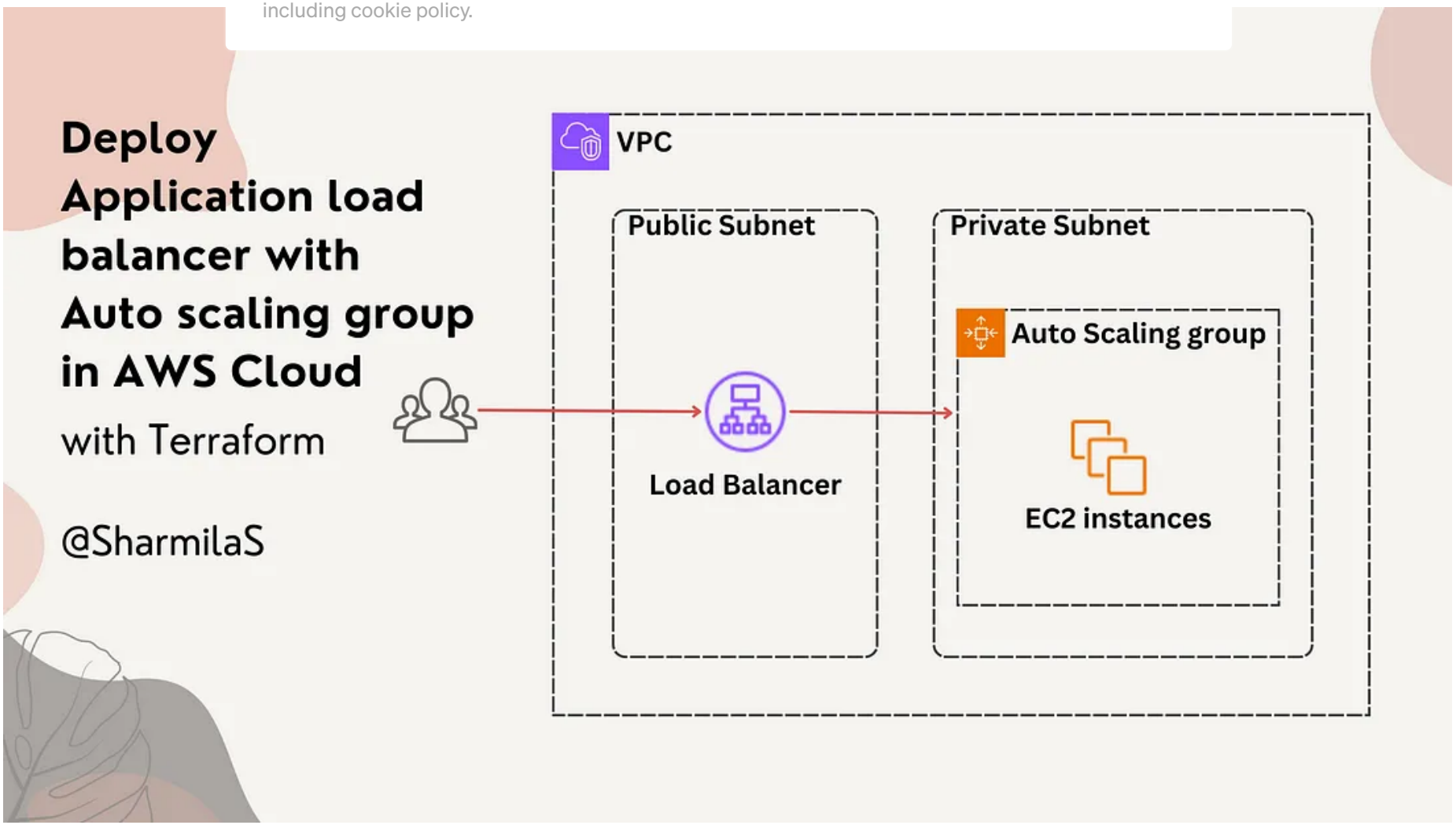
A Step-by-Step Guide to Creating Load Balancer and EC2 with Auto Scaling Group using Terraform

Achieving Scalability and High Availability using Load Balancer and Auto Scaling groups for your server.

Sharmila S · Follow

7 min read · Jun 19



Amazon 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 im

Amazon 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.

Terraform 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 already have the AWS CLI installed, you can skip this step.

Installing or updating the latest version of the AWS CLI

Install the AWS CLI on your system.

docs.aws.amazon.com

The next step is to configure your AWS profile to use in the CLI. Follow the instructions from the official documentation based on your machine type.

Configuring the AWS CLI

Configure the AWS Command Line Interface (AWS CLI) and specify the settings for interacting with AWS.

docs.aws.amazon.com

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

Install Terraform on Mac, Linux, or Windows by downloading the

3. Initialise Project

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

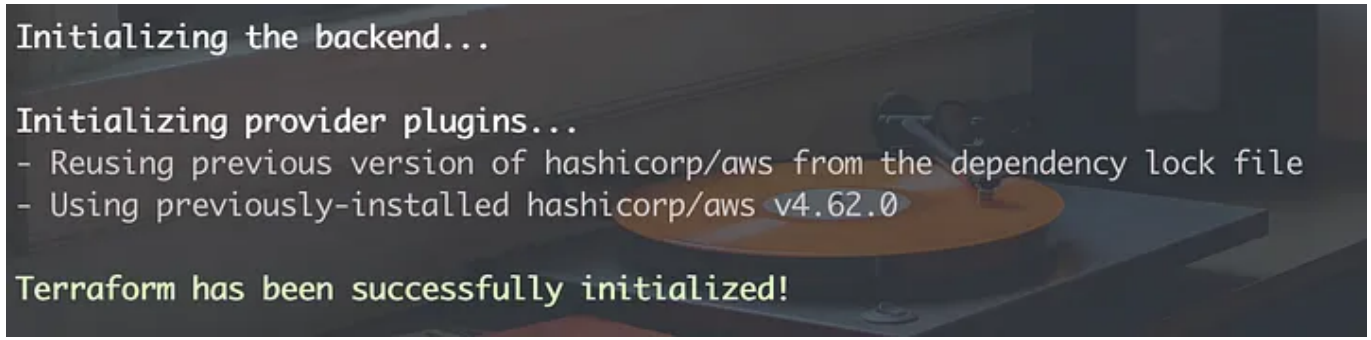
Firstly we need to add information about the provider (Here, we will be using AWS Provider). We will also configure AWS credentials, so we could provision resources in the AWS cloud.

```
terraform {

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

      version = "~> 4.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region                  = "us-east-1"
  shared_config_files     = ["/Path/to/.aws/config"]
  shared_credentials_files = ["/Path/to/.aws/credentials"]
  profile                 = "PROFILE"
}
```

main.tf hosted with ❤️ by GitHub view raw

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

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 and m*


```
# VPC
resource "aws_vpc" "sh_main" {
  cidr_block = "10.0.0.0/23" # 512 IPs
  tags = {
    Name = "sharmi-vpc"
  }
}

# Creating 1st public subnet
resource "aws_subnet" "sh_subnet_1" {
  vpc_id                  = aws_vpc.sh_main.id
  cidr_block              = "10.0.0.0/27" #32 IPs
  map_public_ip_on_launch = true          # public subnet
  availability_zone       = "us-east-1a"
}
# Creating 2nd public subnet
resource "aws_subnet" "sh_subnet_1a" {
  vpc_id                  = aws_vpc.sh_main.id
  cidr_block              = "10.0.0.32/27" #32 IPs
  map_public_ip_on_launch = true           # public subnet
  availability_zone       = "us-east-1b"
}
# Creating 1st private subnet
resource "aws_subnet" "sh_subnet_2" {
  vpc_id                  = aws_vpc.sh_main.id
  cidr_block              = "10.0.1.0/27" #32 IPs
```

```
27    map_public_ip_on_launch = false          # private subnet
28
29
```


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.

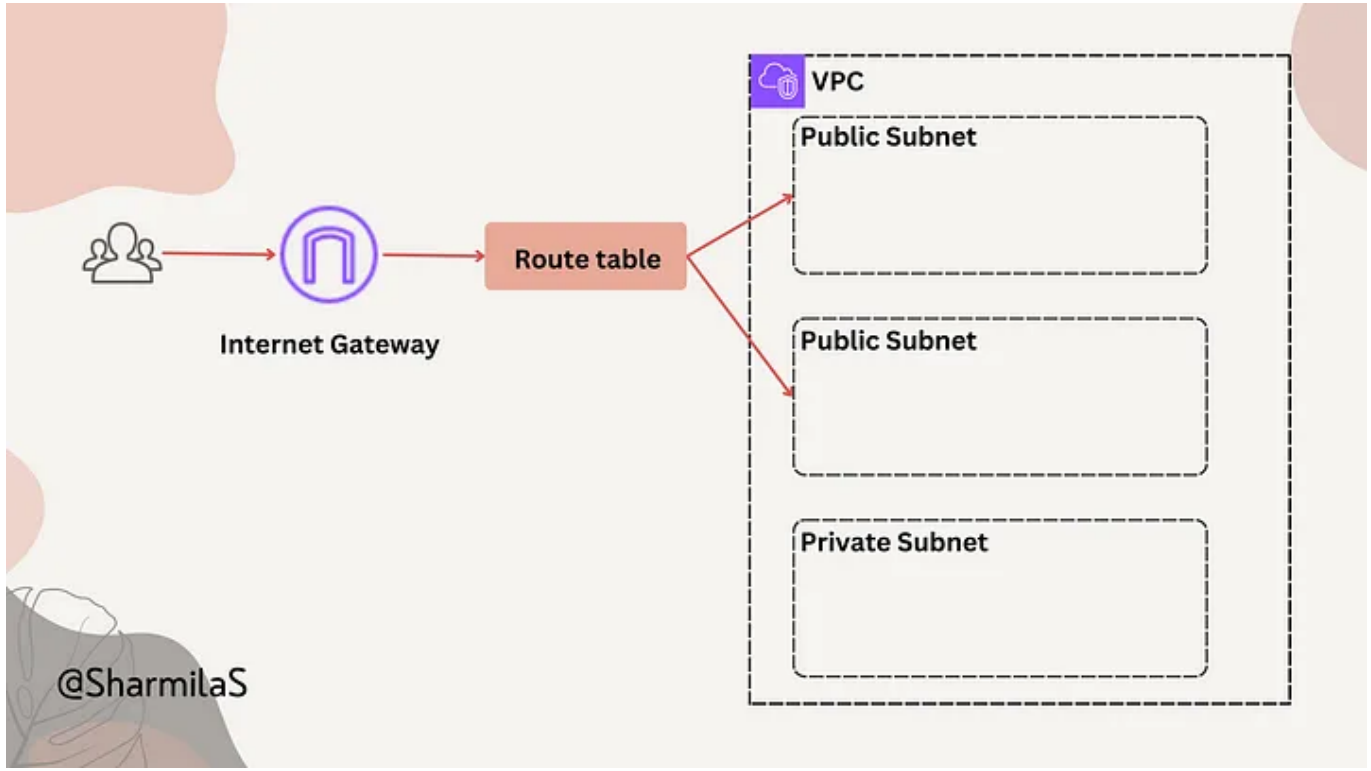
For the public subnet, we have to provide `map_public_ip_on_launch` as *true* (Default — *False*).

Route tables and Gateways

1. Gateway for public subnets

To su...

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 tal


```
# Internet Gateway
resource "aws_internet_gateway" "sh_gw" {
  vpc_id = aws_vpc.sh_main.id
}

# route table for public subnet - connecting to Internet gateway
resource "aws_route_table" "sh_rt_public" {
  vpc_id = aws_vpc.sh_main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.sh_gw.id
  }
}

# associate the route table with public subnet 1
resource "aws_route_table_association" "sh_rta1" {
  subnet_id      = aws_subnet.sh_subnet_1.id
  route_table_id = aws_route_table.sh_rt_public.id
}
# associate the route table with public subnet 2
resource "aws_route_table_association" "sh_rta2" {
  subnet_id      = aws_subnet.sh_subnet_1a.id
  route_table_id = aws_route_table.sh_rt_public.id
}
```

gateways-public.tf hosted with ❤️ by GitHub view raw

Internet gateway for the public subnets

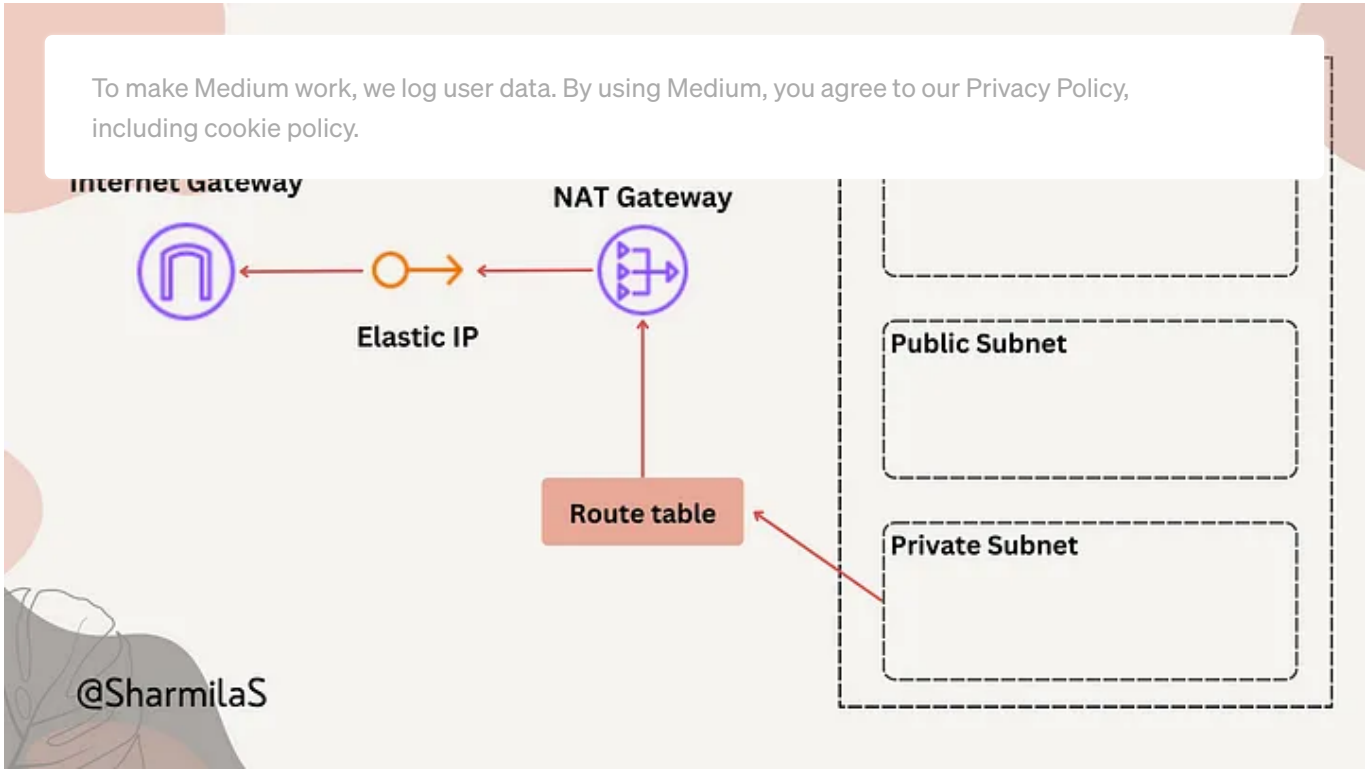
> *N* :e
> table to a subnet.

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.

```
# Elastic IP for NAT gateway
resource "aws_eip" "sh_eip" {
  depends_on = [aws_internet_gateway.sh_gw]
  vpc        = true
  tags = {
    Name = "sh_EIP_for_NAT"
  }
}
```

```
  allocation_id = aws_eip.sh_eip.id
  subnet_id     = aws_subnet.sh_subnet_1.id # nat should be in public subnet

  tags = {
    Name = "Sh NAT for private subnet"
  }

  depends_on = [aws_internet_gateway.sh_gw]
}

# route table - connecting to NAT
resource "aws_route_table" "sh_rt_private" {
  vpc_id = aws_vpc.sh_main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.sh_nat_for_private_subnet.id
  }
}

# associate the route table with private subnet
resource "aws_route_table_association" "sh_rta3" {
  subnet_id      = aws_subnet.sh_subnet_2.id
  route_table_id = aws_route_table.sh_rt_private.id
}
```

gateways-private.tf hosted with ❤️ by GitHub view raw

NAT gateway for the private subnet

Configuring the Load Balancer

We
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 gr*

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` sets 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)

```
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.**

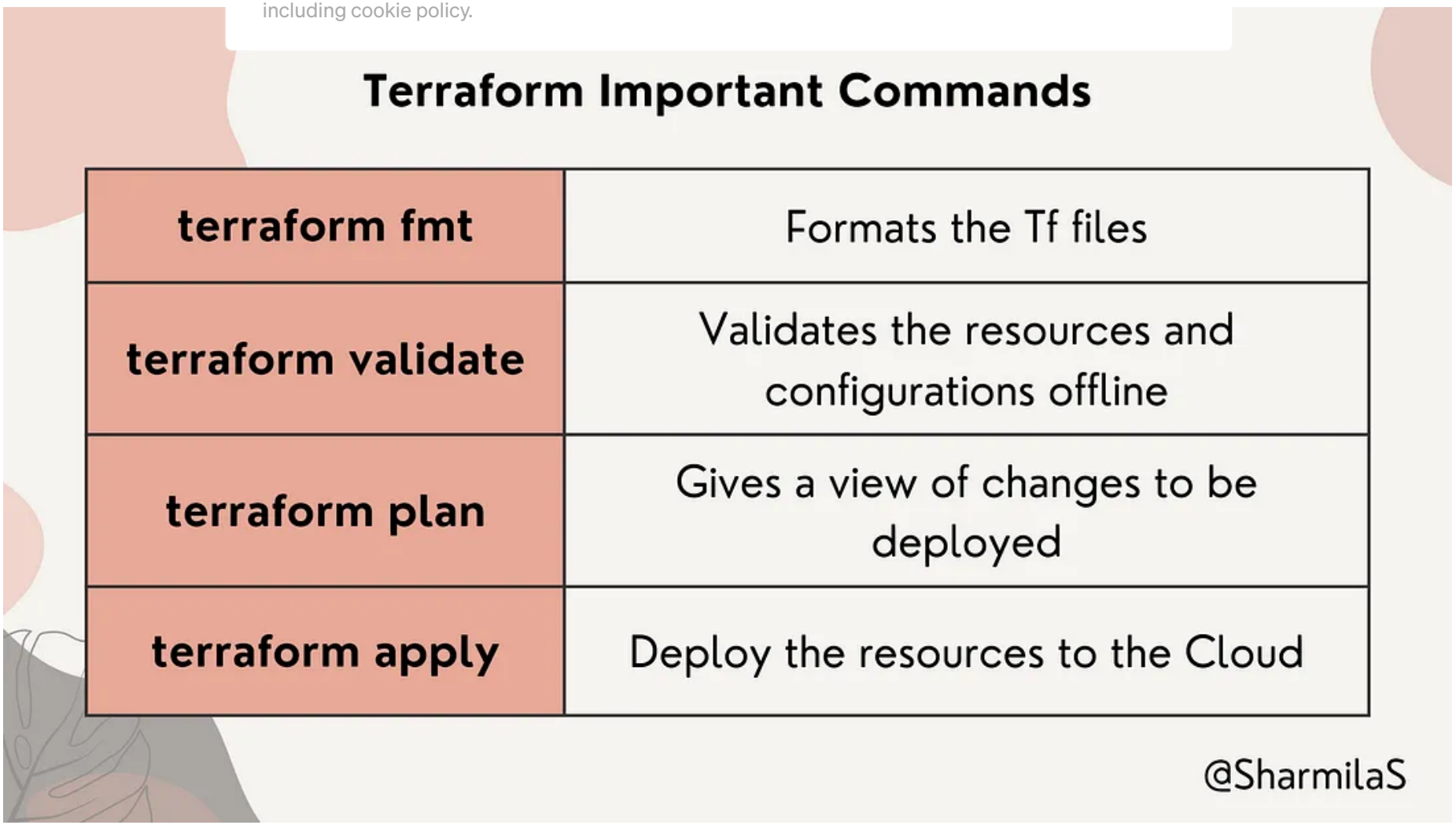
For the EC2 instances, we allow only HTTP requests **from the load balancer.**

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.



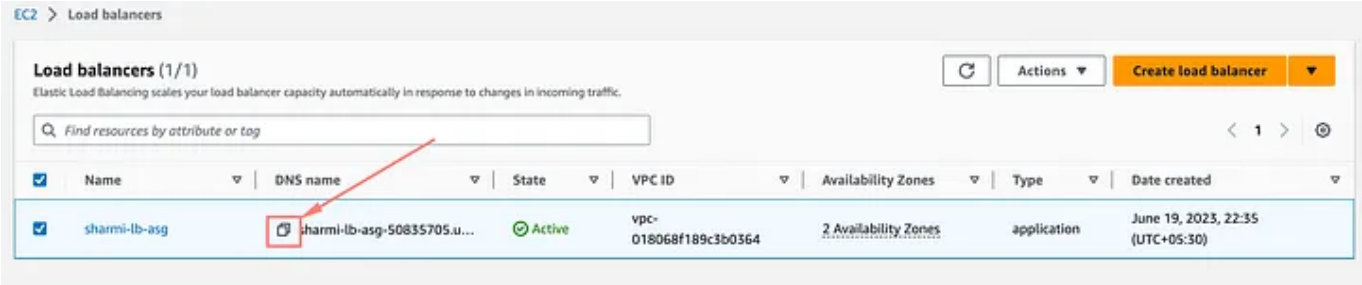
Try the above commands in that order.

`terraform plan` gives you details about the resources that will be deployed an

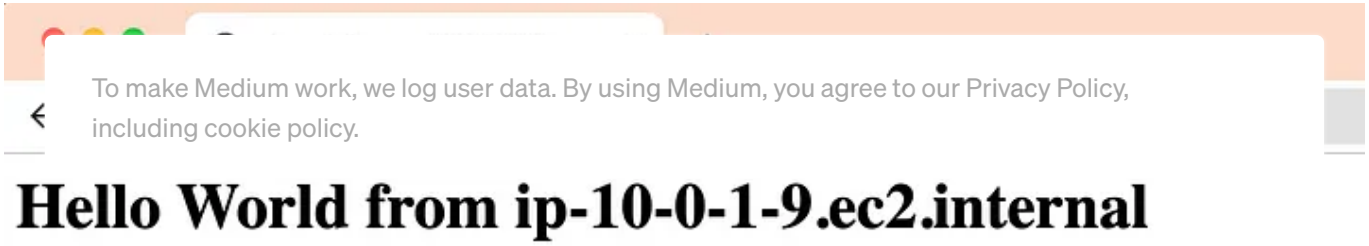
de

After the changes are deployed,

- Go to AWS 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

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:

for creating a load balancer and auto...

Terraform code for creating a load balancer and auto scaling group. - GitHub - SharmilaS22/medium-tf-ec2-alb-asg...

github.com

Hope this article helps! Follow for more!!

AWS | Aws Ec2 | Cloud Computing | Technology | DevOps

Written by Sharmila S

273

Software Engineer | Writes about Full Stack Web Development |
http://sharmilas.bio.link/

More from Sharmila S

Sharmila S in featurepreneur

Connect MongoDB database to Express server—step-by-step

4 min read · May 2, 2021

104 3

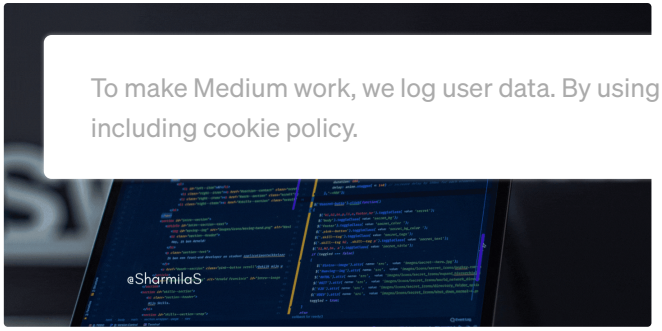
Sharmila S

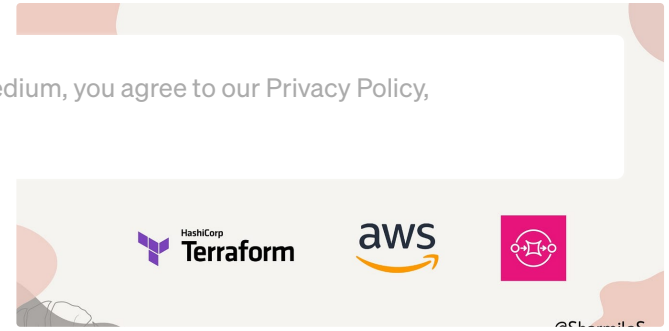
Get Started with RabbitMQ in Node.js

In this article, we are going to connect two node.js applications with a queue using...

6 min read · Jan 28, 2022

238 1





Sharmila S

Download files for the client from the Node.js server using this quic...

Add a download option for files on your Node.js web application for users to...

3 min read · Nov 30, 2021

72

Sharmila S

Creating an SQS Queue Using Terraform: A Step-by-Step Guide

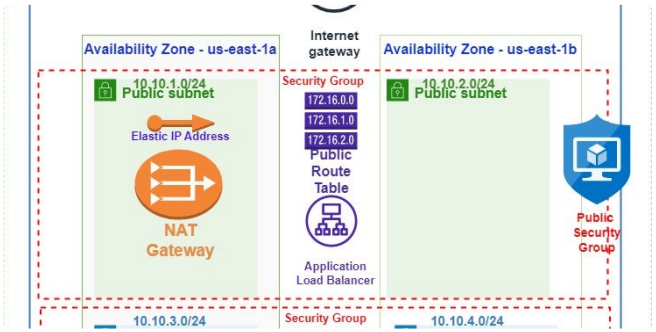
In this article, we will use Terraform to create and manage an SQS Queue while exploring...

7 min read · Jul 5

9

See all from Sharmila S

Recommended from Medium



Jason Li ☸ ☁ in AWS in Plain English

How to create an AWS EC2 Auto-Scaling Group for High Availabilit...

Welcome to week 21 with LUIT! As we continue to explore and learn the Efficiency,...

12 min read · Jun 3

140

B8

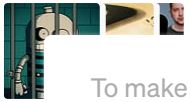
Terraform manage multiple environments

How to manage TF multiple environments in your projects

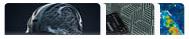
13 min read · Jul 24

67 1

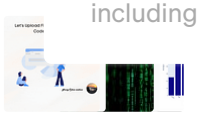
Lists



AI Regulation



ChatGPT prompts

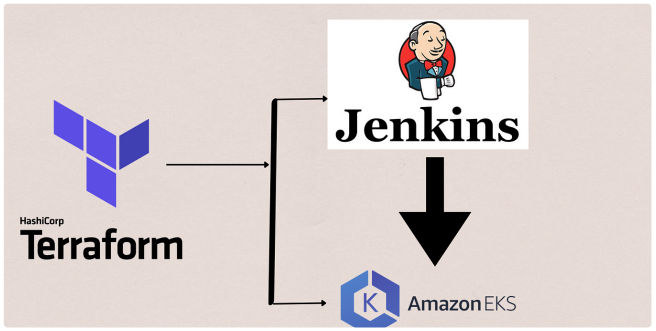


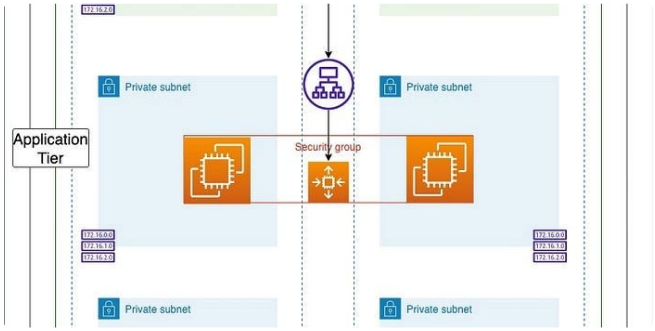
21 stories · 210 saves

ed

Reading

52 stories · 325 saves





Mubin Khalife

DevOps project using Terraform, Jenkins, and EKS

In this article, we are going to create a DevOps project where we'll use Terraform,...

12 min read · Jun 20

22 1

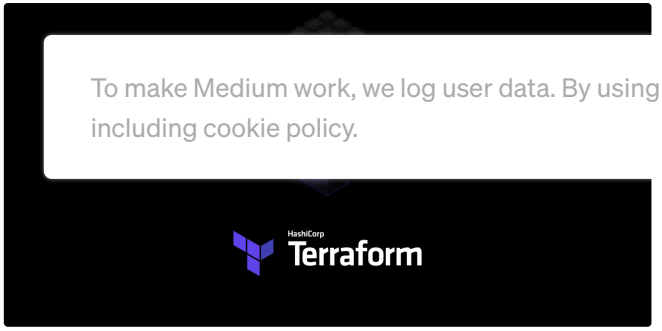
priyanka kumari

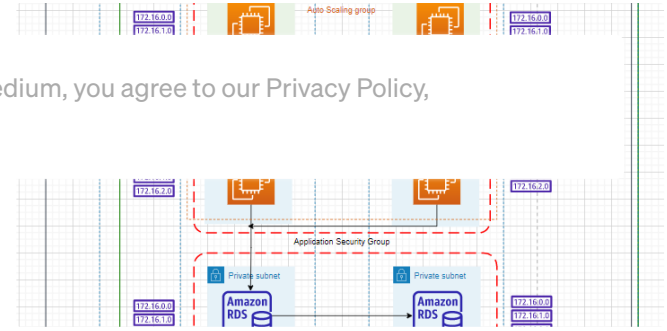
How to build 3 -tier architecture in AWS

What is a 3 Tier Architecture

12 min read · Aug 4

8





Dhruvin Soni

How to import existing AWS resources using Terraform?

How to import existing AWS resources using Terraform?

6 min read · Apr 24

13 1

Navidehbaghaifar

Designing an AWS Three-Tier Architecture

In this week's project, I will guide you through the steps of constructing an AWS Three-Tier...

17 min read · Jun 23

6 1

See more recommendations