# **Terraform Resource Templates**

**Provider.tf**

terraform {

required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 5.0"

    }

  }

}

# Configure the AWS Provider

provider "aws" {

  region = "ap-south-1"

}

**Vpc.tf (subnets, route table, associate route table)**

# Create Vpc

resource "aws\_vpc" "my-vpc" {

  cidr\_block       = "10.10.0.0/16"

  instance\_tenancy = "default"

  tags = {

    Name = "my-vpc"

  }

}

# Create subnets

// Public subnet

resource "aws\_subnet" "my-Publicsubnet" {

  vpc\_id     = aws\_vpc.my-vpc.id

  cidr\_block = "10.10.1.0/24"

  tags = {

    Name = "my-Publicsubnet"

  }

}

// Private subnet

resource "aws\_subnet" "my-Privatesubnet" {

  vpc\_id     = aws\_vpc.my-vpc.id

  cidr\_block = "10.10.2.0/24"

  tags = {

    Name = "my-Privatesubnet"

  }

}

# Create internet gateway

resource "aws\_internet\_gateway" "my-igw" {

  vpc\_id = aws\_vpc.my-vpc.id

  tags = {

    Name = "my-igw"

  }

}

# Create a route table

resource "aws\_route\_table" "my-routetable" {

  vpc\_id = aws\_vpc.my-vpc.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.my-igw.id

  }

  tags = {

    Name = "my-routetable"

  }

}

# Associate subnet with route table

resource "aws\_route\_table\_association" "routetableassociation" {

  subnet\_id      = aws\_subnet.my-Publicsubnet.id

  route\_table\_id = aws\_route\_table.my-routetable.id

}

**S3.tf**

# Create s3 bucket

resource "aws\_s3\_bucket" "david-bucket" {

  bucket = "my-tf-test-bucket-david526"

  versioning {

       enabled = true

  }

  server\_side\_encryption\_configuration {

    rule {

    apply\_server\_side\_encryption\_by\_default {

      sse\_algorithm     = "AES256"

    }

  }

}

 }

**Dynamodb.tf**

# Create dynambodb table

resource "aws\_dynamodb\_table" "statelock" {

  name           = "state-lock"

  billing\_mode   = "PAY\_PER\_REQUEST"

  hash\_key       = "LockID"

  attribute {

    name = "LockID"

    type = "S"

  }

}

* Add below code to provider.tf file

terraform {

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 5.0"

    }

  }

}

# Configure the AWS Provider

provider "aws" {

  region = "ap-south-1" # Go with your region

}

* Create ec2.tf file and paste below code 👇
* **Create ec2 instance and security group**

resource "aws\_instance" "web" {

  ami           = "ami-0ad21ae1d0696ad58" # Update with ami id

  instance\_type = "t2.micro"

  key\_name      = "dock"

  vpc\_security\_group\_ids = [aws\_security\_group.david-sg.id]

  tags = {

    Name = "david-king"

  }

}

resource "aws\_security\_group" "david-sg" {

  name        = "david-sg"

  description = "security groups using terraform"

  vpc\_id      = "vpc-084eea0fba4aa03c1 # Update with your vpc id(here I used default vpc id)☝️

  ingress {

    description      = "HTTPS"

    from\_port        = 443

    to\_port          = 443

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  ingress {

    description      = "HTTP"

    from\_port        = 80

    to\_port          = 80

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  ingress {

    description      = "ssh"

    from\_port        = 22

    to\_port          = 22

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  egress {

    from\_port        = 0

    to\_port          = 0

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  tags = {

    Name = "david-SG"

  }

}

## **How to create elastic/classic load balancer in aws**

* **Below resource templates are required**

**# provider.tf**

terraform {

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 5.0"

    }

  }

}

# Configure the AWS Provider

provider "aws" {

  region = "ap-south-1"

}

**# ec2.tf**

resource "aws\_instance" "web" {

  ami                         = "ami-0ad21ae1d0696ad58"

  instance\_type               = "t2.medium"

  key\_name                    = "dock"

  vpc\_security\_group\_ids      = [aws\_security\_group.david-sg.id]

  subnet\_id                   = aws\_subnet.my-Publicsubnet.id

  associate\_public\_ip\_address = true

  availability\_zone           = "ap-south-1a"

  tags = {

    Name = "david-king"

  }

}

# security.tf

resource "aws\_security\_group" "david-sg" {

  name        = "david-sg"

  description = "security groups using terraform"

  vpc\_id      = aws\_vpc.my-vpc.id

  ingress {

    description      = "HTTPS"

    from\_port        = 443

    to\_port          = 443

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  ingress {

    description      = "HTTP"

    from\_port        = 80

    to\_port          = 80

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  ingress {

    description      = "ssh"

    from\_port        = 22

    to\_port          = 22

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  egress {

    from\_port        = 0

    to\_port          = 0

    protocol         = "-1"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  tags = {

    Name = "david-SG"

  }

}

**# vpc.tf (subnets, internet gateway, route table, associate route table)**

resource "aws\_vpc" "my-vpc" {

  cidr\_block = "10.10.0.0/16"

  tags = {

    Name = "my-vpc"

  }

}

resource "aws\_subnet" "my-Publicsubnet" {

  vpc\_id     = aws\_vpc.my-vpc.id

  cidr\_block = "10.10.1.0/24"

  map\_public\_ip\_on\_launch  = true

  tags = {

    Name = "my-Publicsubnet"

  }

}

resource "aws\_subnet" "my-Privatesubnet" {

  vpc\_id     = aws\_vpc.my-vpc.id

  cidr\_block = "10.10.2.0/24"

  tags = {

    Name = "my-Privatesubnet"

  }

}

# Create internet gateway

resource "aws\_internet\_gateway" "my-igw" {

  vpc\_id = aws\_vpc.my-vpc.id

  tags = {

    Name = "my-igw"

  }

}

# Create a route table

resource "aws\_route\_table" "my-routetable" {

  vpc\_id = aws\_vpc.my-vpc.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.my-igw.id

  }

  tags = {

    Name = "my-routetable"

  }

}

# Associate subnet with route table

resource "aws\_route\_table\_association" "routetableassociation" {

  subnet\_id      = aws\_subnet.my-Publicsubnet.id

  route\_table\_id = aws\_route\_table.my-routetable.id

}

**#elb.tf**

resource "aws\_elb" "main" {

  name            = "main-elb"

  security\_groups = [aws\_security\_group.david-sg.id]

  subnets         = [aws\_subnet.my-Publicsubnet.id]

  listener {

    instance\_port     = 80

    instance\_protocol = "HTTP"

    lb\_port           = 80

    lb\_protocol       = "HTTP"

  }

  instances = [aws\_instance.web.id]

  health\_check {

    target              = "HTTP:80/"

    interval            = 30

    timeout             = 5

    healthy\_threshold   = 2

    unhealthy\_threshold = 2

  }

  tags = {

    Name = "MainELB"

  }

}

output "elb\_dns\_name" {

  value = aws\_elb.main.dns\_name

}

## **How to create ALB In AWS Using Terraform**

* **Here is available a complete resources template**

provider "aws" {

  region = "ap-south-1"

}

resource "aws\_vpc" "my\_vpc" {

  cidr\_block = "10.0.0.0/16"

  tags = {

    Name = "my-vpc"

  }

}

resource "aws\_internet\_gateway" "my\_igw" {

  vpc\_id = aws\_vpc.my\_vpc.id

  tags = {

    Name = "my-igw"

  }

}

resource "aws\_subnet" "public\_subnet\_1" {

  vpc\_id                  = aws\_vpc.my\_vpc.id

  cidr\_block              = "10.0.1.0/24"

  map\_public\_ip\_on\_launch = true

  availability\_zone       = "ap-south-1a"

  tags = {

    Name = "public-subnet-1"

  }

}

resource "aws\_subnet" "public\_subnet\_2" {

  vpc\_id                  = aws\_vpc.my\_vpc.id

  cidr\_block              = "10.0.2.0/24"

  map\_public\_ip\_on\_launch = true

  availability\_zone       = "ap-south-1b"

  tags = {

    Name = "public-subnet-2"

  }

}

resource "aws\_route\_table" "public\_rt" {

  vpc\_id = aws\_vpc.my\_vpc.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.my\_igw.id

  }

  tags = {

    Name = "public-rt"

  }

}

resource "aws\_route\_table\_association" "public\_rt\_assoc\_1" {

  subnet\_id      = aws\_subnet.public\_subnet\_1.id

  route\_table\_id = aws\_route\_table.public\_rt.id

}

resource "aws\_route\_table\_association" "public\_rt\_assoc\_2" {

  subnet\_id      = aws\_subnet.public\_subnet\_2.id

  route\_table\_id = aws\_route\_table.public\_rt.id

}

resource "aws\_security\_group" "alb\_sg" {

  name        = "alb-sg"

  description = "Security group for ALB"

  vpc\_id      = aws\_vpc.my\_vpc.id

  ingress {

    description      = "HTTPS"

    from\_port        = 443

    to\_port          = 443

    protocol         = "tcp"

    cidr\_blocks      = ["0.0.0.0/0"]

    ipv6\_cidr\_blocks = ["::/0"]

  }

  ingress {

    description      = "HTTP"

    from\_port   = 80

    to\_port     = 80

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

  }

ingress {

  description = "SSH"

  from\_port   = 22

  to\_port     = 22

  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 = "alb-sg"

  }

}

resource "aws\_lb\_target\_group" "my\_target\_group" {

  name     = "my-target-group"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.my\_vpc.id

  health\_check {

    path                = "/"

    interval            = 30

    timeout             = 5

    healthy\_threshold   = 2

    unhealthy\_threshold = 2

    matcher             = "200"

  }

  tags = {

    Name = "my-target-group"

  }

}

resource "aws\_lb" "my\_alb" {

  name               = "my-alb"

  internal           = false

  load\_balancer\_type = "application"

  security\_groups    = [aws\_security\_group.alb\_sg.id]

  subnets            = [aws\_subnet.public\_subnet\_1.id, aws\_subnet.public\_subnet\_2.id]

  tags = {

    Name = "my-alb"

  }

}

resource "aws\_lb\_listener" "http\_listener" {

  load\_balancer\_arn = aws\_lb.my\_alb.arn

  port              = "80"

  protocol          = "HTTP"

  default\_action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.my\_target\_group.arn

  }

}

resource "aws\_instance" "web1" {

  ami                         = "ami-0ad21ae1d0696ad58"

  instance\_type               = "t3.medium"

  subnet\_id                   = aws\_subnet.public\_subnet\_1.id

  vpc\_security\_group\_ids      = [aws\_security\_group.alb\_sg.id]

  associate\_public\_ip\_address = true

  key\_name                    = "dock"

  tags = {

    Name = "web1"

  }

}

resource "aws\_instance" "web2" {

  ami                         = "ami-0ad21ae1d0696ad58"

  instance\_type               = "t3.medium"

  subnet\_id                   = aws\_subnet.public\_subnet\_2.id

  vpc\_security\_group\_ids      = [aws\_security\_group.alb\_sg.id]

  associate\_public\_ip\_address = true

  key\_name                    = "dock"

  tags = {

    Name = "web2"

  }

}

resource "aws\_lb\_target\_group\_attachment" "web1\_attachment" {

  target\_group\_arn = aws\_lb\_target\_group.my\_target\_group.arn

  target\_id        = aws\_instance.web1.id

  port             = 80

}

resource "aws\_lb\_target\_group\_attachment" "web2\_attachment" {

  target\_group\_arn = aws\_lb\_target\_group.my\_target\_group.arn

  target\_id        = aws\_instance.web2.id

  port             = 80

}

output "alb\_dns\_name" {

  value = aws\_lb.my\_alb.dns\_name

}

## **How to create ALB context-path based routing**

* Here is available all resources templates

provider "aws" {

  region = "ap-south-1"

}

# Create a VPC

resource "aws\_vpc" "example" {

  cidr\_block = "10.0.0.0/16"

  enable\_dns\_support = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "example-vpc"

  }

}

# Create an Internet Gateway

resource "aws\_internet\_gateway" "example" {

  vpc\_id = aws\_vpc.example.id

  tags = {

    Name = "example-igw"

  }

}

# Create a Public Route Table

resource "aws\_route\_table" "public" {

  vpc\_id = aws\_vpc.example.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.example.id

  }

  tags = {

    Name = "public-route-table"

  }

}

# Create Public Subnets

resource "aws\_subnet" "public" {

  count = 2

  vpc\_id = aws\_vpc.example.id

  cidr\_block = cidrsubnet(aws\_vpc.example.cidr\_block, 8, count.index)

  availability\_zone = element(data.aws\_availability\_zones.available.names, count.index)

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "public-subnet-${count.index}"

  }

}

# Associate the Public Route Table with the Public Subnets

resource "aws\_route\_table\_association" "public" {

  count = 2

  subnet\_id      = aws\_subnet.public[count.index].id

  route\_table\_id = aws\_route\_table.public.id

}

# Create a Security Group

resource "aws\_security\_group" "example" {

  vpc\_id = aws\_vpc.example.id

  ingress {

    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 = "example-security-group"

  }

}

# Create a Network Interface

resource "aws\_network\_interface" "example" {

  subnet\_id   = aws\_subnet.public[0].id

  security\_groups = [aws\_security\_group.example.id]

}

# Create an EC2 Instance

resource "aws\_instance" "example" {

  ami           = "ami-0c2af51e265bd5e0e" # Replace with an appropriate AMI ID for your region

  instance\_type = "t3.medium"

  key\_name      = "dock"

  network\_interface {

    network\_interface\_id = aws\_network\_interface.example.id

    device\_index         = 0

  }

  tags = {

    Name = "example-instance"

  }

}

# Create an Application Load Balancer

resource "aws\_lb" "example" {

  name               = "example-alb"

  internal           = false

  load\_balancer\_type = "application"

  security\_groups    = [aws\_security\_group.example.id]

  subnets            = aws\_subnet.public[\*].id

  enable\_deletion\_protection = false

  enable\_cross\_zone\_load\_balancing = true

}

# Create a Target Group for Static Path

resource "aws\_lb\_target\_group" "static" {

  name     = "static-target-group"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.example.id

  health\_check {

    path                = "/health"

    interval            = 30

    timeout             = 5

    healthy\_threshold   = 2

    unhealthy\_threshold = 2

  }

}

# Create a Target Group for Default Path

resource "aws\_lb\_target\_group" "default" {

  name     = "default-target-group"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.example.id

  health\_check {

    path                = "/health"

    interval            = 30

    timeout             = 5

    healthy\_threshold   = 2

    unhealthy\_threshold = 2

  }

}

# Create a Listener for the ALB

resource "aws\_lb\_listener" "http" {

  load\_balancer\_arn = aws\_lb.example.arn

  port              = 80

  protocol          = "HTTP"

  default\_action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.default.arn

  }

}

# Create a Listener Rule for Context-Path Based Routing

resource "aws\_lb\_listener\_rule" "static" {

  listener\_arn = aws\_lb\_listener.http.arn

  priority     = 100

  condition {

    path\_pattern {

      values = ["/static/\*"]

    }

  }

  action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.static.arn

  }

}

# Attach Target Group to EC2 Instance

resource "aws\_lb\_target\_group\_attachment" "example" {

  target\_group\_arn = aws\_lb\_target\_group.default.arn

  target\_id        = aws\_instance.example.id

  port             = 80

}

# Data source for availability zones

data "aws\_availability\_zones" "available" {}

## **How to create AWS ALB Host Header based Routing using terraform**

provider "aws" {

  region = "ap-south-1" # Update as per your region

}

resource "aws\_vpc" "main" {

  cidr\_block = "10.0.0.0/16"

}

resource "aws\_internet\_gateway" "igw" {

  vpc\_id = aws\_vpc.main.id

}

resource "aws\_subnet" "public" {

  count                   = 2

  vpc\_id                  = aws\_vpc.main.id

  cidr\_block              = cidrsubnet(aws\_vpc.main.cidr\_block, 8, count.index)

  availability\_zone       = element(data.aws\_availability\_zones.available.names, count.index)

  map\_public\_ip\_on\_launch = true

}

data "aws\_availability\_zones" "available" {}

resource "aws\_route\_table" "public" {

  vpc\_id = aws\_vpc.main.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.igw.id

  }

}

resource "aws\_route\_table\_association" "public" {

  count          = 2

  subnet\_id      = element(aws\_subnet.public[\*].id, count.index)

  route\_table\_id = aws\_route\_table.public.id

}

resource "aws\_security\_group" "alb" {

  vpc\_id = aws\_vpc.main.id

  ingress {

    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"]

  }

}

resource "aws\_security\_group" "ec2" {

  vpc\_id = aws\_vpc.main.id

  ingress {

    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"]

  }

}

resource "aws\_lb" "app" {

  name               = "app-load-balancer"

  internal           = false

  load\_balancer\_type = "application"

  security\_groups    = [aws\_security\_group.alb.id]

  subnets            = aws\_subnet.public[\*].id

  enable\_deletion\_protection = false

}

resource "aws\_lb\_target\_group" "app1" {

  name     = "app1-targets"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.main.id

}

resource "aws\_lb\_target\_group" "app2" {

  name     = "app2-targets"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.main.id

}

resource "aws\_lb\_listener" "http" {

  load\_balancer\_arn = aws\_lb.app.arn

  port              = "80"

  protocol          = "HTTP"

  default\_action {

    type = "fixed-response"

    fixed\_response {

      content\_type = "text/plain"

      message\_body = "404: Not Found"

      status\_code  = "404"

    }

  }

}

resource "aws\_lb\_listener\_rule" "app1" {

  listener\_arn = aws\_lb\_listener.http.arn

  priority     = 100

  action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.app1.arn

  }

  condition {

    host\_header {

      values = ["app1.example.com"]

    }

  }

}

resource "aws\_lb\_listener\_rule" "app2" {

  listener\_arn = aws\_lb\_listener.http.arn

  priority     = 200

  action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.app2.arn

  }

  condition {

    host\_header {

      values = ["app2.example.com"]

    }

  }

}

resource "aws\_instance" "app1" {

  ami                    = "ami-0ad21ae1d0696ad58" # Replace with your preferred AMI ID

  instance\_type          = "t2.micro"

  subnet\_id              = aws\_subnet.public[0].id

  vpc\_security\_group\_ids = [aws\_security\_group.ec2.id]

  key\_name               = "dock" # Replace with your key pair name

  user\_data              = <<-EOF

                            #!/bin/bash

                            echo "Hello, World - App1" > /var/www/html/index.html

                            yum install -y httpd

                            systemctl start httpd

                            systemctl enable httpd

                            EOF

  tags = {

    Name = "App1"

  }

  lifecycle {

    create\_before\_destroy = true

  }

}

resource "aws\_instance" "app2" {

  ami                    = "ami-0ad21ae1d0696ad58" # Replace with your preferred AMI ID

  instance\_type          = "t2.micro"

  subnet\_id              = aws\_subnet.public[1].id

  vpc\_security\_group\_ids = [aws\_security\_group.ec2.id]

  key\_name               = "dock" # Replace with your key pair name

  user\_data              = <<-EOF

                            #!/bin/bash

                            echo "Hello, World - App2" > /var/www/html/index.html

                            yum install -y httpd

                            systemctl start httpd

                            systemctl enable httpd

                            EOF

  tags = {

    Name = "App2"

  }

  lifecycle {

    create\_before\_destroy = true

  }

}

resource "aws\_lb\_target\_group\_attachment" "app1\_instance" {

  target\_group\_arn = aws\_lb\_target\_group.app1.arn

  target\_id        = aws\_instance.app1.id

  port             = 80

}

resource "aws\_lb\_target\_group\_attachment" "app2\_instance" {

  target\_group\_arn = aws\_lb\_target\_group.app2.arn

  target\_id        = aws\_instance.app2.id

  port             = 80

}

output "alb\_dns\_name" {

  value = aws\_lb.app.dns\_name

}

## **How to create AWS ALB HTTP Header and Query String Redirects using terraform**

provider "aws" {

  region = "ap-south-1" # Replace with your preferred region

}

resource "aws\_vpc" "main" {

  cidr\_block = "10.0.0.0/16"

}

resource "aws\_internet\_gateway" "igw" {

  vpc\_id = aws\_vpc.main.id

}

resource "aws\_subnet" "public" {

  count                   = 2

  vpc\_id                  = aws\_vpc.main.id

  cidr\_block              = cidrsubnet(aws\_vpc.main.cidr\_block, 8, count.index)

  availability\_zone       = element(data.aws\_availability\_zones.available.names, count.index)

  map\_public\_ip\_on\_launch = true

}

data "aws\_availability\_zones" "available" {}

resource "aws\_route\_table" "public" {

  vpc\_id = aws\_vpc.main.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.igw.id

  }

}

resource "aws\_route\_table\_association" "public" {

  count          = 2

  subnet\_id      = element(aws\_subnet.public[\*].id, count.index)

  route\_table\_id = aws\_route\_table.public.id

}

resource "aws\_security\_group" "alb" {

  vpc\_id = aws\_vpc.main.id

  ingress {

    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"]

  }

}

resource "aws\_security\_group" "ec2" {

  vpc\_id = aws\_vpc.main.id

  ingress {

    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"]

  }

}

resource "aws\_lb" "app" {

  name               = "app-load-balancer"

  internal           = false

  load\_balancer\_type = "application"

  security\_groups    = [aws\_security\_group.alb.id]

  subnets            = aws\_subnet.public[\*].id

  enable\_deletion\_protection = false

}

resource "aws\_lb\_target\_group" "app1" {

  name     = "app1-targets"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.main.id

}

resource "aws\_lb\_target\_group" "app2" {

  name     = "app2-targets"

  port     = 80

  protocol = "HTTP"

  vpc\_id   = aws\_vpc.main.id

}

resource "aws\_lb\_listener" "http" {

  load\_balancer\_arn = aws\_lb.app.arn

  port              = "80"

  protocol          = "HTTP"

  default\_action {

    type = "fixed-response"

    fixed\_response {

      content\_type = "text/plain"

      message\_body = "404: Not Found"

      status\_code  = "404"

    }

  }

}

resource "aws\_lb\_listener\_rule" "http\_header\_redirect" {

  listener\_arn = aws\_lb\_listener.http.arn

  priority     = 100

  action {

    type = "redirect"

    redirect {

      port        = "80"

      protocol    = "HTTP"

      host        = "example.com"

      path        = "/new-path"

## **How to create AWS DNS (Route53) Using Terraform**

provider "aws" {

  region = "us-east-1" # You can change the region as needed

}

# Create a Route 53 Hosted Zone

resource "aws\_route53\_zone" "main" {

  name = "example.local" # Replace with your placeholder domain name

}

# Create an A record

resource "aws\_route53\_record" "www" {

  zone\_id = aws\_route53\_zone.main.zone\_id

  name    = "www.example.local"

  type    = "A"

  ttl     = "300"

  records = ["192.0.2.44"] # Replace with your EC2 instance IP or other target

}

# Create a CNAME record

resource "aws\_route53\_record" "blog" {

  zone\_id = aws\_route53\_zone.main.zone\_id

  name    = "blog.example.local"

  type    = "CNAME"

  ttl     = "300"

  records = ["www.example.local"] # Replace with the target CNAME

}

# Create an MX record

resource "aws\_route53\_record" "mail" {

  zone\_id = aws\_route53\_zone.main.zone\_id

  name    = "example.local" # Replace with your placeholder domain name

  type    = "MX"

  ttl     = "300"

  records = ["10 mail.example.local"] # Replace with your mail server

}

## **How to create AWS RDS Using Terraform**

provider "aws" {

  region = "ap-south-1" # Change as needed

}

resource "aws\_vpc" "example" {

  cidr\_block           = "10.0.0.0/16"

  enable\_dns\_support   = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "example-vpc"

  }

}

resource "aws\_subnet" "subnet1" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.1.0/24"

  availability\_zone       = "ap-south-1a"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-1"

  }

}

resource "aws\_subnet" "subnet2" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.2.0/24"

  availability\_zone       = "ap-south-1b"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-2"

  }

}

resource "aws\_db\_subnet\_group" "example" {

  name       = "mydb-subnet-group"

  subnet\_ids = [aws\_subnet.subnet1.id, aws\_subnet.subnet2.id]

  tags = {

    Name = "My DB subnet group"

  }

}

resource "aws\_db\_instance" "example" {

  allocated\_storage    = 20

  storage\_type         = "gp2"

  engine               = "mysql"

  engine\_version       = "8.0"

  instance\_class       = "db.t3.micro"

  db\_name              = "mydb"

  username             = "admin"

  password             = "mypassword"

  parameter\_group\_name = "default.mysql8.0"

  publicly\_accessible  = false

  db\_subnet\_group\_name = aws\_db\_subnet\_group.example.name

  # Backup and maintenance settings

  backup\_retention\_period = 7

  backup\_window           = "03:00-06:00"

  maintenance\_window      = "Mon:00:00-Mon:03:00"

}

## **How to create AWS autoscaling with launch config and templates**

provider "aws" {

  region = "ap-south-1" # Updated region

}

# VPC

resource "aws\_vpc" "example" {

  cidr\_block           = "10.0.0.0/16"

  enable\_dns\_support   = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "example-vpc"

  }

}

# Subnet 1

resource "aws\_subnet" "subnet1" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.1.0/24"

  availability\_zone       = "ap-south-1a" # Updated AZ

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-1"

  }

}

# Subnet 2

resource "aws\_subnet" "subnet2" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.2.0/24"

  availability\_zone       = "ap-south-1b" # Updated AZ

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-2"

  }

}

# Security Group

resource "aws\_security\_group" "example" {

  vpc\_id = aws\_vpc.example.id

  ingress {

    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 = "example-sg"

  }

}

# IAM Role

resource "aws\_iam\_role" "example" {

  name = "example-role"

  assume\_role\_policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Action = "sts:AssumeRole",

        Effect = "Allow",

        Principal = {

          Service = "ec2.amazonaws.com"

        }

      }

    ]

  })

}

# IAM Role Policy

resource "aws\_iam\_role\_policy" "example" {

  name = "example-policy"

  role = aws\_iam\_role.example.id

  policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Action = [

          "ec2:Describe\*",

          "ec2:AssociateIamInstanceProfile",

          "ec2:CreateNetworkInterface",

          "ec2:AttachNetworkInterface",

          "ec2:DescribeNetworkInterfaces",

          "ec2:DetachNetworkInterface",

          "ec2:DeleteNetworkInterface"

        ],

        Effect   = "Allow",

        Resource = "\*"

      }

    ]

  })

}

# IAM Instance Profile

resource "aws\_iam\_instance\_profile" "example" {

  name = "example-instance-profile"

  role = aws\_iam\_role.example.name

}

# Launch Configuration

resource "aws\_launch\_configuration" "example" {

  name                 = "example-launch-configuration"

  image\_id             = "ami-0ad21ae1d0696ad58" # Updated AMI ID

  instance\_type        = "t2.micro"

  security\_groups      = [aws\_security\_group.example.id]

  iam\_instance\_profile = aws\_iam\_instance\_profile.example.name

  lifecycle {

    create\_before\_destroy = true

  }

  user\_data = base64encode(<<-EOF

              #!/bin/bash

              echo "Hello, World!" > index.html

              nohup busybox httpd -f -p 80 &

              EOF

            )

}

# Launch Template

resource "aws\_launch\_template" "example" {

  name\_prefix          = "example-"

  image\_id             = "ami-0ad21ae1d0696ad58" # Updated AMI ID

  instance\_type        = "t2.micro"

  security\_group\_names = [aws\_security\_group.example.name]

  iam\_instance\_profile {

    name = aws\_iam\_instance\_profile.example.name

  }

  user\_data = base64encode(<<-EOF

              #!/bin/bash

              echo "Hello, World!" > index.html

              nohup busybox httpd -f -p 80 &

              EOF

            )

  tag\_specifications {

    resource\_type = "instance"

    tags = {

      Name = "example-instance"

    }

  }

}

# Auto Scaling Group

resource "aws\_autoscaling\_group" "example" {

  # Uncomment the line below to use launch configuration

  launch\_configuration = aws\_launch\_configuration.example.name

  # Uncomment the lines below to use launch template

  # launch\_template {

  #   id      = aws\_launch\_template.example.id

  #   version = "$Latest"

  # }

  min\_size            = 1

  max\_size            = 3

  desired\_capacity    = 1

  vpc\_zone\_identifier = [aws\_subnet.subnet1.id, aws\_subnet.subnet2.id]

  tag {

    key                 = "Name"

    value               = "example-asg-instance"

    propagate\_at\_launch = true

  }

  health\_check\_type         = "EC2"

  health\_check\_grace\_period = 300

}

# Auto Scaling Policies

resource "aws\_autoscaling\_policy" "scale\_up" {

  name                   = "scale\_up"

  scaling\_adjustment     = 1

  adjustment\_type        = "ChangeInCapacity"

  cooldown               = 300

  autoscaling\_group\_name = aws\_autoscaling\_group.example.name

}

resource "aws\_autoscaling\_policy" "scale\_down" {

  name                   = "scale\_down"

  scaling\_adjustment     = -1

  adjustment\_type        = "ChangeInCapacity"

  cooldown               = 300

  autoscaling\_group\_name = aws\_autoscaling\_group.example.name

}

## **How to create AWS Network Load Balancer with TCP and TLS**

provider "aws" {

  region = "ap-south-1"

}

# VPC

resource "aws\_vpc" "example" {

  cidr\_block           = "10.0.0.0/16"

  enable\_dns\_support   = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "example-vpc"

  }

}

# Subnet 1

resource "aws\_subnet" "subnet1" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.1.0/24"

  availability\_zone       = "ap-south-1a"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-1"

  }

}

# Subnet 2

resource "aws\_subnet" "subnet2" {

  vpc\_id                  = aws\_vpc.example.id

  cidr\_block              = "10.0.2.0/24"

  availability\_zone       = "ap-south-1b"

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "example-subnet-2"

  }

}

# Security Group

resource "aws\_security\_group" "example" {

  vpc\_id = aws\_vpc.example.id

  ingress {

    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 = "example-sg"

  }

}

# IAM Role

resource "aws\_iam\_role" "example" {

  name = "example-role"

  assume\_role\_policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Action = "sts:AssumeRole",

        Effect = "Allow",

        Principal = {

          Service = "ec2.amazonaws.com"

        }

      }

    ]

  })

}

# IAM Role Policy

resource "aws\_iam\_role\_policy" "example" {

  name = "example-policy"

  role = aws\_iam\_role.example.id

  policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Action = [

          "ec2:Describe\*",

          "ec2:AssociateIamInstanceProfile",

          "ec2:CreateNetworkInterface",

          "ec2:AttachNetworkInterface",

          "ec2:DescribeNetworkInterfaces",

          "ec2:DetachNetworkInterface",

          "ec2:DeleteNetworkInterface"

        ],

        Effect   = "Allow",

        Resource = "\*"

      }

    ]

  })

}

# IAM Instance Profile

resource "aws\_iam\_instance\_profile" "example" {

  name = "example-instance-profile"

  role = aws\_iam\_role.example.name

}

# EC2 Instance 1

resource "aws\_instance" "web1" {

  ami                    = "ami-0ad21ae1d0696ad58"

  instance\_type          = "t2.micro"

  subnet\_id              = aws\_subnet.subnet1.id

  security\_groups        = [aws\_security\_group.example.name]

  iam\_instance\_profile   = aws\_iam\_instance\_profile.example.name

  user\_data = <<-EOF

              #!/bin/bash

              echo "Hello from web server 1" > /var/www/html/index.html

              nohup busybox httpd -f -p 80 &

              EOF

  tags = {

    Name = "web1"

  }

}

# EC2 Instance 2

resource "aws\_instance" "web2" {

  ami                    = "ami-0ad21ae1d0696ad58"

  instance\_type          = "t2.micro"

  subnet\_id              = aws\_subnet.subnet2.id

  security\_groups        = [aws\_security\_group.example.name]

  iam\_instance\_profile   = aws\_iam\_instance\_profile.example.name

  user\_data = <<-EOF

              #!/bin/bash

              echo "Hello from web server 2" > /var/www/html/index.html

              nohup busybox httpd -f -p 80 &

              EOF

  tags = {

    Name = "web2"

  }

}

# Target Group

resource "aws\_lb\_target\_group" "example" {

  name        = "example-targets"

  port        = 80

  protocol    = "TCP"

  vpc\_id      = aws\_vpc.example.id

  health\_check {

    healthy\_threshold   = 3

    unhealthy\_threshold = 3

    timeout             = 10

    interval            = 30

    path                = "/"

    matcher             = "200"

  }

  tags = {

    Name = "example-targets"

  }

}

# Register Targets

resource "aws\_lb\_target\_group\_attachment" "web1" {

  target\_group\_arn = aws\_lb\_target\_group.example.arn

  target\_id        = aws\_instance.web1.id

  port             = 80

}

resource "aws\_lb\_target\_group\_attachment" "web2" {

  target\_group\_arn = aws\_lb\_target\_group.example.arn

  target\_id        = aws\_instance.web2.id

  port             = 80

}

# Network Load Balancer

resource "aws\_lb" "example" {

  name               = "example-nlb"

  internal           = false

  load\_balancer\_type = "network"

  subnets            = [aws\_subnet.subnet1.id, aws\_subnet.subnet2.id]

  enable\_deletion\_protection = false

  tags = {

    Name = "example-nlb"

  }

}

# NLB TCP Listener

resource "aws\_lb\_listener" "tcp" {

  load\_balancer\_arn = aws\_lb.example.arn

  port              = 80

  protocol          = "TCP"

  default\_action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.example.arn

  }

}

# NLB TLS Listener (Example for port 443)

resource "aws\_lb\_listener" "tls" {

  load\_balancer\_arn = aws\_lb.example.arn

  port              = 443

  protocol          = "TLS"

  ssl\_policy        = "ELBSecurityPolicy-2016-08"

  certificate\_arn   = "arn:aws:acm:ap-south-1:123456789012:certificate/abcd1234-5678-90ab-cdef-12345678abcd" # Replace with your certificate ARN

  default\_action {

    type             = "forward"

    target\_group\_arn = aws\_lb\_target\_group.example.arn

  }

}