**AWS 3-Tier Architecture: Student Registration Deployment**

**1. Cover Page**

**Project Title:** AWS 3-Tier Architecture: Student Registration Deployment  
**Author:** Rushikesh Dase  
**Environment:** AWS Cloud, Linux-based instances  
**Objective:** Build a secure, scalable student registration system using a LEMP stack with a 3-tier architecture.

**2. Abstract / Objective**

his project demonstrates the deployment of a **3-Tier Student Registration Web Application** on **Amazon Web Services (AWS)**, leveraging a modern cloud architecture approach. The system is designed to be secure, scalable, and modular, following best practices in cloud infrastructure and multi-tier application design.

The main objectives of this project are:

* **To gain hands-on experience with core AWS networking components**, including **Virtual Private Clouds (VPCs), subnets, route tables, security groups, and NAT gateways**. This ensures a practical understanding of how cloud networks are structured and secured.
* **To design and deploy a robust web application stack** using **Linux, Nginx, PHP, and MySQL (LEMP stack)**. This involves configuring web servers, application servers, and databases across separate subnets to maintain security and performance.
* **To understand the end-to-end interaction between the Web, Application, and Database tiers**, including request flow, data handling, and secure communication. This helps reinforce the principles of multi-tier architecture and separation of concerns, which are critical in real-world enterprise systems.
* **To develop problem-solving skills in cloud deployment scenarios**, such as handling private subnet connectivity, proxy configurations, and cost-effective network design, while ensuring the application remains accessible and functional.
* **To build a foundation for further cloud architecture learning**, including automation, load balancing, and monitoring, by providing a working, practical example of a fully functional 3-tier deployment.

Through this project, learners can bridge the gap between theoretical cloud concepts and practical implementation, gaining experience that directly translates to real-world cloud engineering and DevOps practices.

**3. Architecture Overview**

The application follows the **classic 3-Tier model**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tier** | **Component** | **Function** | **Network Placement** |
| Web Tier | Nginx + HTML | Handles frontend requests | Public Subnet |
| Application Tier | PHP | Processes backend logic | Private Subnet |
| Database Tier | MySQL | Stores student data securely | Private Subnet |

**Flow:**  
**User → Web Server → App Server → Database Server**

**Key Features:**

* Separate layers for modularity and security.
* Private subnets for backend and database instances.
* Public subnet for the web interface accessible to users.

**4. Implementation Steps**

**4.1 AWS Infrastructure Setup**

1. **Create Custom VPC**: CIDR 10.0.0.0/16.
2. **Create 3 Subnets**:
   * web-subnet (Public, AZ1)
   * app-subnet (Private, AZ2)
   * db-subnet (Private, AZ3)
3. **Attach Internet Gateway** to the VPC.
4. **Create Route Tables**:
   * Public route table → attach IGW → associate with web-subnet
   * Private route table → attach NAT Gateway → associate with app-subnet and db-subnet
5. **Launch EC2 Instances**:
   * Web Server → web-subnet, public IP, Security Group web-sg (HTTP:80 open)
   * App Server → app-subnet, no public IP, Security Group app-sg (HTTP:80 from web-sg)
   * DB Server → db-subnet, no public IP, Security Group db-sg (MySQL:3306 from app-sg)

**4.2 Web Server Setup**

# Connect to web server

ssh -i s.linux.pem ec2-user@<WEB-PUBLIC-IP>

# Install Nginx

sudo yum install nginx -y

sudo systemctl start nginx

sudo systemctl enable nginx

# Upload HTML form

cd /usr/share/nginx/html

nano form.html

# Paste form.html code and save

# Configure reverse proxy for PHP

sudo nano /etc/nginx/nginx.conf

location ~ \.php$ {

proxy\_pass http://<APP-PRIVATE-IP>;

}

sudo systemctl restart nginx

sudo systemctl reload nginx

**4.3 Application Server Setup**

# Connect via web-server as jump host

ssh -i s.linux.pem ec2-user@<APP-PRIVATE-IP>

# Install PHP and dependencies

sudo yum install php php-mysqlnd mariadb105-server -y

sudo systemctl enable php-fpm

sudo systemctl start php-fpm

# Upload PHP handler

cd /usr/share/nginx/html

nano submit.php

# Paste submit.php code

**4.4 Database Server Setup**

# Connect to DB server

ssh -i s.linux.pem ec2-user@<DB-PRIVATE-IP>

# Install MySQL

sudo yum install mariadb105-server -y

sudo systemctl enable mariadb

sudo systemctl start mariadb

# Configure database and user

mysql -u root -p

CREATE DATABASE studentdb;

CREATE USER 'rushi'@'<APP-PRIVATE-IP>' IDENTIFIED BY 'dase';

GRANT ALL PRIVILEGES ON studentdb.\* TO 'rushi'@'<APP-PRIVATE-IP>';

FLUSH PRIVILEGES;

# Create table

USE studentdb;

CREATE TABLE students (

roll\_no INT AUTO\_INCREMENT PRIMARY KEY,

fullname VARCHAR(50),

email VARCHAR(50),

phone BIGINT(11),

course VARCHAR(20)

);

**5. Security and Network Configuration**

**Security Groups:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Server** | **Allowed Source** | **Ports** | **Purpose** |
| Web SG | 0.0.0.0/0 | 80 (HTTP) | Public access |
| App SG | Web SG | 80 (HTTP) | Allow web tier requests |
| DB SG | App SG | 3306 (MySQL) | Allow app tier DB access |

**Subnets & Route Tables:**

* Public subnet → IGW → web traffic
* Private subnet → NAT Gateway → app/db internet access if needed

**Proxy:**

* PHP requests routed from Web Server → App Server (for private IP access)

**6. Testing & Verification**

1. Open browser: http://<WEB-PUBLIC-IP>/form.html
2. Fill registration form → submit
3. Verify database entries on DB server:

SELECT \* FROM students;

Note – after the implementation Delete the Nat gateway and relese the Elstic IP

**Success Criteria:**

* Data entered in form appears in MySQL table.
* Web interface responds correctly via Nginx and PHP.

**7. Challenges / Learnings**

* **NAT Gateway Costs:** Initially considered NAT for private servers; solved with proxy through web server.
* **Private Server PHP Testing:** Needed reverse proxy because private instances cannot have public IPs.
* **Security Groups:** Understanding traffic flow between tiers took careful planning.
* **SSH Chaining:** Accessing app/db servers required jump host configuration.

**Key Learnings:**

* Importance of network segmentation for security.
* Hands-on experience with multi-tier architecture on AWS.
* Real-world problem-solving for cost and connectivity constraints.

**8. Future Enhancements**

* Add **Elastic Load Balancer (ELB)** for high availability.
* Integrate **SSL/TLS** via AWS Certificate Manager.
* Automate deployments with **Terraform** or **CloudFormation**.
* Monitor infrastructure with **AWS CloudWatch**.
* Implement **RDS** for managed database service.