**QueenOge’s AWS VPC Multi-Tier Architecture Deployment Project Report**

### Project Overview

This project demonstrates the full deployment and automation of a secure multi-tier architecture on AWS using the AWS CLI and Bash scripting. The architecture consists of three layers (Web, App, DB), each in its own subnet, with strict access controls enforced through security groups.

The entire infrastructure is scripted and version-controlled, with connectivity verification and SSH access tests performed for each EC2 instance.

### 1. AWS CLI Configuration

**Command:**

aws configure --profile queenoge

**Profile Configuration:**

* AWS Access Key ID: [Provided by user]
* AWS Secret Access Key: [Provided by user]
* Default region: us-east-1
* Output format: json

All subsequent CLI operations used the queenoge named profile.

### 2. Folder Structure for GitHub Repository

**Command Used:**

mkdir -p vpc-multi-tier-architecture-queenoge/{scripts,keys,screenshots}  
cd vpc-multi-tier-architecture-queenoge

* scripts/ — All Bash scripts used in deployment
* keys/ — Contains PEM key pair for SSH
* screenshots/ — Proof of connectivity tests and AWS Console views

### 3. VPC and Subnets Creation

**Script:** scripts/create-vpc.sh

**Resources Created:**

* VPC CIDR: 10.0.0.0/16
* Subnets:
  + Web: 10.0.1.0/24 (Public)
  + App: 10.0.2.0/24 (Private)
  + DB: 10.0.3.0/24 (Private)
* Internet Gateway and route table for Web subnet
* Route table associations for each subnet

### 4. Key Pair Generation

**Command:**

aws ec2 create-key-pair \  
 --key-name QueenOge-KeyPair \  
 --query 'KeyMaterial' \  
 --output text \  
 --profile queenoge > keys/QueenOge-KeyPair.pem  
chmod 400 keys/QueenOge-KeyPair.pem

**Purpose:** Allows secure SSH access to EC2 instances.

### 5. Security Group Setup

**Script:** scripts/create-security-groups.sh

**Security Groups Created:**

* **Web SG:**
  + Inbound: TCP 22 (SSH), TCP 80 (HTTP) from anywhere
* **App SG:**
  + Inbound: TCP 80 (HTTP) from Web SG
* **DB SG:**
  + Inbound: TCP 3306 (MySQL) from App SG

**All outbound rules** allow all traffic by default.

### 6. EC2 Instance Deployment

**Script:** scripts/launch-ec2.sh

**Instances Launched:**

* **Web Server**:
  + Public IP: 100.27.229.251
  + Private IP: 10.0.1.x
* **App Server**:
  + Private IP: 10.0.2.179
* **DB Server**:
  + Private IP: 10.0.3.226

**Details:**

* AMI: Amazon Linux 2 (ami-0c101f26f147fa7fd)
* Type: t2.micro
* Key: QueenOge-KeyPair

### 7 Connectivity and Access Testing

#### ✅ SSH to Web Server

ssh -i keys/QueenOge-KeyPair.pem ec2-user@100.27.229.251

✅ Connected successfully

#### ✅ Ping from Web to App

ping 10.0.2.179

✅ Ping passed

#### ✅ Ping from App to DB (via jump host)

ping 10.0.3.226

✅ Ping passed

### 8. Full Automation Script

**Script:** scripts/full-deploy.sh

Combines all tasks:

* VPC setup
* Subnet creation
* Security groups
* Key pair
* EC2 provisioning

One command to launch the full infrastructure from scratch.

### 9. Design Choices

* Used **three-tier** model for better separation of concerns.
* CLI scripting for **repeatable, scalable deployment**.
* Used **jump host method** (Web instance) to access private instances.
* Implemented **least privilege** with security groups.

### ️ Challenges and Solutions

| Challenge | Solution |
| --- | --- |
| No public IP for App/DB | Accessed via SSH jump host |
| SSH timeout errors | Manually terminated EC2s via Console |
| “Connection timed out” | Fixed SG rules to allow necessary traffic |
| Directory errors during CLI | Adjusted path and verified file locations |

### Cleanup

Some resources (like EC2s) were terminated manually due to timeout errors from the command line.

### Final Outcome

| Component | Status |
| --- | --- |
| VPC, Subnets | ✅ Created |
| Security Groups | ✅ Functional |
| EC2 Instances | ✅ Launched |
| Connectivity Tests | ✅ Passed |
| Full Automation | ✅ Achieved |

### Deliverables

* Scripts for infrastructure and instance provisioning
* Screenshots of all subnet communications
* Full project structure ready for GitHub:
  + scripts/
  + keys/
  + screenshots/
* Detailed documentation (this report)
* Verified connectivity and tier isolation

### Conclusion

The project showcases a complete cloud infrastructure setup using best practices. From initial configuration to secure access, all layers were deployed, tested, and automated. This setup is ideal for production-ready multi-tier applications and serves as a foundation for future CI/CD or container orchestration integrations.

**Submitted by:** Anyalewechi Esther

**Project:** Deploy a VPC Multi-Tier Architecture and Enforce Access Control