Article Source:  
<https://sharmilas.medium.com/a-step-by-step-guide-to-creating-load-balancer-and-ec2-with-auto-scaling-group-using-terraform-752afd44df8e>  
Github repo:  
<https://github.com/SharmilaS22/medium-tf-ec2-alb-asg?source=post_page-----752afd44df8e-------------------------------->

A**mazon Elastic Load Balancer (ELB)**distributes incoming network traffic across multiple servers, preventing any single resource from being overwhelmed. By evenly distributing the workload, load balancing improves performance and response times.

A**mazon Auto Scaling Group (ASG)**adjusts the number of EC2 instances based on the current demand. It provides elasticity, allowing the application to adapt to changes in traffic patterns. During periods of high demand, ASG provisions additional resources to handle the workload. During low demand, it removes excess resources and thereby reduces costs. It maintains high availability by automatically replacing failed servers with new instances.

T**erraform** is an infrastructure as code (Iac) tool that automates infrastructure provisioning. It is the most popular and offers support for multiple clouds.

## In this article, we will

1. Create a VPC, Subnets, and Configure network requirements.
2. Create EC2 instances with Auto scaling group and Launch template.
3. Create a Load balancer with a target group to the Auto Scaling group.

# Setting up the Environment

## 1. Install AWS CLI and configure

If you don’t have AWS CLI already, install AWS CLI to your system based on your OS by following the instructions [here](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions).

## [Installing or updating the latest version of the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html?source=post_page-----752afd44df8e--------------------------------" \l "getting-started-install-instructions" \t "_blank)

### **[Install the AWS CLI on your system.](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html?source=post_page-----752afd44df8e--------------------------------" \l "getting-started-install-instructions" \t "_blank)**

[docs.aws.amazon.com](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html?source=post_page-----752afd44df8e--------------------------------" \l "getting-started-install-instructions" \t "_blank)

The next step is to configure your AWS profile to use in the CLI. Follow the instructions from the [official documentation](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html) based on your machine type.

## [Configuring the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html?source=post_page-----752afd44df8e--------------------------------" \t "_blank)

### **[Configure the AWS Command Line Interface (AWS CLI) and specify the settings for interacting with AWS.](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html?source=post_page-----752afd44df8e--------------------------------" \t "_blank)**

[docs.aws.amazon.com](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html?source=post_page-----752afd44df8e--------------------------------" \t "_blank)

## 2. Install Terraform

Install Terraform in your system, if you don't have it already. Follow the instructions based on your platform on the official site given below.

## [Install Terraform | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli?source=post_page-----752afd44df8e--------------------------------" \l "install-terraform" \t "_blank)

### **[Install Terraform on Mac, Linux, or Windows by downloading the binary or using a package manager (Homebrew or…](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli?source=post_page-----752afd44df8e--------------------------------" \l "install-terraform" \t "_blank)**

[developer.hashicorp.com](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli?source=post_page-----752afd44df8e--------------------------------" \l "install-terraform" \t "_blank)

## 3. Initialise Project

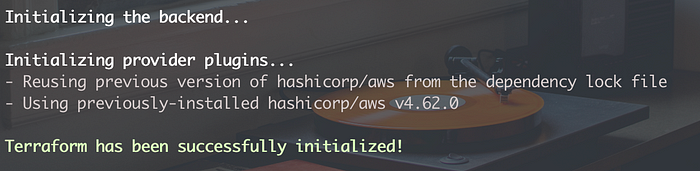
Create an empty project directory and create a file called main.tf within it.

Firstly we need to add information about the provider (Here, we will be using AWS Provider). We will also configure AWS credentials, so we could provision resources in the AWS cloud.

You can configure the AWS credentials in the provider section. Here, I am using the path to AWS config and credential files in my system to authenticate. Also, I am mentioning the region and profile that I will be using by default. (You can also use Access and Secret keys instead of config files)

Now that we have set the provider set, we have to initialise Terraform to use it. (This command will download/update the providers).

terraform init



terraform init

# Networking requirements

We are going to create a VPC with 3 subnets (2 — public, 1 — private) across the region as,

* a public subnet in us-east-1a
* a public subnet in us-east-1b
* a private subnet in us-east-1b

*Note: You can create a new file for VPC configuration or continue in the same file. (Separating resources into different files makes it easier to understand and maintain the infrastructure).*

VPC and subnets

We can create a VPC using the resource aws\_vpc. While creating VPC, it’s important to choose a CIDR block that is large enough to accommodate the desired number of resources, but not too large to waste IP addresses. (Here we have assigned a pool of 512 IP addresses for this tutorial)

We create subnets using the aws\_subnet resource by providing,

* the VPC id
* CIDR block of IP addresses (Make sure the IP address block is within the IP address range of the VPC)
* and the availability zone.

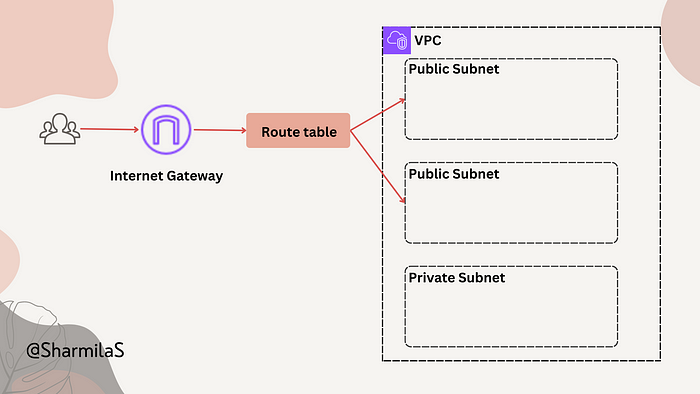
For the public subnet, we have to provide map\_public\_ip\_on\_launch as true (Default — False).

# Route tables and Gateways

## 1. Gateway for public subnets

To have the load balancer open to the internet, we will place it in the public subnet. We also need an Internet Gateway for the public subnet.

We can create a route table for our public subnet to connect it to Internet Gateway.



Let’s create an Internet Gateway and a route table. Then, associate this route table with our public subnets.

Internet gateway for the public subnets

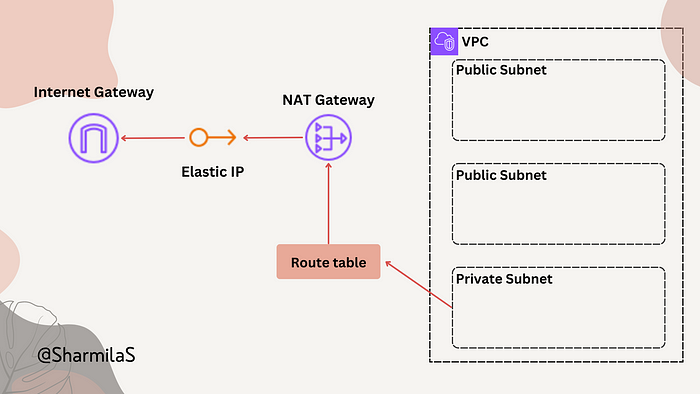
*Note:* aws\_route\_table\_association resource is used to associate the route table to a subnet.

## 2. Gateway for Private Subnet

We will be creating our EC2 instances in the private subnet, allowing only the requests from the load balancer to reach the instances.

But, the instances might need to connect to the internet to download software/tools. To provide access to the internet, we need a NAT gateway for the private subnet.

*The NAT instance must have internet access, So, it must be in a public subnet (a subnet that has a route table with a route to the internet gateway), and it must have a public IP address or an Elastic IP address.*



Creating a route table and associating it with the subnet is the same process as the previous one.

NAT gateway for the private subnet

# Configuring the Load Balancer

We will be creating an application load balancer for our web application, to handle HTTP and HTTPS requests.

Remember to provide a public subnet(s) for our load balancer, since we need it to be internet-facing.

*Let’s proceed to Auto Scaling group configuration first and cover the security group setups later in this article.*

# Creating the Auto Scaling Group

We will be creating and using an EC2 Launch Template to create EC2 instances.

*You can also provide the EC2 configurations directly in the aws\_autoscaling\_group resource instead of creating a launch template to use*

In the launch template, we are providing,

—the AMI we are going to use (make sure to get the AMI id from the same region as the EC2 instance)  
— the instance type  
— user data script (base64 encoded file which has commands to start Apache web server)  
— security group, subnet id for network configurations.

*We will create the security group mentioned here in the next section*

For the Auto scaling group,

1. Configuring the capacity:

* max\_size represents the maximum number of instances that the auto scaling group (ASG) can have, setting a limit to prevent excessive resource usage or costs.
* min\_size sets the minimum number of instances that the group should always have, ensuring a baseline capacity during low demand.
* desired capacity is the desired number of instances that the ASG should aim to have at any given time. It represents the ideal number of instances to handle the current workload.

2. We can connect this Auto Scaling Group to the Load Balancer target group using target\_group\_arns.

3. We add the launch template we created and also the private subnet.

Auto Scaling group with Launch template

# My user data file: (to run Web server)

#!/bin/bash  
sudo yum update -y  
sudo yum install -y httpd  
sudo systemctl start httpd  
sudo systemctl enable httpd  
echo "<h1>Hello World from $(hostname -f)</h1>" > /var/www/html/index.html

# Security access rules — for EC2 and Load Balancer

Let’s create a security group each for load balancer and the EC2 instances.

For the Load Balancer, we are allowing HTTP and HTTPS requests from the **internet**.

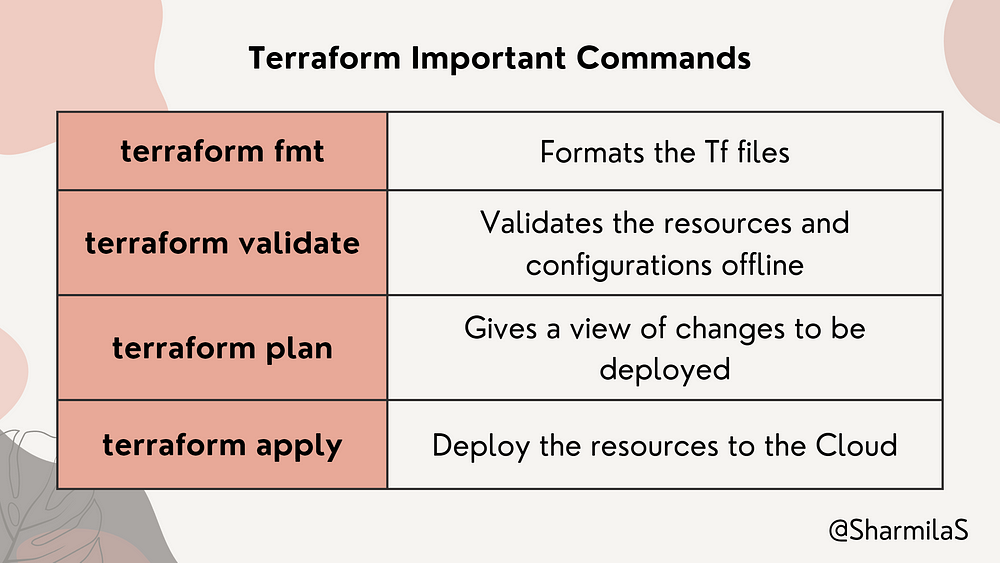
For the EC2 instances, we allow only HTTP requests **from** **the load balancer**.

Security groups for ELB and EC2

# Terraform apply

We have all the resources ready!

Let’s look at the terraform commands that we would be needing every time we deploy any changes.

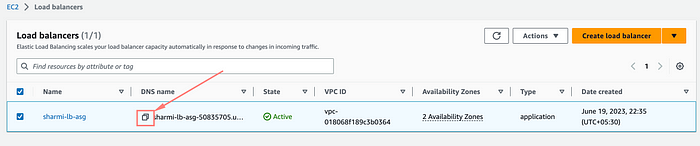


Try the above commands in that order.

terraform plan gives you details about the resources that will be deployed, and if it runs successfully and verified the resources, we can proceed to deploy the resources by using terraform apply command.

After the changes are deployed,

* Go to [AWS Console](https://aws.amazon.com/console/) and sign in.
* Go to Load Balancer under EC2 service.
* Select the load balancer that we created and copy the **DNS name**.



* Hit the URL in the browser



You can make changes in the infrastructure and try the above commands in the same order, to deploy them. The terraform state files save the current state of the infrastructure.

## Backup:

The terraform state files (terraform.tfstate and terraform.tfstate.backup) can be backed up in storage — using which we can provision the resources again anytime if something goes wrong.

**Note**: It is recommended to make changes in the terraform code once deployed and **not directly in the AWS console.**Terraform manages and tracks the state of your infrastructure. If you modify resources in AWS manually, it can lead to inconsistencies between the desired state in your Terraform code and the actual state of the resources. This can cause conflicts and make it difficult to manage and update your infrastructure accurately.

# Terraform destroy

If you want to destroy all the resources created using Terraform in this project, you can use terraform destroy command.  
This could be particularly helpful when you are learning Terraform, you can delete all the resources easily at the end, to avoid costs.

If you have any doubts, or any suggestions to improve this article, feel free to add them in the comments.

## GitHub Repo:

## [GitHub - SharmilaS22/medium-tf-ec2-alb-asg: Terraform code for creating a load balancer and auto…](https://github.com/SharmilaS22/medium-tf-ec2-alb-asg?source=post_page-----752afd44df8e--------------------------------" \t "_blank)

### **[Terraform code for creating a load balancer and auto scaling group. - GitHub - SharmilaS22/medium-tf-ec2-alb-asg…](https://github.com/SharmilaS22/medium-tf-ec2-alb-asg?source=post_page-----752afd44df8e--------------------------------" \t "_blank)**

[github.com](https://github.com/SharmilaS22/medium-tf-ec2-alb-asg?source=post_page-----752afd44df8e--------------------------------" \t "_blank)

Hope this article helps! Follow for more!!