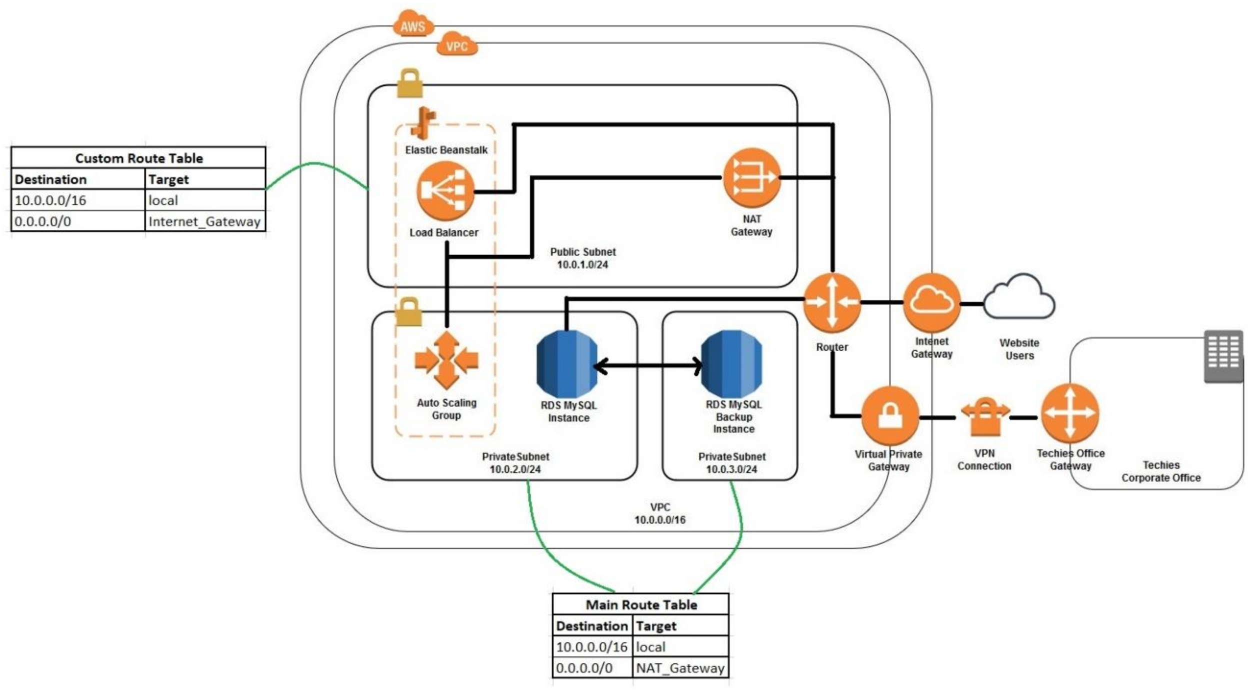
Contents

Proof of Concept for Techies

Technologies Used for the Solution

- Solution is deployed on Amazon Web Services using these services:
  - Amazon EC2
  - Amazon Elastic Beanstalk
  - Amazon RDS
- Solution will be deployed using Hashicorp's Terraform.
- You can download the Terraform files from the Github repository: https://github.com/dinukaj91/VoiceIQ

Infrastructure Overview



Custom Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	Internet_Gateway

Main Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	NAT_Gateway

- The proposed infrastructure has been designed on Amazon Web Services and the network diagram given above describes the proposed infrastructure.
- The solution consists of a high availability PHP application launched on Elastic Beanstalk with an external Amazon RDS database with the multi AZ feature enabled.
- The infrastructure is hosted on a single VPC with three subnets.
- One subnet is a public subnet and the other two subnets are private subnets.
- The public subnet contains the load balancer for the Elastic Beanstalk cluster.

- One private subnet contains the auto scaling group to which the Elastic load balancer connects to and the RDS database.
- To deal with increases in traffic Beanstalk is configured to scale up buy one instance when CPU utilization goes above 80% and scale down when CPU utilization goes below 20%.
- To keep things simple, Beanstalk is configured to work in one availability zone i.e. when CPU utilization goes up a new server is added to the same availability zone (i.e. the same subnet).
- The second subnet contains the replica for the main RDS database in the first subnet.

Connectivity and Security

- The VPC contains an internet gateway to allow communication between the instances in the VPC and the internet.
- The public subnet contains a NAT gateway to enable outgoing internet connectivity from the private subnet.
- It is a requirement for instances in a Beanstalk auto scaling group to connect to the internet and the NAT gateway helps meet this requirement.
- During the creation of the Beanstalk cluster two security groups are automatically generated by Beanstalk.
  - One is attached to the load balancer so that web requests can connect with the load balancer on port 80.
  - The other is attached to the auto scaling group so that requests from the load balancer can connect with the instances in the auto scaling group on port 80.
- The security groups are given below. Note that the names for the security groups when creating the infrastructure is given the name Production automatically. To make my point clear I have used meaningful names in the two tables given below.

Load_Balancer_SG

Type	Protocol	Port Range	Source	Description
http	TCP	80	0.0.0.0/0	Access Web App through the Net

Auto_Scaling_Group_SG

Type	Protocol	Port Range	Source	Description
http	TCP	80	Load_Balancer_SG	Access Web App through the Load Balancer

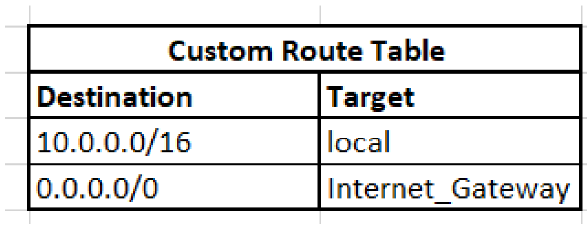
- Another security group is created and attached to the RDS database instance to enable connections from the auto scaling group and to allow connections from the corporate office. This security group is given below:

Database_Server_SG

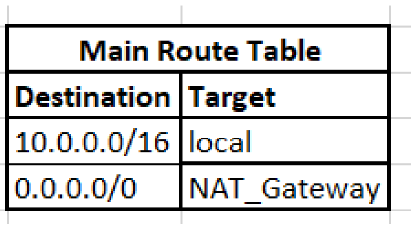
Type	Protocol	Port Range	Source	Description
MySQL/Aurora	TCP	3306	10.0.2.0/24	Access DB from the auto scaling group
MySQL/Aurora	TCP	3306	10.1.0.0/16	Access DB from the Corporate Office

- A virtual private gateway is attached to the VPC and this is connected to the Techies office gateway located at the Techies corporate office through a VPN connection.
- This enables the staff at the corporate office to connect to the database securely.

- The VPC consists of two route tables: the main route table and the custom route table.
- The main route table is attached to the two private subnets and the custom route table is attached to the public subnet.
- The custom route table is used to route traffic from the public subnet to the internet gateway to allow internet connectivity.
- The main route table is used to route traffic from the private subnets to the NAT internet gateway to allow internet connectivity.



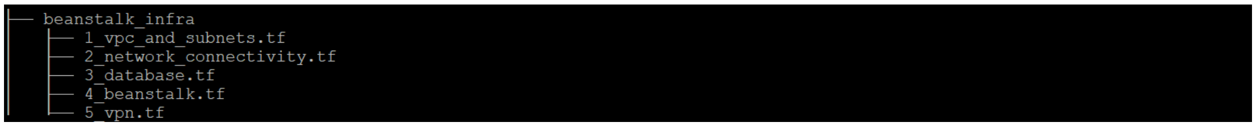
Custom Route Table	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	Internet_Gateway



Main Route Table	
Destination	**Target**
10.0.0.0/16	local
0.0.0.0/0	NAT_Gateway

Deploying the Infrastructure with Terraform

- The terraform files are contained in the github url given at the beginning of the document.
- They follow the following file structure



```
├── beanstalk_infra
│   ├── 1_vpc_and_subnets.tf
│   ├── 2_network_connectivity.tf
│   ├── 3_database.tf
│   ├── 4_beanstalk.tf
│   └── 5_vpn.tf
```

- Each file is used to deploy a certain part of the infrastructure and the name of each file describes which part of the infrastructure that is.
- Modifications can be made to the file according to your requirement.
- It is recommended that you have prior knowledge on Terraform when changing these files.
- The files contain inline comments describing what each resource does and together with the information given in this document you will be able to get a better understanding of what each of these files do.
- The current values will help you deploy the entire infra structure with minimal configuration.

Deploying the Environment

- You can clone the terraform files from the Github repo which is mentioned at the beginning of the document to your Linux environment.
- Then you can export the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY for the AWS account which you pan on deploying the infrastructure to.
- Change directory into the downloaded directory and execute the given commands to deploy the infrastructure.
    - terraform init
    - terraform apply