**Step-by-Step Guide to Deploy Infrastructure with Terraform**

**1. Prerequisites**

Before you begin, ensure you have the following:

* **AWS Account**: You need an active AWS account.
* **Terraform**: Install Terraform on your local machine. You can download it from the official Terraform website.
* **AWS CLI**: Install the AWS CLI tool and configure it with your AWS credentials by running aws configure.

**2. Set Up Your Working Directory**

1. **Create a new directory** on your local machine to store your Terraform configuration files.

mkdir terraform-nodejs-app

cd terraform-nodejs-app

1. **Create a new file** named main.tf in this directory. This file will contain all the Terraform code provided in the script above.

**3. Write the Terraform Configuration**

1. **Open main.tf** in a text editor of your choice (e.g., VS Code, Sublime Text).
2. **Copy and paste** the Terraform script provided below into main.tf.

Ensure you replace the github URL in the EC2 user data section with the actual URL of your Node.js application repository.

**4. Initialize the Terraform Working Directory**

Before deploying the infrastructure, you need to initialize your Terraform working directory, which downloads the necessary provider plugins.

1. **Run the following command** in your terminal from the terraform-nodejs-app directory:

“terraform init”

1. **Check for any errors**. If there are no errors, Terraform has successfully initialized the directory.

**5. Review the Infrastructure Plan**

It's a good practice to review what Terraform will do before it makes any changes.

1. **Run the following command** to create an execution plan:

“terraform plan”

1. **Review the output**. Terraform will show you a list of resources it will create, modify, or delete. Ensure that the plan matches your expectations.

**6. Apply the Terraform Configuration**

After reviewing the plan and confirming it looks good, you can apply the configuration to create the resources.

1. **Run the following command** to deploy the infrastructure:

“terraform apply”

1. **Confirm the apply** when prompted by typing yes.
2. **Wait for Terraform** to finish deploying the resources. This process may take a few minutes.

**7. Access Your Node.js Application**

Once Terraform has finished applying the configuration, it will output the DNS name of the Application Load Balancer (ALB).

1. **Locate the output alb\_dns\_name** in the terminal. This is the DNS name of your ALB.
2. **Open a web browser** and navigate to the ALB DNS name. This should display your Node.js application running on the EC2 instance.

**8. Verify the Deployment**

To ensure everything is running correctly:

* **Check the EC2 instance** to make sure it is running.
* **Verify the Security Groups** are properly configured to allow traffic on ports 80 (HTTP) and 3000 (Node.js application).
* **Test the ALB** by accessing the DNS name. It should route traffic correctly to your Node.js application. Ensure you add the **“/api/greeting”** endpoint to the url as the application is active on that path.

**Additional Notes**

* **Region Configuration**: The script uses the AWS region us-east-1. If you want to deploy the infrastructure in a different region, update the provider "aws" { region = "us-east-1" } line with your desired region.
* **AMI Selection**: The Amazon Machine Image (AMI) ID ami-0c55b159cbfafe1f0 is used in the script. Ensure this AMI is available in your selected region or update it to an AMI ID available in your target region.
* **Security Considerations**: The security group allows inbound traffic on ports 80 and 3000 from any IP address (0.0.0.0/0). For a production environment, restrict this to trusted IP ranges.

**Issues Encountered:**  
I encountered the issue of the application load balancer not able to be provisioned on just one subnet alone. I had to create a second public subnet and attach the security group to it. This way the ALB is able to route request and ensure failover is achieved.   
  
Also, it takes about 10minutes before the web site comes up as the security group changes takes about 10 mins to get fully updated.  
  
  
**Automation Implemented**  
The whole setup was done via terraform using the AWS provider to configure and provision the ALB, Listener, Target Groups, Security Groups, EC2 Instance, VPC, Subnet, Route tables, etc.   
The script is attached below as well as the screenshot of the running instance.

**Terraform script**

#Setting up the provider

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

#setting up the Region

provider "aws" {

region = "us-east-1"

}

# Create VPC

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

enable\_dns\_support = true

enable\_dns\_hostnames = true

}

# Create Internet Gateway

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.main.id

}

# Create Public SubnetTwo

resource "aws\_subnet" "publicTwo" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.2.0/24"

map\_public\_ip\_on\_launch = true

availability\_zone = "us-east-1b"

}

# Associate Route Table with Public Subnet2

resource "aws\_route\_table\_association" "publicTwo" {

subnet\_id = aws\_subnet.publicTwo.id

route\_table\_id = aws\_route\_table.public.id

}

# Create Route Table for Public Subnet

resource "aws\_route\_table" "public" {

vpc\_id = aws\_vpc.main.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

}

# Associate Route Table with Public Subnet

resource "aws\_route\_table\_association" "public" {

subnet\_id = aws\_subnet.public.id

route\_table\_id = aws\_route\_table.public.id

}

# Create Security Group

resource "aws\_security\_group" "web\_sg" {

vpc\_id = aws\_vpc.main.id

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 3000

to\_port = 3000

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"]

}

}

# Launch EC2 Instance with Node.js Application

resource "aws\_instance" "web" {

ami = "ami-066784287e358dad1"

instance\_type = "t2.micro"

subnet\_id = aws\_subnet.public.id

security\_groups = [aws\_security\_group.web\_sg.name]

tags = {

Name = "Expatswap-NodeJSApp"

}

user\_data = <<-EOF

#!/bin/bash

yum update -y

curl -sL https://rpm.nodesource.com/setup\_14.x | bash -

yum install -y nodejs git

mkdir -p /var/www/nodeapp

cd /var/www/nodeapp

git clone https://github.com/emmaxyz90/expatswap.git .

npm install

npm start

EOF

}

# Create Application Load Balancer

resource "aws\_lb" "web\_alb" {

name = "web-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.web\_sg.id]

subnets = [aws\_subnet.public.id]

enable\_deletion\_protection = false

}

# Create Target Group

resource "aws\_lb\_target\_group" "web\_tg" {

name = "web-tg"

port = 3000

protocol = "HTTP"

vpc\_id = aws\_vpc.main.id

health\_check {

path = "/"

port = 3000

interval = 30

timeout = 5

healthy\_threshold = 2

unhealthy\_threshold = 2

}

}

# Attach EC2 Instance to Target Group

resource "aws\_lb\_target\_group\_attachment" "web\_tg\_attachment" {

target\_group\_arn = aws\_lb\_target\_group.web\_tg.arn

target\_id = aws\_instance.web.id

port = 3000

}

# Create Listener for ALB

resource "aws\_lb\_listener" "web\_listener" {

load\_balancer\_arn = aws\_lb.web\_alb.arn

port = 80

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.web\_tg.arn

}

}

output "alb\_dns\_name" {

value = aws\_lb.web\_alb.dns\_name

}

