**[How to Deploy a Two-Tier Architecture in AWS using Terraform | by Achintha Bandaranaike | Medium](https://achinthabandaranaike.medium.com/how-to-deploy-a-two-tier-architecture-in-aws-using-terraform-a4fd0f2e19ae)**

**What is AWS Two-Tier Architecture?**

1. **Web Server Tier**: This tier is responsible for serving web pages and handling user requests. It typically includes load balancers, web servers, and application servers. Let’s take a closer look at each of these components:

**Load Balancers:** A load balancer distributes incoming traffic to multiple web servers, improving the performance and availability of the application. AWS provides two types of load balancers: Application Load Balancers and Network Load Balancers.

**Web Servers:** Web servers are responsible for serving web pages to the user’s browser. They can handle static content such as HTML, CSS, and JavaScript. Web servers can also execute dynamic content, such as PHP, Python, or Ruby on Rails.

1. **Database Tier**: This tier is responsible for storing and managing data used by the web application. It typically includes a database server and associated storage. Let’s take a closer look at each of these components:

* Database Server: A database server stores and manages data used by the application. AWS provides a variety of database options, including Amazon Relational Database Service (RDS), Amazon DynamoDB, and Amazon Aurora.
* Storage: Storage is used by the database server to store data. AWS provides various storage options, such as Amazon Elastic Block Store (EBS), Amazon Simple Storage Service (S3), and Amazon Elastic File System (EFS).

The two tiers are separated by a firewall or security group to restrict access to the database server from the internet. This architecture is highly scalable, and fault-tolerant, and provides a separation of concerns between the web and database tiers.

**Why do We use Two-Tier Architecture?**

There are several reasons why AWS two-tier architecture is a popular choice for deploying web applications in the cloud:

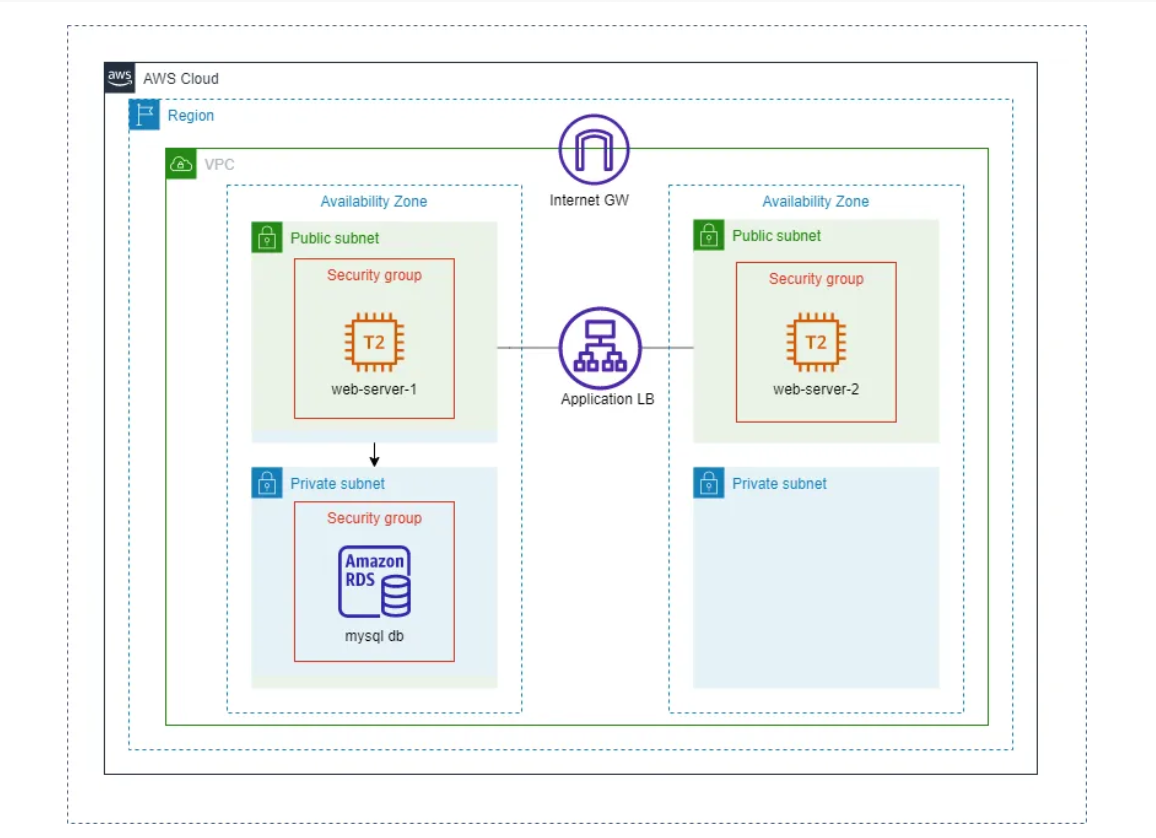
1. **Scalability:** The two-tier architecture allows each tier to scale independently. For example, if the web server tier needs to handle more traffic, additional web server instances can be added, and the load balancer can distribute traffic among them. If the database tier needs to handle more data, additional storage or database instances can be added.
2. **Fault-tolerance:** The two-tier architecture is designed to be fault-tolerant, with multiple web server and database server instances providing redundancy. If one server or instance fails, the load balancer can route traffic to other instances, maintaining application availability.
3. **Security:** The separation of the web server and database tiers allows for better security by restricting access to the database from the internet. By using a firewall or security group to separate the tiers, access to the database can be restricted to only authorized sources.
4. **Performance:** The use of load balancers in the web server tier can improve performance by distributing traffic among multiple web server instances. This can improve the responsiveness of the application and help to reduce latency.

Overall, AWS two-tier architecture provides a flexible and scalable approach to deploying web applications in the cloud, while also providing improved performance, fault tolerance, and security.

**Creating a Two-Tier Architecture in AWS Using Terraform:**

In this article, I will use Terraform to create a two-tier architecture using AWS services. A two-tier architecture consists of a client layer and a database layer. Terraform is an open-source infrastructure as a code tool. With Terraform we can deploy cloud services by declaring the resources in a file.

**Two-Tier Architecture:**



***This two-tier architecture will contain the following components:***

* Deploy a VPC with CIDR 10.0.0.0/16

Within the VPC we will have 2 public subnets with CIDR 10.0.0.0/18 and 10.0.64.0/18. Each public subnet will be in a different Availability Zone for high availability

* Create 2 private subnets with CIDR ‘10.0.128.0/18’ and ‘10.0.192.0/18’ and each will be in a different Availability Zone

Deploy RDS MySQL instance.

* An Application load balancer that will direct traffic to the public subnets.
* Deploy one EC2 instance in each public subnet for high availability.
* Internet Gateway and Elastic IPs for EC2 instance.

1. **Create provider.tf file**

The provider.tf file in Terraform is a configuration file that specifies the cloud provider and its corresponding plugin that Terraform will use to manage resources in that provider.

In Terraform, a provider is a plugin that defines the set of resources and their properties that Terraform can manage in a specific cloud provider, such as AWS, Azure, Google Cloud, or others. The provider plugin is responsible for communicating with the API of the cloud provider, creating, updating, or deleting resources, and handling the authentication and access control.

terraform {  
 required\_providers {  
 aws = {  
 source = "hashicorp/aws"  
 version = "~>4"  
 }  
 }  
}  
  
provider "aws" {  
 region = "ap-southeast-1"  
}

**Create network resources.tf file**

**Create network resources.tf file**

When working with Terraform, one of the most important files you’ll need to create is the network\_resources.tf file. This file typically contains the configuration for the network infrastructure that your application will require to function properly. This can include the creation of a Virtual Private Cloud (VPC), which serves as an isolated network environment for your application, as well as the configuration of subnets, an internet gateway, route tables, and load balancers.

Within the network\_resources.tf file, you may define two public subnets and two private subnets, each with its own unique configurations. The public subnets may be configured to allow inbound traffic from the internet, while the private subnets may be configured to only allow inbound traffic from within the VPC.

Additionally, the network\_resources.tf file may include the configuration of an internet gateway, which serves as a gateway to allow traffic to flow between the VPC and the internet. This file may also include the configuration of route tables and their associations, which are necessary for routing traffic between the subnets within the VPC.

A load balancer may also be included in the network\_resources.tf file. This load balancer can be configured with a listener and a target group, which is used to direct traffic to specific instances within the VPC. The file may also include the attachment of a target group to the load balancer, which specifies which instances to direct traffic to.

Finally, the network\_resources.tf file may include the configuration of an RDS database subnet group. This group specifies which subnets are used for your RDS instances and can be used to ensure that your RDS instances are properly isolated from other resources within the VPC.

Overall, the network\_resources.tf file is an essential part of any Terraform project that requires network infrastructure. Properly configuring this file ensures that your application can communicate with other resources both within and outside of the VPC.

# VPC  
resource "aws\_vpc" "two-tier-vpc" {  
 cidr\_block = "10.0.0.0/16"  
 tags = {  
 Name = "two-tier-vpc"  
 }  
}  
  
# Public Subnets   
resource "aws\_subnet" "two-tier-pub-sub-1" {  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 cidr\_block = "10.0.0.0/18"  
 availability\_zone = "ap-southeast-1a"  
 map\_public\_ip\_on\_launch = "true"  
  
 tags = {  
 Name = "two-tier-pub-sub-1"  
 }  
}  
  
resource "aws\_subnet" "two-tier-pub-sub-2" {  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 cidr\_block = "10.0.64.0/18"  
 availability\_zone = "ap-southeast-1b"  
 map\_public\_ip\_on\_launch = "true"  
 tags = {  
 Name = "two-tier-pub-sub-2"  
 }  
}  
  
# Private Subnets  
resource "aws\_subnet" "two-tier-pvt-sub-1" {  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 cidr\_block = "10.0.128.0/18"  
 availability\_zone = "ap-southeast-1a"  
 map\_public\_ip\_on\_launch = false  
 tags = {  
 Name = "two-tier-pvt-sub-1"  
 }  
}  
resource "aws\_subnet" "two-tier-pvt-sub-2" {  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 cidr\_block = "10.0.192.0/18"  
 availability\_zone = "ap-southeast-1b"  
 map\_public\_ip\_on\_launch = false  
 tags = {  
 Name = "two-tier-pvt-sub-2"  
 }  
}  
  
# Internet Gateway  
resource "aws\_internet\_gateway" "two-tier-igw" {  
 tags = {  
 Name = "two-tier-igw"  
 }  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
}  
  
# Route Table  
resource "aws\_route\_table" "two-tier-rt" {  
 tags = {  
 Name = "two-tier-rt"  
 }  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 route {  
 cidr\_block = "0.0.0.0/0"  
 gateway\_id = aws\_internet\_gateway.two-tier-igw.id  
 }  
}  
  
# Route Table Association  
resource "aws\_route\_table\_association" "two-tier-rt-as-1" {  
 subnet\_id = aws\_subnet.two-tier-pub-sub-1.id  
 route\_table\_id = aws\_route\_table.two-tier-rt.id  
}  
  
resource "aws\_route\_table\_association" "two-tier-rt-as-2" {  
 subnet\_id = aws\_subnet.two-tier-pub-sub-2.id  
 route\_table\_id = aws\_route\_table.two-tier-rt.id  
}  
  
  
# Create Load balancer  
resource "aws\_lb" "two-tier-lb" {  
 name = "two-tier-lb"  
 internal = false  
 load\_balancer\_type = "application"  
 security\_groups = [aws\_security\_group.two-tier-alb-sg.id]  
 subnets = [aws\_subnet.two-tier-pub-sub-1.id, aws\_subnet.two-tier-pub-sub-2.id]  
  
 tags = {  
 Environment = "two-tier-lb"  
 }  
}  
  
resource "aws\_lb\_target\_group" "two-tier-lb-tg" {  
 name = "two-tier-lb-tg"  
 port = 80  
 protocol = "HTTP"  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
}  
  
# Create Load Balancer listener  
resource "aws\_lb\_listener" "two-tier-lb-listner" {  
 load\_balancer\_arn = aws\_lb.two-tier-lb.arn  
 port = "80"  
 protocol = "HTTP"  
 default\_action {  
 type = "forward"  
 target\_group\_arn = aws\_lb\_target\_group.two-tier-lb-tg.arn  
 }  
}  
  
# Create Target group  
resource "aws\_lb\_target\_group" "two-tier-loadb\_target" {  
 name = "target"  
 depends\_on = [aws\_vpc.two-tier-vpc]  
 port = "80"  
 protocol = "HTTP"  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
   
}  
  
resource "aws\_lb\_target\_group\_attachment" "two-tier-tg-attch-1" {  
 target\_group\_arn = aws\_lb\_target\_group.two-tier-loadb\_target.arn  
 target\_id = aws\_instance.two-tier-web-server-1.id  
 port = 80  
}  
resource "aws\_lb\_target\_group\_attachment" "two-tier-tg-attch-2" {  
 target\_group\_arn = aws\_lb\_target\_group.two-tier-loadb\_target.arn  
 target\_id = aws\_instance.two-tier-web-server-2.id  
 port = 80  
}  
  
# Subnet group database  
resource "aws\_db\_subnet\_group" "two-tier-db-sub" {  
 name = "two-tier-db-sub"  
 subnet\_ids = [aws\_subnet.two-tier-pvt-sub-1.id, aws\_subnet.two-tier-pvt-sub-2.id]  
}

**Create security resources.tf file**

In addition to the network\_resources.tf file, another essential file in Terraform is the security\_resources.tf file. This file is responsible for configuring the security measures for your application, such as defining security groups for both EC2 instances and load balancers.

Within the security\_resources.tf file, you may define the configuration of a security group for your EC2 instance. This security group may specify which ports are open and which protocols are allowed for inbound and outbound traffic. For example, the EC2 security group may be configured to allow inbound traffic on ports 80 and 22, which correspond to HTTP and SSH traffic, respectively. This ensures that your EC2 instances can be accessed from the internet for web traffic and from your local machine for SSH access.

The security\_resources.tf file may also include the configuration of a security group for your load balancer. This security group specifies the rules that allow traffic to flow from the load balancer to the EC2 instances within the VPC. These rules may include which ports are open and which protocols are allowed for inbound and outbound traffic.

Properly configuring the security\_resources.tf file is crucial for ensuring that your application is secure and protected from unauthorized access. By defining security groups for your EC2 instances and load balancers, you can control the flow of traffic both within and outside of the VPC. This helps to minimize the risk of security breaches and ensures that your application is compliant with any relevant regulations or standards.

# Security Group  
resource "aws\_security\_group" "two-tier-ec2-sg" {  
 name = "two-tier-ec2-sg"  
 description = "Allow traffic from VPC"  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 depends\_on = [  
 aws\_vpc.two-tier-vpc  
 ]  
  
 ingress {  
 from\_port = "0"  
 to\_port = "0"  
 protocol = "-1"  
 }  
 ingress {  
 from\_port = "80"  
 to\_port = "80"  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 ingress {  
 from\_port = "22"  
 to\_port = "22"  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 egress {  
 from\_port = "0"  
 to\_port = "0"  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 tags = {  
 Name = "two-tier-ec2-sg"  
 }  
}  
  
# Load balancer security group  
resource "aws\_security\_group" "two-tier-alb-sg" {  
 name = "two-tier-alb-sg"  
 description = "load balancer security group"  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
 depends\_on = [  
 aws\_vpc.two-tier-vpc  
 ]  
  
  
 ingress {  
 from\_port = "0"  
 to\_port = "0"  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 egress {  
 from\_port = "0"  
 to\_port = "0"  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
  
 tags = {  
 Name = "two-tier-alb-sg"  
 }  
}  
  
# Database tier Security gruop  
resource "aws\_security\_group" "two-tier-db-sg" {  
 name = "two-tier-db-sg"  
 description = "allow traffic from internet"  
 vpc\_id = aws\_vpc.two-tier-vpc.id  
  
 ingress {  
 from\_port = 3306  
 to\_port = 3306  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.two-tier-ec2-sg.id]  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 ingress {  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.two-tier-ec2-sg.id]  
 cidr\_blocks = ["10.0.0.0/16"]  
 }  
  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
}

**Create ec2-resources.tf file**

The ec2-resources.tf file is a Terraform configuration file that is responsible for defining the resources required to provision EC2 instances in your AWS environment. This file type includes the configuration of two EC2 instances, each with an NGINX web server installed, as well as the allocation of two Elastic IPs for those instances.

The EC2 instances defined in the ec2-resources.tf file may be configured with specific instance types, depending on the requirements of your application. Additionally, the file may include the definition of the AMI used to launch the instances, which specifies the operating system and any additional software or configurations required to run your application.

Once the EC2 instances are defined, the ec2-resources.tf file may allocate two Elastic IPs, which are used to provide static public IP addresses for your EC2 instances. This is useful for scenarios where you need a fixed IP address for your instances, such as for setting up DNS records or for firewall configurations.

Finally, the ec2-resources.tf file may configure the EC2 instances with the NGINX web server, which allows the instances to serve web pages to users. This configuration may include the installation of the NGINX software, as well as any required configurations for serving web pages.

Overall, the ec2-resources.tf file is an essential part of any Terraform project that requires EC2 instances. Properly configuring this file ensures that your instances are provisioned with the correct configurations, software, and network settings and that they can be accessed from the internet with a static IP address.

# Public subnet EC2 instance 1  
resource "aws\_instance" "two-tier-web-server-1" {  
 ami = "ami-064eb0bee0c5402c5"  
 instance\_type = "t2.micro"  
 security\_groups = [aws\_security\_group.two-tier-ec2-sg.id]  
 subnet\_id = aws\_subnet.two-tier-pub-sub-1.id  
 key\_name = "two-tier-key"  
  
 tags = {  
 Name = "two-tier-web-server-1"  
 }  
  
 user\_data = <<-EOF  
#!/bin/bash  
sudo yum update -y  
sudo amazon-linux-extras install nginx1 -y   
sudo systemctl enable nginx  
sudo systemctl start nginx  
EOF  
}  
  
# Public subnet EC2 instance 2  
resource "aws\_instance" "two-tier-web-server-2" {  
 ami = "ami-064eb0bee0c5402c5"  
 instance\_type = "t2.micro"  
 security\_groups = [aws\_security\_group.two-tier-ec2-sg.id]  
 subnet\_id = aws\_subnet.two-tier-pub-sub-2.id  
 key\_name = "two-tier-key"  
  
 tags = {  
 Name = "two-tier-web-server-2"  
 }  
  
 user\_data = <<-EOF  
#!/bin/bash  
sudo yum update -y  
sudo amazon-linux-extras install nginx1 -y   
sudo systemctl enable nginx  
sudo systemctl start nginx  
EOF  
}  
  
#EIP  
  
resource "aws\_eip" "two-tier-web-server-1-eip" {  
 vpc = true  
  
 instance = aws\_instance.two-tier-web-server-1.id  
 depends\_on = [aws\_internet\_gateway.two-tier-igw]  
}  
  
resource "aws\_eip" "two-tier-web-server-2-eip" {  
 vpc = true  
  
 instance = aws\_instance.two-tier-web-server-2.id  
 depends\_on = [aws\_internet\_gateway.two-tier-igw]  
}

**Create db-resources.tf file**

The db-resources.tf file in Terraform is responsible for configuring the resources required to provision an RDS database in your AWS environment. This file typically includes the configuration of a MySQL RDS database, including the definition of the database instance class, allocated storage, and the desired backup window.

The RDS database defined in the db-resources.tf file may be configured with specific database instance classes and storage types, depending on the requirements of your application. Additionally, the file may specify the desired backup window for the RDS instance, which determines when automated backups are taken and how long they are retained.

Properly configuring the db-resources.tf file is crucial for ensuring that your RDS database is configured with the correct settings and backup policies. By defining the database instance class, allocated storage, and backup window, you can ensure that your RDS instance is optimized for performance, durability, and availability.

Overall, the db-resources.tf file is an essential part of any Terraform project that requires an RDS database. Properly configuring this file ensures that your RDS database is provisioned with the correct configurations, storage, backup settings, and any other desired policies. This helps to ensure the reliability and scalability of your application and minimize the risk of data loss.

# RDS MYSQL database  
resource "aws\_db\_instance" "two-tier-db-1" {  
 allocated\_storage = 5  
 storage\_type = "gp2"  
 engine = "mysql"  
 engine\_version = "5.7"  
 instance\_class = "db.t2.micro"  
 db\_subnet\_group\_name = "two-tier-db-sub"  
 vpc\_security\_group\_ids = [aws\_security\_group.two-tier-db-sg.id]  
 parameter\_group\_name = "default.mysql5.7"  
 db\_name = "two\_tier\_db1"  
 username = "admin"  
 password = "password"  
 allow\_major\_version\_upgrade = true  
 auto\_minor\_version\_upgrade = true  
 backup\_retention\_period = 35  
 backup\_window = "22:00-23:00"  
 maintenance\_window = "Sat:00:00-Sat:03:00"  
 multi\_az = false  
 skip\_final\_snapshot = true  
}

Summary view of the terraform files in VS code:

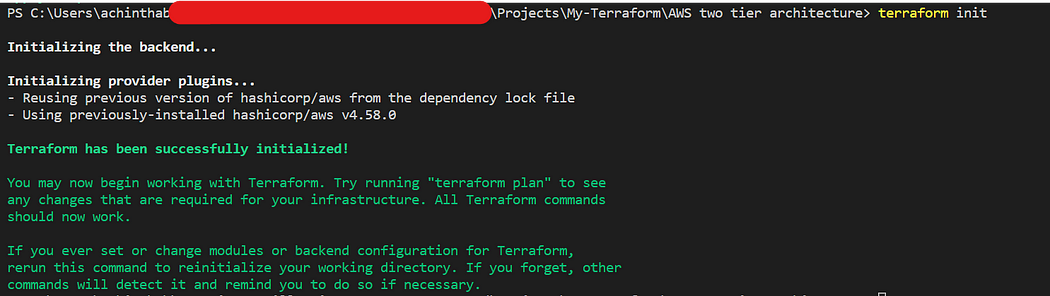
***Let's deploy!***

Make sure you have already inserted your AWS credentials and are operating from the root directory before starting these Terraform commands.

1. **terraform init**

The terraform init the command is used to initialize a new or existing Terraform configuration. This command downloads the required provider plugins and sets up the backend for storing state.

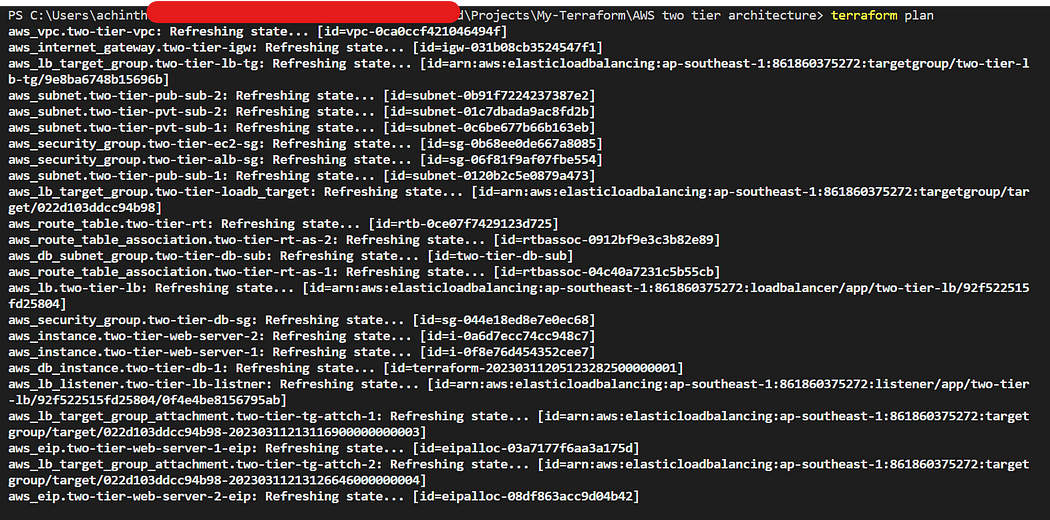
terraform init



2. **terraform plan**

The terraform plan the command is used to create an execution plan for the Terraform configuration. This command shows what resources Terraform will create, modify, or delete when applied.

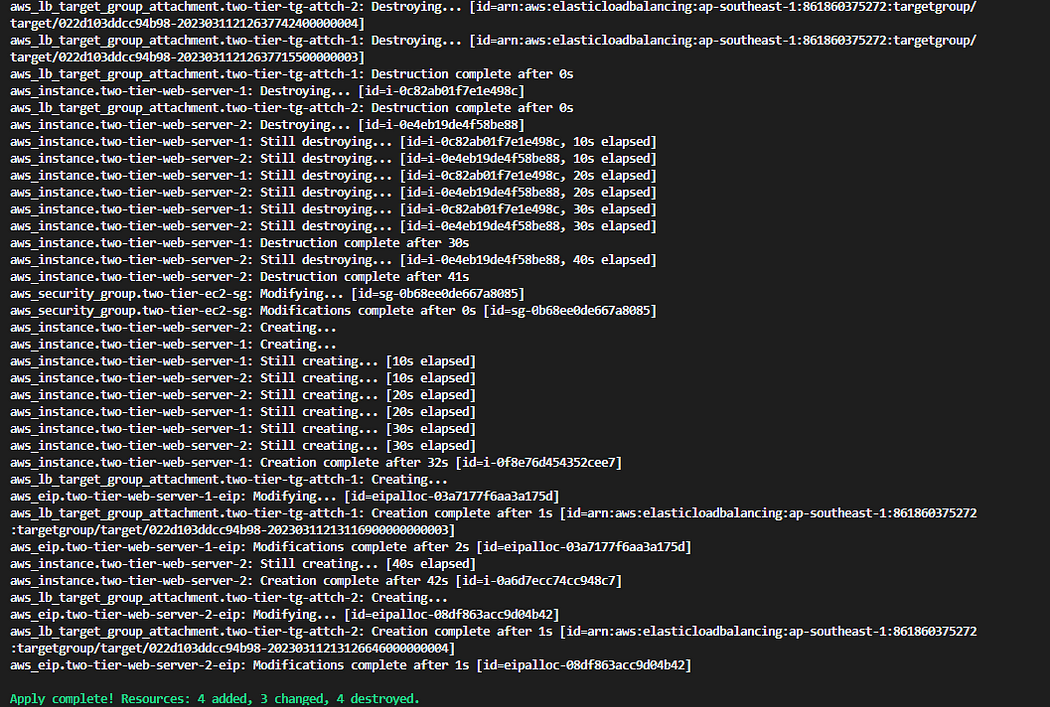
terraform plan



3. **terraform apply**

The terraform apply the command is used to apply the Terraform configuration and create or modify resources in the target environment.

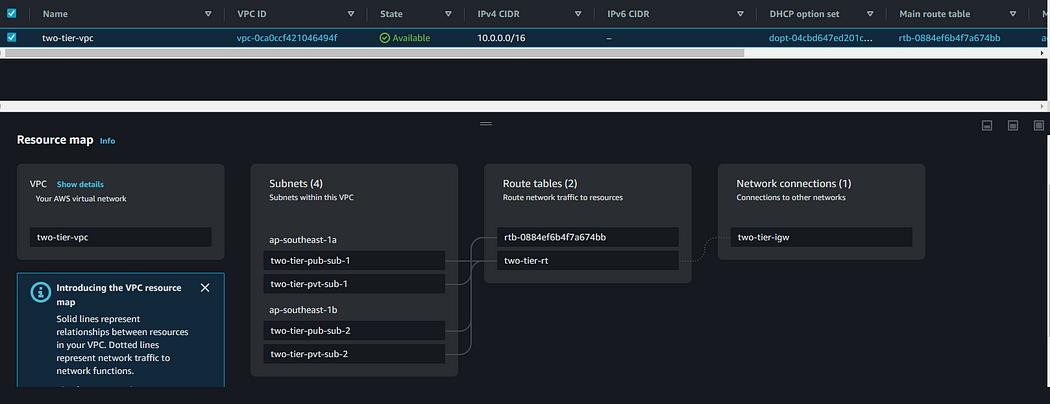
terraform apply



all resources are created

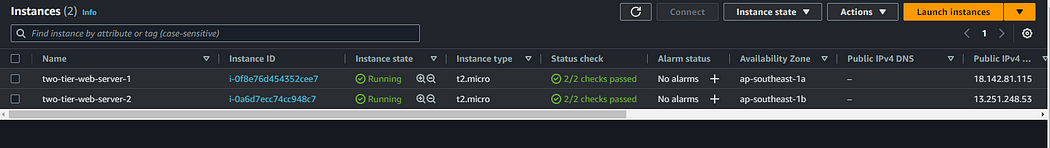
**Go to the AWS console and verify**

1. VPC and Network resources



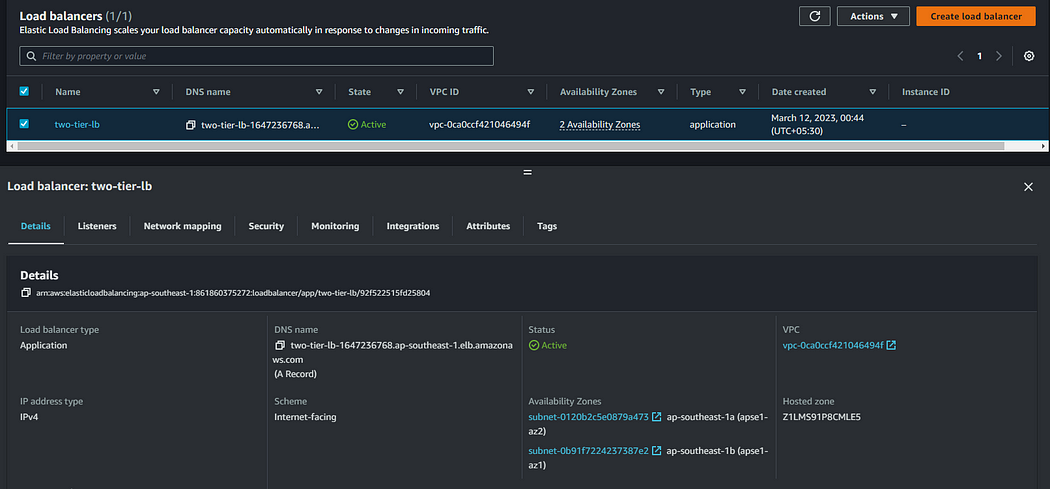
vpc, public and private subnets,route table, and internet gateway

2. EC2 instances



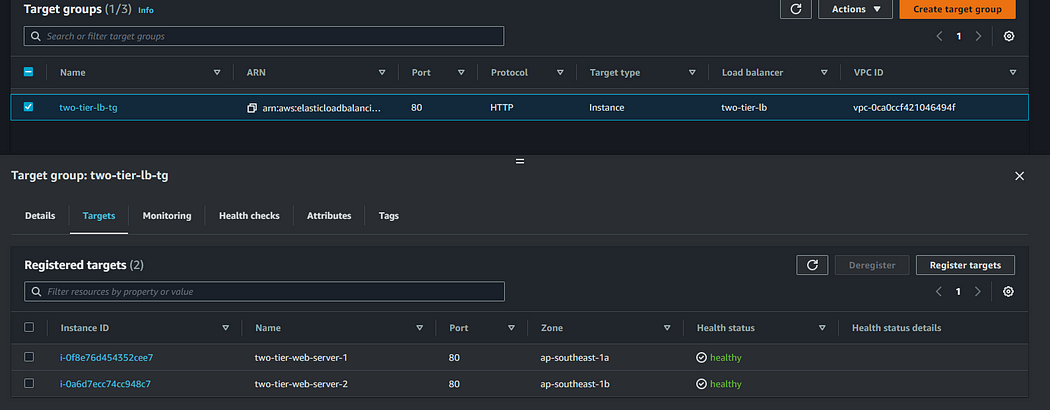
web server 1 and web server 2 instances

3. Load Balancer



application load balancer

4. Target Group



Target Group and targets are healthy

5. RDS MYSQL Database

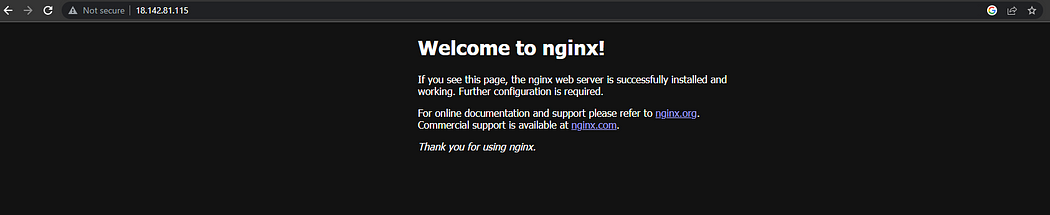
A screenshot of a computer

AI-generated content may be incorrect.

RDS MYSQL Database

6. EC2 NGINX web server is also working fine

*web server-1*



web server 1 nginx web page

*web server-2*

A screenshot of a computer

AI-generated content may be incorrect.

web server 2 nginx web page

7. Security Groups