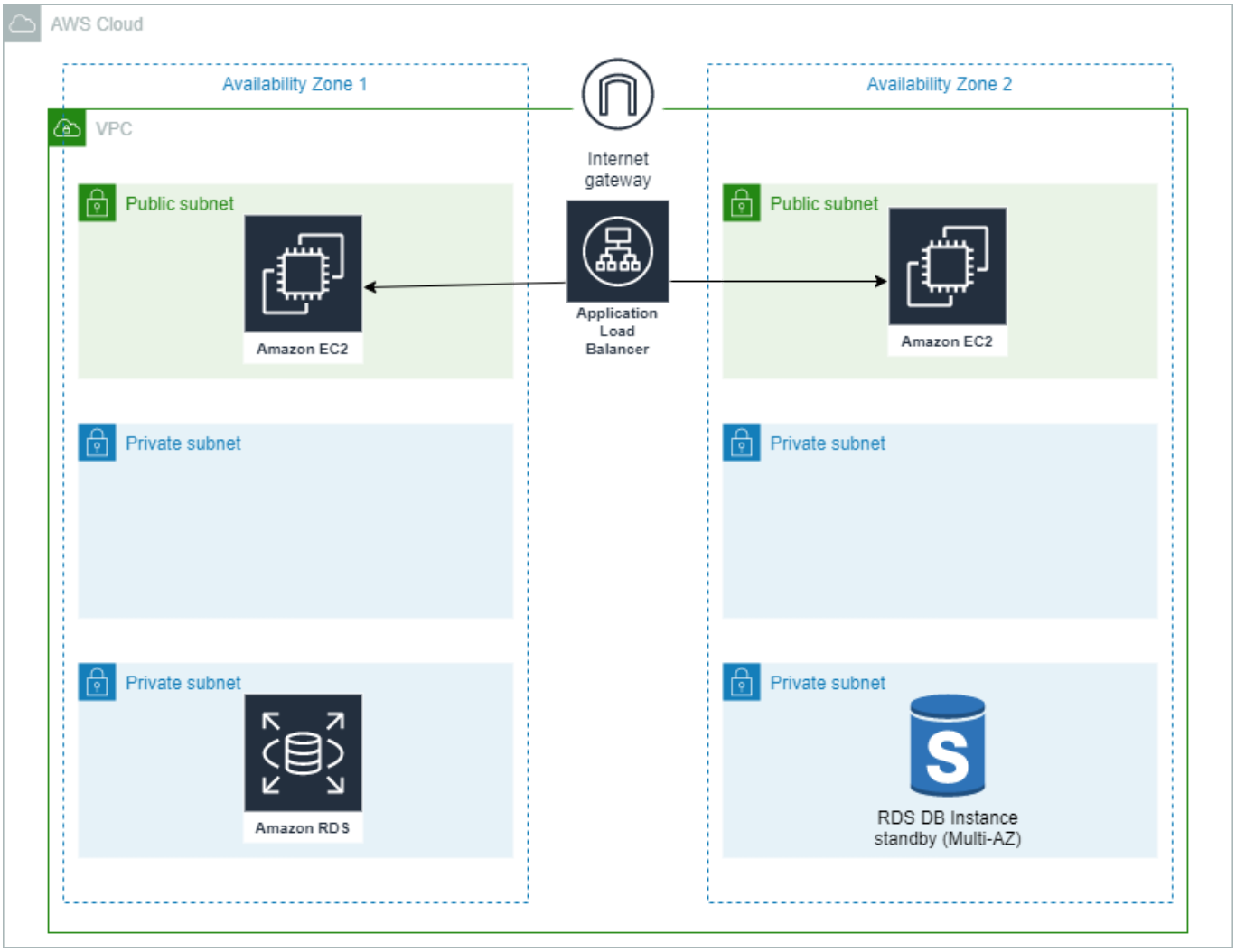
Terraform: Deploy A Three-Tier Architecture in AWS

Troy Ingram Following

Jun 2 · 6 min read



Infrastructure as Code (IaC)

The cloud gives us the ability to create our environments quickly, but the problem that arises is how to configure and manage the environments. Manually updating from the console may be acceptable for a small organization in a single region, but what if you have to create and maintain environments in multiple regions? Not only is it an inefficient use of time to create and maintain everything, but it's also error-prone.

Imagine that you are asked to create an environment in a single Region. Not really a big deal and you are able to complete the task with relative ease. Now you need to do the

exact same thing for five more Regions. Not only that, once you have completed the excruciatingly repetitive task, your leadership asks you to make a change that then needs to be applied to all Regions. This example is inefficiency at its finest.

Think about infrastructure as code as a scalable blueprint for your environment. It allows you to provision and configure your environments in a reliable and safe way. By using code to deploy your infrastructure you gain the ability to use the same tools as developers such as version control, testing, code reviews, and CI/CD.

Terraform

HashiCorp Terraform is a tool for building, changing, and versioning infrastructure that has an open-source and enterprise version. Terraform is cloud agnostic and can be used to create multi-cloud infrastructure. It allows IaC in a human readable language called HashiCorp Configuration Language (HCL).

Prerequisites

- Install Terraform
- Install the AWS CLI
- Sign up for an AWS Account
- Your preferred IDE (I used Visual Studio Code)

Notes Before Getting Started

- I'll be going through sections of my main.tf Terraform file, but if you'd just like to get to the full code, skip to the end or visit my GitHub.
- This Terraform file does not follow best practice of DRY (Don't Repeat Yourself) code. The best practice would be to use variables and modules rather than using a single file and hard coding.
- I've just started using Terraform and in a future post will update this file to incorporate variables and modules.
- A typical three-tier architecture uses a web, application and database layer. In this project, although I've created an application layer, I'm not deploying any instances in the application layer and have not created a Security Group for the application layer. If you decide to do so, you will need to modify some Security Group rules and create an application layer security group.
- RDS has Multi-AZ set to true for high availability. If you'd like to save money be sure to set this to false.

The Future Updates As Promised

- Terraform: Using Variables and Count

Alright, let's get started.

Website Script

1. Create a new 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 web server on our instances and create a unique landing page for each so we can verify the Application Load Balancer is working.

Configure Provider

Providers are plugins that Terraform requires so that it can install and use for your Terraform configuration. Terraform offers the ability to use a variety of **Providers**, so it

doesn't make sense to use all of them for each file. We will declare our **Provider** as AWS.

1. Create a **main.tf** file and add each of the following sections to the **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.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}
```

three-tier-providers.tf hosted with ♡ by GitHub view raw

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.

```
# Create a VPC
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"
  }
}
```

three-tier-vpc-subnets.tf hosted with ♡ by GitHub view raw

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. Just saying that our subnets are public does not make it so. We will need to create a route table and Associate our Web Layer subnets.

3. The **web-rt** route table creates a route in our **VPC** to our **Internet Gateway** for CIDR 0.0.0.0/0.

4. The next two blocks are associating **web-subnet-1** and **web-subnet-2** with the **web-rt** route table. I feel like you should be able to associate more than one subnet in a single block but I kept getting errors and wasn't able to find any helpful documentation.

Note: We only create one new public route table and do not need to create a private route table. By default all subnets are associated with the default route table which is set to private by default.

```
# Create Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.my-vpc.id

  tags = {
    Name = "Demo IGW"
  }
}

# Create Web layber 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
  }
```

three-tier-igw-rt.tf hosted with ♡ by GitHub view raw

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. If using a different Region then you'd need to update.
3. **vpc_security_group_ids** is set to a not yet created Security Group, which will be created in the next section for our Application Load Balancer. Terraform doesn't create infrastructure in order. It is smart enough to know what needs to be created before others (for the most part, I'll talk more about this later).
4. **user_data** is used to boot strap our instance. Rather than type our the code directly, 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.

```
  #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")
```

```
9
10    tags = {
11      Name = "Web Server"
12    }
13
14  }
15
16  resource "aws_instance" "webserver2" {
17    ami                    = "ami-0d5eff06f840b45e9"
18    instance_type          = "t2.micro"
19    availability_zone      = "us-east-1b"
20    vpc_security_group_ids = [aws_security_group.webserver-sg.id]
21    subnet_id              = aws_subnet.web-subnet-2.id
22    user_data              = file("install_apache.sh")
23
24    tags = {
25      Name = "Web Server"
26    }
27
28  }
```

three-tier-webservers.tf hosted with ♡ by GitHub view raw

Create Security Groups

1. Create a Security Group named **web-sg** with 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 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 allow the inbound traffic from the **webserver-sg** Security Group. We open outbound traffic to all the ephemeral ports.

```
1   # Create Web Security Group
2   resource "aws_security_group" "web-sg" {
3     name        = "Web-SG"
4     description = "Allow HTTP inbound traffic"
5     vpc_id      = aws_vpc.my-vpc.id
6
7     ingress {
8       description = "HTTP from VPC"
9       from_port   = 80
10      to_port     = 80
11      protocol    = "tcp"
12      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"
```

```
64      security_groups = [aws_security_group.webserver-sg.id]
65    }
66
67    egress {
68      from_port   = 32768
69      to_port     = 65535
70      protocol    = "tcp"
71      cidr_blocks = ["0.0.0.0/0"]
72    }
73
74    tags = {
75      Name = "Database-SG"
76    }
77  }
```

three-tier-sg.tf hosted with ♡ by GitHub view raw

Create Application Load Balancer

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

- **internal** is set to false, making it an external Load Balancer.
- **load_balancer_type** is set to application designating it an Application Load Balancer.
- **security_groups** is set to our **web-sg** Security Group which allows access from the internet over port 80.
- **subnets** is set to both of our web subnets. This designates where the ALB will send traffic and requires a minimum of two subnets in two different AZs.

2. Create an Application Load Balancer Target Group.

3. The **aws_lib_target_group_attachment** Resource attaches our instances to the **Target Group**. Note that I've added **depends_on** to both of these. I kept experiencing an issue where my instances kept showing as unhealthy in the Target Group because they weren't done initializing. By setting the **depends_on** to their respective web server, the issue was resolved.

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

```
1   resource "aws_lb" "external-elb" {
2     name               = "External-LB"
3     internal           = false
4     load_balancer_type = "application"
5     security_groups    = [aws_security_group.web-sg.id]
6     subnets            = [aws_subnet.web-subnet-1.id, aws_subnet.web-subnet-2.id]
7   }
```

```
8
9   resource "aws_lb_target_group" "external-elb" {
10    name     = "ALB-TG"
11    port     = 80
12    protocol = "HTTP"
13    vpc_id   = aws_vpc.my-vpc.id
14  }
15
16  resource "aws_lb_target_group_attachment" "external-elb1" {
17    target_group_arn = aws_lb_target_group.external-elb.arn
18    target_id        = aws_instance.webserver1.id
19    port             = 80
20
21    depends_on = [
22      aws_instance.webserver1,
23    ]
24  }
25
26  resource "aws_lb_target_group_attachment" "external-elb2" {
27    target_group_arn = aws_lb_target_group.external-elb.arn
28    target_id        = aws_instance.webserver2.id
29    port             = 80
30
31    depends_on = [
32      aws_instance.webserver2,
33    ]
34  }
35
36  resource "aws_lb_listener" "external-elb" {
37    load_balancer_arn = aws_lb.external-elb.arn
38    port              = "80"
39    protocol          = "HTTP"
40
41    default_action {
42      type             = "forward"
43      target_group_arn = aws_lb_target_group.external-elb.arn
44    }
45  }
```

three-tier-elb.tf hosted with ♡ by GitHub view raw

Create RDS Instance

1. Create an 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.

2. Create a DB Subnet Group. **subnet_ids** identifies which subnets will be used by the Database.

```
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"
  }
}
```

three-tier-rds.tf hosted with ♡ by GitHub view raw

Output

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

1. We will use 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
}
```

three-tier-output.tf hosted with ♡ by GitHub view raw

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 validate` 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.

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.

Clean Up

1. To delete our infrastructure run `terraform destroy` . When prompted type **Yes**. This command will delete all the infrastructure that we created.

Congratulations on creating a Three-Tier AWS Architecture!

Complete Code

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
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 Public 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"
  }
}

resource "aws_subnet" "database-subnet" {
  vpc_id            = aws_vpc.my-vpc.id
  cidr_block        = "10.0.3.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "Database"
  }
}

# Create Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.my-vpc.id

  tags = {
    Name = "Demo IGW"
  }
}

# Create Web layber 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"
  }
}
```

```
123
124 # Create Web Subnet association with Web route table
125 resource "aws_route_table_association" "a" {
126   subnet_id      = aws_subnet.web-subnet-1.id
127   route_table_id = aws_route_table.web-rt.id
128 }
129
130 resource "aws_route_table_association" "b" {
131   subnet_id      = aws_subnet.web-subnet-2.id
132   route_table_id = aws_route_table.web-rt.id
133 }
134
135 #Create EC2 Instance
136 resource "aws_instance" "webserver1" {
137   ami                    = "ami-0d5eff06f840b45e9"
138   instance_type          = "t2.micro"
139   availability_zone      = "us-east-1a"
140   vpc_security_group_ids = [aws_security_group.webserver-sg.id]
141   subnet_id              = aws_subnet.web-subnet-1.id
142   user_data              = file("install_apache.sh")
143
144   tags = {
145     Name = "Web Server"
146   }
147
148 }
149
150 resource "aws_instance" "webserver2" {
151   ami                    = "ami-0d5eff06f840b45e9"
152   instance_type          = "t2.micro"
153   availability_zone      = "us-east-1b"
154   vpc_security_group_ids = [aws_security_group.webserver-sg.id]
155   subnet_id              = aws_subnet.web-subnet-2.id
156   user_data              = file("install_apache.sh")
157
158   tags = {
159     Name = "Web Server"
160   }
161
162 }
163
164 # Create Web Security Group
165 resource "aws_security_group" "web-sg" {
166   name        = "Web-SG"
167   description = "Allow HTTP inbound traffic"
168   vpc_id      = aws_vpc.my-vpc.id
169
170   ingress {
171     description = "HTTP from VPC"
172     from_port   = 80
173     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 Application 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"
  }
}

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
  }
}

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 "lb_dns_name" {
  description = "The DNS name of the load balancer"
  value       = aws_lb.external-elb.dns_name
}
```

three-tier-main.tf hosted with ♡ by GitHub view raw

*More content at **plainenglish.io***

Terraform DevOps AWS Programming Cloud Computing