Three-Tier Network Architecture Setup in AWS

July 14, 2025

Contents

1 Introduction

This document outlines the configuration of a three-tier network architecture in Amazon Web Services (AWS) for an internship assignment. The architecture consists of three subnets: Web Tier (public), App Tier (private), and DB Tier (private). Specific access rules are enforced: the Web Tier accesses only the App Tier, the App Tier accesses both the Web and DB Tiers, and the DB Tier cannot initiate connections to any tier. Only the Web Tier has internet access. Each tier hosts two virtual machines (VMs): one Linux with Apache and one Windows with IIS.

2 Architecture Overview

The three-tier architecture separates the application into presentation (Web Tier), application logic (App Tier), and data storage (DB Tier). This setup enhances scalability, security, and isolation.

- **Web Tier**: Public subnet, accessible from the internet, hosts web servers.
- **App Tier**: Private subnet, processes application logic, communicates with Web and DB Tiers.
- **DB Tier**: Private subnet, hosts data services, accessible only by App Tier.

3 VPC and Subnet Configuration

3.1 Creating the VPC

1. Navigate to the AWS Management Console > VPC Dashboard > Your VPCs.
2. Click *Create VPC*.
3. Set:
    - Name Tag: `demo-vpc`
    - IPv4 CIDR Block: `10.0.0.0/16`
4. Click *Create VPC*.

3.2 Creating Subnets

1. In the VPC Dashboard, under *Subnets*, click *Create Subnet*.
2. Create three subnets in `demo-vpc`:
    - **Web Tier Subnet**:
        - Name Tag: `web-public`
        - Availability Zone: `us-east-2a`
        - IPv4 CIDR Block: `10.0.1.0/24`

- **App Tier Subnet**:
  - Name Tag: `app-private`
  - Availability Zone: `us-east-2a`
  - IPv4 CIDR Block: `10.0.2.0/24`
- **DB Tier Subnet**:
  - Name Tag: `db-private`
  - Availability Zone: `us-east-2a`
  - IPv4 CIDR Block: `10.0.3.0/24`

4 Internet Gateway and NAT Gateway

4.1 Internet Gateway

1. In the VPC Dashboard, under *Internet Gateways*, click *Create Internet Gateway*.
2. Name Tag: `demo-igw`.
3. Click *Create*.
4. Select `demo-igw`, click *Actions* > *Attach to VPC*, choose `demo-vpc`.

4.2 NAT Gateway

1. In the VPC Dashboard, under *NAT Gateways*, click *Create NAT Gateway*.
2. Set:
   - Subnet: `web-public`
   - Elastic IP: Allocate a new Elastic IP.
3. Click *Create NAT Gateway*.

5 Route Tables

1. In the VPC Dashboard, under *Route Tables*, click *Create Route Table*.
2. Create three route tables:
   - **Web Tier Route Table**:
     - Name Tag: `web-rt`
     - VPC: `demo-vpc`
     - Add Route: Destination `0.0.0.0/0`, Target `demo-igw`
   - **App Tier Route Table**:

- Name Tag: app-rt
- VPC: demo-vpc
- Add Route: Destination 0.0.0.0/0, Target NAT Gateway

- **DB Tier Route Table**:
  - Name Tag: db-rt
  - VPC: demo-vpc
  - No additional routes (only local 10.0.0.0/16)

3. Associate subnets:
   - web-rt with web-public
   - app-rt with app-private
   - db-rt with db-private

6 Security Groups

Table 1: Security Group Rules

Security Group	Inbound Rules	Outbound Rules
Web Tier (web-sg)	HTTP (80) from 0.0.0.0/0 HTTP (80) from app-sg	HTTP (80) to app-sg
App Tier (app-sg)	HTTP (80) from web-sg	HTTP (80) to db-sg HTTP (80) to web-sg
DB Tier (db-sg)	HTTP (80) from app-sg	None (all outbound denied)

1. In the VPC Dashboard, under *Security Groups*, create:
   - **Web Tier SG** (web-sg):
     - Inbound: HTTP (80) from 0.0.0.0/0, HTTP (80) from app-sg
     - Outbound: HTTP (80) to app-sg
   - **App Tier SG** (app-sg):
     - Inbound: HTTP (80) from web-sg
     - Outbound: HTTP (80) to db-sg, HTTP (80) to web-sg
   - **DB Tier SG** (db-sg):
     - Inbound: HTTP (80) from app-sg
     - Outbound: None (remove default allow-all rule)

7 Virtual Machine Deployment

7.1 Web Tier

- **Linux VM**:
  1. Launch EC2 instance:
     - AMI: Amazon Linux 2
     - Instance Type: `t2.micro`
     - Network: VPC: `demo-vpc`, Subnet: `web-public`
     - Auto-assign Public IP: Yes
     - Security Group: `web-sg`
  2. Connect via SSH and configure Apache:

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
echo ”Web Tier Linux” | sudo tee /var/www/html/index.html
```

- **Windows VM**:
  1. Launch EC2 instance:
     - AMI: Windows Server 2019
     - Instance Type: `t2.micro`
     - Network: VPC: `demo-vpc`, Subnet: `web-public`
     - Auto-assign Public IP: Yes
     - Security Group: `web-sg`
  2. Connect via RDP, open Server Manager, add *Web Server (IIS)* role.
  3. Create `index.html` in `C:` with content: ”Web Tier Windows”.

7.2 App Tier

- **Linux VM**:
  1. Launch EC2 instance:
     - AMI: Amazon Linux 2
     - Instance Type: `t2.micro`
     - Network: VPC: `demo-vpc`, Subnet: `app-private`
     - Auto-assign Public IP: No

- Security Group: `app-sg`

2. Connect via SSH (use EC2 Instance Connect or a bastion host in Web Tier):

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
echo "App Tier Linux" | sudo tee /var/www/html/index.html
```

- **Windows VM**:

1. Launch EC2 instance:
   - AMI: Windows Server 2019
   - Instance Type: `t2.micro`
   - Network: VPC: `demo-vpc`, Subnet: `app-private`
   - Auto-assign Public IP: No
   - Security Group: `app-sg`
2. Connect via RDP (use EC2 Instance Connect or bastion host), add *Web Server (IIS)* role.
3. Create `index.html` in `C:` with content: "App Tier Windows".

7.3 DB Tier

- **Linux VM**:

1. Launch EC2 instance:
   - AMI: Amazon Linux 2
   - Instance Type: `t2.micro`
   - Network: VPC: `demo-vpc`, Subnet: `db-private`
   - Auto-assign Public IP: No
   - Security Group: `db-sg`
2. Connect via SSH (use EC2 Instance Connect or bastion host):

```
sudo yum update -y
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
echo "DB Tier Linux" | sudo tee /var/www/html/index.html
```

- **Windows VM**:

1. Launch EC2 instance:
   - AMI: Windows Server 2019
   - Instance Type: `t2.micro`
   - Network: VPC: `demo-vpc`, Subnet: `db-private`
   - Auto-assign Public IP: No
   - Security Group: `db-sg`
2. Connect via RDP (use EC2 Instance Connect or bastion host), add *Web Server (IIS)* role.
3. Create `index.html` in `C:` with content: ”DB Tier Windows”.

8 Verification

1. **Web Tier Access**:
   - From a browser, access the public IPs of Web Tier VMs on port 80.
   - Expected: ”Web Tier Linux” or ”Web Tier Windows” displayed.
2. **Web to App Tier**:
   - From a Web Tier VM, use `curl <App Tier VM private IP>:80`.
   - Expected: ”App Tier Linux” or ”App Tier Windows” displayed.
3. **App to DB Tier**:
   - From an App Tier VM, use `curl <DB Tier VM private IP>:80`.
   - Expected: ”DB Tier Linux” or ”DB Tier Windows” displayed.
4. **App to Web Tier**:
   - From an App Tier VM, use `curl <Web Tier VM private IP>:80`.
   - Expected: ”Web Tier Linux” or ”Web Tier Windows” displayed.
5. **DB Tier Restrictions**:
   - From a DB Tier VM, attempt `curl <Web or App Tier VM private IP>:80`.
   - Expected: Connection fails due to no outbound rules in `db-sg`.

9 Additional Notes

- For production, consider using multiple Availability Zones for high availability.
- Use AWS Systems Manager Session Manager for secure access to private subnet VMs.

- Monitor costs, as NAT Gateways and Elastic IPs incur charges.
- For a database tier, typically, a managed service like Amazon RDS would be used instead of web servers.

10 References

- AWS VPC Documentation: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html
- AWS Security Groups: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html
- AWS EC2 Documentation: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
- SUDO Consultants: https://sudoconsultants.com/crafting-a-web-architectu
- Stratus10 AWS Best Practices: https://stratus10.com/blog/aws-best-practices