**CloudFormation Implementation**

This repository contains scripts and CloudFormation config files to deploy Django App on Amazon Web Services (AWS). Below is an overview of the architecture and services utilized in this deployment.

**Architecture Overview**

The deployment is based on a microservices architecture, leveraging various AWS services for scalability, reliability, and security

**Services Utilized**

* Virtual Private Cloud (VPC): Configured with public and private subnets spanning two availability zones for enhanced availability and fault tolerance.
* Internet Gateway: Enables connectivity between VPC instances and the wider internet.
* Security Groups: Acts as a network firewall mechanism to control traffic to and from instances.
* Availability Zones: Utilized for increased system reliability and fault tolerance.
* Public Subnets: Host infrastructure components like the NAT Gateway and Application Load Balancer.
* EC2 Instance Connect Endpoint: Facilitates secure connections to assets within both public and private subnets.
* Identity and Access Management (IAM): Used to create roles allowing EC2 instances to access and communicate with data files stored in S3.
* Amazon Relational Database Service (RDS): Provides database service for the application.
* Private Subnets: Hosts web servers (EC2 instances) and databases (RDS instances) for enhanced security.
* NAT Gateway: Allows instances in private subnets to access the internet.
* Application Load Balancer (ALB): Distributes web traffic evenly to an Auto Scaling Group of EC2 instances across multiple Availability Zones.
* Auto Scaling Group: Automatically manages EC2 instances for availability, scalability, fault tolerance, and elasticity.
* Simple Notification Service (SNS): Configured to alert about activities within the Auto Scaling Group.
* Amazon S3: Stores application code, migration scripts, and logs

**Deployment steps**

1. Save the templates above as files: vpc.yaml, iam\_s3.yaml, asg\_alb.yaml, rds.yaml, cloudwatch.yaml, and main-root.yaml.
2. Upload vpc.yaml, iam\_s3.yaml, asg\_alb.yaml, rds.yaml, cloudwatch.yaml to an S3 bucket in us-east-1 (make sure the bucket is accessible by CloudFormation; we recommend restricting access but ensure CloudFormation can read the TemplateURL).

Example:

aws s3 cp vpc.yaml s3://my-cf-templates/vpc.yaml

aws s3 cp iam\_s3.yaml s3://my-cf-templates/iam\_s3.yaml

aws s3 cp asg\_alb.yaml s3://my-cf-templates/asg\_alb.yaml

aws s3 cp rds.yaml s3://my-cf-templates/rds.yaml

aws s3 cp cloudwatch.yaml s3://my-cf-templates/cloudwatch.yaml

1. Edit main-root.yaml and replace each TemplateURL with the actual S3 URL (e.g. https://my-cf-templates.s3.amazonaws.com/vpc.yaml).
2. Create a CloudFormation stack using main-root.yaml:

aws cloudformation create-stack \

--stack-name django-prod-root \

--template-body file://main-root.yaml \

--parameters ParameterKey=StaticBucketName,ParameterValue=your-unique-static-bucket-name \

ParameterKey=DBPassword,ParameterValue='YourStrongDBPassw0rd' \

ParameterKey=KeyName,ParameterValue='DjangoKey.pem' \

ParameterKey=AlarmEmail,ParameterValue='ops@example.com' \

--capabilities CAPABILITY\_IAM CAPABILITY\_NAMED\_IAM

(You can also use AWS Console > CloudFormation and upload the root template and pass parameters there.)

1. After stack creation completes, fetch the ALB DNS from stack outputs (or console). Use that to verify traffic is reaching instances.
2. In your EC2 UserData or via SSM, deploy your Django code, set environment variables (database host from output django-rds-endpoint, DB port, DB user and password), run migrations, collectstatic (upload static files to S3), and start Gunicorn / uWSGI.

Example environment variables:

DATABASES={

'default': {

'ENGINE': 'django.db.backends.postgresql\_psycopg2',

'NAME': 'django\_db',

'USER': 'django\_user',

'PASSWORD': '<DBPassword you set>',

'HOST': '<RDS endpoint>',

'PORT': '<RDS port>'

}

}