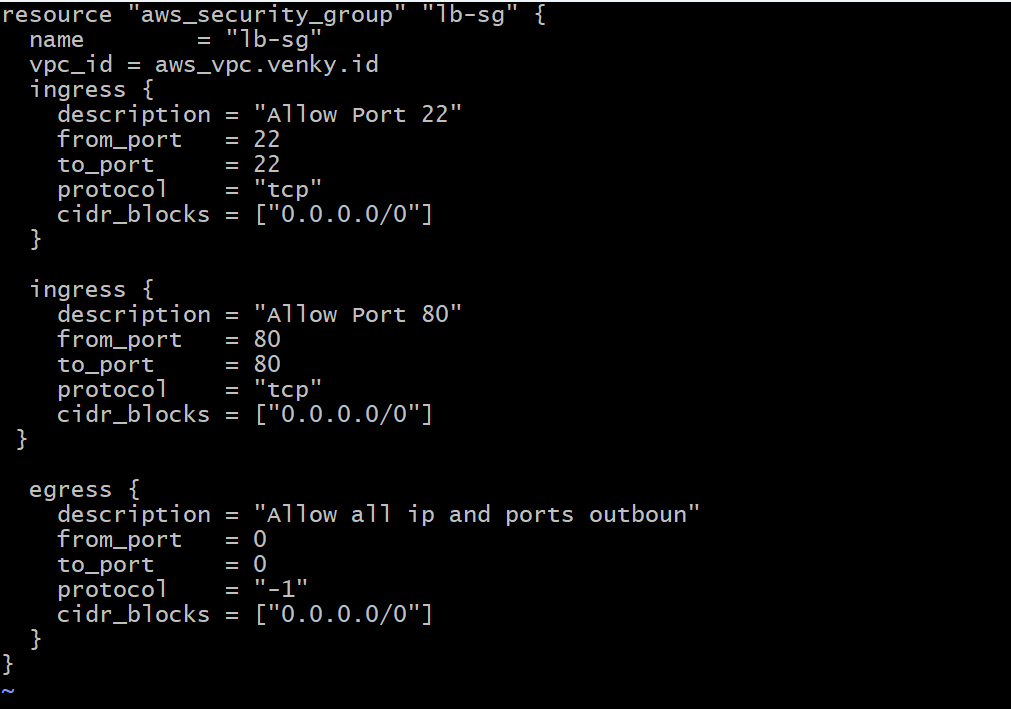
**Creation of Rout Table and Association**

In this section, we will create a route table and we will attach to subnets



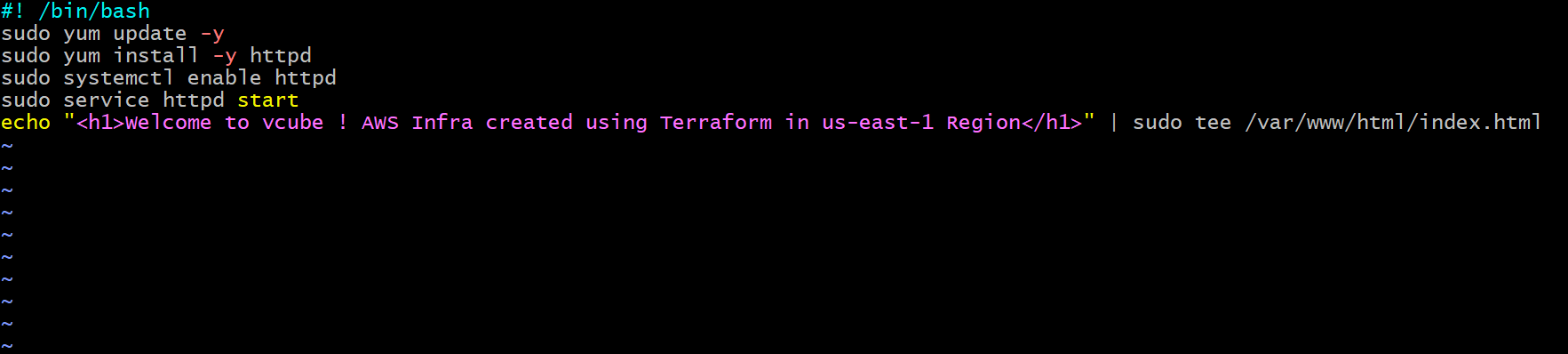
**Creation of Security Group**

In this section, we will create a security group to allow incoming and outgoing traffic through ingress and egress rules.



**User Data for EC2 Instance**

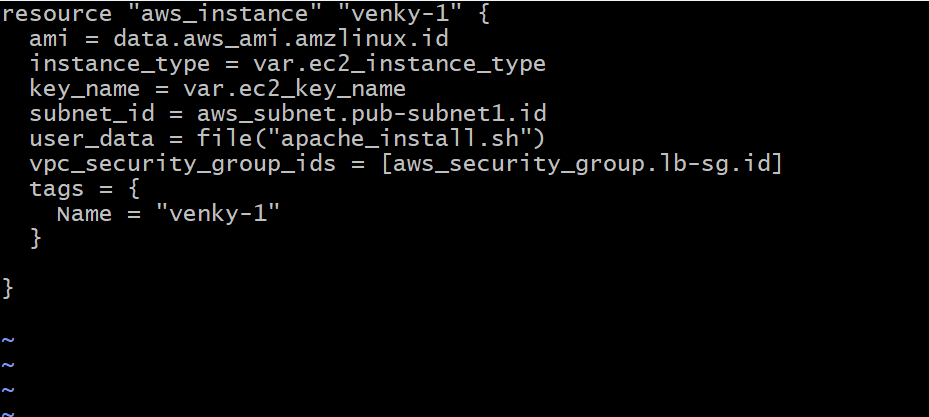
In this section, we will provide user data for the ec2 instance because of the host application.



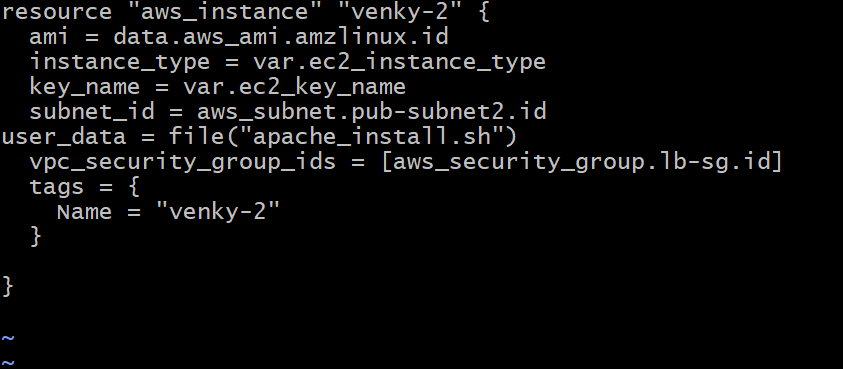
**Creation of EC2 Instance**

In this section, we will create two instances to balance the load between them and maintain high availability in different regions.

* **1st Instance:**

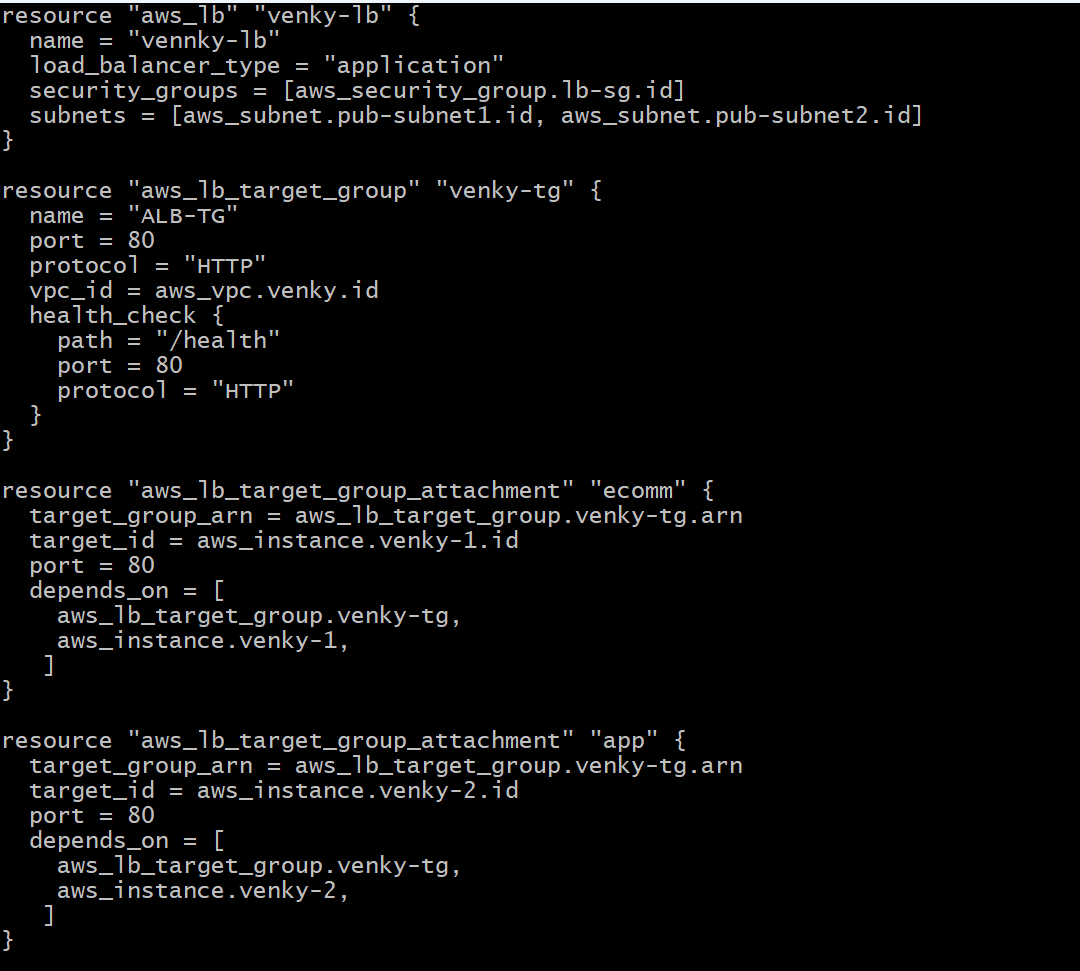


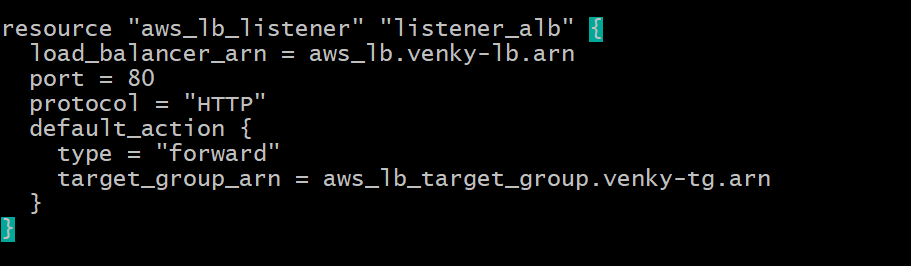
* **2nd instance:**



**Creation of Load Balancer**

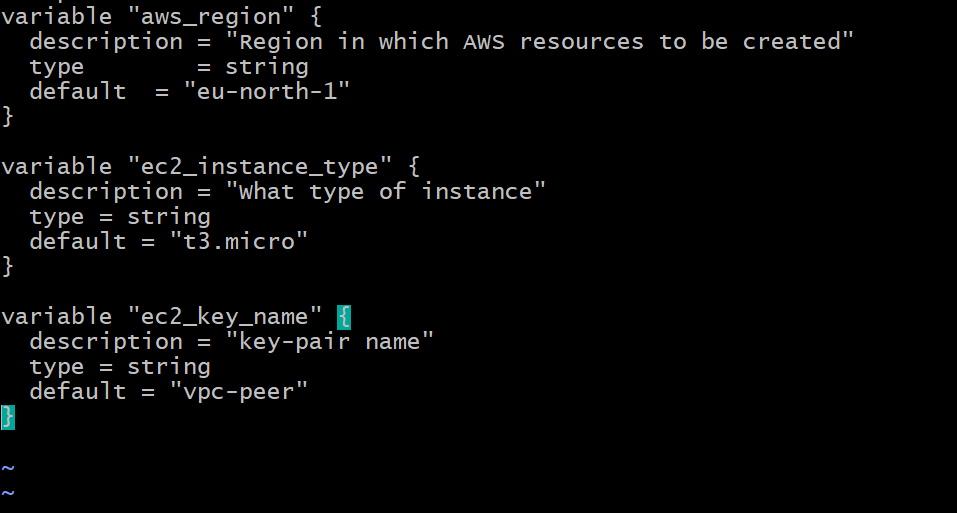
In this section, we will create a load balancer to distribute traffic across multiple servers.





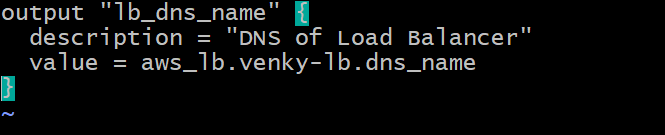
**Creation of variable file**

The variable file in Terraform serves as a centralized location for defining and managing input variables used across multiple Terraform configurations. These variables allow users to parameterize their infrastructure code, making it more flexible, reusable, and maintainable.

****

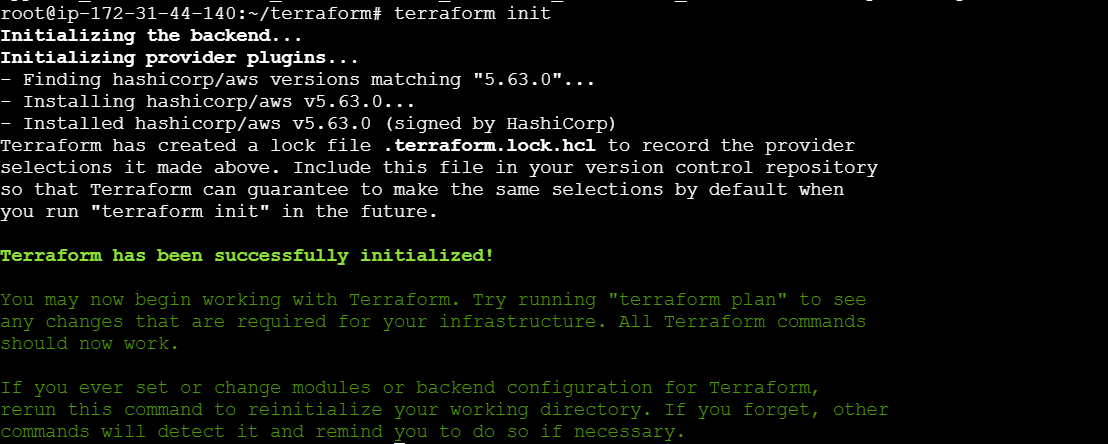
**Creation of Output block**

In this section, we will create output values block, to get the output by using public\_ip, dns\_name.

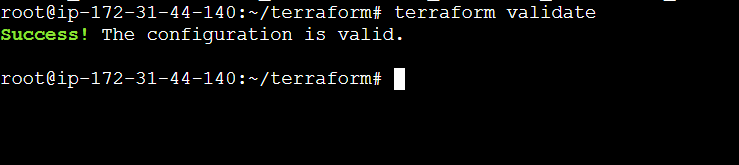


**Step 4: Now Initialize Terraform And Execute Terraform Commands**

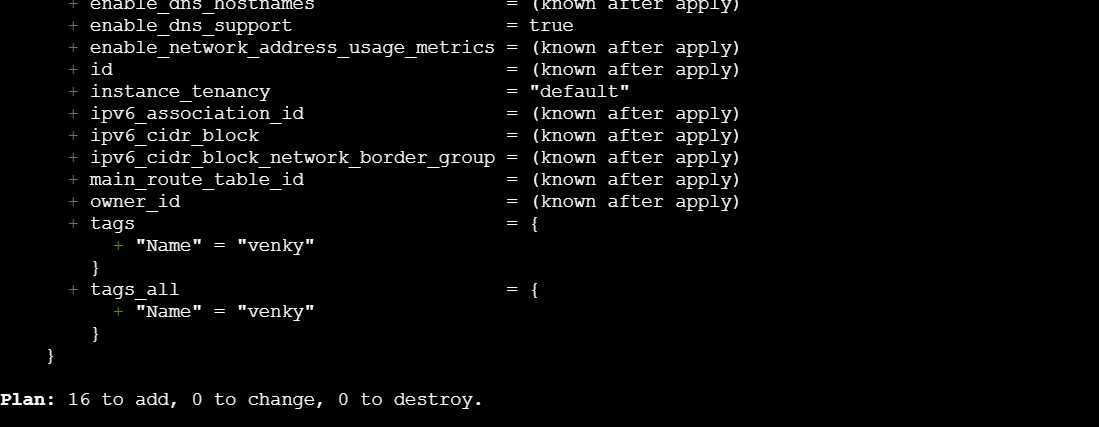
Now initialize terraform by using “terraform init” command



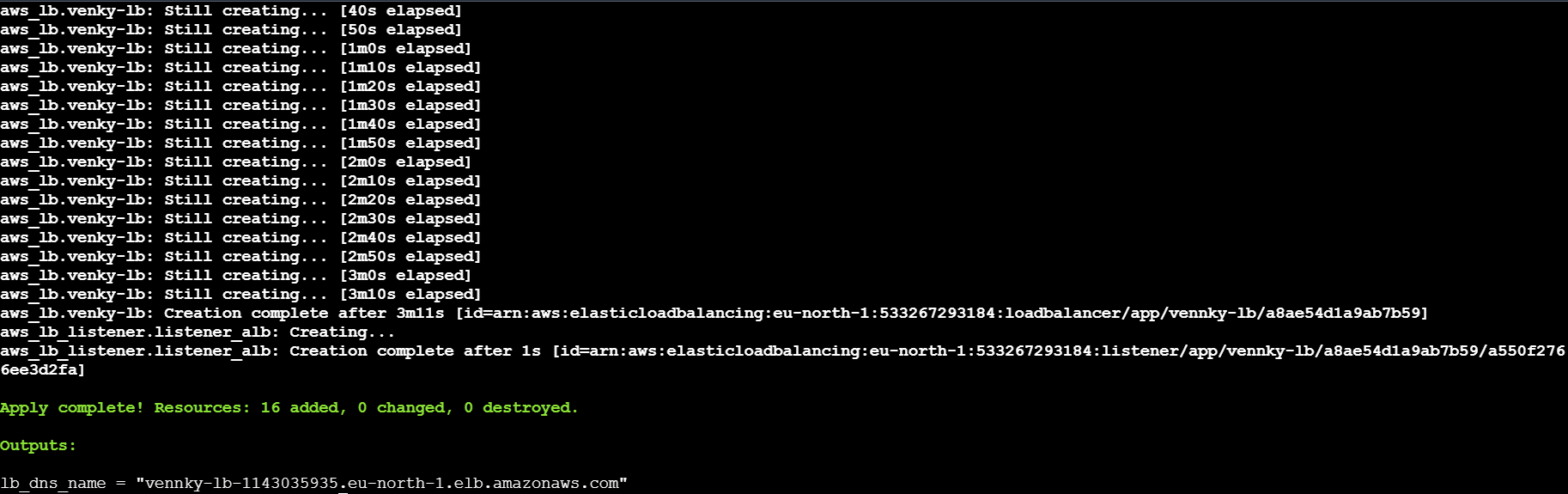
Now validate terraform configuration files by using “terraform validate”



Now plan to terraform script using “terraform plan”



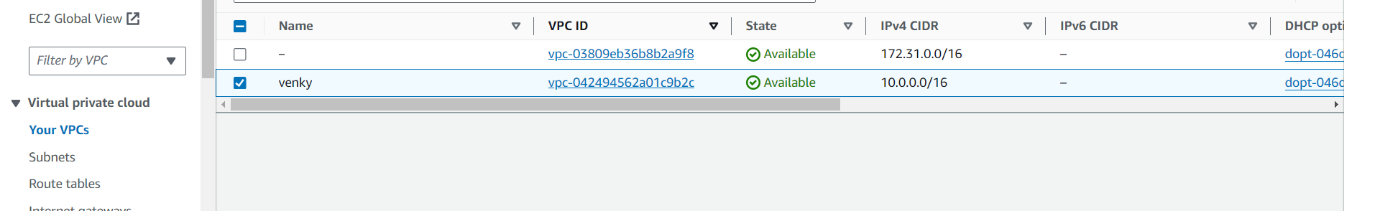
Now apply the terraform script to launch all our resources by using “terraform apply”



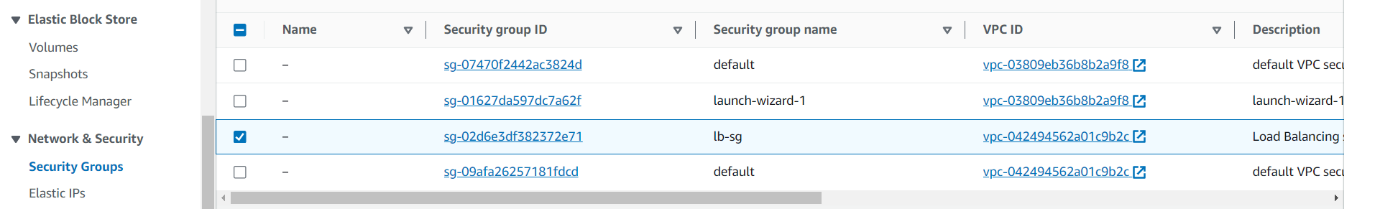
After our load balancer is created we can see that “load balancer dns name”

**The following screenshot shows that we successfully created a load-balancer topic in AWS using Terraform**

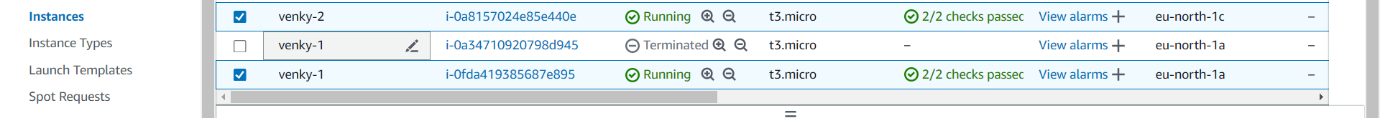
**VPC**

****

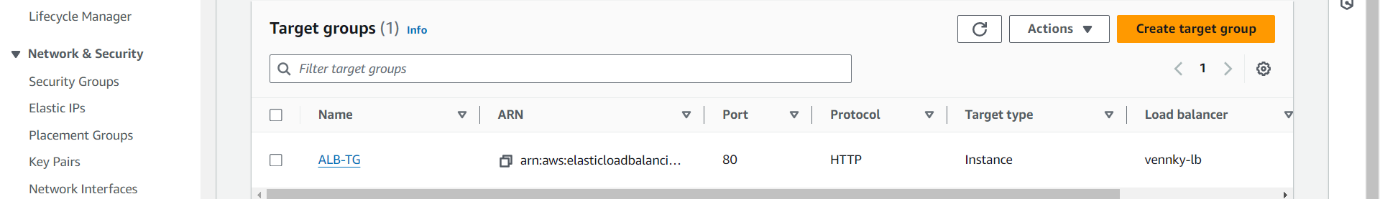
**Security group**

****

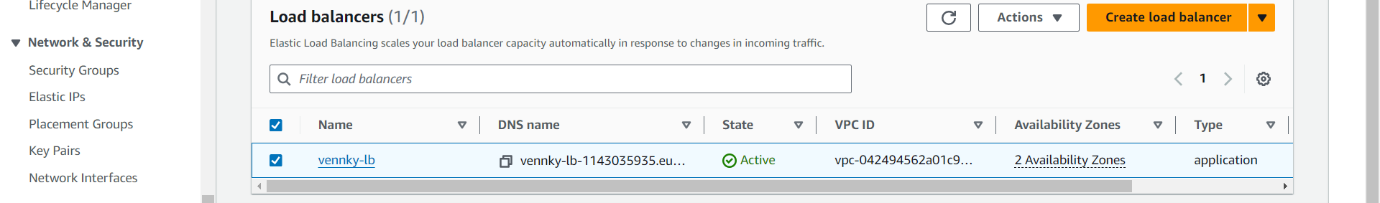
**EC2 Instance**

****

**Target Groups**

****

**Load Balancer**

****

**Conclusion**

By Utilizing Terraform for deploying AWS load balancers provides a streamlined and efficient way to deal with managing infrastructure as code. By arranging load balancer configurations, listeners, target groups, and related resources, groups can automate the provisioning system, ensuring consistency and reliability quality across deployments. Terraforms declarative syntax structure works on infrastructure management, empowering users to define the ideal condition of their AWS environment and apply changes reliably. The integration with AWS services, for example, Elastic Load Balancing (ELB) works with high availability, fault tolerance, and scalability of applications.