**Challenge #1**

A 3-tier environment is a common setup. Use a tool of your choosing/familiarity create these resources on a cloud environment (**AWS Cloud**).

Solution:-

This can be accomplished by using AWS and terraform

We can follow below steps to create common setup :

1. Create a Virtual Private Cloud (VPC) network for the resources in the 3-tier environment.
2. Create a firewall rule to allow incoming traffic to the web tier
3. Create a managed instance group for the web tier and configure it with a startup script to install a web server and configure it with a basic website. Similarly we need to do the above steps for Application tier and Database tier.
4. Create a load balancer to distribute incoming traffic to the web tier
5. Create an instance template for each tier and attach it to the corresponding managed instance group.
6. Create a managed disk and attach it to each instance in the database tier for persistence of data.

Terraform------------

1. Create a directory for this Terraform project.
2. Inside the new directory create a file named **install\_apache.sh**and use the below code. This code will install an Apache webserver on our instances

|  |  |
| --- | --- |
|  | #!/bin/bash |
|  |  |
|  | yum update -y |
|  | Yum, install -y httpd.x86\_64 |
|  | systemctl start httpd.service |
|  | systemctl enable httpd.service |
|  | echo "Hello World from $(hostname -f)" > /var/www/html/index.html |

Declare our **Provider** as AWS.

1. Create a **main.tf** file.
2. From the terminal in the Terraform directory containing install\_apache.sh and main.tf run terraform init
3. Use the below code to set our **Provider** to AWS and set our Region to us-east-1.

**Filename: three-tier-providers.tf**

|  |
| --- |
| terraform { |
|  | required\_providers { |
|  | aws = { |
|  | source = "hashicorp/aws" |
|  | version = "~> 3.0" |
|  | } |
|  | } |
|  | } |
|  |  |
|  | # Configure the AWS Provider |
|  | provider "aws" { |
|  | region = "us-east-1" |
|  | } |

**Create VPC and Subnets**

1. Our first Resource is creating our VPC with CIDR 10.0.0.0/16.
2. **web-subnet-1** and **web-subnet-2** resources create our web layer in two availability zones. Notice that we have map\_public\_ip\_on\_launch = true
3. **application-subnet-1** and **application-subnet-2** resources create our application layer in two availability zones. This will be a private subnet.
4. **database-subnet-1** and **database-subnet-2** resources create our database layer in two availability zones. This will be a private subnet.

**three-tier-vpc-subnets.tf**

|  |
| --- |
|  |
|  |
|  | resource "aws\_vpc" "my-vpc" { |
|  | cidr\_block = "10.0.0.0/16" |
|  | tags = { |
|  | Name = "Demo VPC" |
|  | } |
|  | } |
|  |  |
|  | # Create Web Public Subnet |
|  | resource "aws\_subnet" "web-subnet-1" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.1.0/24" |
|  | availability\_zone = "us-east-1a" |
|  | map\_public\_ip\_on\_launch = true |
|  |  |
|  | tags = { |
|  | Name = "Web-1a" |
|  | } |
|  | } |
|  |  |
|  | resource "aws\_subnet" "web-subnet-2" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.2.0/24" |
|  | availability\_zone = "us-east-1b" |
|  | map\_public\_ip\_on\_launch = true |
|  |  |
|  | tags = { |
|  | Name = "Web-2b" |
|  | } |
|  | } |
|  |  |
|  | # Create Application Private Subnet |
|  | resource "aws\_subnet" "application-subnet-1" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.11.0/24" |
|  | availability\_zone = "us-east-1a" |
|  | map\_public\_ip\_on\_launch = false |
|  |  |
|  | tags = { |
|  | Name = "Application-1a" |
|  | } |
|  | } |
|  |  |
|  | resource "aws\_subnet" "application-subnet-2" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.12.0/24" |
|  | availability\_zone = "us-east-1b" |
|  | map\_public\_ip\_on\_launch = false |
|  |  |
|  | tags = { |
|  | Name = "Application-2b" |
|  | } |
|  | } |
|  |  |
|  | # Create Database Private Subnet |
|  | resource "aws\_subnet" "database-subnet-1" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.21.0/24" |
|  | availability\_zone = "us-east-1a" |
|  |  |
|  | tags = { |
|  | Name = "Database-1a" |
|  | } |
|  | } |
|  |  |
|  | resource "aws\_subnet" "database-subnet-2" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | cidr\_block = "10.0.22.0/24" |
|  | availability\_zone = "us-east-1b" |
|  |  |
|  | tags = { |
|  | Name = "Database-2b" |
|  | } |
|  | } |

**Create Internet Gateway and Route Table**

1. Our first resource block will create an **Internet Gateway**. We will need an **Internet Gateway** to allow our public subnets to connect to the Internet.
2. The **web-rt** route table creates a route in our **VPC** to our **Internet Gateway** for CIDR 0.0.0.0/0.
3. The next two blocks are associating **web-subnet-1** and **web-subnet-2** with the **web-rt**route table.

**Filename: three-tier-in-rt.tf**

|  |
| --- |
|  |
|  | resource "aws\_internet\_gateway" "igw" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  |  |
|  | tags = { |
|  | Name = "Demo IGW" |
|  | } |
|  | } |
|  |  |
|  | # Create Web layer route table |
|  | resource "aws\_route\_table" "web-rt" { |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  |  |
|  |  |
|  | route { |
|  | cidr\_block = "0.0.0.0/0" |
|  | gateway\_id = aws\_internet\_gateway.igw.id |
|  | } |
|  |  |
|  | tags = { |
|  | Name = "WebRT" |
|  | } |
|  | } |
|  |  |
|  | # Create Web Subnet association with Web route table |
|  | resource "aws\_route\_table\_association" "a" { |
|  | subnet\_id = aws\_subnet.web-subnet-1.id |
|  | route\_table\_id = aws\_route\_table.web-rt.id |
|  | } |
|  |  |
|  | resource "aws\_route\_table\_association" "b" { |
|  | subnet\_id = aws\_subnet.web-subnet-2.id |
|  | route\_table\_id = aws\_route\_table.web-rt.id |

**Create Web Servers**

1. **webserver1** resource creates a Linux 2 EC2 instance in the us-east-1a Availability Zone.
2. **ami** is set to the AMI id for the Linux 2 AMI for the us-east-1 Region.
3. **vpc\_security\_group\_ids** are set to a not yet created Security Group, which will be created in the next section for our Application Load Balancer.
4. **user\_data** is used to bootstrap our instance. we will reference the **install\_apache.sh** file we created earlier.
5. **webserver2** is almost identical except that availability\_zone is set to us-east-1b.

**Filename: three-tier-webservers.tf**

|  |
| --- |
| #Create EC2 Instance |
|  | resource "aws\_instance" "webserver1" { |
|  | ami = "ami-0d5eff06f840b45e9" |
|  | instance\_type = "t2.micro" |
|  | availability\_zone = "us-east-1a" |
|  | vpc\_security\_group\_ids = [aws\_security\_group.webserver-sg.id] |
|  | subnet\_id = aws\_subnet.web-subnet-1.id |
|  | user\_data = file("install\_apache.sh") |
|  |  |
|  | tags = { |
|  | Name = "Web Server" |
|  | } |
|  |  |
|  | } |
|  |  |
|  | resource "aws\_instance" "webserver2" { |
|  | ami = "ami-0d5eff06f840b45e9" |
|  | instance\_type = "t2.micro" |
|  | availability\_zone = "us-east-1b" |
|  | vpc\_security\_group\_ids = [aws\_security\_group.webserver-sg.id] |
|  | subnet\_id = aws\_subnet.web-subnet-2.id |
|  | user\_data = file("install\_apache.sh") |
|  |  |
|  | tags = { |
|  | Name = "Web Server" |
|  | } |
|  |  |
|  | } |

**Create Security Groups**

1. Create a Security Group named **web-sg** with an inbound rule opening **HTTP port 80** to **CIDR 0.0.0.0/0** and allowing all outbound traffic.
2. Create a Security Group named **webserver-sg** with an inbound rule opening **HTTP port 80**, but this time it’s not open to the world. Instead, we are only allowing traffic from our **web-sg** Security Group.
3. Create a Security Group named **database-sg** with inbound rule opening **MySQL port 3306** and once again we keep security tight by only allowing the inbound traffic from the **webserver-sg** Security Group. We open outbound traffic to all the ports.

**Filename: three-tier-sg.tf**

|  |
| --- |
|  |
|  | resource "aws\_security\_group" "web-sg" { |
|  | name = "Web-SG" |
|  | description = "Allow HTTP inbound traffic" |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  |  |
|  | ingress { |
|  | description = "HTTP from VPC" |
|  | from\_port = 80 |
|  | to\_port = 80 |
|  | 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"] |
|  | } |
|  |  |
|  | tags = { |
|  | Name = "Web-SG" |
|  | } |
|  | } |
|  |  |
|  | # Create Web Server Security Group |
|  | resource "aws\_security\_group" "webserver-sg" { |
|  | name = "Webserver-SG" |
|  | description = "Allow inbound traffic from ALB" |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  |  |
|  | ingress { |
|  | description = "Allow traffic from web layer" |
|  | from\_port = 80 |
|  | to\_port = 80 |
|  | protocol = "tcp" |
|  | security\_groups = [aws\_security\_group.web-sg.id] |
|  | } |
|  |  |
|  | egress { |
|  | from\_port = 0 |
|  | to\_port = 0 |
|  | protocol = "-1" |
|  | cidr\_blocks = ["0.0.0.0/0"] |
|  | } |
|  |  |
|  | tags = { |
|  | Name = "Webserver-SG" |
|  | } |
|  | } |
|  |  |
|  | # Create Database Security Group |
|  | resource "aws\_security\_group" "database-sg" { |
|  | name = "Database-SG" |
|  | description = "Allow inbound traffic from application layer" |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  |  |
|  | ingress { |
|  | description = "Allow traffic from application layer" |
|  | from\_port = 3306 |
|  | to\_port = 3306 |
|  | protocol = "tcp" |
|  | security\_groups = [aws\_security\_group.webserver-sg.id] |
|  | } |
|  |  |
|  | egress { |
|  | from\_port = 32768 |
|  | to\_port = 65535 |
|  | protocol = "tcp" |
|  | cidr\_blocks = ["0.0.0.0/0"] |
|  | } |
|  |  |
|  | tags = { |
|  | Name = "Database-SG" |
|  | } |
|  | } |

* **Create Application Load Balancer**

1.Create an external **Application Load Balancer**.

2. Create an Application Load Balancer Target Group.

3. The **aws\_lib\_target\_group\_attachment** Resource attaches our instances to the **Target Group**.

4. Add a listener on port 80 that forwards traffic to our **Target Group**.

**Filename: three-tier-elb.tf**

|  |
| --- |
| resource "aws\_lb" "external-elb" { |
|  | name = "External-LB" |
|  | internal = false |
|  | load\_balancer\_type = "application" |
|  | security\_groups = [aws\_security\_group.web-sg.id] |
|  | subnets = [aws\_subnet.web-subnet-1.id, aws\_subnet.web-subnet-2.id] |
|  | } |
|  |  |
|  | resource "aws\_lb\_target\_group" "external-elb" { |
|  | name = "ALB-TG" |
|  | port = 80 |
|  | protocol = "HTTP" |
|  | vpc\_id = aws\_vpc.my-vpc.id |
|  | } |
|  |  |
|  | resource "aws\_lb\_target\_group\_attachment" "external-elb1" { |
|  | target\_group\_arn = aws\_lb\_target\_group.external-elb.arn |
|  | target\_id = aws\_instance.webserver1.id |
|  | port = 80 |
|  |  |
|  | depends\_on = [ |
|  | aws\_instance.webserver1, |
|  | ] |
|  | } |
|  |  |
|  | resource "aws\_lb\_target\_group\_attachment" "external-elb2" { |
|  | target\_group\_arn = aws\_lb\_target\_group.external-elb.arn |
|  | target\_id = aws\_instance.webserver2.id |
|  | port = 80 |
|  |  |
|  | depends\_on = [ |
|  | aws\_instance.webserver2, |
|  | ] |
|  | } |
|  |  |
|  | resource "aws\_lb\_listener" "external-elb" { |
|  | load\_balancer\_arn = aws\_lb.external-elb.arn |
|  | port = "80" |
|  | protocol = "HTTP" |
|  |  |
|  | default\_action { |
|  | type = "forward" |
|  | target\_group\_arn = aws\_lb\_target\_group.external-elb.arn |
|  | } |
|  | } |

**Create RDS Instance**

1. Create a MySQL RDS Instance. Some attributes to note:

* **db\_subnet\_group\_name** is a required field and is set to the **aws\_db\_subnet\_group.default**.
* **instance\_class** is set to a DB.t2.micro.
* **multi\_az** is set to true for high availability, but if you’d like to keep costs low, set this to false.
* **username** & **password** will need to be changed.
* **vpc\_secuiryt\_group\_ids** is set to our database-sg Security Group.

1. Create a DB Subnet Group. **subnet\_ids** identify which subnets will be used by the Database.

**Filename: view rawthree-tier-rds.tf**

|  |
| --- |
| resource "aws\_db\_instance" "default" { |
|  | allocated\_storage = 10 |
|  | db\_subnet\_group\_name = aws\_db\_subnet\_group.default.id |
|  | engine = "mysql" |
|  | engine\_version = "8.0.20" |
|  | instance\_class = "db.t2.micro" |
|  | multi\_az = true |
|  | name = "mydb" |
|  | username = "username" |
|  | password = "password" |
|  | skip\_final\_snapshot = true |
|  | vpc\_security\_group\_ids = [aws\_security\_group.database-sg.id] |
|  | } |
|  |  |
|  | resource "aws\_db\_subnet\_group" "default" { |
|  | name = "main" |
|  | subnet\_ids = [aws\_subnet.database-subnet-1.id, aws\_subnet.database-subnet-2.id] |
|  |  |
|  | tags = { |
|  | Name = "My DB subnet group" |
|  | } |
|  | } |

**Output**

After your infrastructure completes, Output will print out the requested values.

1. We will use the output to print out our ALB DNS so we can test our web servers.

|  |
| --- |
| Output "lb\_dns\_name" { |
|  | description = "The DNS name of the load balancer" |
|  | value = aws\_lb.external-elb.dns\_name |
|  | } |

**Provision Infrastructure**

1. If you didn’t do so earlier or you just want to do it again, from the terminal run terraform init.
2. Run terraform fmt. This ensures your formatting is correct and will modify the code for you to match.
3. Run terraform validation to ensure there are no syntax errors.
4. Run terraform plan to see what resources will be created.
5. Run terraform apply to create your infrastructure. Type **Yes**when prompted.
6. To Delete infrastructure run- terraform destroy.

**Testing**

1. After your infrastructure has been created there should be an Output displayed on your terminal for the Application Load Balancer DNS Name.
2. Copy and paste (without quotations) into a new browser tab. Refresh the page to see the load balancer switch between the two instances.