AWS Web App Deployment with Terraform & GitHub Actions

This repository automates the deployment and destruction of AWS infrastructure using Terraform and GitHub Actions. It supports multiple environments (`dev`, `staging`, `prod`) and regions, and integrates with s3/dynamodb for remote state management.

A comprehensive Terraform project for deploying a scalable, secure web application infrastructure on AWS with multi-environment support.

This project will provision infrastructure for a simple, scalable web application. The app will run on EC2 instances in an Auto Scaling Group (ASG) behind a Load Balancer, use RDS for data, and store static assets in S3, distributed via CloudFront. Infrastructure Requirements 1. Networking • Create a VPC with: o 2 public subnets (for ALB + NAT) o 2 private subnets (for EC2 + RDS) o Internet Gateway + NAT Gateway o Proper route tables for each tier 2. Compute • Deploy an Auto Scaling Group with: o EC2 instances running Amazon Linux 2 o User data script that installs Nginx and serves a simple HTML "Hello from Terraform" page o Instance type configurable via variable (default: t3.micro) o Scaling policy based on CPU utilization (optional) 3. Load Balancing • Create an Application Load Balancer (ALB): o Public-facing o Listener on port 80 o Target group that includes the ASG instances o Health check endpoint (/) 4. Database • Create an RDS instance (MySQL or PostgreSQL): o Deployed in private subnets o Accessible only from EC2 instances o Configurable parameters (storage, instance type, engine version) o Credentials managed via Terraform variables (safely) 5. Terraform Best Practices • Organize configuration into modules (e.g., vpc, compute, alb, rds, s3_cloudfront) • Use a remote backend (S3 + DynamoDB) for Terraform state • Implement variables, outputs, and a terraform.tfvars file • Apply consistent resource tagging (Environment, Owner, Project) • Follow the principle of least privilege for IAM roles/policies 6. Monitoring • Add CloudWatch alarms: o High CPU usage on EC2 o Low free storage on RDS

□ Architecture Overview

Design Diagrams

(https://github.com/sthakur1985/aws-web-app-deploy/blob/main/aws-architecture-webapp-1.jpg)

(https://github.com/sthakur1985/aws-web-app-deploy/blob/main/aws-architecture-webapp-2.jpg)

This project proposes a complete 3-tier web application infrastructure:

- **Presentation Tier**: Application Load Balancer (ALB) with SSL/TLS termination. (SSL is not considered as of now)
- **Application Tier**: Auto Scaling Group with EC2 instances in private subnets
- **Data Tier**: RDS MySQL is deployed in Multi-Az mode. Diagram depicts provsion of read replica as well
- **Content Delivery**: S3 + CloudFront for static content
- **DNS Management**: Route53 for domain management
- **Security**: Centralized IAM roles and security groups. DB credentails are managed by secret manager
- **Monitoring**: CloudWatch alarms and logging

Project Structure

```
aws-web-app-deploy/
└── workflows/
    └── terraform.yml        # Github actions workflow
├── main.tf                  # Root module configuration
├── variables.tf             # Input variables
├── outputs.tf              # Output values
├── backend.tf              # Terraform backend configuration
├── environments/           # Environment-specific configurations
│   ├── dev/
│   │   ├── terraform.tfvars
│   │   ├── backend.hcl
│   │   └── secrets.tfvars.example
│   ├── staging/
│   └── prod/
└── modules/                # Reusable Terraform modules
    ├── vpc/               # VPC, subnets, routing
    ├── alb/               # Application Load Balancer
    ├── compute/           # EC2 Auto Scaling Group
    ├── rds/               # RDS database
    ├── iam/               # IAM roles and policies
    ├── s3/                # S3 bucket for static content
    ├── cloudfront/        # CloudFront CDN
    ├── route53/           # DNS management
    └── cloudwatch/        # Monitoring and alarms
```

Network Architecture

Subnet Design

- **2 Public Subnets** (ALB) - Multi-AZ for high availability
- **2 EC2 Private Subnets** (Application servers) - Multi-AZ with NAT Gateway access
- **2 RDS Private Subnets** (Database) - Multi-AZ, isolated from internet

Security Groups

- **ALB Security Group**: HTTP/HTTPS from internet
- **EC2 Security Group**: HTTP from ALB only
- **RDS Security Group**: Database port from EC2 only

Quick Start

Prerequisites

- AWS CLI configured with appropriate credentials
- Terraform >= 1.5.0
- Provider Version ~> 5.0
- An AWS account with necessary permissions

1. Clone and Setup

```
git clone <repository-url>
cd aws-web-app-deploy
```

2. Configure Environment

```
# Copy and edit environment configuration
cp environments/dev/terraform.tfvars.example environments/dev/terraform.tfvars
cp environments/dev/secrets.tfvars.example environments/dev/secrets.tfvars

# Edit with your values
```

```
vi environments/dev/terraform.tfvars
vi environments/dev/secrets.tfvars  #  may not be used as we are using secrets from
        pipeline secret env variables. Can be used in user local unit testing.
```

Prerequisites

- [Terraform CLI]
- [Terraform Cloud]
- AWS account with access keys
- GitHub repository secrets configured:
  - `AWS_ACCESS_KEY_ID`
  - `AWS_SECRET_ACCESS_KEY`
  - `TF_DB_CRED` (optional: database credentials or other sensitive vars)

▯ Security & Secrets

Secrets must be configured in your GitHub repository:

- `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`: for AWS authentication
- `TF_DB_CRED`: optional, passed as a variable to Terraform

Required Variables

```
# Basic Configuration
aws_region   = "eu-west-2"
env          = "dev"
project      = "webapp"
project_name = "webapp"
costcenter   = "development"
owner        = "dev-team"

# Network Configuration
vpc_cidr                  = "10.0.0.0/16"
public_subnet_cidrs       = ["10.0.1.0/24", "10.0.2.0/24"]
ec2_private_subnet_cidrs  = ["10.0.3.0/24", "10.0.4.0/24"]
rds_private_subnet_cidrs  = ["10.0.5.0/24", "10.0.6.0/24"]

# Compute Configuration
instance_type = "t3.micro"
key_name     = "my-keypair"

# Database Configuration
db_engine         = "mysql"
db_instance_class = "db.t3.micro"
db_name          = "appdb"
db_username      = "admin"
```

Secrets Configuration

Create `environments/{env}/secrets.tfvars`:

```
db_password = "your-secure-password-here"
```

Multi-Environment Support

Environment Differences

Feature	Dev	Staging	Production
Instance Type	t3.micro	t3.small	t3.medium
RDS Multi-AZ	No	Yes	Yes

Monitoring	Basic	Enhanced	Full
Backup Retention	7 days	14 days	30 days
Auto Scaling	1-2 instances	1-3 instances	2-10 instances

Deployment Commands

Development:

```
terraform init -backend-config=environments/dev/backend.hcl
terraform apply -var-file=environments/dev/terraform.tfvars -var="<secret-key>=<secret-value>"
```

Staging:

```
terraform init -backend-config=environments/staging/backend.hcl
terraform apply -var-file=environments/staging/terraform.tfvars -var="<secret-key>=<secret-value>"
```

Production:

```
terraform init -backend-config=environments/prod/backend.hcl
terraform apply -var-file=environments/prod/terraform.tfvars -var="<secret-key>=<secret-value>"
```

Security Features

Infrastructure Security

- **VPC Flow Logs** for network monitoring
- **Private subnets** for application and database tiers
- **Security groups** with least privilege access
- **NAT Gateways** for secure outbound internet access
- **RDS encryption** at rest and in transit
- **SECRET management** is done aws secret manager or pipeline secrets

IAM Security

- **Centralized IAM module** for role management
- **EC2 instance profiles** with minimal permissions
- **RDS monitoring roles** for enhanced monitoring
- **Principle of least privilege** throughout

Data Security

- **AWS Secrets Manager** for RDS password management
- **S3 bucket encryption** for static content
- **CloudFront HTTPS** enforcement
- **ALB SSL/TLS** termination

Monitoring & Logging

CloudWatch Alarms

- **EC2 CPU utilization** monitoring
- **RDS CPU and storage** monitoring
- **ALB response time** and error rate monitoring
- **Custom thresholds** per environment

Logging

- **VPC Flow Logs** to CloudWatch
- **ALB access logs** to S3 (optional)
- **RDS logs** to CloudWatch (configurable)

CDN & DNS

CloudFront Configuration

- **S3 origin** with Origin Access Control (OAC)
- **HTTPS redirect** enforcement
- **Compression** enabled
- **Custom domains** support with SSL certificates

Route53 Integration

- **Hosted zone** management (optional)
- **A records** for ALB and CloudFront
- **Health checks** for high availability

Troubleshooting

Common Issues

ALB Creation Failed: - Check if your AWS account has ALB permissions - Verify subnet configuration and availability zones

RDS Connection Issues: - Verify security group rules - Check subnet group configuration - Ensure RDS is in private subnets

Terraform State Issues: - Ensure S3 bucket exists for state storage - Check backend configuration in `.hcl` files - Verify AWS credentials and permissions

Useful Commands

```
# Check Terraform state
terraform state list

# Import existing resources
terraform import aws_instance.example i-1234567890abcdef0

# Refresh state
terraform refresh

# Validate configuration
terraform validate

# Format code
terraform fmt -recursive
```

Outputs

After successful deployment, you'll receive:

```
# Network Information
vpc_id                = "vpc-xxxxxxxxx"
public_subnet_ids     = ["subnet-xxxxxxxx", "subnet-yyyyyyyy"]
private_subnet_ids    = ["subnet-xxxxxxxx", "subnet-yyyyyyyy"]
```

```
# Application Information
alb_dns_name         = "webapp-dev-alb-xxxxxxxxx.eu-west-2.elb.amazonaws.com"
cloudfront_domain    = "xxxxxxxxx.cloudfront.net"

# Database Information
rds_endpoint         = "webapp-dev-rds.xxxxxxxxx.eu-west-2.rds.amazonaws.com"

# DNS Information (if configured)
alb_fqdn            = "app.example.com"
cloudfront_fqdn     = "cdn.example.com"
```

Features

- Modular Terraform setup for AWS resources (VPC, EC2, RDS, ALB, S3, CloudFront)
- GitHub Actions workflow for:
  - `terraform init`, `fmt`, `plan`, and `apply`
  - Environment and region selection
  - Secure secret management
- Supports both `terraform_apply` and `terraform_destroy` actions
- Uses s3 bucket and dynamodb for statefile management. (for dev and staging s3 lockfile is enabled instead for dynamodb)

Testing (Optional)

- Unit testing code using terratest and go.
- Integration test code will help to check the all modules are working as expected together.
- format and syntax checking.

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test thoroughly
5. Submit a pull request

License

This project is licensed under the MIT License - see the LICENSE file for details.

Support

For issues and questions: 1. Check the troubleshooting section 2. Review AWS documentation 3. Open an issue in the repository 4. Contact Soumya Thakur (soumyathakur85@gmail.com)

Note: Always review and test configurations in a development environment before applying to production. Ensure you understand the AWS costs associated with the resources created by this infrastructure.