AWS DevOps Project: Scalable E-Commerce Platform

Objective

Deploy a robust, scalable e-commerce platform using AWS services and DevOps best practices.

Architecture Overview

- **Frontend: React (deployed on Amazon S3 + CloudFront)**
- **Backend API: FastAPI (deployed on ECS Fargate)**
- **Database: Amazon RDS (PostgreSQL/MySQL)**
- **Authentication: Amazon Cognito**
- **Storage: Amazon S3 for product images and static files**
- **CI/CD: AWS CodePipeline + CodeBuild + CodeDeploy + SonarQube (for code quality)**
- **Infrastructure as Code (IaC): Terraform**
- **Monitoring & Logging: Amazon CloudWatch + Kibana**
- **Security: AWS WAF, IAM roles, and Security Groups**
- **Automation: Lambda functions for scheduled tasks**
- **Secrets Management: AWS Secrets Manager for secure credential handling**

Step 1: Infrastructure Setup (IaC with Terraform)

1. **Create VPC**
    - **Create a VPC with CIDR block 10.0.0.0/16**
2. **Add Subnets**
    - **Public subnet 1: 10.0.1.0/24**
    - **Public subnet 2: 10.0.2.0/24**
    - **Private subnet 1: 10.0.3.0/24**
    - **Private subnet 2: 10.0.4.0/24**
3. **Add Internet Gateway and NAT Gateway**
    - **Create an IGW and attach it to the VPC.**

- Add a NAT Gateway in the public subnet.

4. Create Route Tables
    - Public Route Table with IGW route.
    - Private Route Table with NAT Gateway route.
5. Deploy ECS Cluster with Fargate tasks
    - Use Terraform to create:
        - ECS Cluster
        - ECS Service with Fargate launch type
6. Create an ALB for routing traffic.
7. Launch RDS Instance for Database
    - Enable Multi-AZ for redundancy.
8. Deploy S3 Buckets for Storage
    - One bucket for static content and another for product images.

Step 2: Backend API Setup (FastAPI + Docker)

1. Write a Dockerfile for FastAPI:

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

2. Push the Docker image to Amazon ECR:

```
docker build -t fastapi-app .

docker tag fastapi-app:latest <AWS_ACCOUNT_ID>.dkr.ecr.<AWS_REGION>.amazonaws.com/fastapi-app:latest

docker push <AWS_ACCOUNT_ID>.dkr.ecr.<AWS_REGION>.amazonaws.com/fastapi-app:latest
```

3. Create an ECS Task Definition and ECS Service.

Step 3: Frontend Deployment (React + S3 + CloudFront)

1. **Build the React app:**

```
npm run build
```

2. **Upload build files to S3:**

```
aws s3 sync ./build s3://<YOUR_BUCKET_NAME> --delete
```

3. **Create a CloudFront distribution with your S3 bucket as the origin.**

Step 4: CI/CD Pipeline Setup

1. **Create a CodePipeline with the following steps:**

- **Source: Connect to your GitHub repository.**
- **Build: Use CodeBuild to build the backend Docker image and React app.**
- **Code Quality Check: Integrate SonarQube for code quality analysis.**
- **Deploy: ECS service for backend and S3 for frontend.**

2. **Sample buildspec.yml file for CodeBuild with SonarQube:**

```
version: 0.2
phases:
 install:
  commands:
   - echo "Installing dependencies..."
   - npm install
 build:
  commands:
   - npm run build
   - docker build -t fastapi-app .
   - sonar-scanner -Dsonar.projectKey=my_project -
Dsonar.host.url=<SONARQUBE_URL> -Dsonar.login=<SONARQUBE_TOKEN>
 post_build:
```

commands:

- docker push <AWS_ACCOUNT_ID>.dkr.ecr.<AWS_REGION>.amazonaws.com/fastapi-app:latest

- echo "Build complete."

Step 5: Database Setup (RDS)

1. **Launch a PostgreSQL or MySQL RDS instance.**
2. **Create a security group allowing ECS tasks to connect securely.**
3. **Use Secrets Manager to manage database credentials.**

Step 6: Monitoring & Logging

1. **Enable CloudWatch metrics for ECS service.**
2. **Create CloudWatch alarms for:**

- **High CPU utilization**
- **Memory usage spikes**
- **Failed ECS task deployments**

3. **Use Kibana for improved log visualization.**

Step 7: Security & Scaling

1. **Use AWS WAF for enhanced security.**
2. **Implement IAM policies with least-privilege access.**
3. **Enable ECS auto-scaling for dynamic resource allocation.**

Step 8: Automation with AWS Lambda

1. **Create a Lambda function to purge old Docker images in ECR.**
2. **Automate RDS backup using scheduled Lambda functions.**

Step 9: Documentation & Cleanup

1. **Write clear documentation on setup, deployment, and maintenance.**
2. **Use cleanup scripts for temporary resources.**

Bonus Enhancements

- ✅ **Add caching with Amazon ElastiCache (Redis)**
- ✅ **Implement Blue-Green Deployment for zero downtime**
- ✅ **Integrate Amazon SES for email notifications**
- ✅ **Use CloudFormation for faster replication**

Key Learning Outcomes

- ✅ **Hands-on with ECS, RDS, and S3**
- ✅ **Infrastructure as Code (IaC) using Terraform**
- ✅ **Efficient CI/CD pipeline using CodePipeline**
- ✅ **Enhanced code quality check with SonarQube integration**
- ✅ **Secure architecture with WAF, IAM, and VPC**
- ✅ **Real-time monitoring and alerting using CloudWatch and Kibana**