AWS RDS

What is a Database?

A database is a structured collection of data organized in a way that allows for efficient storage, retrieval, and manipulation of information. It serves as a centralized repository where data is stored in tables, each consisting of rows and columns.
In a database, data is organized according to a predefined schema, which defines the structure of the data and the relationships between different pieces of information. This schema ensures data integrity and consistency by enforcing rules and constraints on how data can be stored and accessed.
Databases are used in various applications and industries to store and manage large volumes of data, ranging from simple lists to complex datasets. They provide mechanisms for querying and analyzing data, enabling users to extract meaningful insights and make informed decisions based on the information stored within them.
Overall, databases play a critical role in modern information systems by providing a reliable and efficient means of storing, organizing, and managing data.

Amazon Relational Database Service

RDS is a service from Amazon Web Services (AWS) that offers ready-to-use databases in the cloud. It helps businesses manage their databases without worrying about the underlying infrastructure.

A relational database is a type of database that organizes data into tables, with each table consisting of rows and columns. It establishes relationships between tables using keys, enabling efficient storage and retrieval of data.

A relational database is like a big organized collection of tables. Each table is like a different category, and each row in the table is like a single item. The columns in the table describe different details about each item, like its name, age, or address.

What's special about relational databases is they can connect information between tables using special keys. These keys help create relationships between different pieces of information. This makes it easier to organize and find information quickly, kind of like how you might organize your toys in different boxes with labels to find them easily when you need them."

Key Points about RDS

Ready-to-Use Databases: ***RDS provides pre-configured databases like MySQL, PostgreSQL, etc., which you can start using immediately without setup hassles.***

Managed Service: ***AWS takes care of routine database tasks such as backups, updates, and scaling, allowing users to focus on their applications.***

Easy Scalability*: RDS allows you to easily adjust the size of your database as your needs grow, without complex configurations.*

High Availability*: RDS ensures high availability of your databases through features like automated backups and failover support.*

Security: ***RDS offers various security features such as encryption, access control, and network isolation to protect your data.***

Monitoring and Alerts: ***RDS provides monitoring tools to keep track of your database's performance and alerts you in case of any issues.***

Automatic Backups: ***RDS automatically takes backups of your database, enabling point-in-time recovery if needed.***

Pay-as-You-Go Pricing*: With RDS, you only pay for the resources you use, making it cost-effective for businesses of all sizes.*

Use of RDS

RDS is commonly used by businesses and developers to set up, operate, and scale relational databases for various applications such as web applications, mobile apps, e-commerce platforms, and more. It's especially useful for organizations that want to focus on developing their applications without worrying about managing the underlying database infrastructure. RDS offers a reliable and scalable solution for storing and managing data in the cloud, making it an essential component of many cloud-based applications.

RDS DB Instance

An RDS instance is a virtual database server managed by Amazon Relational Database Service (RDS) within the Amazon Web Services (AWS) cloud infrastructure. It represents a single running database deployment and can be thought of as a container for your database.

Key points about RDS instances

Database Engine: ***Each RDS instance runs a specific database engine, such as MySQL, PostgreSQL, MariaDB, Oracle, or Microsoft SQL Server. You choose the database engine when creating the RDS instance.***

Instance Class: ***This refers to the computational and memory capacity allocated to the RDS instance. Instance classes range from small, single-core instances to larger, multi-core instances with more memory. You can choose the instance class based on your application's performance and resource requirements.***

Storage: ***RDS instances include storage for the database files. You can specify the storage type (e.g., General Purpose SSD, Provisioned IOPS SSD, Magnetic) and size when creating the instance. Storage can be scaled up or down as needed.***

Endpoint: ***Each RDS instance has an endpoint, which is a DNS address that your applications use to connect to the database. The endpoint includes the hostname and port number needed to establish a connection.***

Security: ***RDS instances are secured through various mechanisms such as security groups, network access control lists (ACLs), encryption at rest, and encryption in transit. These security features help protect your data from unauthorized access.***

Backup and Recovery: ***RDS automatically takes backups of your database instance according to the backup retention period you specify. You can also manually initiate backups and create snapshots for point-in-time recovery.***

High Availability: ***RDS offers Multi-AZ deployments, where your database is automatically replicated across multiple Availability Zones (AZs) for redundancy and failover support. In the event of a failure, RDS can automatically fail over to the standby instance to minimize downtime.***

Overall, an RDS instance provides a managed environment for running your relational database, allowing you to focus on building and running your applications without worrying about managing the underlying infrastructure. It offers scalability, reliability, security, and ease of use, making it a popular choice for hosting databases in the cloud

‘’’

RDS is categorized as a Platform as a Service (PaaS) because it provides a managed environment for hosting and managing relational databases. It abstracts away much of the complexity associated with infrastructure management, allowing users to focus on their database configurations and applications. This makes RDS an ideal choice for developers and businesses seeking a convenient, scalable, and reliable database solution without the burden of managing the underlying infrastructure.

Important Points to remember before creating a DB Instance and attaching it to the web application.

- ***It is necessary to allow MySQL port 3306 in the security group of the Ec2 instance and RDS instance.***
- ***SSH, HTTP, and Port 8080 are necessary for the Ec2 Instance to host the Apache Tomcat Application.***
- ***We need to download the MySQL connector/j into the Ec2 and move it to the lib directory of the Apache Tomcat Server***

- *The **MySQL Connector/J** is a software library or driver used by Java applications to connect to MySQL databases. It provides Java programs with the necessary tools to interact with MySQL databases seamlessly. Think of it as a bridge between Java and MySQL, allowing Java applications to send queries and receive data from MySQL databases using standard Java Database Connectivity (JDBC) APIs.*
- *The Connector/J handles tasks such as establishing connections to MySQL databases, executing SQL queries, managing transactions, and processing query results. It's packaged as a JAR (Java Archive) file, which developers include in their Java projects to enable database connectivity.*
- *To use the MySQL Connector/J in an Apache Tomcat environment, developers typically include the Connector/J JAR file in the 'lib' directory of Apache Tomcat. This allows the Tomcat server to access the Connector/J library and enables web applications deployed on Tomcat to communicate with MySQL databases.*
- *With the MySQL Connector/J, Java developers can create strong applications that use MySQL databases for storing and getting data. It's really important for Java developers who need to work with MySQL databases.*
- *The MySQL Connector/J is just one type of connector provided by MySQL for Java applications. MySQL also provides other types of connectors for different programming languages and platform*

- ***Then we need to configure the database connection pool in Apache Tomcat configuration.***

Setting up a connection pool is important for:

- ***Efficient Use of Resources:***
  Creating and closing database connections requires a lot of computer resources.
  Connection pooling saves these resources by reusing existing connections instead of creating new ones every time.
- ***Faster Performance:***
  Reusing connections from the pool saves time compared to establishing new connections.
  This results in faster database operations and improves the overall performance of the web application.
- ***Scalability:***
  Connection pooling helps handle more users by efficiently managing connections.
  It prevents the database from being overwhelmed during periods of high traffic, ensuring that the application remains responsive.
- ***Reliability:***
  Connection pooling includes features like connection validation.
  It ensures that connections are valid and usable before they are given to the application, reducing the risk of errors.
- ***Concurrency Control:***
  Connection pooling manages concurrent database access to prevent issues like data corruption.
  It ensures that database operations are synchronized when multiple threads are involved, maintaining data integrity.

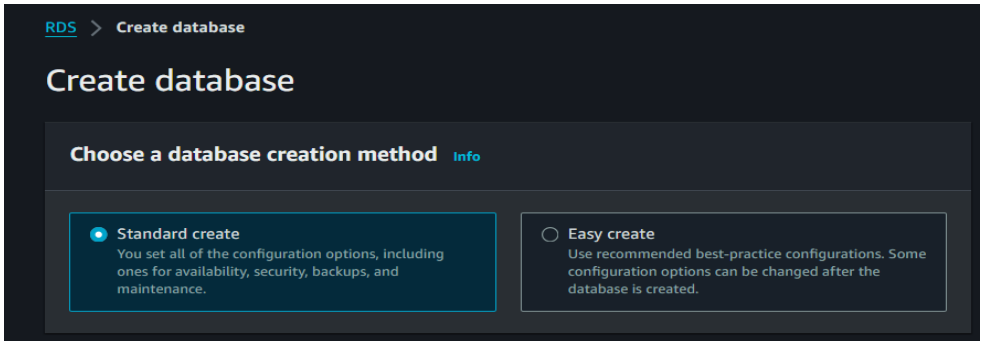
- ***Need to install mariadb105 or MySQL package. (MariaDB is a modified version of MySQL)***
  Installing MariaDB or MySql allows you to set up a database instance, essential for SSH access to manage and interact with your database. Once installation is done, then we can run the MySQL command.

CLOUD-BASED THREE-TIER STUDENT APPLICATION DEPLOYMENT PRACTICE

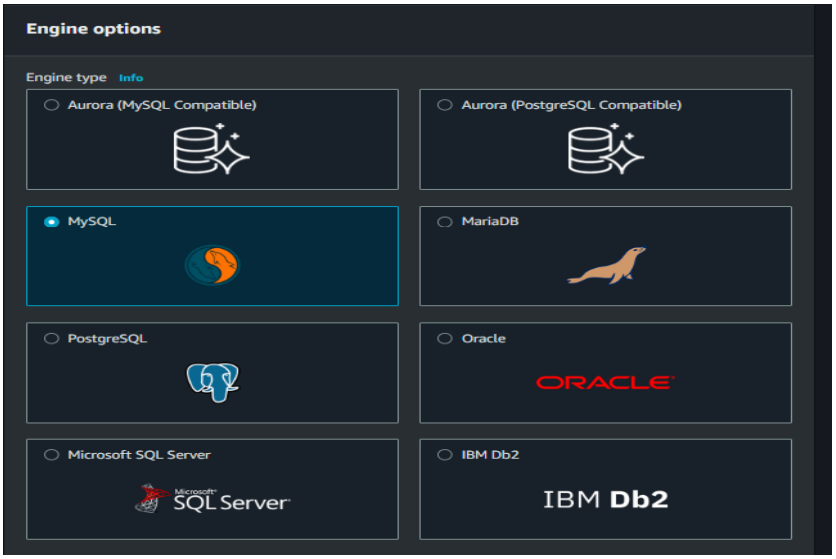
This project involves setting up a three-tier architecture for a student application on AWS, integrating a proxy server for enhanced security and request management. It includes creating a MySQL database instance on Amazon RDS, launching an EC2 instance with Apache Tomcat, and configuring the proxy server to facilitate single IP access to the application. Through practical steps, this project offers valuable insights into deploying and configuring a secure, scalable, and efficient cloud-based architecture for student applications.

Step 1) ***Go to the RDS Service, and click on the Create database button.***

Step 2) ***Select the Standard database creation method.***

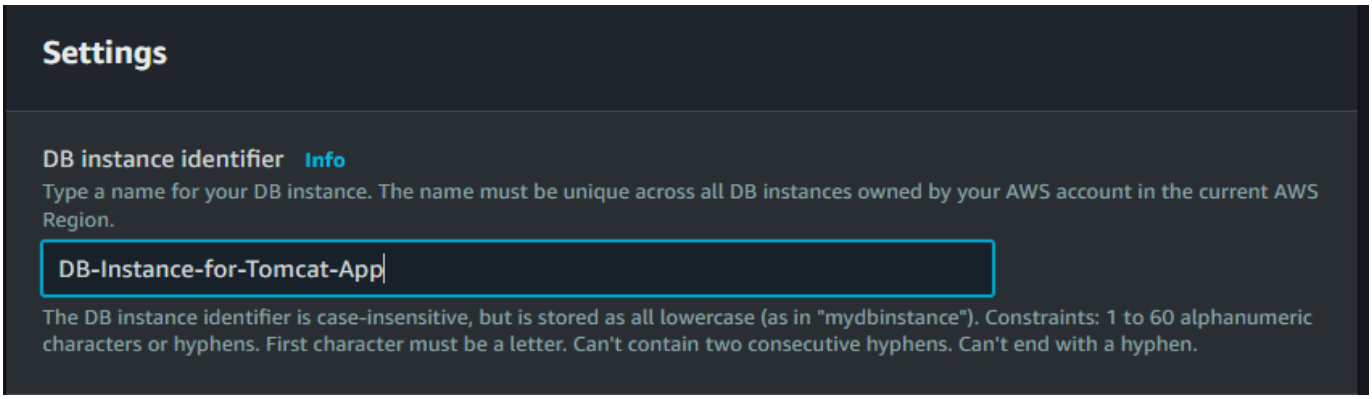


Step 3) ***Select MySQL engine type and version.***

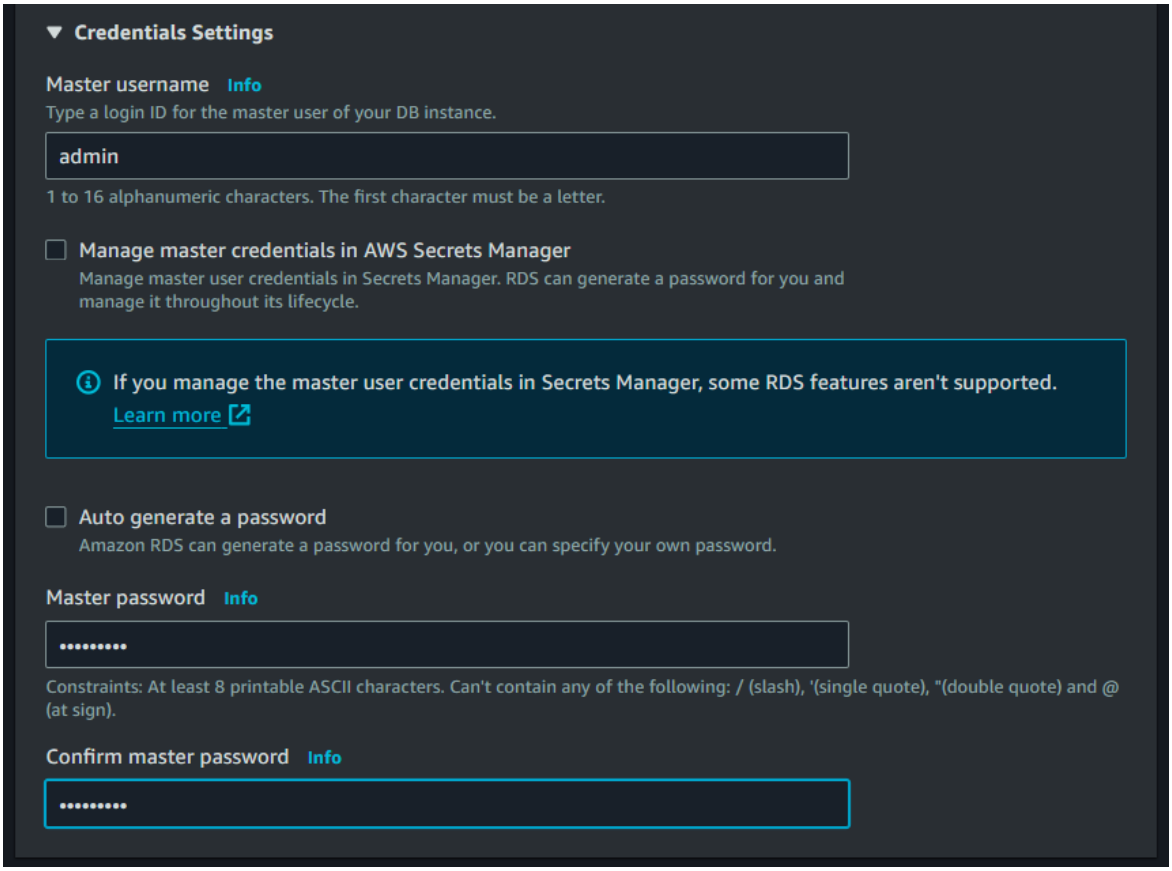


Step 4) ***Select the free tier template.***

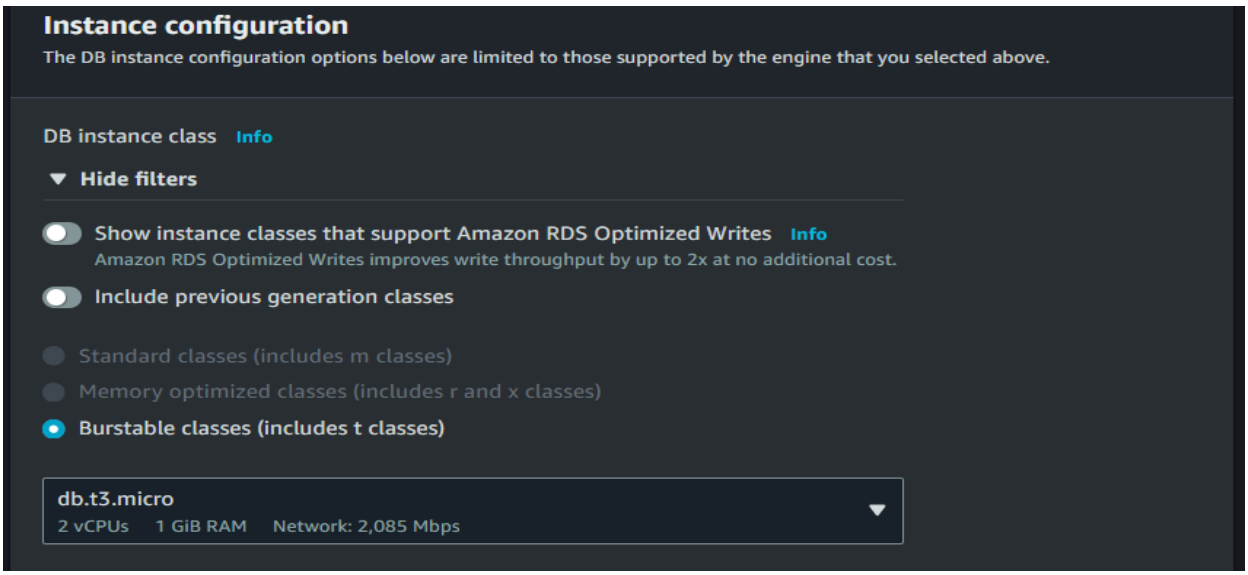
Step 5) ***Name the database Instance.***



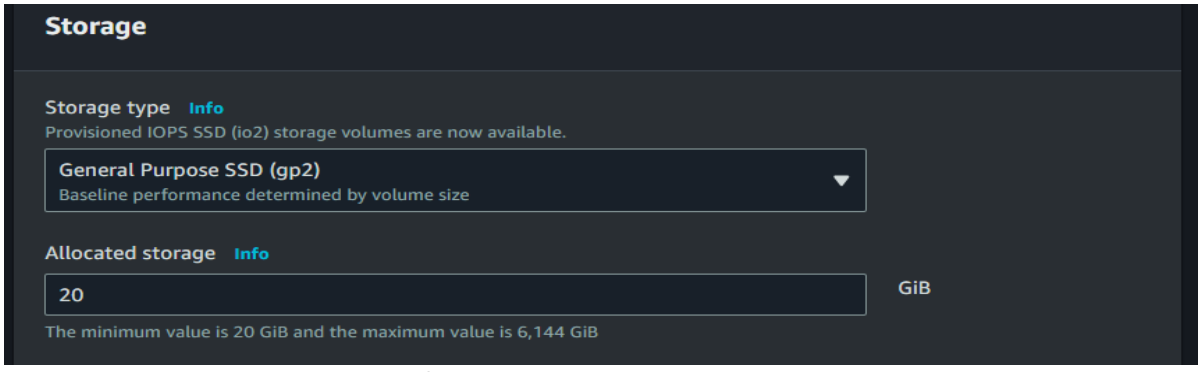
Step 6) ***In the Credential Setting, specify the user name (is a login ID) and only alphanumerical and start with later, Ex: admin, admin123, or darsh86. Then set the password, it is at least 8 characters (only alphanumerical), no special character. Ex: MyPass123, secure123.***



Step 7) ***Select Instance type.***

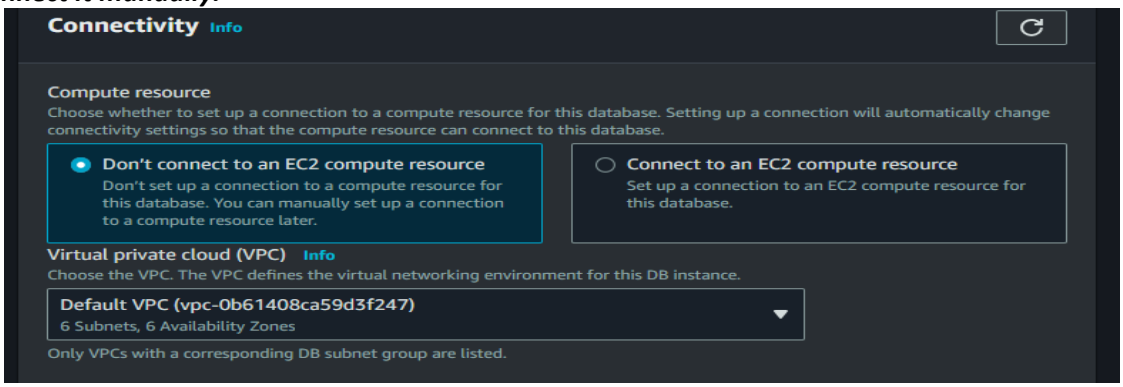


Step 8) ***Select Storage type and size minimum 20 GB (for SSD - gp2) and maximum 6144 GB for all storage types.***

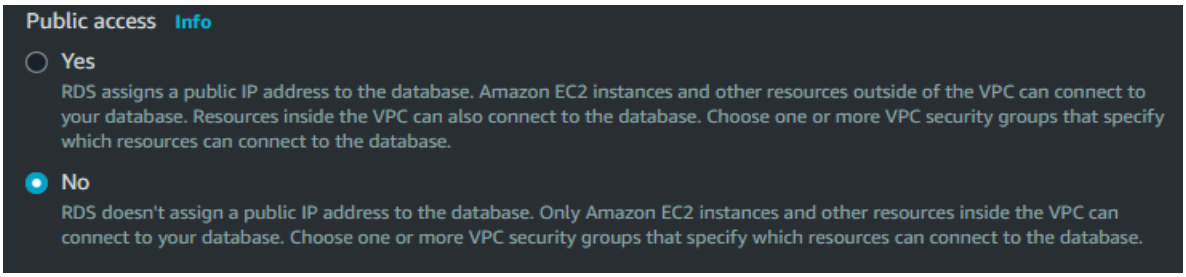


In the Autoscaling option by default, Autoscaling is enabled as now not needed so disable it.

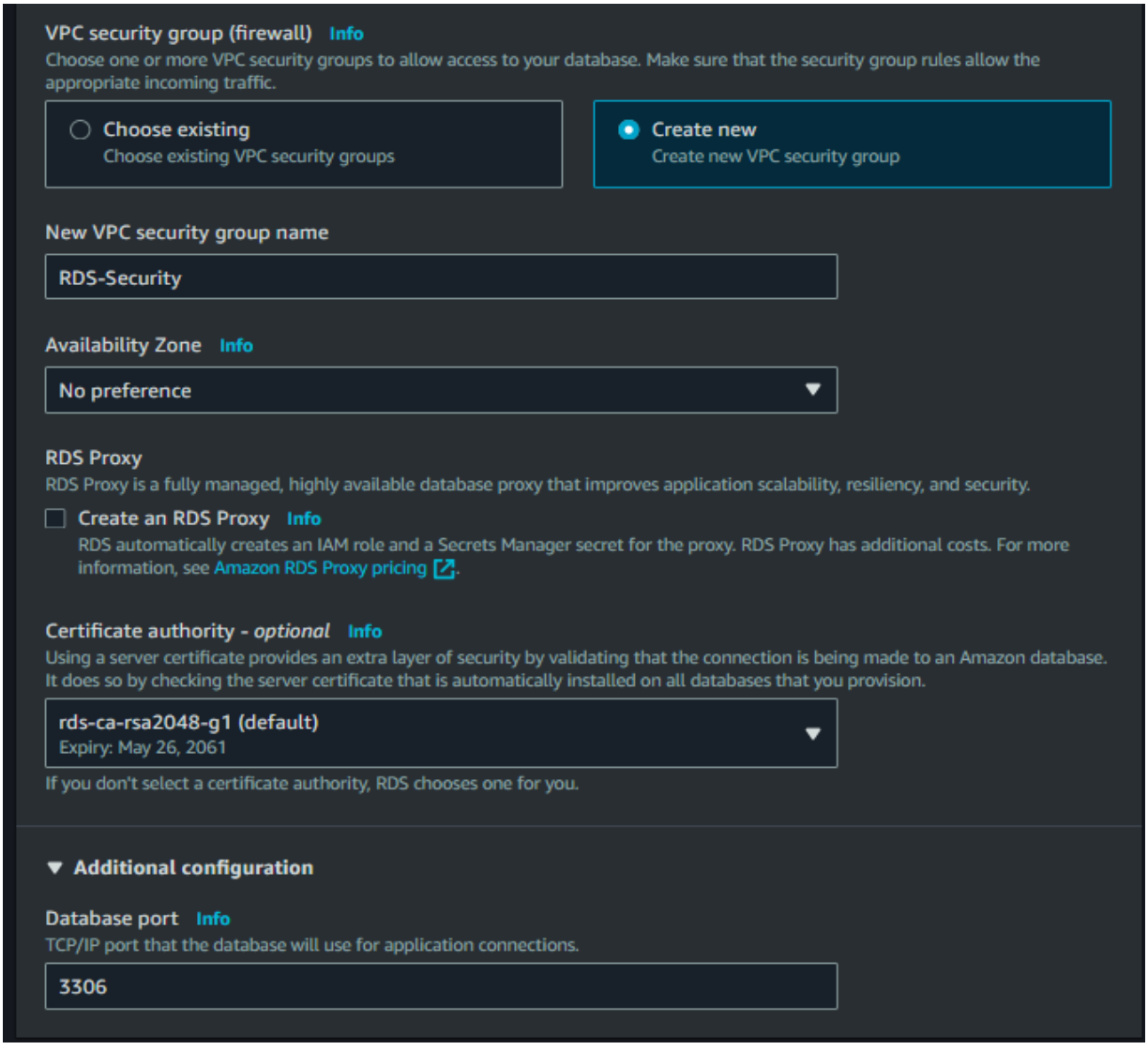
Step 9) ***In connectivity, we can directly select connect database to the Ec2 instance and don't connect to the Ec2 Instance, then we need to connect it manually.***



Step 10) ***Select VPC, and define Public Access yes or no. If yes database will get the IP address and other resources outside of VPC can connect to the database. Select no to the only resources inside the VPC that can connect to the database.***

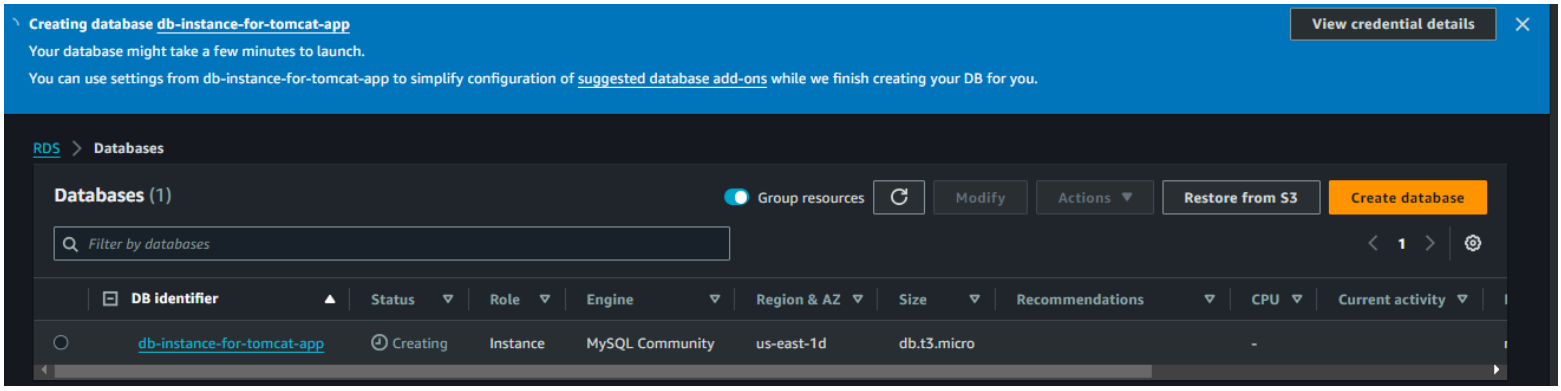


Step 11) ***Create a Security group.***



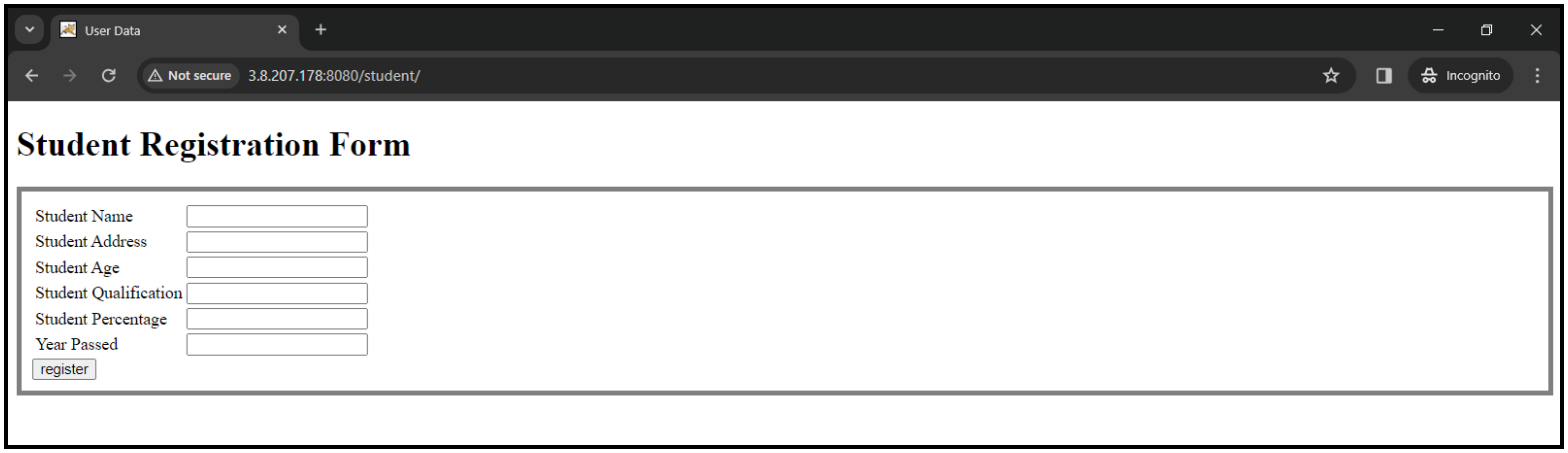
Step 12) ***In addition setting by default Backup, encryption, and maintenance are enabled, that setting affects on cost, if you disable cost will decrease. Now we disable it. But in the real scenario of the workplace is important to enable it.***

Step 13) ***Now simply click on the Create Database button. It will take a few minutes to create the database.***



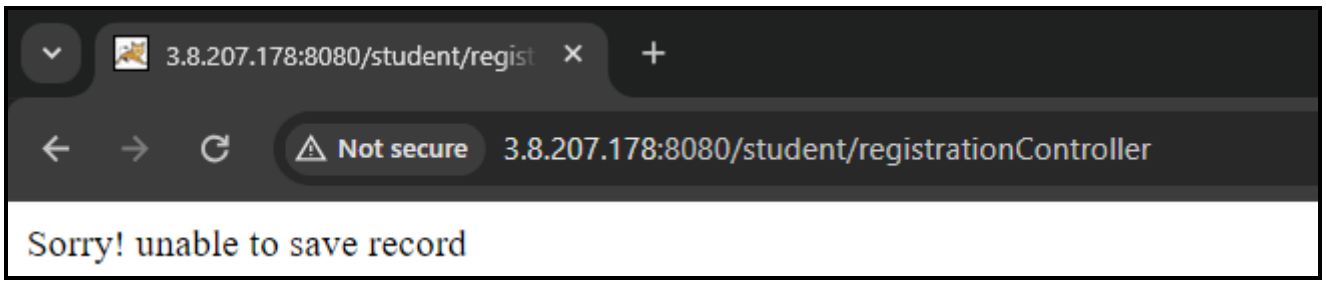
Step 14) ***Now we will launch the Ec2 Instance.***

- ***Launch instance with inbound rule SSH, HTTP, custom TCP port 8080, and MySQL port 3306.***
- ***Install Apache Tomcat in the Ec2 Instance, and configure it.***
- ***Install the Java package.***
- ***We have a pre-configured link to the Student App template, download the template and move it to Apache Tomcat's webapp directory.***
- ***Give full permission to catalina.sh file. Then start the Apache web server.***
- ***Now copy your public IP address of the Instance and browse it with the application path.***



We can see our application is running.

Step 15) ***Now we try to register into the application. Simply fill details and click on the register button.***



We can see the application is unable to save the details because there is no database backend.

Step 16) Now we need to attach our DB Instance to this application.

- *curl –O https://db-tom.s3.ap-northeast-1.amazonaws.com/mysql-connector.jar → to download the connector.*
- *mv mysql-connector.jar apache-tomcat-9.0.87/lib/ → to move the connector into the lib directory of apache tomcat.*
- *vim apache-tomcat-9.0.87/conf/context.xml → to configure Instance RDS in Apache tomcat.*

  Benefits: Enables efficient and secure interaction between Tomcat-based web applications and RDS databases.
- *Configuration: (add in 20th line of file)*

```
Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
               maxTotal="100" maxIdle="30" maxWaitMillis="10000"
               username="USERNAME" password="PASSWORD"
driverClassName="com.mysql.jdbc.Driver"
               url="jdbc:mysql://DB-ENDPOINT:3306/DATABASE"/>
```

Make sure to replace USERNAME, PASSWORD, DB-ENDPOINT, and DATABASE with the actual values specific to your MySQL database configuration. Once configured, your application should be able to use this DataSource to establish connections to the MySQL database and perform database operations.

```
<Context>
<Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000"
              username="admin" password="darsh8699" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://db-instacnce-for-tomcat-app.c5oy6iwa65lu.eu-west-2.rds.amazonaws.com:3306/DATABASE"/>
```

- *Save the file.*

  Now here we configured a database connection pool in Tomcat, allowing our web application to efficiently manage database connections.
- *Now we need to install mariadb105*

  Installing MariaDB allows you to set up a database instance, essential for SSH access to manage and interact with your database. Once installation is done, we can run the MySQL command.
- *yum install mariadb105 -y → to install mariadb105*

  Here our configuration and setup are done to connect with our DB Instance.

Step 16) Now get SSH of the DB Instance over the Ec2 Instance.

- *mysql -h <endpoint of db-instance> -u admin -p[password] → run command.*

```
[ec2-user@ip-172-31-34-1 ~]$ mysql -h db-instacnce-for-tomcat-app.c5oy6iwa65lu.eu-west-2.rds.amazonaws.com -u admin -pdarsh8699
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 55
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

We have logged into our database. Now we can run the MySQL command

MySQL commands:

1) *show databases; - Lists all databases on the server*
2) *use database_name; - Selects a specific database to work with*
3) *show tables; - Lists all tables in the current database*
4) *describe table_name; - Shows the structure of a specific table*
5) *create database database_name; - Creates a new database*
6) *create table table_name (column1 datatype, column2 datatype, ...); - Creates a new table*
7) *insert into table_name (column1, column2, ...) values (value1, value2, ...); - Inserts new records into a table*
8) *select column1, column2, ... from table_name where condition; - Retrieves data from one or more tables*

9) *update table_name set column1 = value1, column2 = value2, ... where condition;* - **Modifies existing records in a table**
10) *delete from table_name where condition;* - **Removes records from a table**
11) *alter table table_name add column_name datatype;* - **Modifies an existing table**
12) *drop table table_name;* - **Deletes a table**
13) *drop database database_name;* - **Deletes a database**
14) *create user 'username'@'localhost' identified by 'password';* - **Creates a new database user**
15) *grant all privileges on database_name.* to 'username'@'localhost';* - **Grants privileges to a user on a database or tables**
16) *revoke all privileges on database_name.* from 'username'@'localhost';* - **Revokes previously granted privileges from a user**
17) *select User, Host from mysql.user;* - **Lists all MySQL users**
18) *show grants for 'username'@'localhost';* - **Lists the privileges granted to a user**
19) *flush privileges;* - **Reloads the privileges from the grant tables in the MySQL database**

Step 17) **Now we create a database for studentapp and then create a student table within it using the following script.**

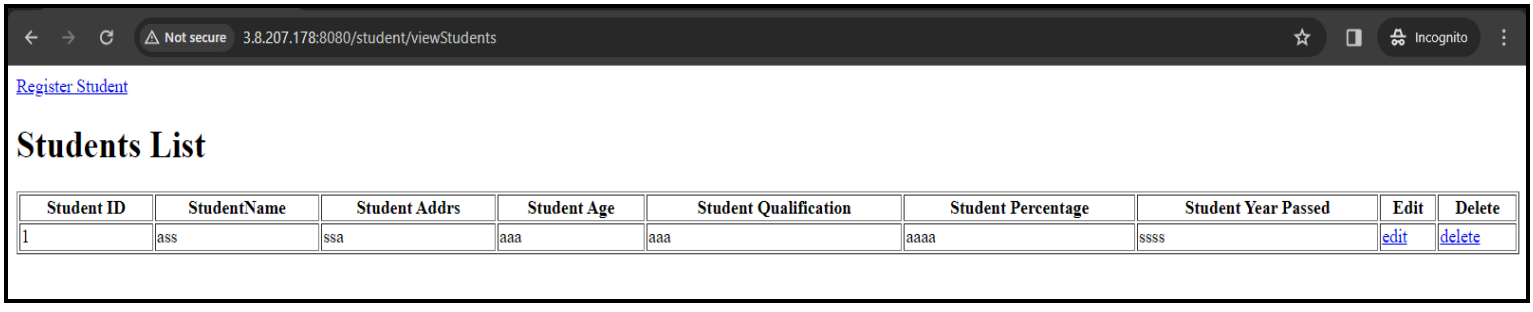
```
CREATE DATABASE studentapp;
USE studentapp;
CREATE TABLE IF NOT EXISTS students (
   student_id INT NOT NULL AUTO_INCREMENT,
   student_name VARCHAR(100) NOT NULL,
   student_addr VARCHAR(100) NOT NULL,
   student_age VARCHAR(3) NOT NULL,
   student_qual VARCHAR(20) NOT NULL,
   student_percent VARCHAR(10)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

This script will create the "studentapp" database if it doesn't already exist, then switch to that database, and finally create the "students" table within it, with columns for student name, address, age, qualification, percentage, and year passed. Each column has a specified data type and is marked as "NOT NULL", meaning it cannot contain NULL values. The "student_id" column is set as the primary key for the table.

In the connection pool configuration, we specified the database name as "studentapp" so here we had needed to create the database "studentapp". And created a table to store student information.

Here we have successfully configured our Apache Tomcat "studentapp" application to use the "studentapp" database via the JDBC DataSource. Now our "studentapp" application should be able to save the details to the database

Step 17) **Now to register the student again fill the details in "studentapp" application and click on the register button.**



Here we can see student details saved successfully.

Step 19) Now we launch a new Ec2 instance for the proxy server using NGINX.

We're setting up NGINX as a proxy server to make our student application more secure and easier to manage. With the proxy, we can control who can access our app and limit it to just one IP address. This way, we can make sure only authorized users can use our app. The proxy server helps us keep things organized and adds an extra layer of security to our setup.

- *In the inbound rule, SSH and HTTP rules allow into the security group of instance*
- *Get SSH.*
- *Run the command:*

  sudo yum install nginx -y → To install NGINX
  sudo systemctl start nginx → To start NGINX
  sudo systemctl enable nginx → To enable NGINX.

- *Now we need to add a proxy configuration within an NGINX server block.*

When setting up NGINX as a proxy server, we would locate or create a server block within the NGINX configuration file and add the proxy configurations inside that block. This ensures that the proxy settings apply only to the specific domain or IP address configured in that server block.

- *Run the command: sudo vim /etc/nginx/nginx.conf in this configuration file we add the proxy server configuration.*
- *Now add configuration in this file as below then save and exit from file.*

```
location / {
    proxy_pass http://ApplicationPublicIP:8080/student/;
    }
```

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

        error_page 404 /404.html;
        location = /404.html {
        }
        location / {
      proxy_pass http://3.8.207.178:8080/student/;
      }
        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }

# Settings for a TLS enabled server.
#
#    server {
#        listen       443 ssl http2;
```

i-0821b40791332db0c (Proxy-Server)
PublicIPs: 18.132.12.95 PrivateIPs: 172.31.41.14

This NGINX configuration forwards requests from the root location ("/") to a backend application server located at "http://ApplicationPublicIP:8080/student/". NGINX serves as a bridge, passing incoming requests to the specified backend endpoint ("/student/"). This setup allows NGINX to facilitate communication between clients and the backend application server.

- *Now we have to restart or reload NGINX to apply the changes in the configuration file.*
  Run the command: sudo systemctl restart nginx or sudo nginx -s reload

Step 20) Now copy the public IP address of our NGINX instance and browse it.

We can see the NGINX proxy server IP address correctly forwarding requests to "studentapp". So here we have successfully deployed a three-tier student application with NGINX proxy server integration, achieving enhanced security and accessibility."