**Process flow of the Cloud Provision IaC**

**Prerequisites of setting up a new Cloud Provider to deploy the application.**

* Need to have Terraform installed on the client desktop.
* Need to have ssh installed for remote login to the EC2 instance.
* Create a free-tier AWS account for the deployment using a new email ID and credit card number.
* Could use windows or MacBook desktop for provision of the AWS cloud.

**Infrastructure-as-Code to be deployed for the github application**

In order for Terraform to be able to make changes in your AWS account, you will need to configure the AWS credentials for the user you created earlier. There are several ways to do this (see [A Comprehensive Guide to Authenticating to AWS on the Command Line](https://blog.gruntwork.io/a-comprehensive-guide-to-authenticating-to-aws-on-the-command-line-63656a686799))

Update the Access code and Secret key into the the terraform code inside the variable.tf file

So that new credentials of the root are updated from the new aws account.

The Github repository link for the terraform IaC.

/terraform/

**Network.tf** 🡪 This script helps to create the VPC and access code.

**Routing.tf** 🡪 This script helps to create the route table and IGW

**Subnets.tf** 🡪 this script helps to create the subnet and AZ of network

**Securitygroups.tf** 🡪 This setups the firewall rules and security policy

**Dns-and-dhcp.tf** 🡪 This script setup the DNS zones and network access.

**Ec2-machine.tf** 🡪 This script create the Key pair and EC2 instance for deploying the application packages and codes.

**Asg.tf** 🡪 This script for HA and load balancing of the web and DB to create another instance in the AZ

**Variables.tf** 🡪 This are the variable file of the parameters in the terraform files.

**Architecture Diagram attached**

The architect diagrams is created with AWS RDS service for better security and RDS Managed services is better option to use as DB service. Due to cost involve in the having RDS services in Free tier Account, I had not included it with Terraform scripts but loaded the PostgreSQL DB in EC2 instance.

But in production environment, there is always a Licensed account from AWS, so we can use RDS service to pay-on-demand as we use the DB. Best practice is to private subnet for the RDS service to have a secure DB from the public.

**Deployment of the application from the github repository(https://github.com/servian/TechChallengeApp)**

|  |
| --- |
|  |
|  | *#Load the packages from linux*  Sudo yum update -y  sudo yum install -y git golang postgresql10 postgresql10-server postgresql10-contrib postgresql10-libs docker |
|  | *#service start of the docker* |
|  | sudo systemctl enable docker.service |
|  | sudo systemctl enable postgresql.service |
|  | sudo systemctl start docker.service |
|  | sudo cat <<EOF >>/var/lib/pgsql/data/pg\_hba.conf |
|  | local all all trust |
|  | host all 127.0.0.1/32 trust |
|  | EOF  *# service start of the postgresql* |
|  | sudo systemctl start postgresql.service |
|  | # download the github respostory from the net |
|  | git clone https://github.com/servian/TechChallengeApp |
|  | cd TechChallengeApp |
|  | go get -d github.com/Servian/TechChallengeApp |
|  | ./build.sh |
|  | cd dist  *#Run the application as according to the challenge document.* |
|  | ./TechChallengeApp updatedb |
|  | ./TechChallengeApp serve |

There are three commands which are pretty much required to provision the infrastructure using Terraform.

* $ terraform init
* $ terraform plan
* $ terraform apply

**Note : AWS RDS services is not created in the terraform scripts since Free tier Account charges for the RDS services and testing. Since charges are involve, loaded PostgreSQL in the EC2 instance itself to do the testing and demo**.