

## **Project Documentation: AWS Three-Tier Architecture with CloudFront and S3**

### ****Project Overview****

### This project involves the deployment of a **highly scalable and cost-efficient** AWS-based three-tier architecture using **AWS CloudFront, S3, EC2, RDS, and other networking components**. The infrastructure is provisioned using **Terraform Infrastructure as Code (IaC)**, ensuring automation, scalability, and security.

1. **Presentation Tier (Frontend)**
2. **Logic Tier (Backend)**
3. **Database Tier**

### ****Architecture Components****

#### ****1. Networking Layer****

* **VPC (Virtual Private Cloud):** Provides network isolation.
* **Public and Private Subnets:** Organizes application tiers securely.
* **Internet Gateway (IGW):** Enables internet access for public subnet components.
* **NAT Gateway:** Allows private subnets to access the internet securely.
* **Network ACLs (NACLs):** Controls inbound and outbound traffic at the subnet level.

#### ****2. Presentation Tier (Public Subnet)****

* **AWS CloudFront:** Acts as a Content Delivery Network (CDN) for optimized performance and security.
* **Amazon S3:** Hosts static content and serves as the origin for CloudFront.
* **AWS WAF (Web Application Firewall):** Protects against common web threats.
* **Amazon Route 53:** Manages domain routing and DNS configuration.

#### ****3. Application Tier (Logic Layer - Public Subnet)****

* **AWS Load Balancer (ALB):** Distributes traffic across backend instances.
* **3 EC2 Instances:** Processes application logic dynamically.
* **IAM Policies & Roles:** Ensures secure access control.

#### ****4. Database Tier (Private Subnet)****

* **AWS Load Balancer (Optional for HA):** Balances database traffic.
* **3 Amazon RDS Instances (MySQL):** Provides high availability and reliability.
* **IAM Policies & Roles:** Ensures secure database access.

#### ****5. Terraform Infrastructure Management****

* **Terraform Backend Storage (S3):** Stores Terraform state files securely.
* **Terraform Modules:** Automates infrastructure provisioning.

### ****Key Features & Benefits****

✅ **Highly Scalable & Available:** Uses AWS Auto Scaling and CloudFront caching. ✅ **Cost-Optimized:** No EC2 instances in the presentation tier (CloudFront + S3 reduces compute costs). ✅ **Secure:** Implements AWS WAF, IAM, and NACLs for enhanced security. ✅ **Automated Deployment:** Terraform automates infrastructure provisioning and state management. ✅ **High Performance:** CloudFront and Load Balancers optimize content delivery and request handling.

## **Terraform Code for CloudFront + S3 + Three-Tier AWS Architecture**

### ****Key AWS Services in the Architecture****

1. **Networking Components:**
   * **VPC (Virtual Private Cloud)** for network isolation.
   * **Public Subnets:** Hosts **AWS CloudFront + S3 (Presentation Tier)** and **Application Load Balancer, 3 EC2 Instances (Logic Tier)**.
   * **Private Subnets:** Hosts **AWS Load Balancer, 3 Amazon RDS instances (Database Tier).**
   * **Internet Gateway (IGW)** for internet access.
   * **NAT Gateway** for private instances to access the internet securely.
   * **Network ACLs (NACLs) instead of Security Groups** to control inbound/outbound traffic.
2. **Security & Access Control:**
   * **AWS WAF (Web Application Firewall)** for protection.
   * **IAM Roles & Policies** for secure access.
   * **Network ACLs (NACLs)** to manage traffic control at the subnet level.
3. **Frontend Layer (Presentation Tier - Public Subnet):**
   * **Amazon Route 53** (Domain Name System for global routing).
   * **AWS CloudFront** (Content Delivery Network to optimize content delivery).
   * **Amazon S3** (Static Content Hosting).
4. **Application Layer (Logic Tier - Public Subnet):**
   * **Application Load Balancer (ALB) to distribute traffic.**
   * **3 EC2 Instances for processing application logic.**
   * **IAM configured for secure access.**
5. **Database Layer (Data Tier - Private Subnet):**
   * **AWS Load Balancer (Optional for HA).**
   * **3 Amazon RDS Instances.**
   * **IAM configured for secure database access.**

### ****1. provider.tf - AWS Provider****

provider "aws" {

region = var.aws\_region

}

### ****2. variables.tf - Define Input Variables****

variable "aws\_region" {

description = "AWS Region"

default = "us-east-1"

}

variable "bucket\_name" {

description = "S3 Bucket Name for Static Website"

default = "my-static-website-bucket"

}  
  
variable "terraform\_state\_bucket" {

description = "S3 Bucket for Terraform State"

default = "my-terraform-state-bucket"

}

### ****3. backend.tf - Store Terraform State in S3****

terraform {

backend "s3" {

bucket = var.terraform\_state\_bucket

key = "terraform/state.tfstate"

region = "us-east-1"

encrypt = true

versioning = true

}  
}

### ****4. main.tf - Define AWS Infrastructure****

#create VPC

resource "aws\_vpc" "MyVPC" {

  cidr\_block = "10.0.0.0/16"

  enable\_dns\_support = true

}

# internet gateway

resource "aws\_internet\_gateway" "MyIGW" {

  vpc\_id = aws\_vpc.MyVPC.id

}

#Route tabler for public and private subnet's

# Public Route Table

resource "aws\_route" "public\_internet\_access" {

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

  destination\_cidr\_block = "0.0.0.0/0"

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

}

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

  count = length(aws\_subnet.public)

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

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

}

# Private Route Table

resource "aws\_route" "private\_nat\_access" {

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

  destination\_cidr\_block = "0.0.0.0/0"

  nat\_gateway\_id         = aws\_nat\_gateway.MyNatGateway.id

}

resource "aws\_route\_table\_association" "private\_assoc" {

  count = length(aws\_subnet.private)

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

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

}

# Nat Gateway

# Elastic IP for NAT Gateway

resource "aws\_eip" "MyEIP" {

  domain = "vpc"

}

resource "aws\_nat\_gateway" "MyNatGateway" {

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

  allocation\_id = aws\_eip.MyEIP.id

}

#create public subnet and private subnet

resource "aws\_subnet" "public" {

  vpc\_id = aws\_vpc.MyVPC.id

  cidr\_block = "10.0.${count.index + 1}.0/24"

  map\_public\_ip\_on\_launch = true

  count = 2

}

resource "aws\_subnet" "private" {

  vpc\_id = aws\_vpc.MyVPC.id

  cidr\_block = "10.0.${count.index + 3}.0/24"

  count = 2

}

#network acl for public subnet and private subnet

resource "aws\_network\_acl" "public\_nacl" {

  vpc\_id = aws\_vpc.MyVPC.id

}

resource "aws\_network\_acl" "private\_nacl" {

  vpc\_id = aws\_vpc.MyVPC.id

}

#route table for public subnet and private subnet

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

  vpc\_id = aws\_vpc.MyVPC.id

}

resource "aws\_route\_table" "private\_route\_table" {

  vpc\_id = aws\_vpc.MyVPC.id

}

#AWS Security Configuration (WAF, IAM)

resource "aws\_waf\_web\_acl" "WAF" {

  name = "MyWAF"

  metric\_name = "MyWAF"

  default\_action {

    type = "ALLOW"

  }

}

#Deploy AWS CloudFront, S3, EC2, and RDS

# Create S3 Bucket

resource "aws\_s3\_bucket" "MyBucket" {

  bucket = "mybucketswaras"

}

# Configure S3 Bucket for Static Website Hosting

resource "aws\_s3\_bucket\_website\_configuration" "website\_config" {

  bucket = aws\_s3\_bucket.MyBucket.id

  index\_document {

    suffix = "index.html"

  }

  error\_document {

    key = "error.html"

  }

}

# Enable Public Access (If Required)

resource "aws\_s3\_bucket\_public\_access\_block" "public\_access" {

  bucket = aws\_s3\_bucket.MyBucket.id

  block\_public\_acls       = false

  block\_public\_policy     = false

  ignore\_public\_acls      = false

  restrict\_public\_buckets = false

}

# Define S3 Bucket Policy for Public Read Access

resource "aws\_s3\_bucket\_policy" "public\_policy" {

  bucket = aws\_s3\_bucket.MyBucket.id

  policy = jsonencode({

    Version = "2012-10-17",

    Statement = [

      {

        Sid       = "AllowCloudFrontAccess",

        Effect    = "Allow",

        Principal = {

          Service = "cloudfront.amazonaws.com"

        },

        Action    = "s3:GetObject",

        Resource  = "${aws\_s3\_bucket.MyBucket.arn}/\*",

        Condition = {

          StringEquals = {

            "AWS:SourceArn" = "arn:aws:cloudfront::YOUR\_ACCOUNT\_ID:distribution/YOUR\_CLOUDFRONT\_ID"

          }

        }

      }

    ]

  })

}

# create cloudfront

resource "aws\_cloudfront\_distribution" "cdn" {

    origin {

        domain\_name = aws\_s3\_bucket.MyBucket.bucket\_regional\_domain\_name

        origin\_id   = aws\_s3\_bucket.MyBucket.bucket\_regional\_domain\_name

    }

    enabled = true

    default\_cache\_behavior {

        allowed\_methods = ["GET", "HEAD"]

        cached\_methods  = ["GET", "HEAD"]

        target\_origin\_id = aws\_s3\_bucket.MyBucket.bucket\_regional\_domain\_name

        viewer\_protocol\_policy = "allow-all"

        forwarded\_values {

            query\_string = false

            cookies {

                forward = "none"

            }

        }

    }

    restrictions {

        geo\_restriction {

            restriction\_type = "none"

        }

    }

    viewer\_certificate {

        cloudfront\_default\_certificate = true

    }

}

#create EC2 instance

resource "aws\_instance" "MyEC2" {

  count         = 3

  ami          = "ami-023a307f3d27ea427"

  instance\_type = "t2.micro"

  key\_name      = "Santhu"

  subnet\_id     = aws\_subnet.public[count.index % 2].id

}

#create RDS instance

resource "aws\_db\_instance" "MyRDS" {

  count = 3

  allocated\_storage    = 20

  engine               = "mysql"

  engine\_version       = "5.7"

  instance\_class       = "db.t2.micro"

  identifier           = "rds-instance-${count.index}"

  username             = "admin"  # ✅ Required field

  password             = "Swaras@09!"  # ✅ Required field (Use AWS Secrets Manager in production)

  publicly\_accessible  = false

  skip\_final\_snapshot  = true

}

#IAM role for EC2 and RDS

resource "aws\_iam\_role" "ec2\_role" {

  name = "MyEC2Role"

  assume\_role\_policy = <<EOF

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "Service": "ec2.amazonaws.com"

      },

      "Action": "sts:AssumeRole"

    }

  ]

}

EOF

}

resource "aws\_iam\_role" "rds\_role" {

  name = "MyRDSRole"

  assume\_role\_policy = <<EOF

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "Service": "rds.amazonaws.com"

      },

      "Action": "sts:AssumeRole"

    }

   ]

  }

    EOF

}

### ****7. Outputs****

output "s3\_website\_url" {

value = aws\_s3\_bucket.website\_bucket.website\_endpoint

}

output "cloudfront\_url" {

value = aws\_cloudfront\_distribution.cdn.domain\_name

}

### ****Final Fixes & Enhancements****

✅ **AWS CloudFront serves static content from S3 for fast delivery.**  
✅ **3 EC2 instances deployed in Logic Tier for backend processing.**  
✅ **3 RDS instances deployed in Private Subnets for database storage.**  
✅ **IAM Roles configured for EC2 and RDS security.**  
✅ **Network ACLs implemented instead of Security Groups.**  
✅ **NAT Gateway configured between Public and Private Subnets.**  
✅ **Outputs now include CloudFront and S3 Website URLs.  
  
  
Terraform Yml file to create the infra.**name: 'Terraform CI/CD'

on:

push:

branches: [ "main" ]

pull\_request:

jobs:

terraform:

runs-on: ubuntu-latest

steps:

- name: Checkout Repository

uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v1

- name: Configure AWS Credentials # 🛠 FIX: Authenticate AWS

uses: aws-actions/configure-aws-credentials@v2

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ap-south-1 # Change to your AWS region

- name: Initialize Terraform

run: terraform init

- name: Format Check

run: terraform fmt -recursive

- name: Terraform Plan

run: terraform plan -input=false

- name: Apply Changes (Only on Main Branch)

if: github.ref == 'refs/heads/main' && github.event\_name == 'push'

run: terraform apply -auto-approve -input=false