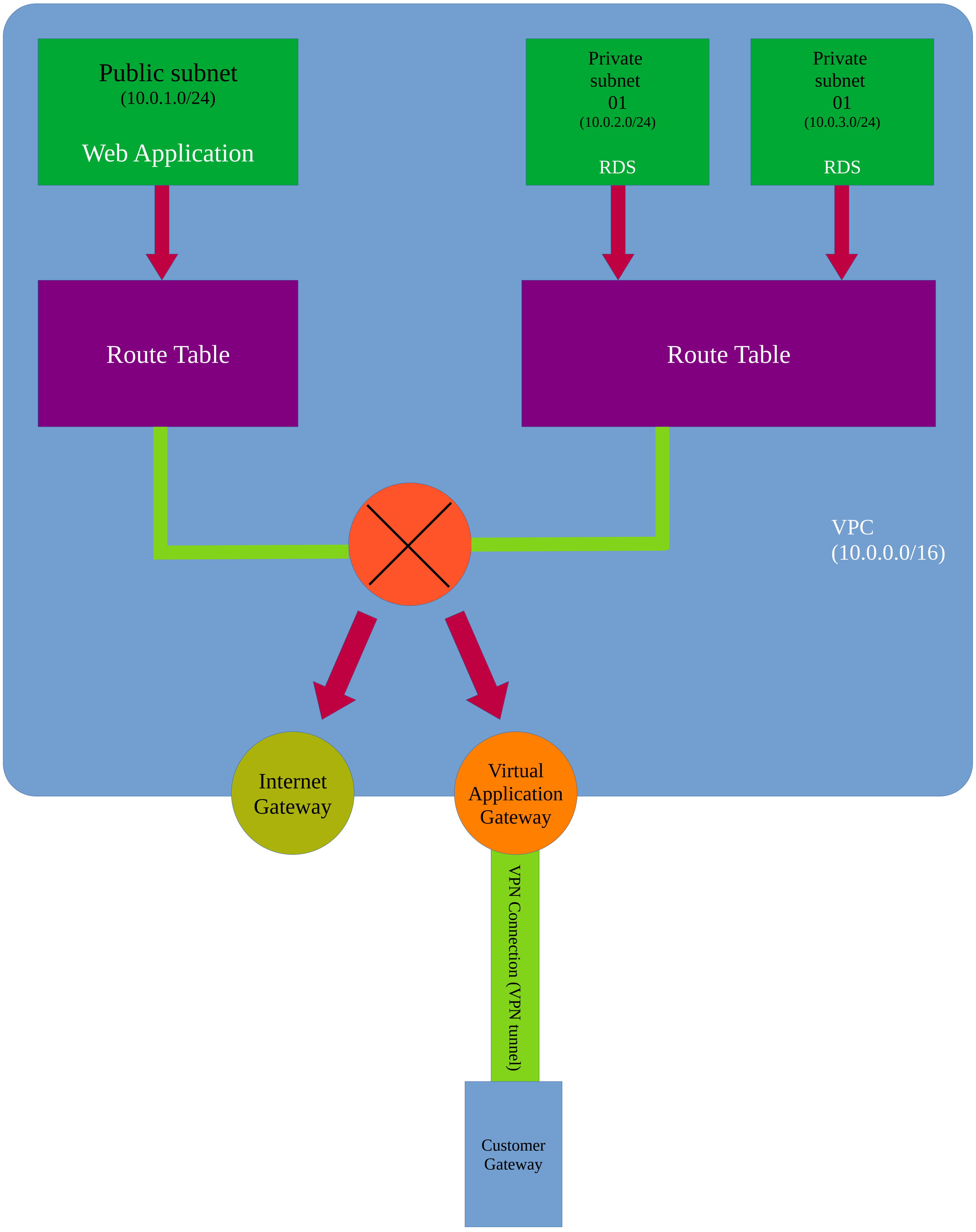
Automating the process of resource provisioning using Terraform

Infrastructure diagram :



Prerequisites :

- **AWS CLI**

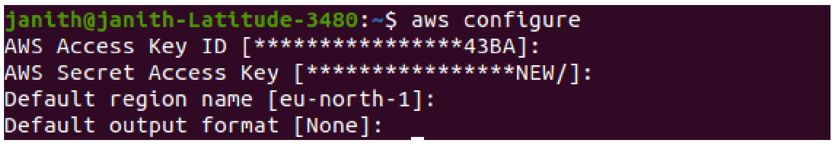
AWS is needed to be installed and configured as the first step

Installation :

Run following commands to download and install AWS CLI v2

- `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"`
- `unzip awscliv2.zip`
- `sudo ./aws/install`

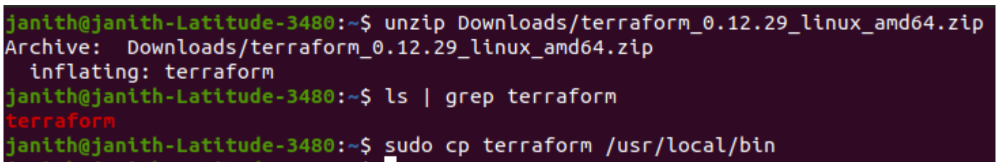
Then configure aws cli, adding Access key ID and Secret access key, as follows.



```
janith@janith-Latitude-3480:~$ aws configure
AWS Access Key ID [****************43BA]:
AWS Secret Access Key [****************NEW/]:
Default region name [eu-north-1]:
Default output format [None]:
```

- **Terraform**

In order to automate the infrastructure provisioning process, Terraform is used. First download Terraform from the official website (https://www.terraform.io/). Then unzip the downloaded zip file, and copy the terraform file to the ***/usr/local/bin/*** (verify that the installation is running as expected by executing *terraform -v* command)



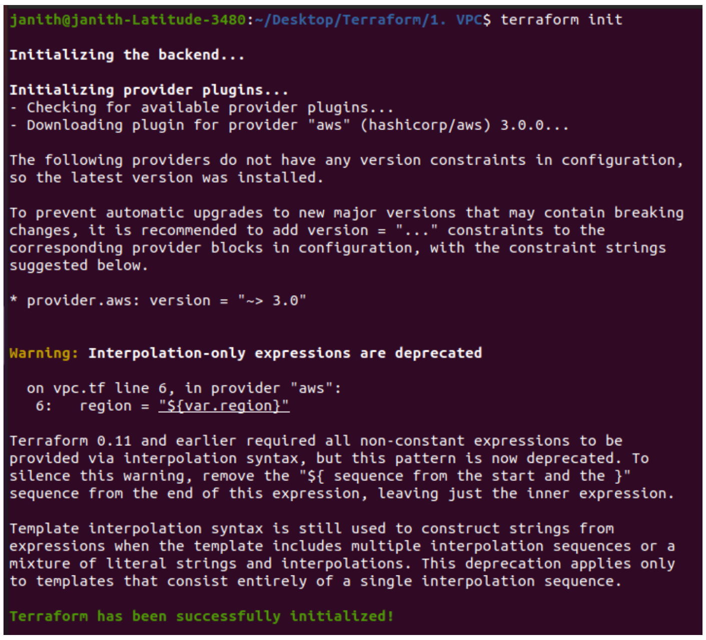
```
janith@janith-Latitude-3480:~$ unzip Downloads/terraform_0.12.29_linux_amd64.zip
Archive:  Downloads/terraform_0.12.29_linux_amd64.zip
  inflating: terraform
janith@janith-Latitude-3480:~$ ls | grep terraform
terraform
janith@janith-Latitude-3480:~$ sudo cp terraform /usr/local/bin
```

STEP 01

Provisioning VPC

Steps :

1. Navigate to the **1. VPC** directory. Then run the *terraform init* command as follows, in order to initialize **1. VPC** directory such that it contains appropriate Terraform configuration files for AWS



```
janith@janith-Latitude-3480:~/Desktop/Terraform/1. VPC$ terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 3.0.0...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 3.0"


Warning: Interpolation-only expressions are deprecated

  on vpc.tf line 6, in provider "aws":
   6:   region = "${var.region}"

Terraform 0.11 and earlier required all non-constant expressions to be
provided via interpolation syntax, but this pattern is now deprecated. To
silence this warning, remove the "${ sequence from the start and the }"
sequence from the end of this expression, leaving just the inner expression.

Template interpolation syntax is still used to construct strings from
expressions when the template includes multiple interpolation sequences or a
mixture of literal strings and interpolations. This deprecation applies only
to templates that consist entirely of a single interpolation sequence.

Terraform has been successfully initialized!
```

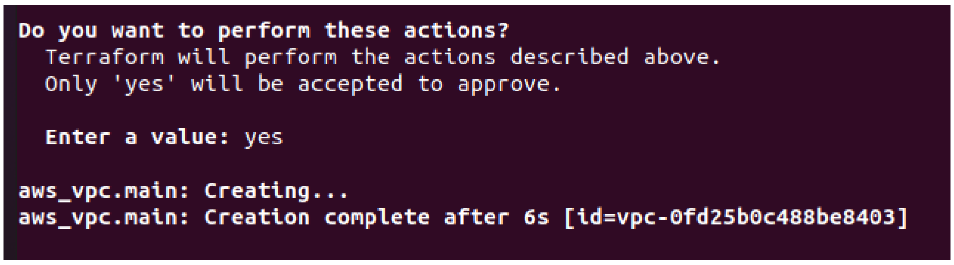
2. Do the necessary changes to the **vpc.tf** file according to the requirements of the organization (fields such as **cidr_block** and **tags**)

Example :

```
   Defining region and the other essential variable for the VPC to be deployed
variable "region" {default = "ap-southeast-1 "}

   Defining the provider as AWS
provider "aws" {
 region = "${var.region}"
}

   Provisioning resource
resource "aws_vpc" "main" {
 cidr_block       = "10.0.0.0/16 "
 instance_tenancy = "default"

 tags = {
  Name = "voiceiq"
 }

}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the VPC.



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.main: Creating...
aws_vpc.main: Creation complete after 6s [id=vpc-0fd25b0c488be8403]
```

NOTE : Please note down the ID of the just created VPC as it is required for future configurations

aws_vpc.main: Creation complete after 6s [id=vpc-0fd25b0c488be8403]

STEP 02

Subnets and the Internet Gateway

As the next step, one public subent and two private subnets need to be created. The public subnet is for the EC2 instance, which will be created with the Elastic beanstalk environment and two private subnets are for RDS instance which will create as we proceed. (The two private subnets are created in different availability zones, since the RDS instance is going to be a multi AZ one). Apart from that, an Internet gateway is also created and attached to the VPC, which was created previously.

Steps :

1. Navigate to the **2. Subnets** directory. Then run the *terraform init* command, in order to initialize **2. Subnets** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **subnets.tf** file

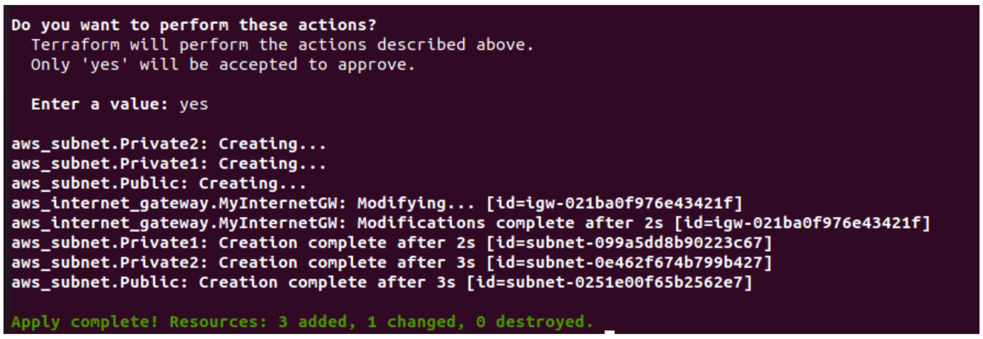
Example :

```
  Defining region and the other essential variables for the required Subnets to be deployed

variable "region" {default = "ap-southeast-1"}
variable "vpc_id" {default = "vpc-0fd25b0c488be8403"}
variable "AZ_a" {default = "ap-southeast-1a"}
variable "AZ_b" {default = "ap-southeast-1b"}
variable "AZ_c" {default = "ap-southeast-1c"}

  Defining the provider as AWS

provider "aws" {
 region = "${var.region}"
}

  Provisioning resource (Subents & Internet gateway)
resource "aws_subnet" "Public" {
 vpc_id     = "${var.vpc_id}"
 cidr_block = "10.0.1.0/24"
 availability_zone = "${var.AZ_a}"
 map_public_ip_on_launch = "true"

 tags = {
   Name = "Public"
 }
}

resource "aws_subnet" "Private1" {
 vpc_id     = "${var.vpc_id}"
 cidr_block = "10.0.2.0/24"
 availability_zone = "${var.AZ_b}"
 map_public_ip_on_launch = "false"

 tags = {
   Name = "Private1"
 }
}

resource "aws_subnet" "Private2" {
 vpc_id     = "${var.vpc_id}"
 cidr_block = "10.0.3.0/24"
 availability_zone = "${var.AZ_c}"
 map_public_ip_on_launch = "false"

 tags = {
   Name = "Private2"
 }
}

resource "aws_internet_gateway" "MyInternetGW" {
 vpc_id = "${var.vpc_id}"

 tags = {
   Name = "MyInternetGW"
 }
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to create all three subnets and the internet gateway.

NOTE : Please note down the subnet IDs and internet gateway ID just created, as they are required for future configurations



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_subnet.Private2: Creating...
aws_subnet.Private1: Creating...
aws_subnet.Public: Creating...
aws_internet_gateway.MyInternetGW: Modifying... [id=igw-021ba0f976e43421f]
aws_internet_gateway.MyInternetGW: Modifications complete after 2s [id=igw-021ba0f976e43421f]
aws_subnet.Private1: Creation complete after 2s [id=subnet-099a5dd8b90223c67]
aws_subnet.Private2: Creation complete after 3s [id=subnet-0e462f674b799b427]
aws_subnet.Public: Creation complete after 3s [id=subnet-0251e00f65b2562e7]

Apply complete! Resources: 3 added, 1 changed, 0 destroyed.
```

STEP 03

Route-table for the VPC

As the next step, a route-tables should be created in order to create a route to the internet gateway.

Steps :

1. Navigate to the **3. RouteTable** directory. Then run the *terraform init* command, in order to initialize **3. RouteTable** directory such that it contains appropriate Terraform configuration files for AWS.

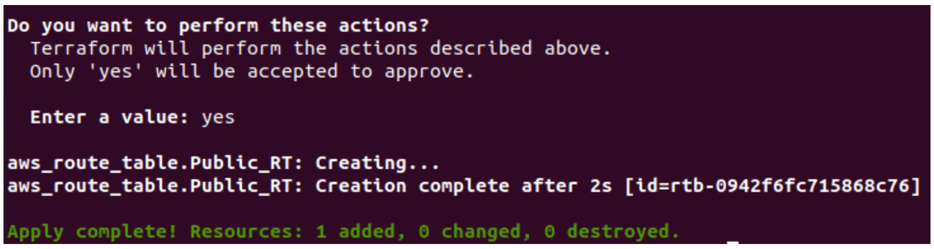
2. Do the necessary changes to the **route_table.tf** file

Example :

```
  Defining region and the other essential variables for the required Route table to be deployed

variable "region" {default = "ap-southeast-1"}
variable "vpc_id" {default = "vpc-0fd25b0c488be8403"}
variable "igw_id" {default = "igw-021ba0f976e43421f"}

  Defining the provider as AWS

provider "aws" {
 region = "${var.region}"
}

  Provisioning resource (Route Table)

resource "aws_route_table" "Public_RT" {
 vpc_id = "${var.vpc_id}"

 route {
  cidr_block = "0.0.0.0/0"
  gateway_id = "${var.igw_id}"
 }

 tags = {
  Name = "Public_RT"
 }

}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Route-table.



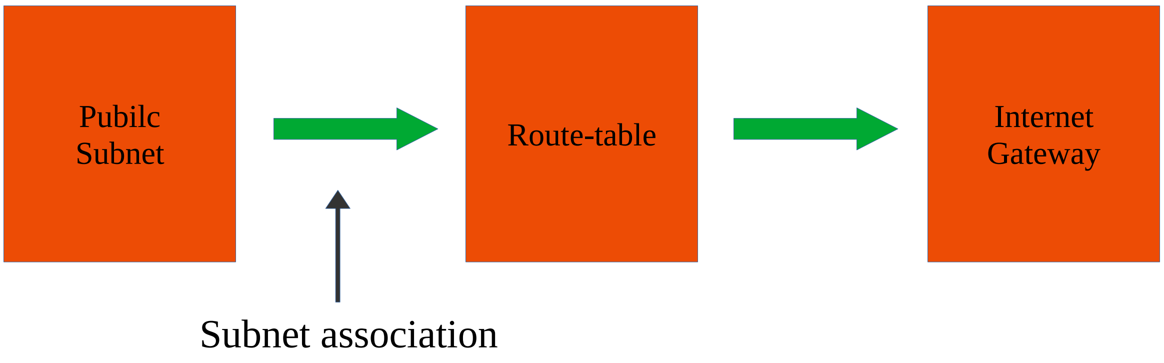
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_route_table.Public_RT: Creating...
aws_route_table.Public_RT: Creation complete after 2s [id=rtb-0942f6fc715868c76]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

NOTE : Please note down the ID of the just created Route-table as it is required for future configurations

STEP 04

Subnet association for the Route-table

Purpose of this is to create a route from public subnet to the previously created route-table.



Steps :

1. Navigate to the **4. SubnetAssociation** directory. Then run the *terraform init* command, in order to initialize **4. SubnetAssociation** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **subnet_association.tf** file

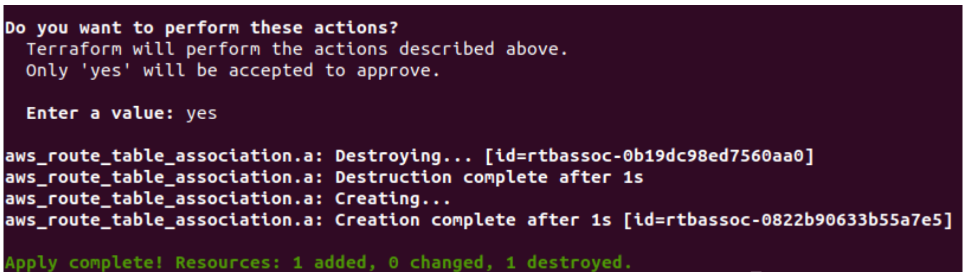
Example :

```
   Defining region and the other essential variables for the required Subnet association to be deployed

variable "region" {default = "ap-southeast-1"}
variable "public_subnet_id" {default = "subnet-0f984d87a7d07b735"}
variable "rt_id" {default = "rtb-0942f6fc715868c76"}

   Defining the provider as AWS

provider "aws" {
 region = "${var.region}"
}
```

```
   Provisioning resource (Subnet association)
resource "aws_route_table_association" "a" {
  subnet_id      = "${var.public_subnet_id}"
  route_table_id = "${var.rt_id}"
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to create subnet association.



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_route_table_association.a: Destroying... [id=rtbassoc-0b19dc98ed7560aa0]
aws_route_table_association.a: Destruction complete after 1s
aws_route_table_association.a: Creating...
aws_route_table_association.a: Creation complete after 1s [id=rtbassoc-0822b90633b55a7e5]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

STEP 05

Create Security Groups

Steps :

1. Navigate to the **5. SecuriyGroups** directory. Then run the *terraform init* command, in order to initialize **5. SecuriyGroups** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **sg.tf** file

Example :

```
   Defining region and the subnet group for the RDS instance to be deployed

variable "region"           { default = "ap-southeast-1" }
variable "vpc_id"           { default = "vpc-069ffe9976319a412" }
variable "vpc_cidr"          { default = "10.0.0.0/16" }

   Defining the provider as AWS

provider "aws" {
  region = "${var.region}"
}

   Provisioning resource (Security group for Elastic Beanstalk)

resource "aws_security_group" "allow_http" {
  name        = "allow_http"
  description = "Allow HTTP inbound traffic"
  vpc_id      = "${var.vpc_id}"
```

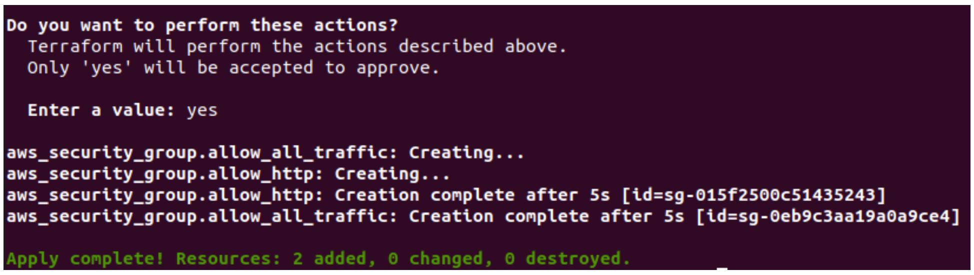
```
 ingress {
  from_port   =
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 }

 egress {
  from_port   =
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
 }

 tags = {
  Name = "allow_HTTP"
 }
}
```

Provisioning resource (Security group for RDS)

```
resource "aws_security_group" "allow_all_traffic" {
 name        = "allow_all"
 description = "Allow ALL inbound traffic"
 vpc_id      = "${var.vpc_id}"

 ingress {
  from_port   =
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
 }

 egress {
  from_port   =
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
 }

 tags = {
  Name = "allow_ALL"
 }
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Security Groups



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_security_group.allow_all_traffic: Creating...
aws_security_group.allow_http: Creating...
aws_security_group.allow_http: Creation complete after 5s [id=sg-015f2500c51435243]
aws_security_group.allow_all_traffic: Creation complete after 5s [id=sg-0eb9c3aa19a0a9ce4]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

NOTE : Please note-down the IDs of the Security groups just created, as they are required for future configurations

STEP 06

Create Subnet Group

Steps :

1. Navigate to the **6. SubnetGroup** directory. Then run the *terraform init* command, in order to initialize **6. SubnetGroup** directory such that it contains appropriate Terraform configuration files for AWS.

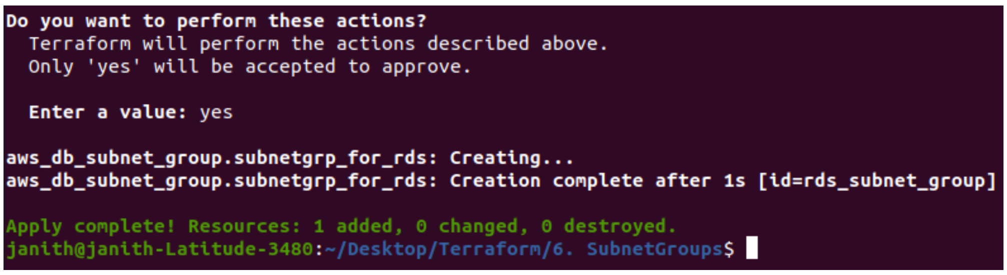
2. Do the necessary changes to the **subnet_grp.tf** file

Example :

```
   Defining region and the subnet group for the RDS instance to be deployed

variable "region"              { default = "ap-southeast-1" }
variable "private_subnet_id_1"     { default = "subnet-0590e5a5a9a9d2dbb" }
variable "private_subnet_id_2"     { default = "subnet-050fcc314238c0c65" }

   Defining the provider as AWS

provider "aws" {
  region = "${var.region}"
}

   Provisioning resource (Subnet Group)

resource "aws_db_subnet_group" "subnetgrp_for_rds" {
 name      = "rds_subnet_group"
 subnet_ids = [var.private_subnet_id_1, var.private_subnet_id_2]

 tags = {
  Name = "RDS_SubnetGroup"
 }
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Subnet Group.



```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_db_subnet_group.subnetgrp_for_rds: Creating...
aws_db_subnet_group.subnetgrp_for_rds: Creation complete after 1s [id=rds_subnet_group]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
janith@janith-Latitude-3480:~/Desktop/Terraform/6. SubnetGroups$
```

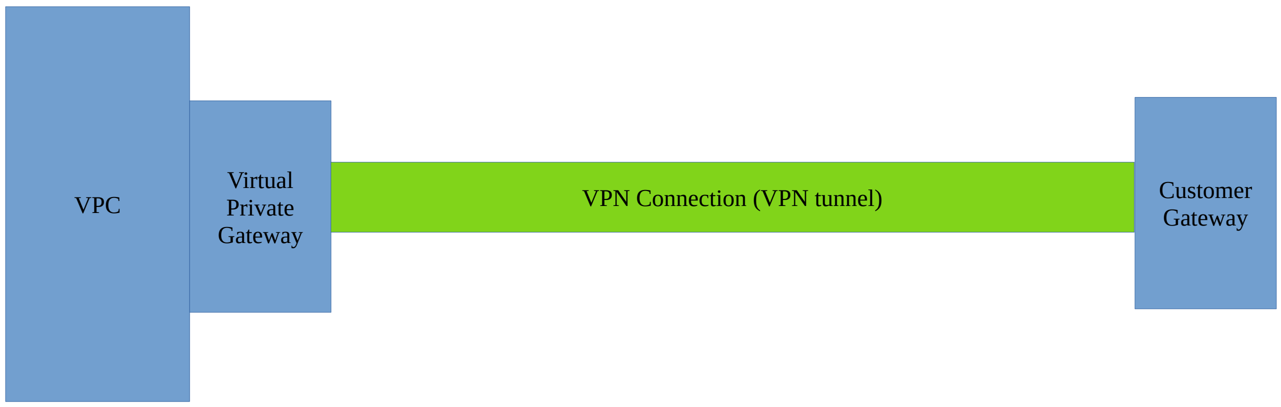
STEP 07

Create Customer gateway (For VPN connection)

In order to create the VPN tunnel, to access, there three main mandatory components. They are,

- Customer Gateway
- Virtual Private Gateway
- Site-to-site VPN connection

Those three components are organized according to the following illustration.



Steps :

1. Navigate to the **7. CustomerGateway** directory. Then run the *terraform init* command, in order to initialize **7. CustomerGateway** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **customer_gw.tf** file

```
  Defining region for the Customer Gateway to be deployed

variable "region" {default = "ap-southeast-1"}

  Defining the provider as AWS

provider "aws" {
 region = "${var.region}"
}

  Provisioning resource (Customer Gateway)
resource "aws_customer_gateway" "my_customer_gw" {
 bgp_asn    =
 ip_address = "54.23.69.4"
 type       = "ipsec.1"

 tags = {
  Name = "my_customer_gw"
 }
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Customer Gateway.

STEP 08

Create Virtual Private Gateway (For the VPN connection)

Steps :

1. Navigate to the **8. VirtualPrivateGateway** directory. Then run the *terraform init* command, in order to initialize **8. VirtualPrivateGateway** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **vpn_gw.tf** file

Example :

```
   Defining region for the Virtual Private Gateway to be deployed

variable "region" {default = "ap-southeast-1"}
variable "vpc_id" {default = "vpc-069ffe9976319a412"}

   Defining the provider as AWS

provider "aws" {
 region = "${var.region}"
}

   Provisioning resource (Virtual Private Gateway)

resource "aws_vpn_gateway" "my_vpn_gw" {
 vpc_id = var.vpc_id

 tags = {
  Name = "my_vpn_gw"
 }
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Virtual Private Gateway.

STEP 09

Create VPN Tunnel (For the VPN connection)

Steps :

1. Navigate to the **9. VPNConnection** directory. Then run the *terraform init* command, in order to initialize **9. VPNConnection** directory such that it contains appropriate Terraform configuration files for AWS.

2. Do the necessary changes to the **vpn.tf** file

Example :

```
   Defining region, customer gateway ID and vpn gateway ID for the VPN tunnel to be deployed

variable "region" {default = "ap-southeast-1"}
variable "vpn_gw_id" {default = "vgw-0b7997108fcfa4b10"}
variable "customer_gw_id" {default = "cgw-0e662426a26fba0cb"}

   Defining the provider as AWS

provider "aws" {
  region = "${var.region}"
}

   Provisioning resource (VPN Connection)

resource "aws_vpn_connection" "my_vpn_connection" {
  vpn_gateway_id      = var.vpn_gw_id
  customer_gateway_id = var.customer_gw_id
  type                = "ipsec.1"
  static_routes_only  = true
}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Virtual Private Network connection.

STEP 09

Launching Web application (Elastic Beanstalk)

Steps :

1. Navigate to the **ElasticBeanstalk** directory. Then run the *terraform init* command, in order to initialize **ElasticBeanstalk** directory such that it contains appropriate Terraform configuration files for AWS.

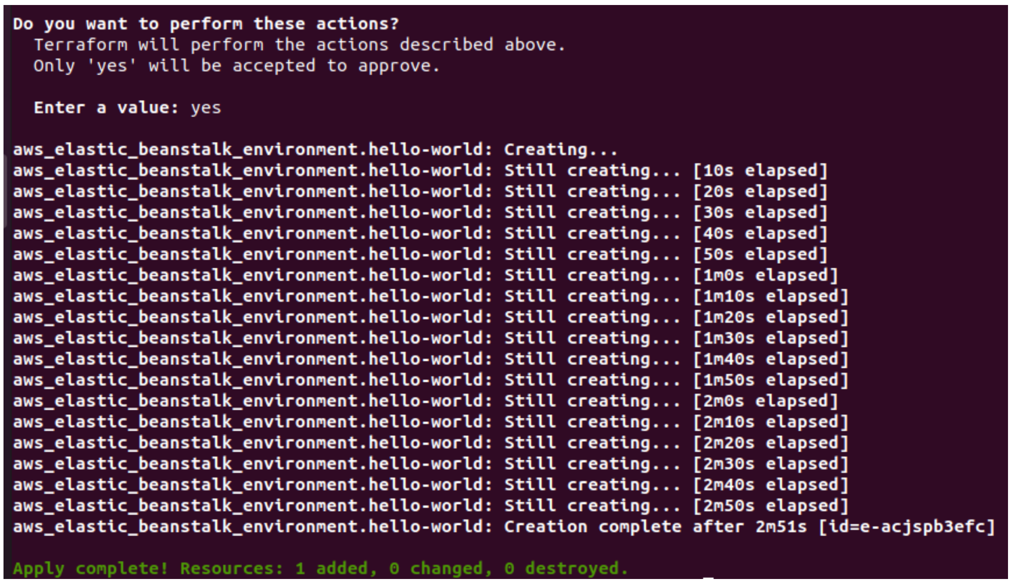
2. Do the necessary changes to the **eb.tf** file

Example :

```
   Defining region, VPC ID and Public subnet ID for the Web application to be deployed
variable "vpc_id" {default = "vpc-069ffe9976319a412"}
variable "region" {default = "ap-southeast-1"}
variable "public_subnet" {default = "subnet-0f984d87a7d07b735"}


   Defining the provider as AWS
provider "aws" {
 region = "${var.region}"
}


   Provisioning resources
resource "aws_elastic_beanstalk_application" "hello-world" {
 name        = "hello-world"
 description = "Test of beanstalk deployment"
}

resource "aws_elastic_beanstalk_environment" "hello-world" {
 name = "hello-world"
 application = "hello-world"
 solution_stack_name = "64bit Amazon Linux 2 v3.0.3 running Docker"
 cname_prefix = "hello-world"

 setting {
  namespace = "aws:autoscaling:launchconfiguration"
  name = "IamInstanceProfile"
  value = "aws-elasticbeanstalk-ec2-role"
 }
 setting {
  namespace = "aws:ec2:vpc"
  name = "VPCId"
  value = "${var.vpc_id}"
 }
 setting {
  namespace = "aws:ec2:vpc"
  name = "AssociatePublicIpAddress"
  value = "true"
 }
 setting {
  namespace = "aws:ec2:vpc"
  name = "Subnets"
  value = "${var.public_subnet}"
 }
 setting {
   namespace = "aws:autoscaling:launchconfiguration"
   name = "InstanceType"
   value = "t2.micro"
 }


}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.

4. Finally execute *terraform apply* command to provision the Elastic Beanstalk Environment.



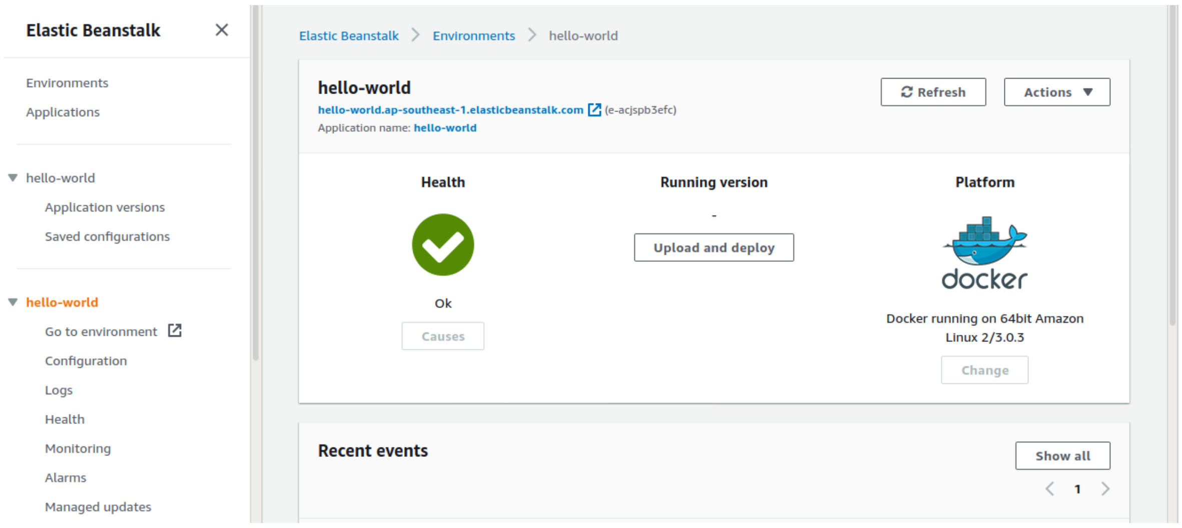
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_elastic_beanstalk_environment.hello-world: Creating...
aws_elastic_beanstalk_environment.hello-world: Still creating... [10s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [20s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [30s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [40s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [50s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m0s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m10s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m20s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m30s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m40s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [1m50s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m0s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m10s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m20s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m30s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m40s elapsed]
aws_elastic_beanstalk_environment.hello-world: Still creating... [2m50s elapsed]
aws_elastic_beanstalk_environment.hello-world: Creation complete after 2m51s [id=e-acjspb3efc]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Next step is to launch the Docker image (as an Elastic Beanstalk Application). For that follow the below steps.

1. Login to the AWS console

2. Navigate to **Elastic Beanstalk** service (Services → Elastic Beanstalk)

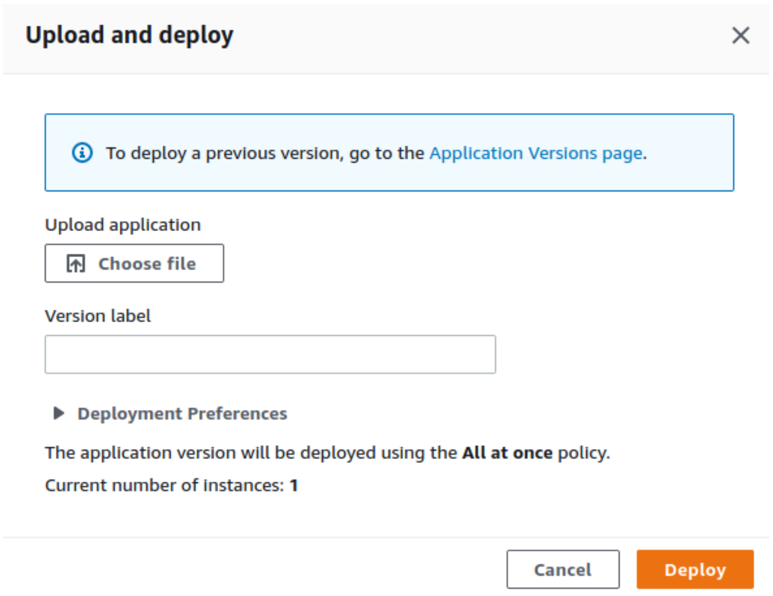
3. Click **Environments** from the Navigation pane of the Elastic Beanstalk (Newly created hello-world should be visible there. Just lick on it)



hello-world
hello-world.ap-southeast-1.elasticbeanstalk.com (e-acjspb3efc)
Application name: hello-world

Health: Ok
Running version: -
Platform: Docker running on 64bit Amazon Linux 2/3.0.3

4. Click on **Upload and Deploy** button



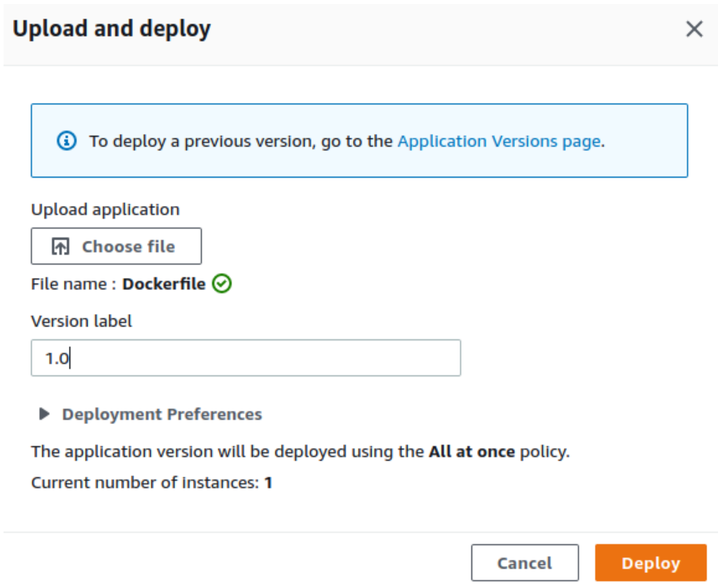
To deploy a previous version, go to the Application Versions page.

Upload application

Version label

The application version will be deployed using the **All at once** policy.
Current number of instances: **1**

5. Click on choose file. The navigate and select the Dockerfile of the application. After providing the version of the application of the application click deploy



To deploy a previous version, go to the Application Versions page.

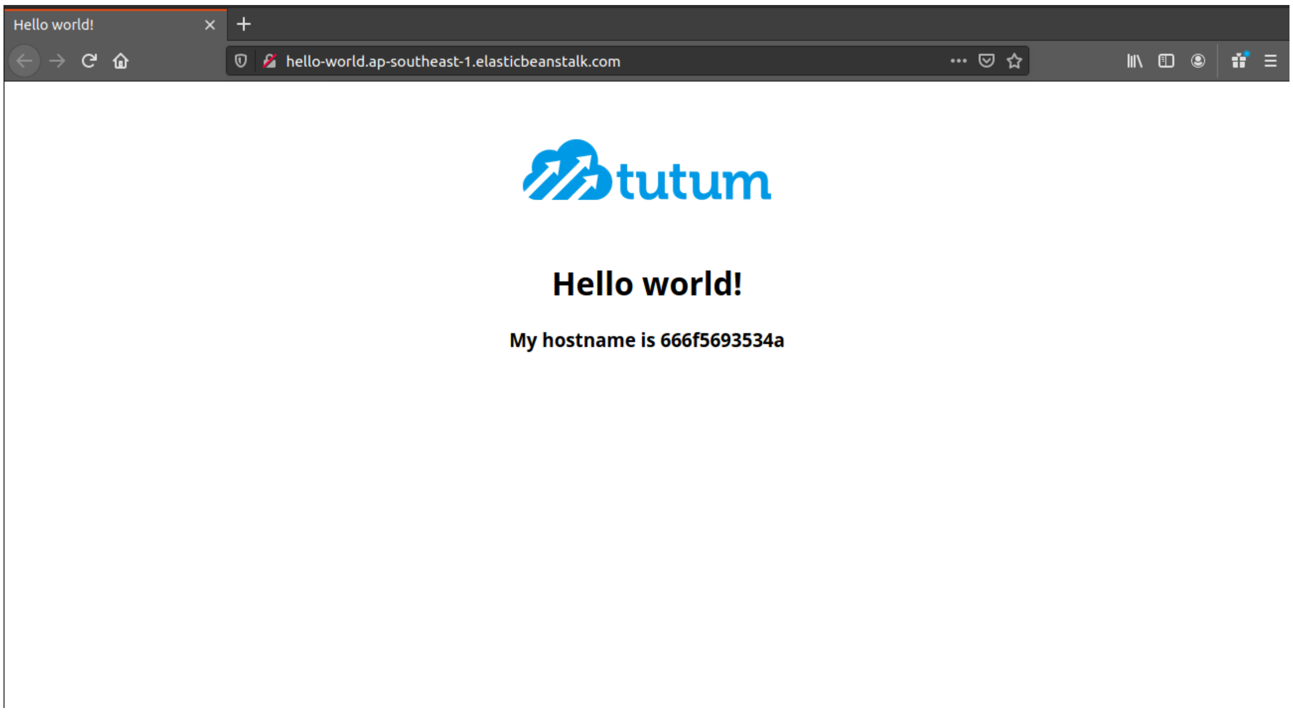
Upload application

File name : **Dockerfile**

Version label: 1.0

The application version will be deployed using the **All at once** policy.
Current number of instances: **1**

Elastic Beanstalk environment is going to be updated. After several minutes, the application can be accessed through hello-world.ap-southeast-1.elasticbeanstalk.com



Hello world!

My hostname is 666f5693534a

STEP 10

Provisioning Database as a Service MySQL instance (Amazon RDS for MySQL instance)

Steps :

1. Navigate to the ./RDS directory. Then run the *terraform init* command, in order to initialize **./ RDS** directory such that it contains appropriate Terraform configuration files for AWS

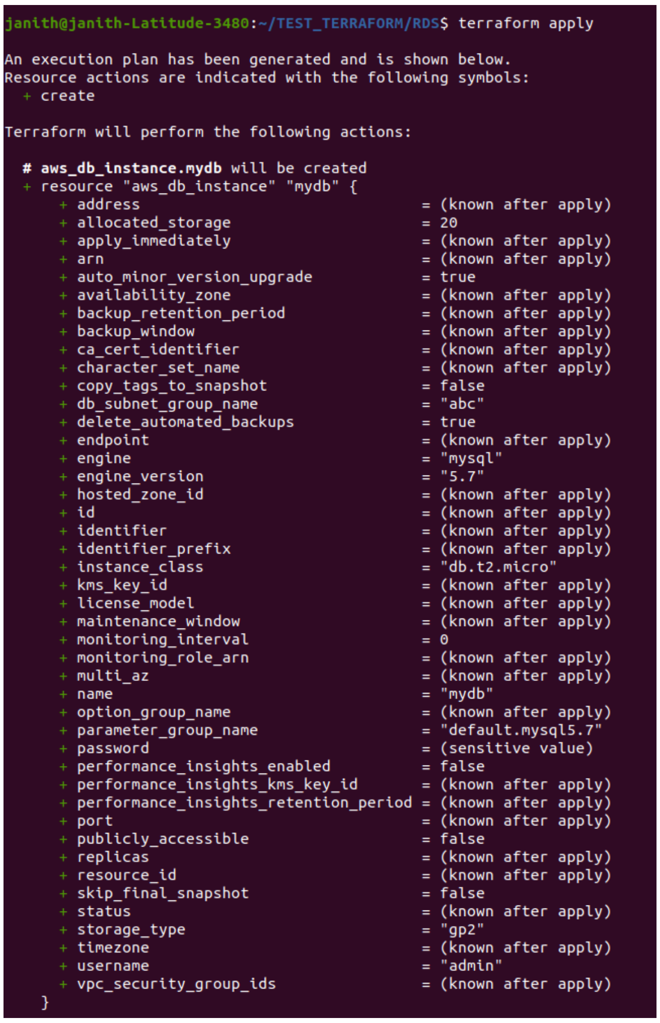
2. Do the necessary changes to the fields such as **username, password, region** and the **db_subnet_group_name** in the **rds.tf** file.

Example :

```
#Defining region and the subnet group for the RDS instance to be deployed

variable "region" { default = "ap-southeast-1" }
variable "subnet_group_name" { default = "abc" }

#Defining the provider as AWS

provider "aws" {
region = "${var.region}"
}

#Provisioning the resource (RDS)

resource "aws_db_instance" "mydb" {
```

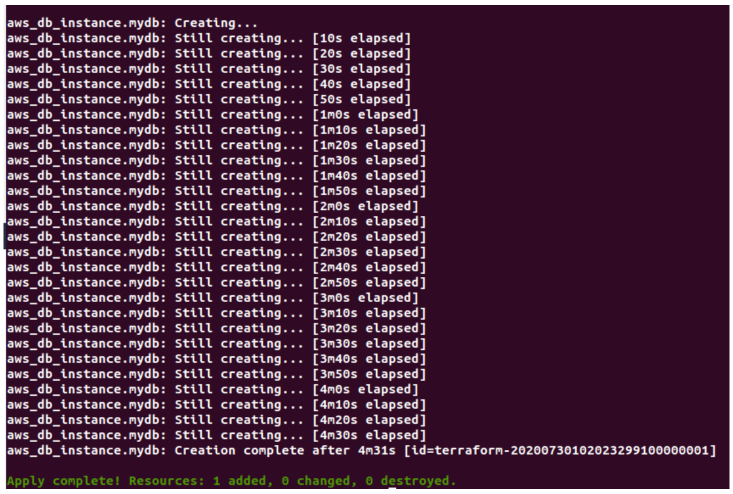
```
allocated_storage = 20
storage_type = "gp2"
engine = "mysql"
engine_version = "5.7"
instance_class = "db.t2.micro"
name = "mydb"
username = "admin"
password = "&8Hwg($soQ=L"
parameter_group_name = "default.mysql5.7"
db_subnet_group_name = "${var.subnet_group_name}"

}
```

3. Execute *terraform plan* command to make sure what are the changes which are going to be done to the system.



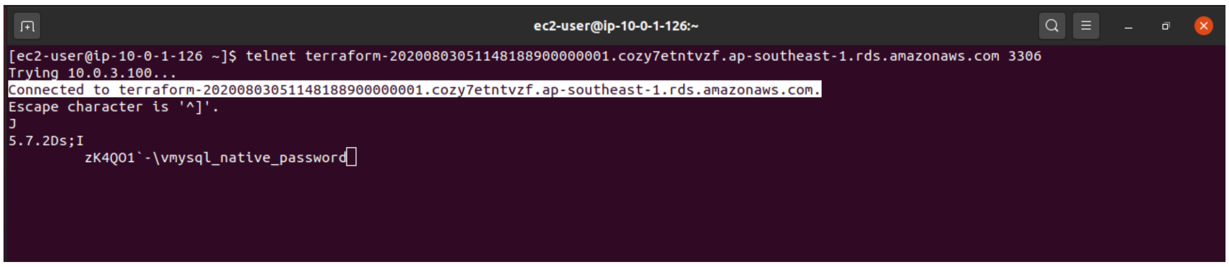
```
janith@janith-Latitude-3480:~/TEST_TERRAFORM/RDS$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_db_instance.mydb will be created
  + resource "aws_db_instance" "mydb" {
      + address                               = (known after apply)
      + allocated_storage                     = 20
      + apply_immediately                     = (known after apply)
      + arn                                   = (known after apply)
      + auto_minor_version_upgrade            = true
      + availability_zone                     = (known after apply)
      + backup_retention_period               = (known after apply)
      + backup_window                         = (known after apply)
      + ca_cert_identifier                    = (known after apply)
      + character_set_name                    = (known after apply)
      + copy_tags_to_snapshot                 = false
      + db_subnet_group_name                  = "abc"
      + delete_automated_backups              = true
      + endpoint                              = (known after apply)
      + engine                                = "mysql"
      + engine_version                        = "5.7"
      + hosted_zone_id                        = (known after apply)
      + id                                    = (known after apply)
      + identifier                            = (known after apply)
      + identifier_prefix                     = (known after apply)
      + instance_class                        = "db.t2.micro"
      + kms_key_id                            = (known after apply)
      + license_model                         = (known after apply)
      + maintenance_window                    = (known after apply)
      + monitoring_interval                   = 0
      + monitoring_role_arn                   = (known after apply)
      + multi_az                              = (known after apply)
      + name                                  = "mydb"
      + option_group_name                     = (known after apply)
      + parameter_group_name                  = "default.mysql5.7"
      + password                              = (sensitive value)
      + performance_insights_enabled          = false
      + performance_insights_kms_key_id       = (known after apply)
      + performance_insights_retention_period = (known after apply)
      + port                                  = (known after apply)
      + publicly_accessible                   = false
      + replicas                              = (known after apply)
      + resource_id                           = (known after apply)
      + skip_final_snapshot                   = false
      + status                                = (known after apply)
      + storage_type                          = "gp2"
      + timezone                              = (known after apply)
      + username                              = "admin"
      + vpc_security_group_ids                = (known after apply)
    }
```

4. Finally execute *terraform apply* command to provision the RDS instance. If the following message pops up in the terminal, Congratulations !! The RDS instance is provisioned successfully.



```
aws_db_instance.mydb: Creating...
aws_db_instance.mydb: Still creating... [10s elapsed]
aws_db_instance.mydb: Still creating... [20s elapsed]
aws_db_instance.mydb: Still creating... [30s elapsed]
aws_db_instance.mydb: Still creating... [40s elapsed]
aws_db_instance.mydb: Still creating... [50s elapsed]
aws_db_instance.mydb: Still creating... [1m0s elapsed]
aws_db_instance.mydb: Still creating... [1m10s elapsed]
aws_db_instance.mydb: Still creating... [1m20s elapsed]
aws_db_instance.mydb: Still creating... [1m30s elapsed]
aws_db_instance.mydb: Still creating... [1m40s elapsed]
aws_db_instance.mydb: Still creating... [1m50s elapsed]
aws_db_instance.mydb: Still creating... [2m0s elapsed]
aws_db_instance.mydb: Still creating... [2m10s elapsed]
aws_db_instance.mydb: Still creating... [2m20s elapsed]
aws_db_instance.mydb: Still creating... [2m30s elapsed]
aws_db_instance.mydb: Still creating... [2m40s elapsed]
aws_db_instance.mydb: Still creating... [2m50s elapsed]
aws_db_instance.mydb: Still creating... [3m0s elapsed]
aws_db_instance.mydb: Still creating... [3m10s elapsed]
aws_db_instance.mydb: Still creating... [3m20s elapsed]
aws_db_instance.mydb: Still creating... [3m30s elapsed]
aws_db_instance.mydb: Still creating... [3m40s elapsed]
aws_db_instance.mydb: Still creating... [3m50s elapsed]
aws_db_instance.mydb: Still creating... [4m0s elapsed]
aws_db_instance.mydb: Still creating... [4m10s elapsed]
aws_db_instance.mydb: Still creating... [4m20s elapsed]
aws_db_instance.mydb: Still creating... [4m30s elapsed]
aws_db_instance.mydb: Creation complete after 4m31s [id=terraform-20200730102023299100000001]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Accessing RDS instance from web application :



```
[ec2-user@ip-10-0-1-126 ~]$ telnet terraform-20200803051148188900000001.cozy7etntvzf.ap-southeast-1.rds.amazonaws.com 3306
Trying 10.0.3.100...
Connected to terraform-20200803051148188900000001.cozy7etntvzf.ap-southeast-1.rds.amazonaws.com.
Escape character is '^]'.
J
5.7.2Ds;I
         zK4QO1`-\vmysql_native_password
```