

# Three-Tier Project Deployment on AWS (Student Registration Form)

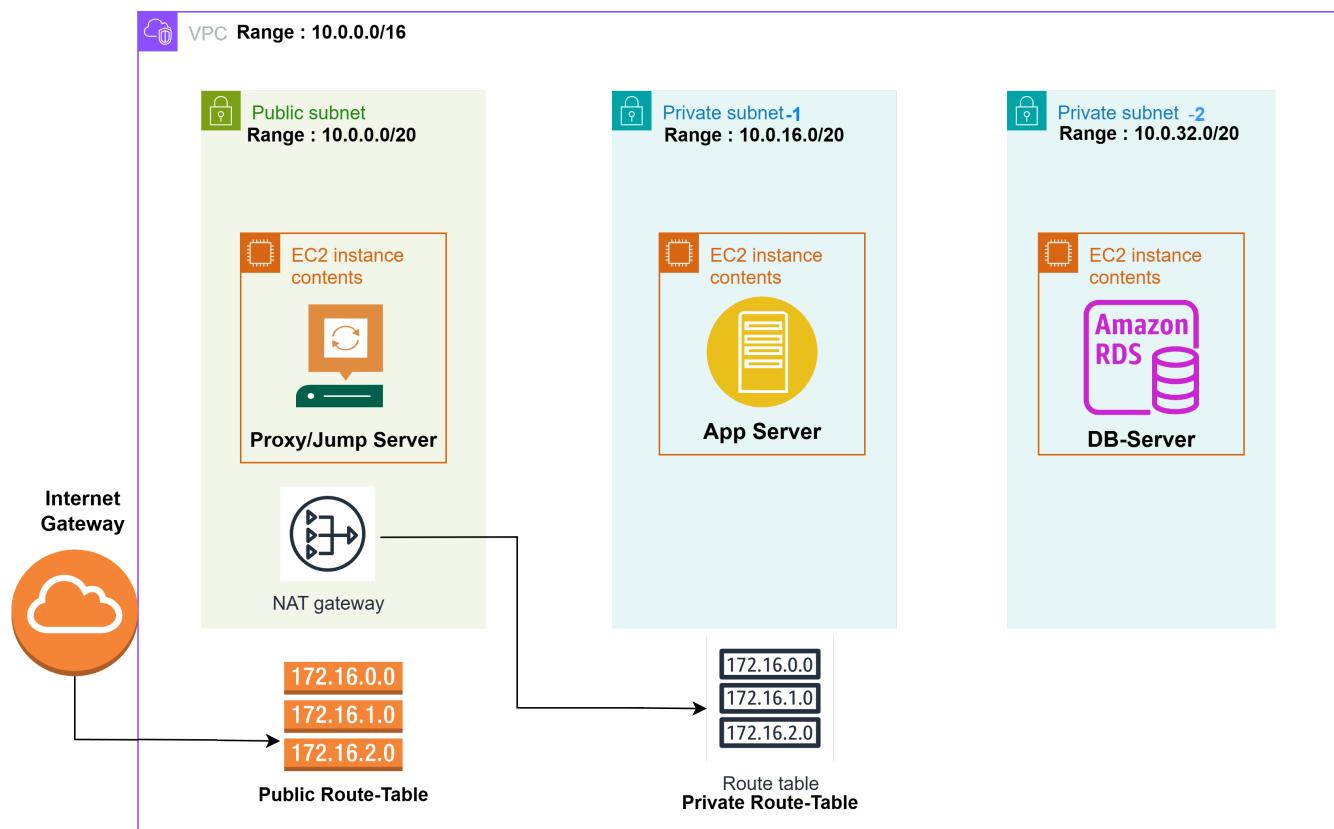
---

## Overview

I deployed a **3-Tier Student Registration Web Application** on AWS using a complete **VPC** setup with **NGINX** as a reverse proxy, **Tomcat** as the application server, and **MariaDB on RDS** as the database layer.

- The architecture is divided into three layers:
    - Presentation Layer** – Proxy Server (Public Subnet)
    - Application Layer** – App Server (Private Subnet-1)
    - Database Layer** – RDS & DB Server (Private Subnet-2)
  - It ensures **high availability, security, and scalability** by distributing components across multiple availability zones and using appropriate networking configurations.
- 

## Three-Tier Architecture



- The **public subnet** only hosts the proxy (exposed to the internet).
- The **app and DB subnets** remain private for security.
- A **NAT Gateway** allows private EC2s to download updates without public exposure.

### 1. Presentation Tier (Proxy Layer)

- Hosted in a public subnet

- Uses NGINX as a reverse proxy
- Handles all incoming HTTP traffic
- Also works as a Jump Server (Bastion Host) to connect securely to private instances

## 2. Application Tier (Private Subnet)

- Runs Apache Tomcat
- Hosts the Java-based Student Registration Web Application
- Connects to the database through JDBC

## 3. Database Tier (Private Subnet)

- Uses Amazon RDS (MariaDB)
- Stores student registration data
- Only accessible from the Application Tier (port 3306)"

# Steps to deploy the Student Application Form :

## 1. Create the VPC

### 1. Create VPC:

- Name: **three-tier-VPC**
- CIDR: **10.0.0.0/16**

```
# The VPC defines your isolated network in AWS.
# The `/16` block provides enough IPs to subdivide into multiple subnets.
# (Optional) Tag the VPC so you can filter easily in the console.
```

Name	VPC ID	State	Block Public...	IPv4 CIDR	IPi
Three-Tier-Project	vpc-036fbede387b0723d	Available	Off	10.0.0.0/16	-

## 2. Create subnets (different AZs for High Availability)

### 1. public-subnet

- CIDR: **10.0.0.0/20**

- AZ: **us-east-1a**
- Enable auto-assign public IPv4 if you want EC2 with public IP.

## 2. private-subnet-1

- CIDR: **10.0.16.0/20**
- AZ: **us-east-1b**

## 3. private-subnet-2

- CIDR: **10.0.32.0/20**
- AZ: **us-east-1c**

```
# Public subnet : hosts internet-facing resources (proxy, NAT).
# Private subnets : isolate app and DB tiers for security and compliance.
```

Subnets (3) <a href="#">Info</a>					
<a href="#">Actions</a> ▾ <a href="#">Create subnet</a>					
<span>Last updated less than a minute ago</span>					
<span><input type="checkbox"/> Show more (+1)</span> <span><a href="#">Clear filters</a></span>					
Name	Subnet ID	State	VPC	Block Public.	
Public-subnet	<a href="#">subnet-0ffca66d799e19df4</a>	<span>Available</span>	<a href="#">vpc-036fbebe387b0723d   Thre...</a>	<span>Off</span>	<span>Off</span>
Private-subnet-2	<a href="#">subnet-0e5298955331d8118</a>	<span>Available</span>	<a href="#">vpc-036fbebe387b0723d   Thre...</a>	<span>Off</span>	<span>Off</span>
Private-subnet-1	<a href="#">subnet-061d7a1eaf84631a3</a>	<span>Available</span>	<a href="#">vpc-036fbebe387b0723d   Thre...</a>	<span>Off</span>	<span>Off</span>

- Ensure different AZs are used for high availability.
- Ensure each private subnet is associated with the private route table later.

## 3. Internet Gateway (IGW) + Public Route Table

### 1. Create an Internet Gateway:

- Name: **three-tier-IGW**

**Create internet gateway** [Info](#)

An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.

<b>Internet gateway settings</b>	
<b>Name tag</b> Creates a tag with a key of 'Name' and a value that you specify. <input type="text" value="Three-Tier-IGW"/>	
<b>Tags - optional</b> A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.	
<b>Key</b> <input type="text" value="Name"/>	<b>Value - optional</b> <input type="text" value="Three-Tier-IGW"/>
<a href="#">Add new tag</a> <small>You can add 49 more tags.</small>	

- Attach it to **three-tier-VPC**.

```
# Enables public subnets to access the Internet.
```

## 2. Create / or edit Route Table for public traffic:

- Rename route table to **public-RT** (or create new).
- Edit routes: add **0.0.0.0/0** → target **three-tier-IGW**.

- **(Optional)** Subnet association: associate **public-RT** with **public-subnet**.

```
# Public-RT → route `0.0.0.0/0` → IGW → associate with `public-subnet`.
```

## 4. NAT Gateway + Private Route Table

### 1. Create a private route table:

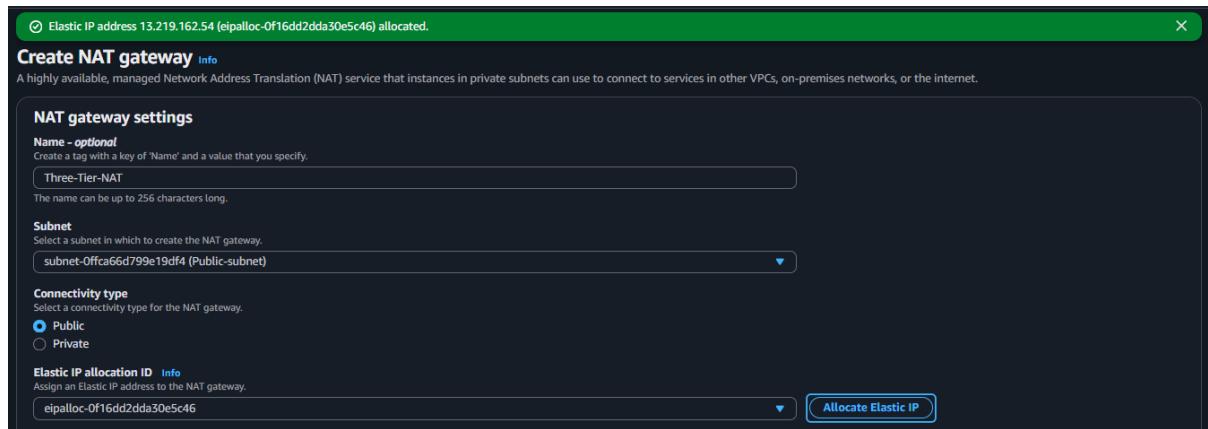
- Name: **private-RT**
- Associate **private-RT** with **private-subnet-1** and **private-subnet-2**.

```
# Private-RT → route `0.0.0.0/0` → NAT Gateway → associate with private subnets.
```

### 2. Create NAT Gateway:

- Name: **three-tier-NAT**
- Subnet: **public-subnet**

- Allocate an **Elastic IP** for it.



```
# Lets private EC2s download updates or dependencies from the internet securely.
```

### 3. Edit **private-RT** routes:

- Add **0.0.0.0/0** → target **three-tier-NAT**.

```
# This allows instances in private subnets to access the Internet for
updates/patches without public IPs.
```

## 5. Security Group (three-tier-SG)

- Name:** **three-tier-SG**
- Description:** Allow SSH, HTTP, MySQL, and custom TCP (8080)
- VPC:** **three-tier-VPC**
- Inbound Rules:**
  - SSH: Anywhere (IPv4)
  - HTTP: Anywhere (IPv4)
  - MySQL/Aurora: Custom TCP 3306
  - Custom TCP: 8080

```
# Security groups act as firewalls controlling inbound/outbound traffic.
```

## 6. Launch Instances

### 6.1 Proxy Server (public)

- Name:** Proxy Server
- AMI:** Amazon Linux 2 (A1)

- **VPC:** three-tier-VPC
- **Subnet:** public-subnet
- **SG:** three-tier-SG

## 6.2 App Server (private)

- **Name:** App Server
- **AMI:** Amazon Linux 2 (A1)
- **VPC:** three-tier-VPC
- **Subnet:** private-subnet-1
- **SG:** three-tier-SG

## 6.3 DB Server (private)

- **Name:** DB Server
- **AMI:** Amazon Linux 2 (A1)
- **VPC:** three-tier-VPC
- **Subnet:** private-subnet-2
- **SG:** three-tier-SG

Instances (3) <a href="#">Info</a>										
<a href="#">Find Instance by attribute or tag (case-sensitive)</a> <span>Last updated  less than a minute ago</span> <span><a href="#">Connect</a></span> <span><a href="#">Instance state ▾</a></span> <span><a href="#">Actions ▾</a></span> <span><a href="#">Launch instances</a> ▾</span>										
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv6	
<input type="checkbox"/>	Proxy-Server	i-04b5db04d9c063fc	Running	t3.micro	3/3 checks passed <a href="#">View alarms</a> +	3/3 checks passed <a href="#">View alarms</a> +	us-east-1a	-	52.90.20.	
<input type="checkbox"/>	App-Server	i-043cc6dacc85f4efc	Running	t3.micro	3/3 checks passed <a href="#">View alarms</a> +	3/3 checks passed <a href="#">View alarms</a> +	us-east-1b	-	-	
<input type="checkbox"/>	Db-Server	i-0da3ccb1fafcd360	Running	t3.micro	3/3 checks passed <a href="#">View alarms</a> +	3/3 checks passed <a href="#">View alarms</a> +	us-east-1c	-	-	

## 7. RDS MariaDB (managed DB)

### 1. Create DB instance:

- **Type:** Standard Create
- **Engine:** MariaDB
- **DB Identifier:** **three-tier-RDS**
- **Authentication:** Auto-generate password (store securely)
- **VPC:** **three-tier-VPC**
- **Security Group:** **three-tier-SG**
- **Availability Zone:** **us-east-1c** (or use Multi-AZ for higher availability)

## 2. Note the **RDS endpoint and saved password** — you'll use this in the app **DB config**.

The screenshot shows the AWS RDS 'Databases' page. A green success message at the top says 'Successfully created database three-tier-rds'. It includes instructions to view connection details and modify the DB instance. Below the message is a table with one row for the database 'three-tier-rds'. The table columns are: DB identifier, Status, Role, Engine, Region, and Size. The database is listed as 'Available' and is a 'MariaDB' instance in the 'us-east-1c' region, size 'db.t4g.micro'.

DB identifier	Status	Role	Engine	Region ...	Size
three-tier-rds	Available	Instance	MariaDB	us-east-1c	db.t4g.micro

```
# RDS can replace the DB EC2 instance for managed performance and backups.
```

## 8. SSH & Setup Proxy Server (nginx)

- SSH in (from your workstation)

```
ssh -i <key.pem> ec2-user@<proxy-public-ip>
sudo hostnamectl hostname proxy
exit

# SSH :Connect to the public EC2 using your SSH key.
# hostnamectl : sets a logical name for the host.
```

```
sadguru@DESKTOP-8JENEDM MINGW64 /c/Poonam AWS/SSh-key
$ ssh -i "north-v-key2.pem" ec2-user@52.90.20.169
The authenticity of host '52.90.20.169 (52.90.20.169)' can't be established.
ED25519 key fingerprint is SHA256:RXXihhzKF1wfByVTnk7v0GB4bCvj02dNLtBA+81KnQE.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.90.20.169' (ED25519) to the list of known hosts.

      #_
      ~\_ #####_          Amazon Linux 2023
      ~~ \#####\
      ~~  \###|
      ~~   \#/ __  https://aws.amazon.com/linux/amazon-linux-2023
      ~~    \~' '-'>
      ~~~      /
      ~~.~.     /
      ~~/_/_/
      _/m/'|_/

[ec2-user@ip-10-0-3-114 ~]$ sudo hostnamectl hostname proxy-server
[ec2-user@ip-10-0-3-114 ~]$ exit
Logout
Connection to 52.90.20.169 closed.

sadguru@DESKTOP-8JENEDM MINGW64 /c/Poonam AWS/SSh-key
$ ssh -i "north-v-key2.pem" ec2-user@52.90.20.169
      #_
      ~\_ #####_          Amazon Linux 2023
      ~~ \#####\
      ~~  \###|
      ~~   \#/ __  https://aws.amazon.com/linux/amazon-linux-2023
      ~~    \~' '-'>
      ~~~      /
      ~~.~.     /
      ~~/_/_/
      _/m/'|__

Last login: Wed Oct 29 14:55:41 2025 from 157.32.139.245
[ec2-user@proxy-server ~]$ |
```

- Update and Install Nginx:

```
sudo yum update
sudo yum install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx

# yum update : ensures system packages are current.
# systemctl : starts and auto-enables NGINX at boot.
```

- Edit nginx config to proxy to app server

- Open the nginx config (or site conf):

```
cd /etc/nginx/
sudo vim nginx.conf
```

- Add inside server block:

```

    location / {
        proxy_pass http://<app-server-private-ip>:8080/student/;
    }
    # NGINX forwards external HTTP traffic to the private app server's internal IP on
    port 8080.

```

```

include /etc/nginx/conf.d/*.conf;

server {
    listen      80;
    listen      [::]:80;
    server_name ;
    root       /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    error_page 404 /404.html;
    location = /404.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }

    location /{
        proxy_pass http://10.0.21.19:8080/student/;
    }
}

```

- Restart Nginx:

```
sudo systemctl restart nginx
```

## 9. Transfer private key to proxy (if you plan to SSH into app via proxy)

- From your local machine:

```

scp -i <key.pem> <key.pem> ec2-user@<proxy-public-ip>:/home/ec2-user/
# Transfers key to proxy for secure access into private EC2 via bastion model.

```

```

sadguru@DESKTOP-8JENEDM MINGW64 /c/Poonam AWS/ssh-key
$ scp -i north-v-key2.pem north-v-key2.pem ec2-user@52.90.20.169:/home/ec2-user/
north-v-key2.pem                                         100% 1678      4.9KB/s   0
sadguru@DESKTOP-8JENEDM MINGW64 /c/Poonam AWS/ssh-key
$ |

```

## 10. SSH to App Server (via proxy or using SSM)

- From proxy (if using private key on proxy):

```
sudo ssh -i <key.pem> ec2-user@<app-server-private-ip>
sudo hostnamectl hostname app-server
exit
```

- SSH again to app-server :

## 11. Install Java and Tomcat on App Server

- Install Java

```
sudo yum update -y
sudo yum install java -y
java -version
# `yum install java` installs the JRE runtime for Tomcat.
```

- Download and extract Tomcat (example using curl)

1. On Tomcat website pick the **apache-tomcat-9.x** core tar.gz and copy its link.
2. On the app server:

```
cd /opt
sudo wget <tomcat-core-tar.gz-link>
sudo tar -xvzf apache-tomcat-9.x.xx.tar.gz -C /opt/
# `curl` downloads the Tomcat package.
```

```
[ec2-user@app-server ~]$ sudo wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.111/bin/apache-tomcat-9.0.111.tar.gz
--2025-10-29 15:14:42-- https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.111/bin/apache-tomcat-9.0.111.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13037002 (12M) [application/x-gzip]
Saving to: 'apache-tomcat-9.0.111.tar.gz'

apache-tomcat-9.0.111.tar 100%[=====] 12.43M --.-KB/s   in 0.1s
2025-10-29 15:14:43 (118 MB/s) - 'apache-tomcat-9.0.111.tar.gz' saved [13037002/13037002]

[ec2-user@app-server ~]$ ls
apache-tomcat-9.0.111.tar.gz
```

- Start Tomcat

```
cd /opt/<apache-tomcat-folder>/bin
./catalina.sh start
# `catalina.sh start` launches the Tomcat service.
```

## 12. Deploy Student Application

- On the app server:

```
cd /opt/<apache-tomcat-folder>/webapps/
```

- **download the WAR provided by your source (e.g., from a PDF link or artifact storage)**

```
wget <link-to-studentapp.war>
```

```
[root@app-server bin]# cd ..
[root@app-server apache-tomcat-9.0.111]# cd webapps/
[root@app-server webapps]# wget https://s3-us-west-2.amazonaws.com/studentapi-cit/student.war
--2025-10-29 15:36:39-- https://s3-us-west-2.amazonaws.com/studentapi-cit/student.war
Resolving s3-us-west-2.amazonaws.com (s3-us-west-2.amazonaws.com)... 52.218.132.104, 52.92.227.128, 52.92.212.40, ...
Connecting to s3-us-west-2.amazonaws.com (s3-us-west-2.amazonaws.com)|52.218.132.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 89423 (87K) [binary/octet-stream]
Saving to: 'student.war'

student.war          100%[=====] 87.33K --.-KB/s   in 0.1s

2025-10-29 15:36:40 (682 kB/s) - 'student.war' saved [89423/89423]

[root@app-server webapps]# ls
ROOT  docs  examples  host-manager  manager  student  student.war
[root@app-server webapps]# cd ..
[root@app-server apache-tomcat-9.0.111]# |
```

- restart Tomcat to pick up the WAR

```
cd .. /bin/
./catalina.sh stop
./catalina.sh start
```

- After Tomcat starts, app should be available on port **8080** (internally).
- Restart Nginx:

```
sudo systemctl restart nginx
```

Paste the public ip of proxy-server on the Google

## 13.Database Setup

- SSH to DB Server and change the hostname:

```
sudo ssh -i <key.pem> ec2-user@<private-IP of db-server>
sudo hostnamectl hostname db
exit
```

- Login again to db-server:

```
sudo yum update -y
sudo yum install mariadb105-server -y
sudo systemctl start mariadb
# Installs and starts MariaDB locally, or connects to RDS via client.
```

## 14. Configure DB (if using RDS)

- From the **DB server** (or from an EC2 in same VPC with mysql client):

```
# if you installed mariadb client
mysql -h <RDS-endpoint> -u admin -p
# enter RDS password when prompted
```

- SQL to create DB and table:

```
CREATE DATABASE studentapp;
USE studentapp;

CREATE TABLE IF NOT EXISTS students (
    student_id INT NOT NULL AUTO_INCREMENT,
    student_name VARCHAR(100) NOT NULL,
    student_addr VARCHAR(100) NOT NULL,
    student_age VARCHAR(3) NOT NULL,
    student_qual VARCHAR(20) NOT NULL,
    student_percent VARCHAR(10) NOT NULL,
    student_year_passed VARCHAR(10) NOT NULL,
    PRIMARY KEY (student_id)
);

SHOW TABLES;
SELECT * FROM students;
exit
```

```
MariaDB [(none)]> create database studentapp;
Query OK, 1 row affected (0.006 sec)

MariaDB [(none)]> use studentapp;
Database changed
MariaDB [studentapp]> CREATE TABLE students (
    -> student_id INT NOT NULL AUTO_INCREMENT,
    -> student_name VARCHAR(100) NOT NULL,
    -> student_addr VARCHAR(100) NOT NULL,
    -> student_age VARCHAR(3) NOT NULL,
    -> student_qual VARCHAR(20) NOT NULL,
    -> student_percent VARCHAR(10) NOT NULL,
    -> student_year_passed VARCHAR(10) NOT NULL,
    -> PRIMARY KEY (student_id)
    -> );
Query OK, 0 rows affected (0.016 sec)

MariaDB [studentapp]> show tables;
+-----+
| Tables_in_studentapp |
+-----+
| students             |
+-----+
1 row in set (0.001 sec)

MariaDB [studentapp]> select * from students;
Empty set (0.001 sec)

MariaDB [studentapp]> exit
Bye
```

```
# Initializes schema for the web app to store student records
```

## 15. Add MySQL connector to Tomcat (app → RDS)

- On app server:

```
sudo -i
cd /opt/<apache-tomcat-folder>/lib/
ls

# download connector
curl -O <mysql-connector-link>
ls
# The connector JAR provides JDBC connectivity, and the datasource config enables
Tomcat to pool DB connections securely.
```

- Edit Tomcat `context.xml` to add resource:

```
# /opt/<apache-tomcat-folder>/conf/context.xml
<Context>
    ...
        <Resource name="jdbc/TestDB" auth="Container"
type="javax.sql.DataSource"
            maxTotal="500" maxIdle="30" maxWaitMillis="1000"
            username="admin" password=<your-rds-password>
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://<RDS-ENDPOINT>:3306/studentapp?useUnicode
            =yes&characterEncoding=utf8"/>
    ...
</Context>
```

```
<Context>
    <!-- Default set of monitored resources. If one of these changes, the -->
    <!-- web application will be reloaded. -->
    <watchedResource>WEB-INF/web.xml</watchedResource>
    <watchedResource>WEB-INF/tomcat-web.xml</watchedResource>
    <watchedResource>${catalina.base}/conf/web.xml</watchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->
    <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
        maxTotal="500" maxIdle="30" maxWaitMillis="1000"
        username="admin" password="Oaas3B1TsrebtGnXKRyf" driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://three-tier-rds.cirk6si6wi85.us-east-1.rds.amazonaws.com:3306/studentapp?useUni
        de=yes&characterEncoding=utf8"/>
```

- Restart Tomcat:

```
cd /opt/<apache-tomcat-folder>/bin/
./catalina.sh stop
./catalina.sh start
```

## Verification

1. Visit the **Proxy Public IP** in a browser: [http://<proxy-public-ip>/](http://<proxy-public-ip>)

**Student Registration Form**

Student Name   
 Student Address   
 Student Age   
 Student Qualification   
 Student Percentage   
 Year Passed

- Nginx proxies requests to <http://<app-private-ip>:8080/student>

2. Fill in the student data form.
- 

## Student Registration Form

Student Name   
 Student Address   
 Student Age   
 Student Qualification   
 Student Percentage   
 Year Passed

3. Verify data is saved successfully.

[Register Student](#)

## Students List

Student ID	StudentName	Student Addrs	Student Age	Student Qualification	Student Percentage	Student Year Passed	Edit	Delete
1	Ram	Nagar	23	BE	82	2024	<a href="#">edit</a>	<a href="#">delete</a>
2	Sham	Pune	22	Bcom	78	2025	<a href="#">edit</a>	<a href="#">delete</a>

4. Connect to RDS or DB Server:

```
USE studentapp;
SELECT * FROM students;
```

- You should see the inserted records.

```
[ec2-user@db-server ~]$ sudo mysql -h three-tier-rds.cirk6si6wi85.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 171
Server version: 11.4.8-MariaDB-log managed by https://aws.amazon.com/rds/
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use studentapp;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [studentapp]> select * from students;
+-----+-----+-----+-----+-----+-----+-----+
| student_id | student_name | student_addr | student_age | student_qual | student_percent | student_yea
r_passed |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | Ram | Nagar | 23 | BE | 82 | 2024
| 2 | Sham | Pune | 22 | Bcom | 78 | 2025
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.001 sec)
```

---

## Troubleshooting tips

- If the app is not reachable, check:
  - Security group inbound rules (ports open correctly).
  - NACLs on subnets (should allow traffic).
  - Route tables (public subnet routes to IGW, private to NAT).
  - Tomcat status and logs ([catalina.out](#)).
  - Nginx error logs ([/var/log/nginx/error.log](#)).
  - RDS inbound rule allows app SG to connect on 3306.

---

## Conclusion

- This project successfully sets up a Three-Tier Web Application on AWS, complete with:
  - Secure networking (VPC, subnets, routing)
  - Scalable compute (EC2)
  - Managed database (RDS)
  - Load separation (proxy → app → DB)
- It demonstrates fundamental AWS networking, EC2 configuration, and integration between compute and database layers in a real-world architecture.