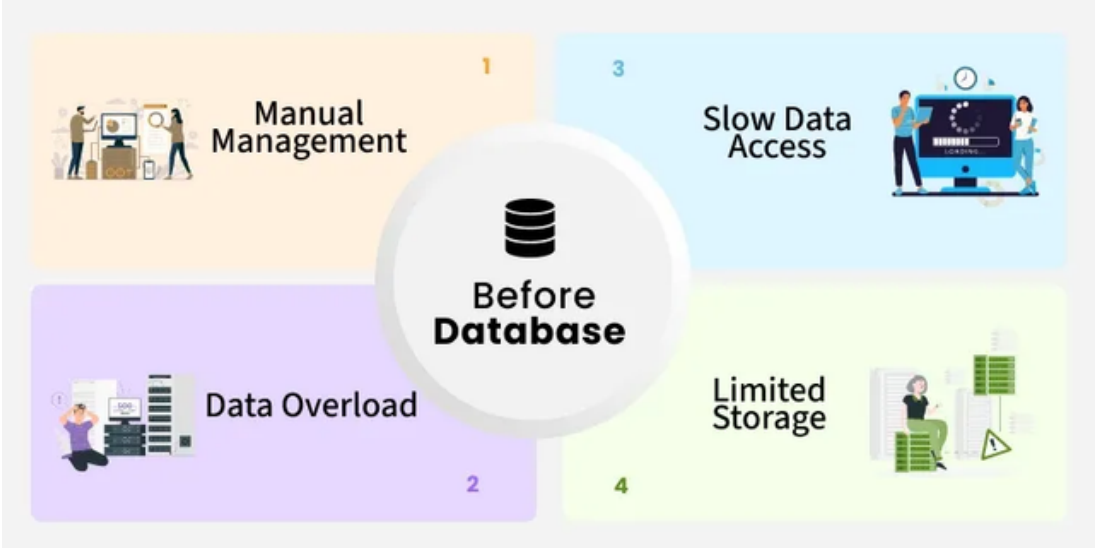
AWS RDS

Relational Database Service

Concept Overview:

What is Database:

A database is an electronically stored, systematic collection of data. It can contain any type of data, including words, numbers, images, videos, and files. You can use software called a database management system (DBMS) to store, retrieve, and edit data. In computer systems, the word database can also refer to any DBMS, to the database system, or to an application associated with the database.



Relational Database:

Relational database is a database that stores data in tables (rows & columns) with relationships between them.

- Data organized into tables
- Each table has rows (records) & columns (fields)
- Uses SQL (Structured Query Language) for queries
- Supports relationships (Primary Key, Foreign Key)
- Ensures data integrity & consistency

Relational databases:

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server
- MariaDB
- IBM Db2
- Amazon Aurora
- SQLite

Non-Relational Database:

Non-relational database is a database that stores data in flexible formats (not only tables), designed for scalability

- Data stored as documents, key-value pairs, graphs, or wide-columns
- No fixed schema (schema-less or flexible schema)
- Optimized for scalability & high performance
- Great for big data & real-time applications

Non-Relational databases:

- MongoDB
- Cassandra
- Redis
- DynamoDB

About RDS:

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

Benefits of it:
Amazon RDS provides the following principal advantages over database deployments that aren't fully managed:

- You can use database engines that you are already familiar with: IBM Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL.
- Amazon RDS manages backups, software patching, automatic failure detection, and recovery.
- You can turn on automated backups, or manually create your own backup snapshots. You can use these backups to restore a database. The Amazon RDS restore process works reliably and efficiently.
- You can get high availability with a primary DB instance and a synchronous secondary DB instance that you can fail over to when problems occur. You can also use read replicas to increase read scaling.
- In addition to the security in your database package, you can control access by using AWS Identity and Access Management (IAM) to define users and permissions. You can also help protect your databases by putting them in a virtual private cloud (VPC).

Features of RDS:

Availability:

- Automated Backups → easy recovery, quick access
- Database Snapshots → user-driven, shareable across AWS accounts

Security:

- Strong password (Admin role by default)
- Encryption with AWS KMS keys

Backups:

- Automated Backups (configured at creation)
- Snapshots (non-editable, for record keeping)

Scalability:

- Horizontal Scaling → add multiple instances to handle high traffic
- Vertical Scaling → upgrade existing resources (CPU, storage)

Performance:

- General Purpose SSD (cost-effective, broad use)
- Provisioned SSD (higher performance needs)

Pricing:

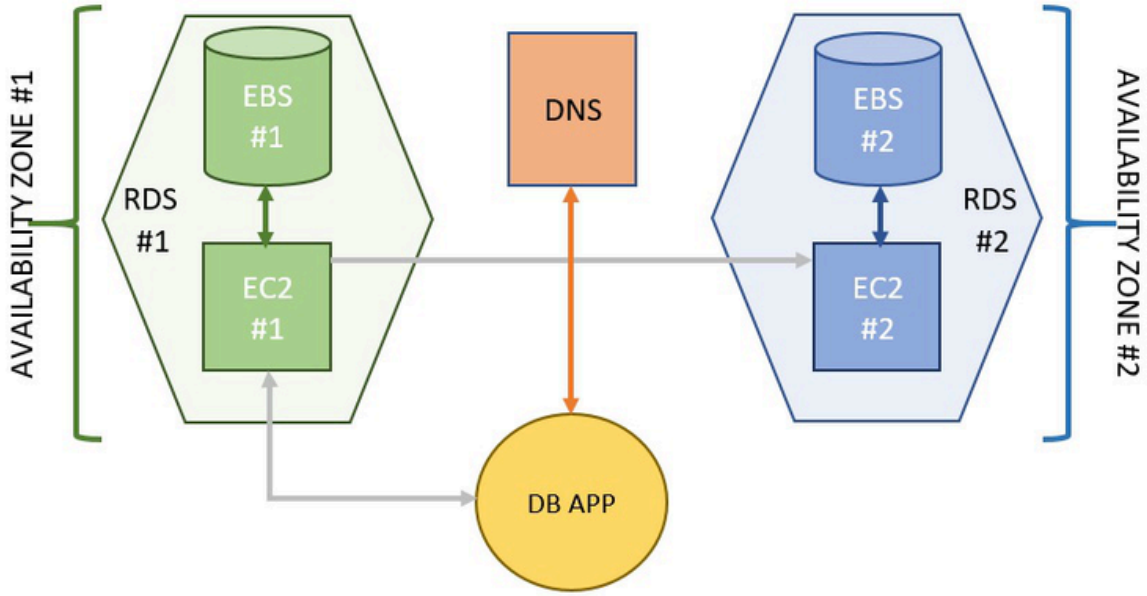
- Pay-as-you-go model
- No minimum charge, free tier available with limits

Work process of RDS:

Traditionally, database management was scattered across multiple layers, the web server, the application server, and finally the database, requiring a dedicated team for maintenance. To simplify this, AWS introduced RDS, an all-in-one managed database service.

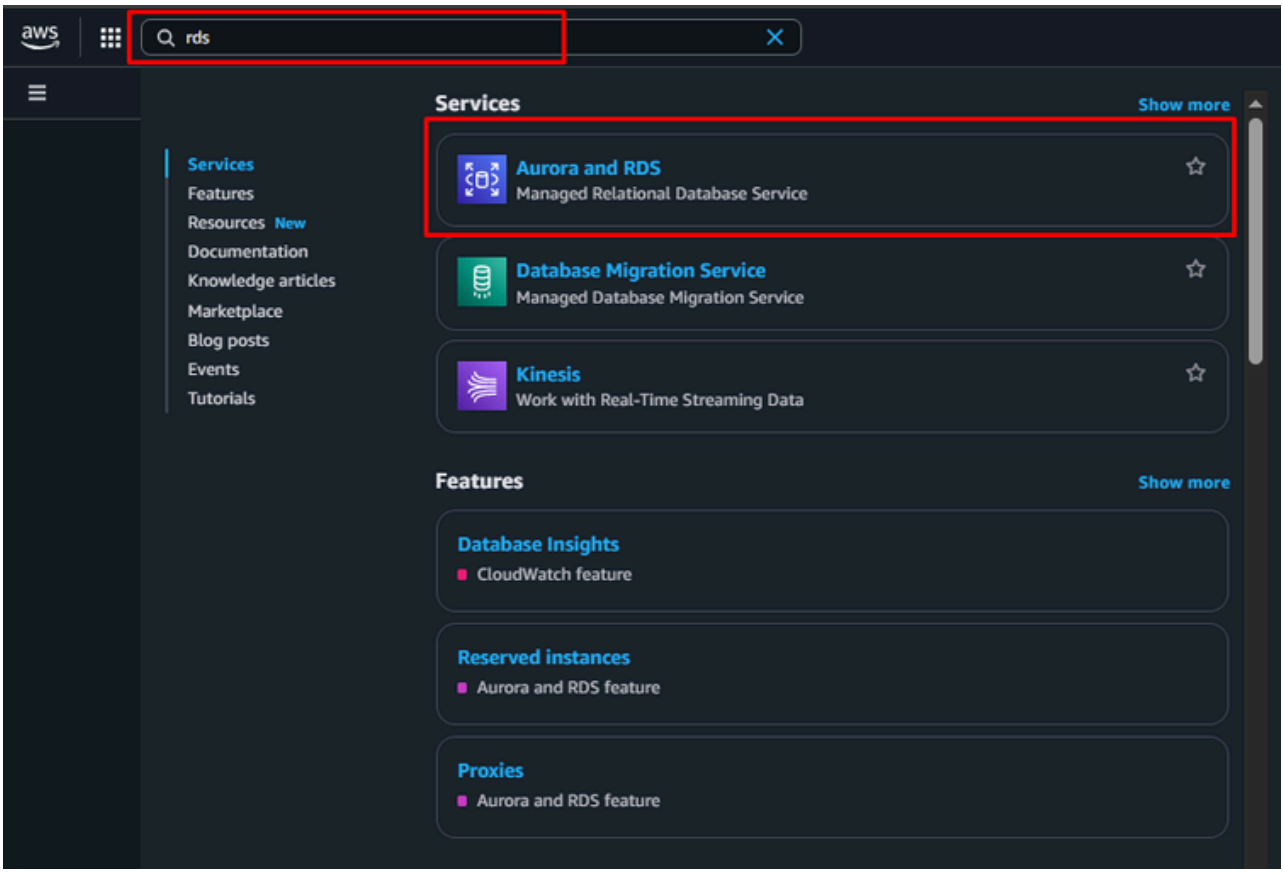
RDS integrates all components of traditional database architecture, including compute (EC2) and networking (DNS), into a single managed solution. Each part of the RDS architecture comes with its own set of features, making database management easier, more efficient, and highly scalable.

A diagram illustrating the RDS architecture is provided below.

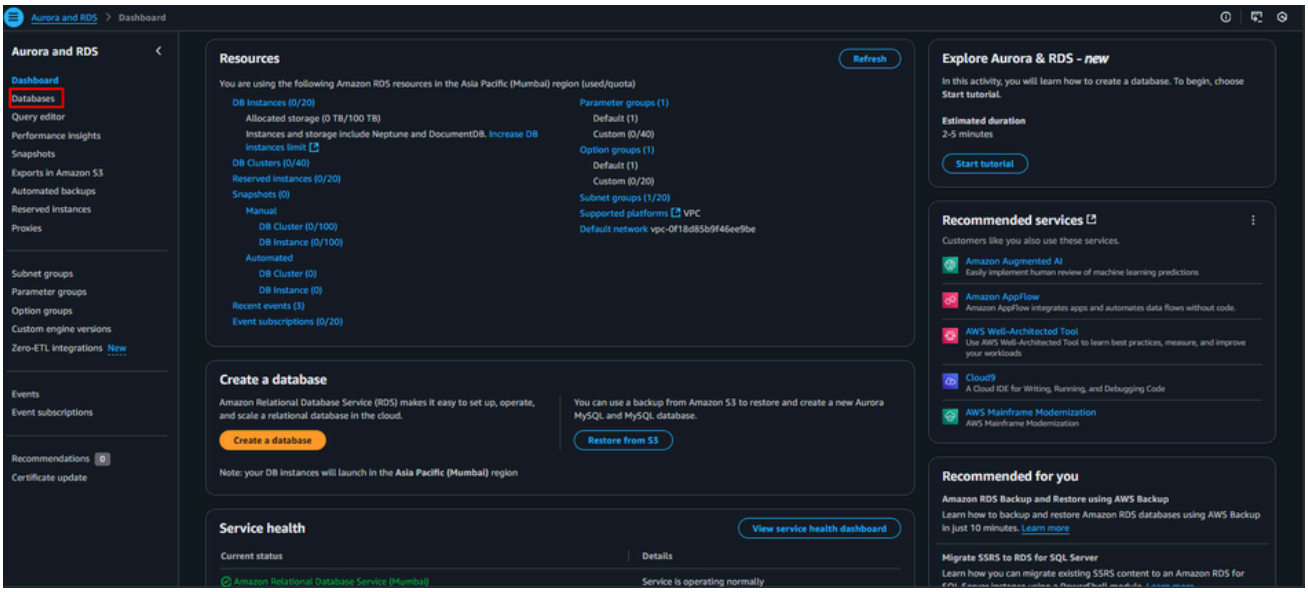


Hands-on with (RDS + MySQL + EC2):

Step 1: Sign in to your AWS account and search for RDS.

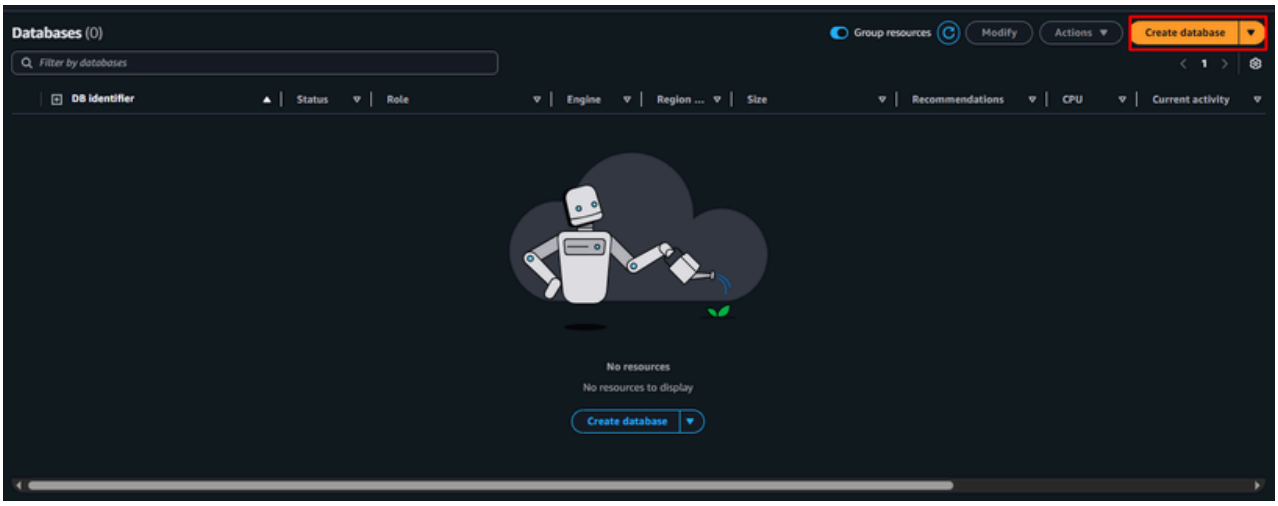


Get this page now click on databases section.

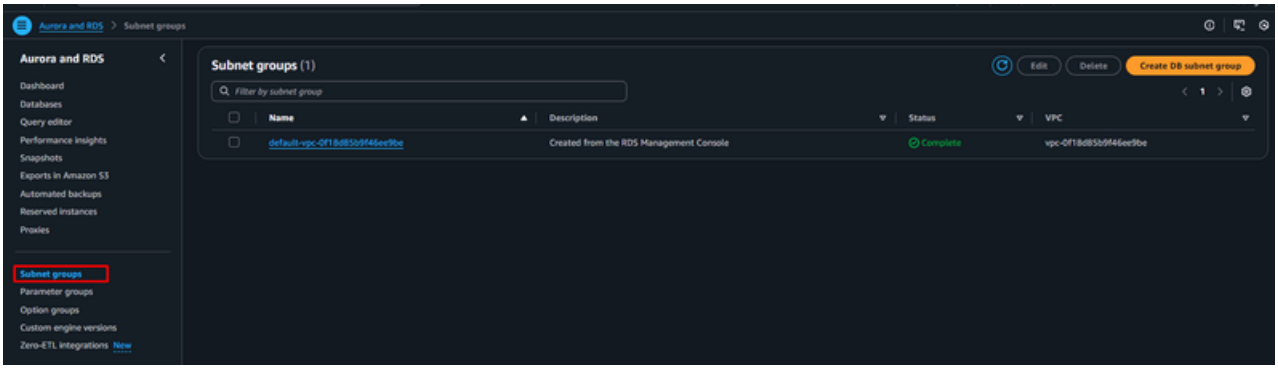


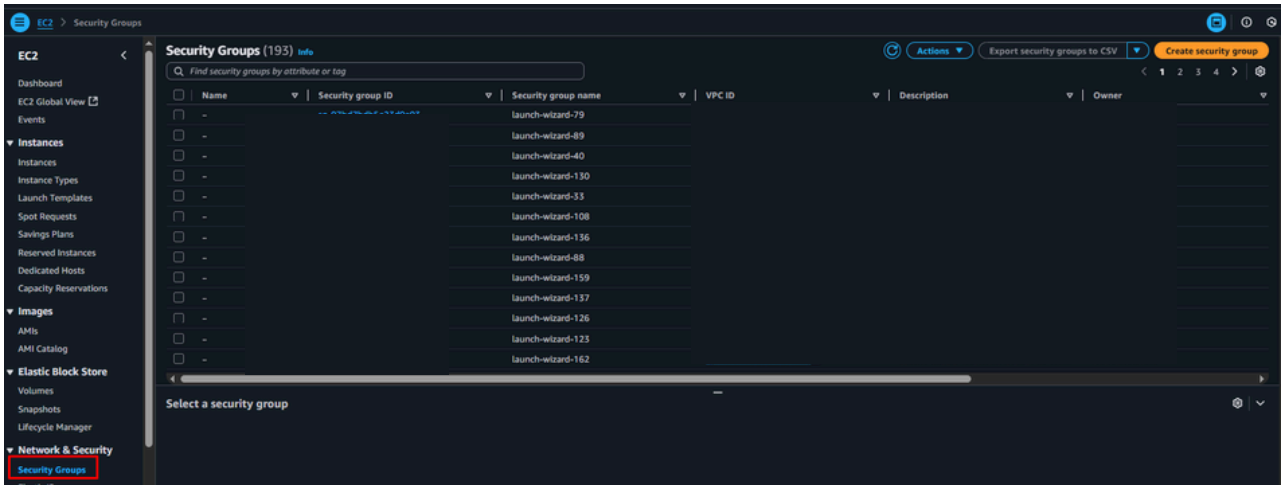
Hands-on with (RDS + MySQL + EC2):

Step 2: After click on database section get this page. Now click on create databse button.



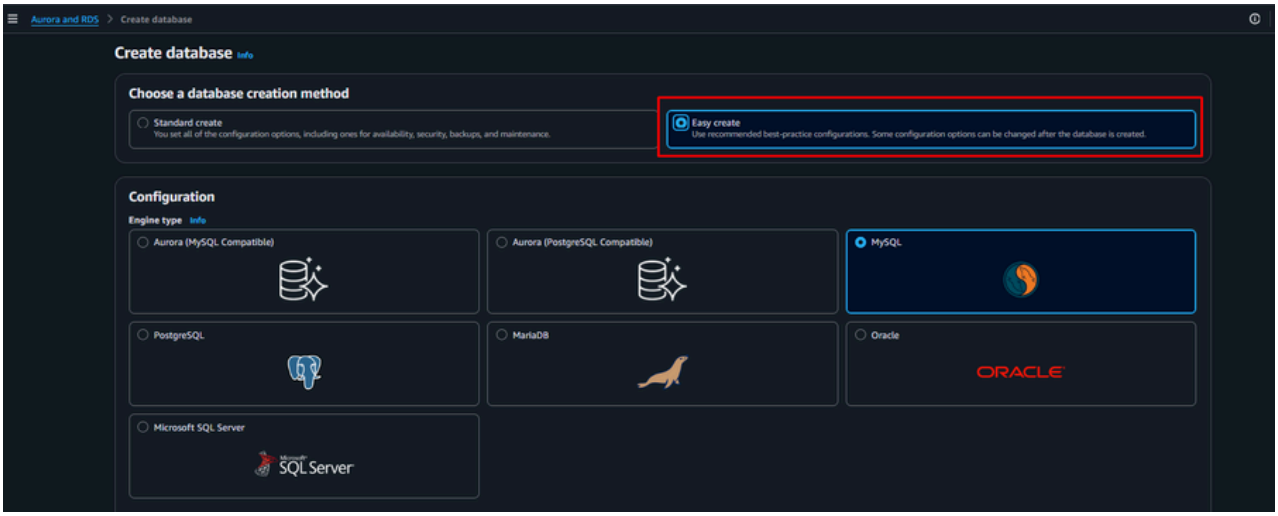
Note: Before creating a new database, you need to create a subnet group and a security group.



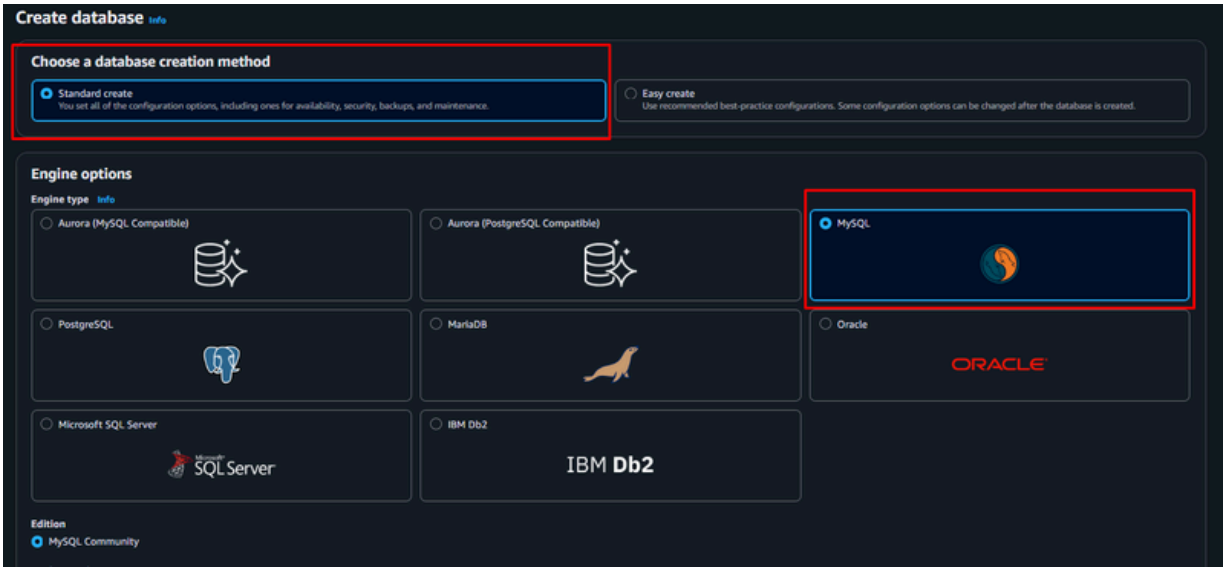


Hands-on with (RDS + MySQL + EC2):

Step 3: You can create a database easily with one click, but I chose to create it manually with detailed configuration.

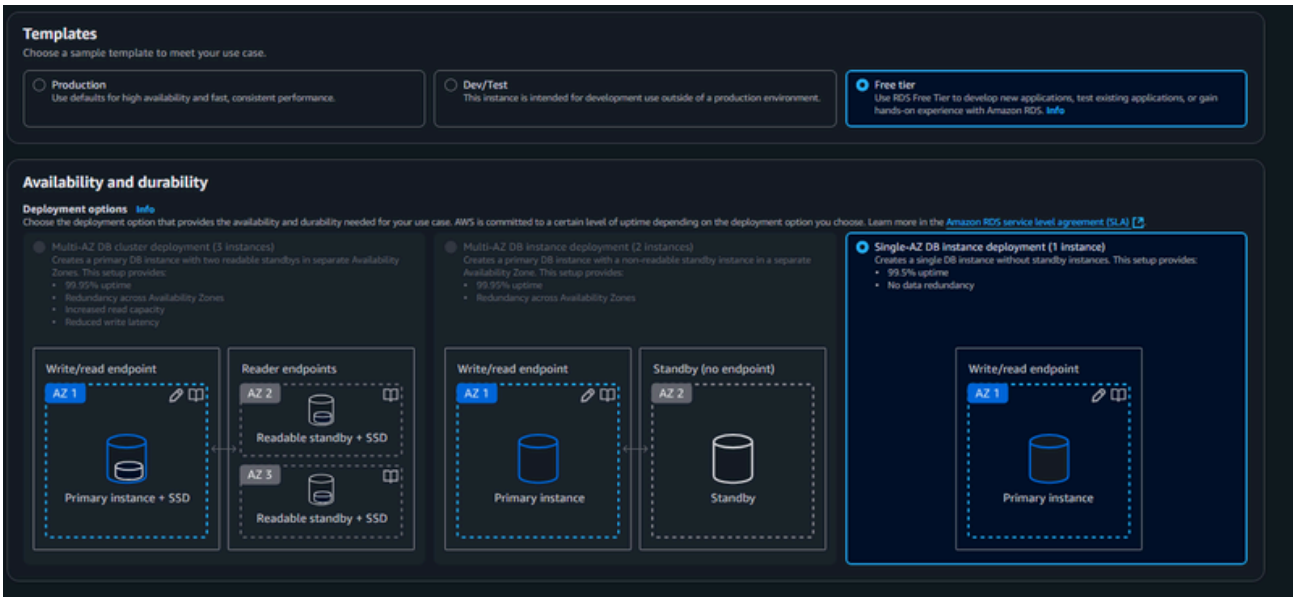


Step 4: Let's create db manually. Choose standard create and database which one you want to use.

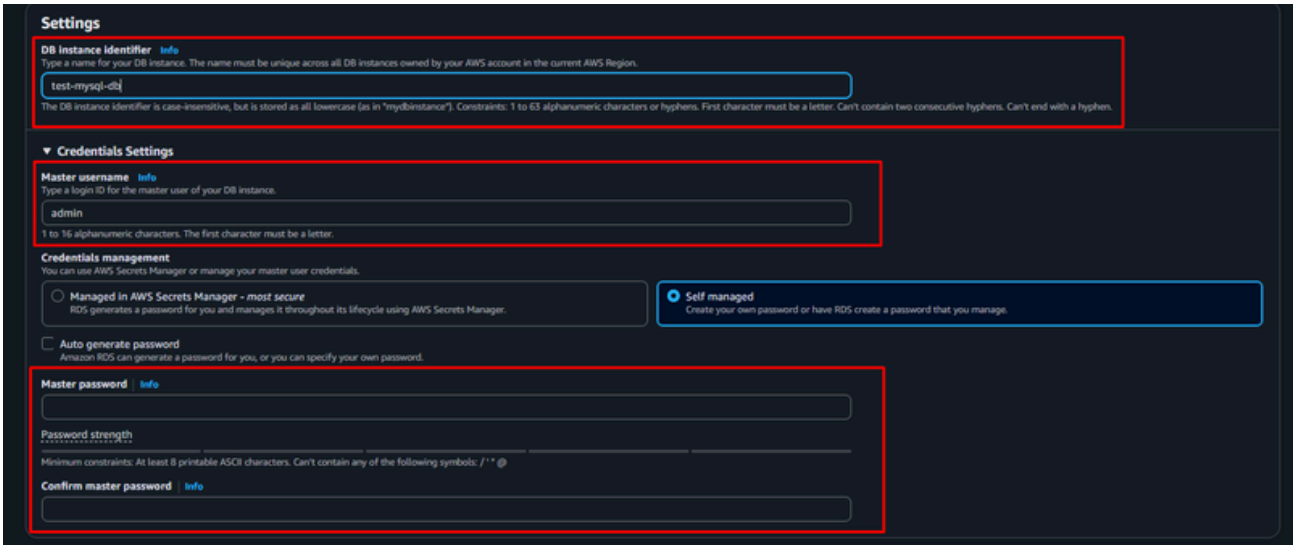


Hands-on with (RDS + MySQL + EC2):

In the Template section, I chose the Free Tier option, so it hides paid resources for more configuration.

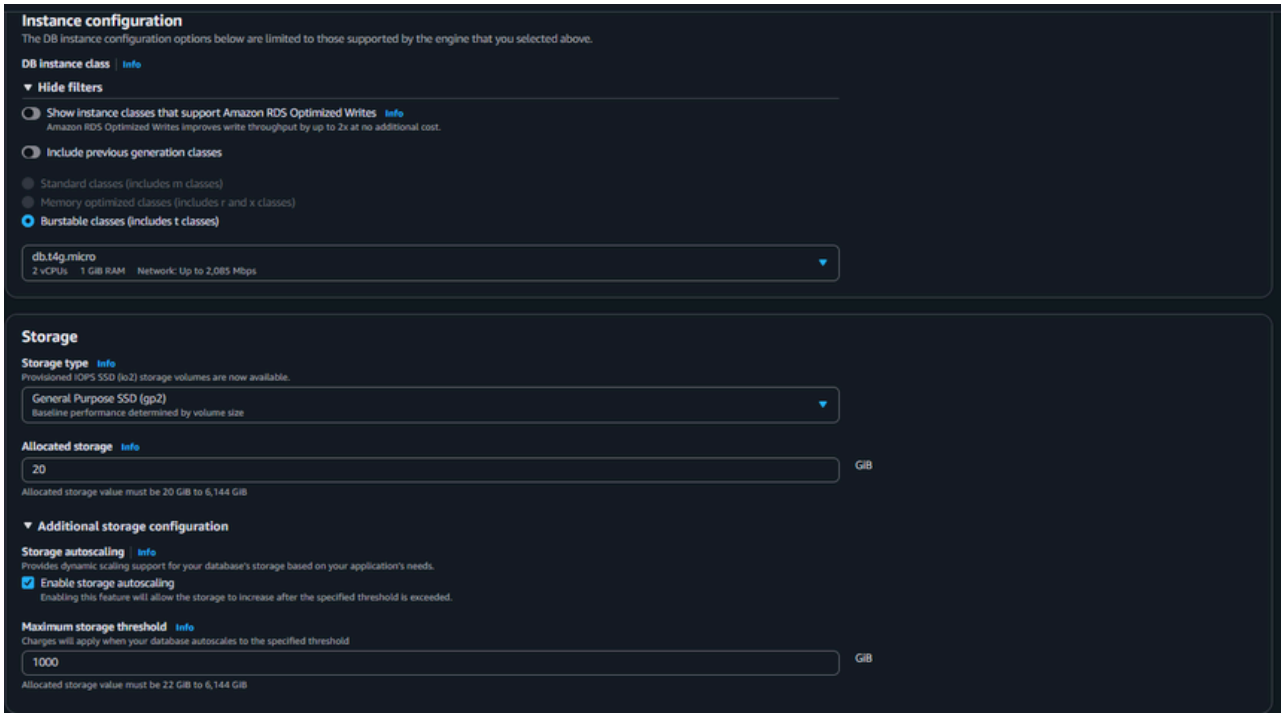


In settings section, type db name, master username and set password for access it later.

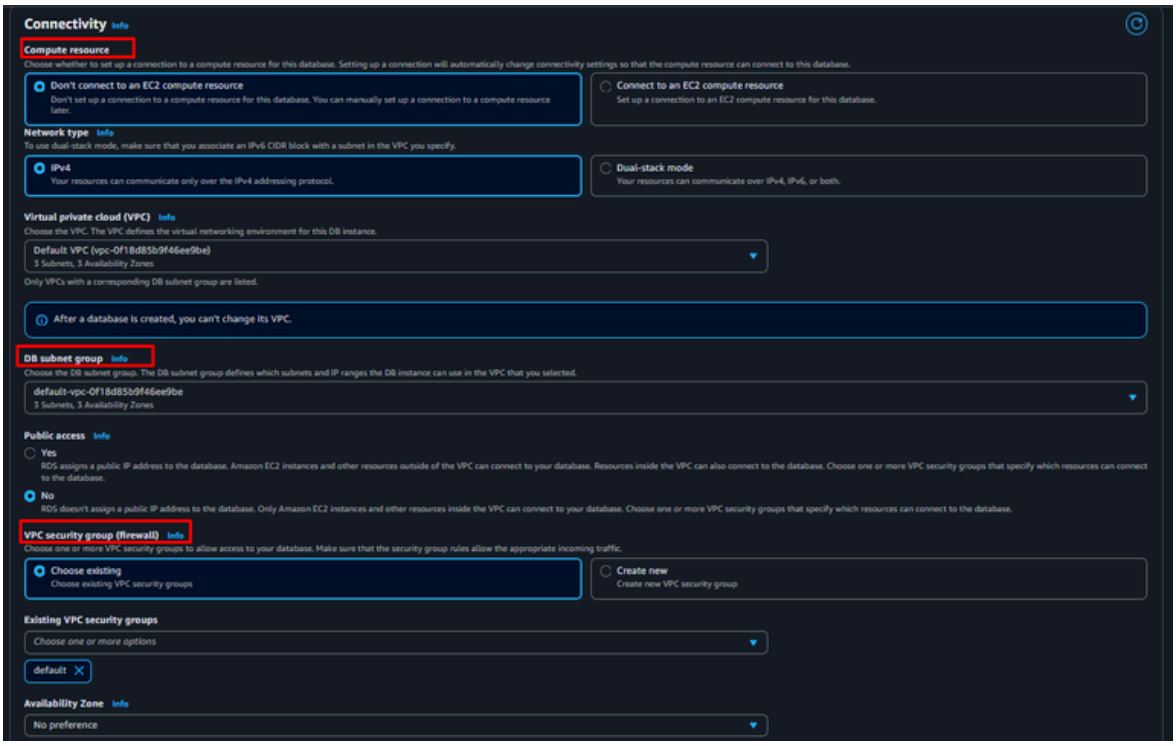


Hands-on with (RDS + MySQL + EC2):

In the Instance configuration & Storage section, I keep all as it is, if you need any changes you can do.

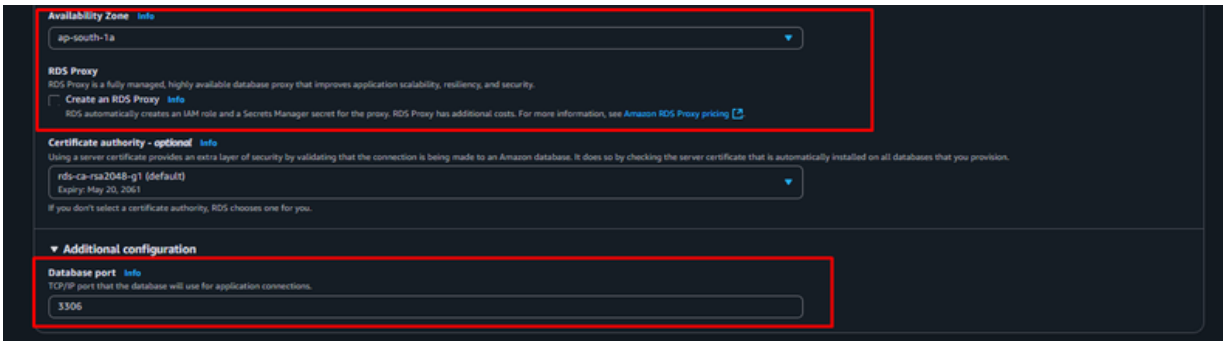


In the Connectivity section, configure your subnet groups and security groups, SG will be the same for EC2 instance.

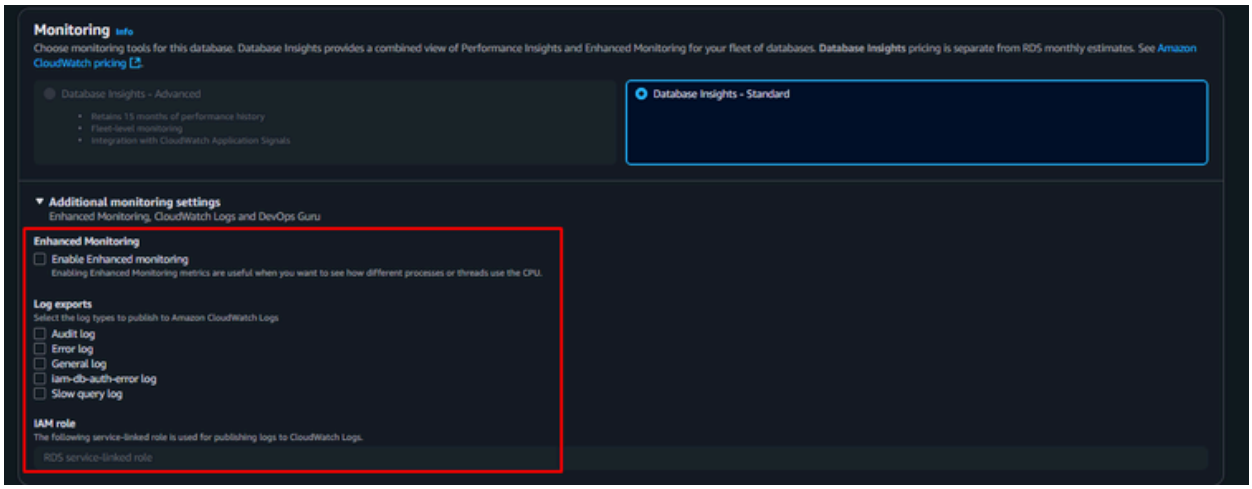


Hands-on with (RDS + MySQL + EC2):

Don’t forget to choose AZ. If you need you can set a custom port for your MySql connection.



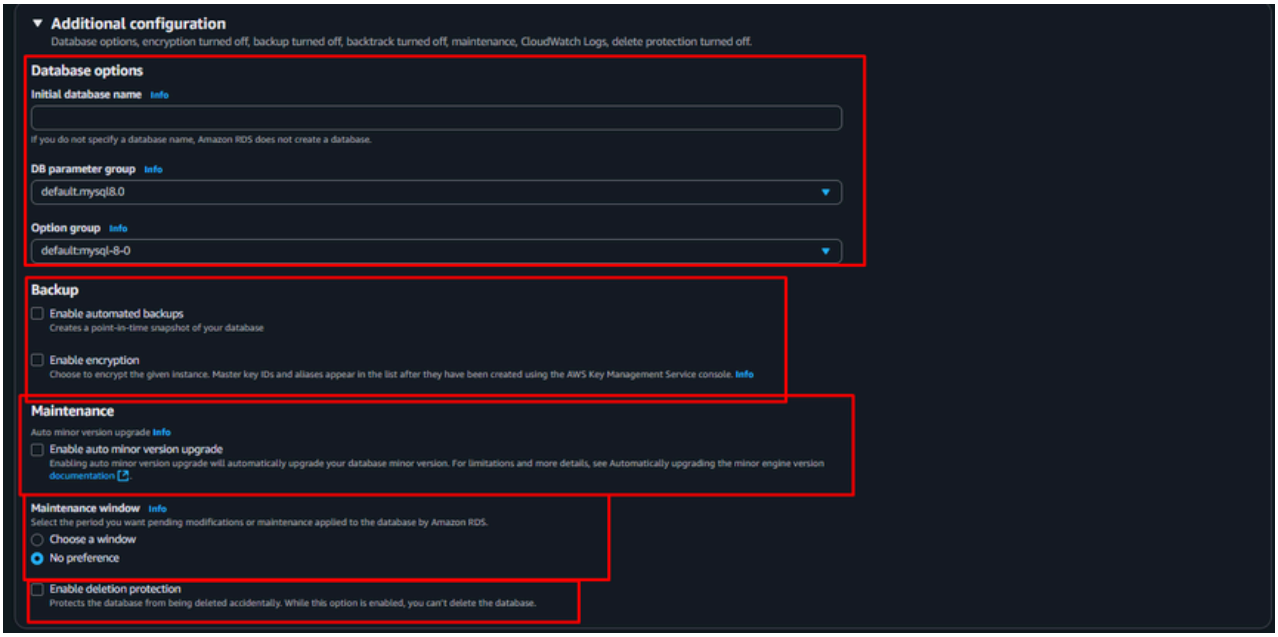
In the Monitoring section, you can choose the logs option, i keep it as it is.



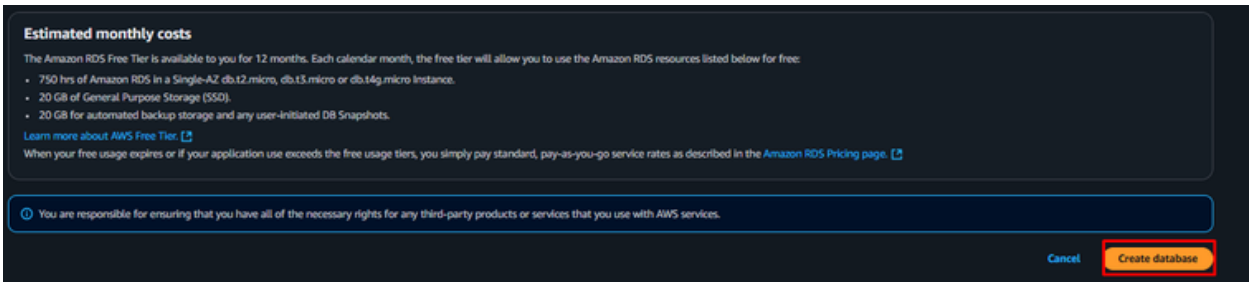
In the Additional configuration, you can choose the backup, maintance and maintaince window option. I keep it default i don’t need an backup.

Hands-on with (RDS + MySQL + EC2):

In the Additional Configuration, you can choose backup, maintenance, and maintenance window options. I kept it default as I don't need a backup.



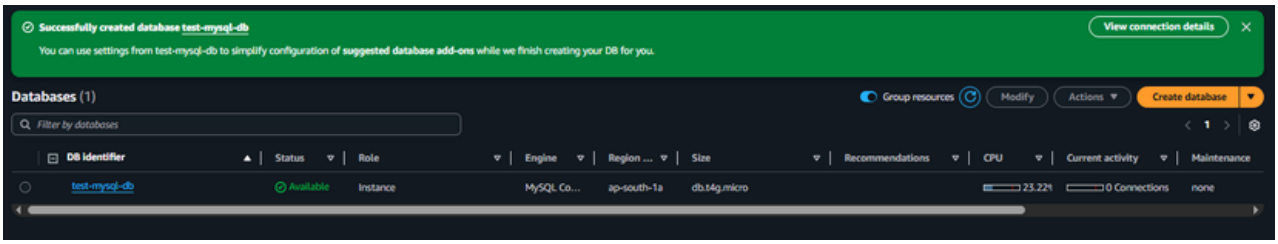
One important thing: enable deletion protection by checking this option. Your database won't be deleted when it is selected.



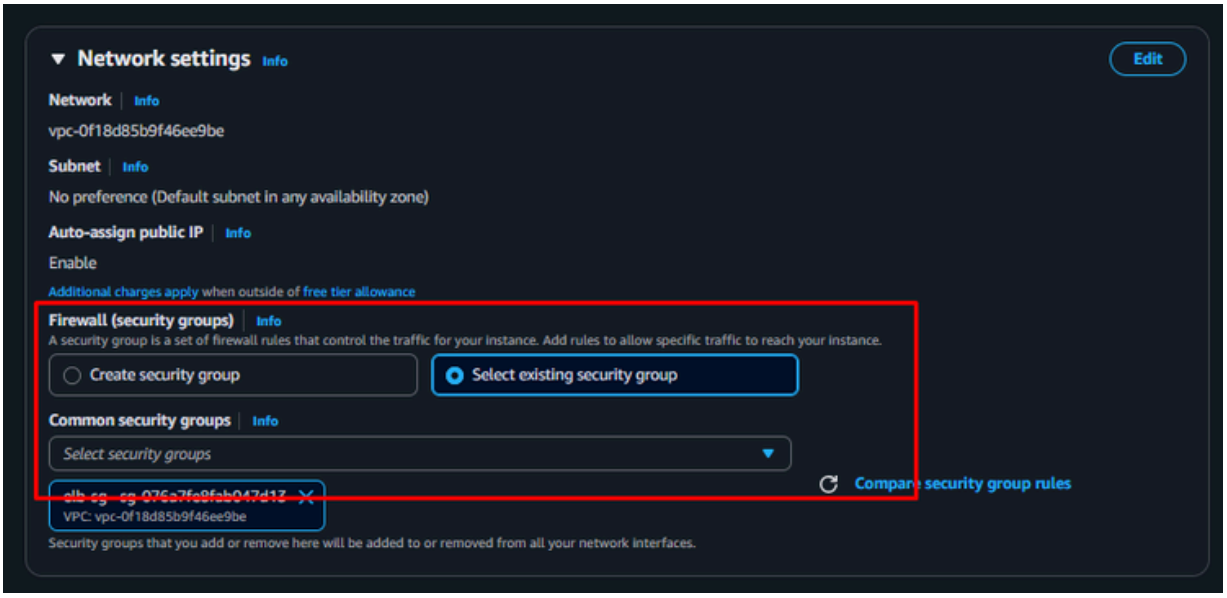
Now, click on create database button and create it.

Hands-on with (RDS + MySQL + EC2):

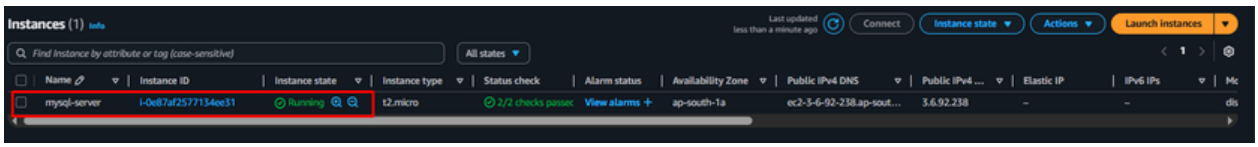
Databse created successfully.



Step 5: Now launch an EC2 instance with the same Security Group that is used in the database.



Here is my EC2 instance:



Hands-on with (RDS + MySQL + EC2):

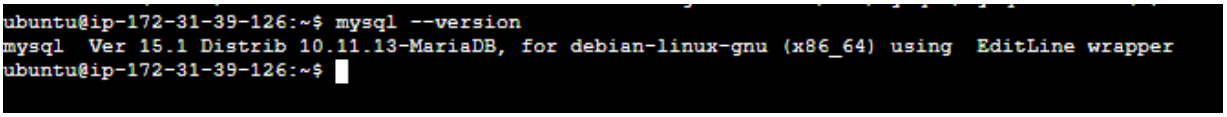
Step 6: Now it's time to connect our database with the EC2 instance. Before doing this, I provide a command-line file that can be used to connect to and test the RDS from EC2.

```
# update packages
sudo apt update && sudo apt upgrade -y

# install mysql
sudo apt install mariadb-client

# test
mysql --version

# retrieved endpoint, username, and password to connect
 mysql -h <RDS_Endpoint> -P 3306 -u <RDS_Username> -p

# after connect show db lists
SHOW DATABASES;

# create new database
CREATE DATABASE dbname;

# switch to your db
USE dbname;

# create user table
CREATE TABLE users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL
);

# insert sample data
INSERT INTO users (name, email)
VALUES
('Alamgir Hosen', 'alamgir@example.com'),
('Abir Ahmed', 'abir@example.com');

# see table lists
SHOW TABLES;

# show data from specific table
SELECT * FROM users;

# leave from db
exit;
```

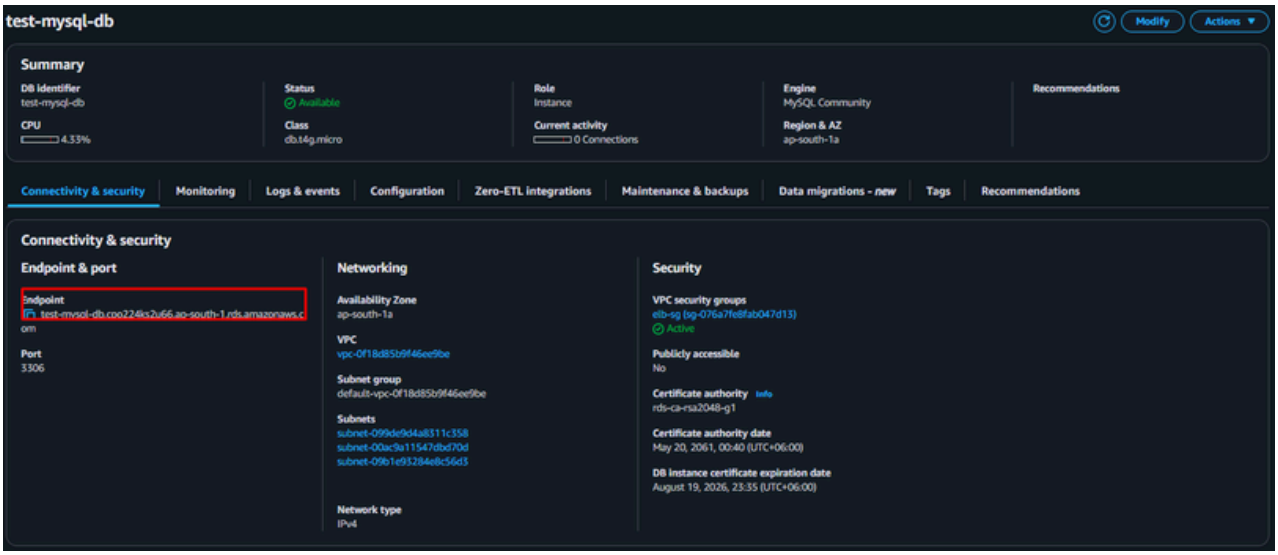
Hands-on with (RDS + MySQL + EC2):

Don’t forget to set inbound roles of Security Groups for your MySql connection.

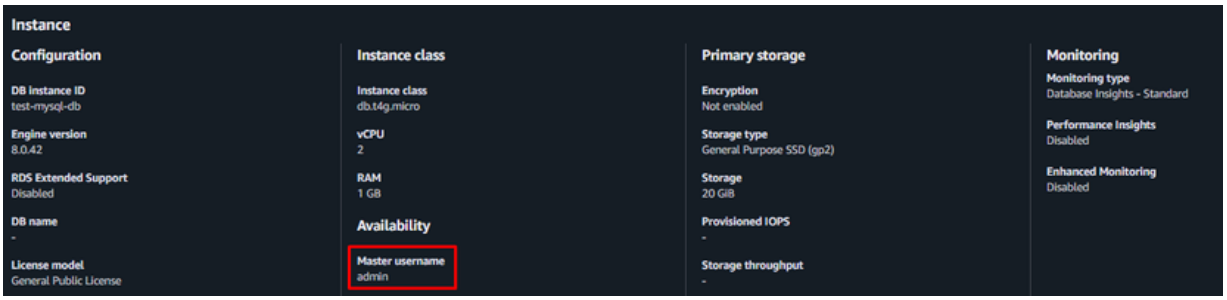
Now, connect to your EC2 instance and install the necessary configurations. You can do this via SSH or directly from the EC2 terminal. In my case, I used the EC2 terminal.



MySQL installed successfully.
Now, connect to the RDS MySQL. To do this, you need the endpoint and master username, which you can find in your database details.

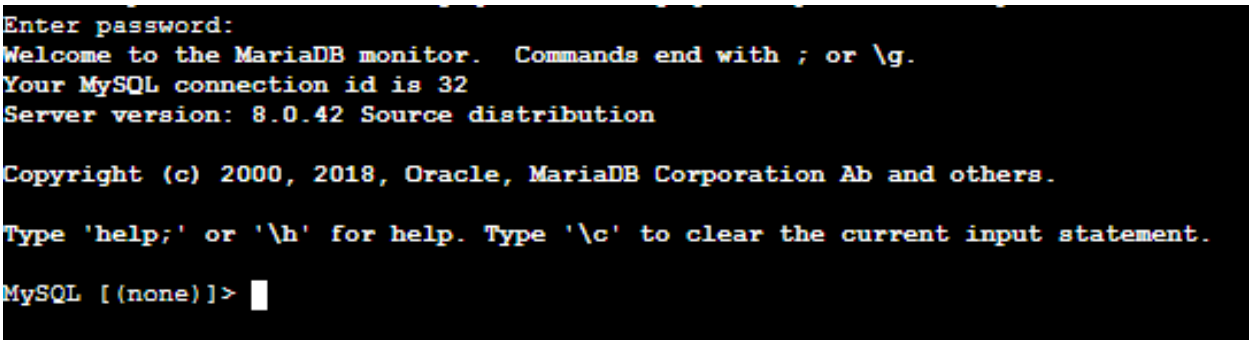


Hands-on with (RDS + MySQL + EC2):



Now run this command with your configuration details:
mysql -h <RDS_Endpoint> -P 3306 -u <RDS_Username> -p

Type your database password and press enter.



Successfully connected to the database. Now, test it by creating a new database and table, and inserting some records as well.

Hands-on with (RDS + MySQL + EC2):

Here are the results of the operations I performed using my command file: First, I created a new database, then a new table called users, inserted some user data, and finally displayed the list of users.

```
MySQL [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.009 sec)

MySQL [(none)]> CREATE DATABASE testdb;
Query OK, 1 row affected (0.009 sec)

MySQL [(none)]> use testdb;
Database changed
MySQL [testdb]> CREATE TABLE users (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(100) NOT NULL,
    ->     email VARCHAR(100) UNIQUE NOT NULL
    -> );
Query OK, 0 rows affected (0.045 sec)

MySQL [testdb]> INSERT INTO users (name, email)
    -> VALUES
    -> ('Alamgir Hosen', 'alamgir@example.com'),
    -> ('Abir Ahmed', 'abir@example.com');
Query OK, 2 rows affected (0.006 sec)
Records: 2  Duplicates: 0  Warnings: 0

MySQL [testdb]> SHOW TABLES;
+------------------+
| Tables_in_testdb |
+------------------+
| users            |
+------------------+
1 row in set (0.002 sec)

MySQL [testdb]> SELECT * FROM users;
+----+---------------+---------------------+
| id | name          | email               |
+----+---------------+---------------------+
|  1 | Alamgir Hosen | alamgir@example.com |
|  2 | Abir Ahmed    | abir@example.com    |
+----+---------------+---------------------+
2 rows in set (0.003 sec)

MySQL [testdb]>
```

Successfully connected the RDS MySQL database with the EC2 instance.

Thank You

Stay Connect:

/in/alamgirweb11

/alamgirweb11

Dynamo DB

Amazon DynamoDB is a serverless, NoSQL, fully managed database with single-digit millisecond performance at any scale.

It is a **fully managed NoSQL database service** provided by AWS.

- **NoSQL** means it doesn't use the traditional table–row–column schema like relational databases (MySQL, PostgreSQL). Instead, it stores data as **key-value pairs** and **documents**.
- It's designed for **fast, consistent performance at any scale**.
- AWS takes care of infrastructure management, scaling, backups, patching, and availability — so developers can focus on building apps instead of managing DB servers.

Capabilities of DynamoDB

1. Query & Scan Operations

- **Query**: Efficient lookup based on primary key or indexes.
- **Scan**: Reads all items in a table (less efficient).

2. Indexes

- **Primary Key**: Partition key (mandatory) + optional sort key.
- **Secondary Indexes**:
  - **Global Secondary Index (GSI)** – query using a different partition key.
  - **Local Secondary Index (LSI)** – same partition key, different sort key.

3. DAX (DynamoDB Accelerator)

- In-memory caching layer.
- Reduces read latency from milliseconds to microseconds.

4. Streams

- Captures table activity (insert/update/delete).
- Used for event-driven applications, analytics, auditing.

5. Transactions

- Supports ACID transactions for multi-item, multi-table operations.

6. TTL (Time to Live)

- Automatically delete expired items (useful for session data, logs).

7. Backup & Restore

- On-demand backups and point-in-time recovery (PITR).

8. Integration

- Works seamlessly with **AWS Lambda, API Gateway, Kinesis, S3, Athena, EMR, SageMaker**, etc.

Key Features of DynamoDB

1. Managed & Serverless

- No need to provision or manage servers.
- Auto-scaling based on traffic.
- Pay only for what you use.

2. High Performance (Single-digit ms Latency)

- Optimized for low-latency reads/writes.
- Suitable for real-time applications.

3. Scalability

- Handles **millions of requests per second**.
- Scales seamlessly without downtime.

4. Flexible Data Model

- **Tables, Items, and Attributes**:
  - A **Table** = collection of items (like a database table).
  - An **Item** = single record (like a row).
  - **Attributes** = fields of an item (like columns).
- Supports nested documents (JSON-like).

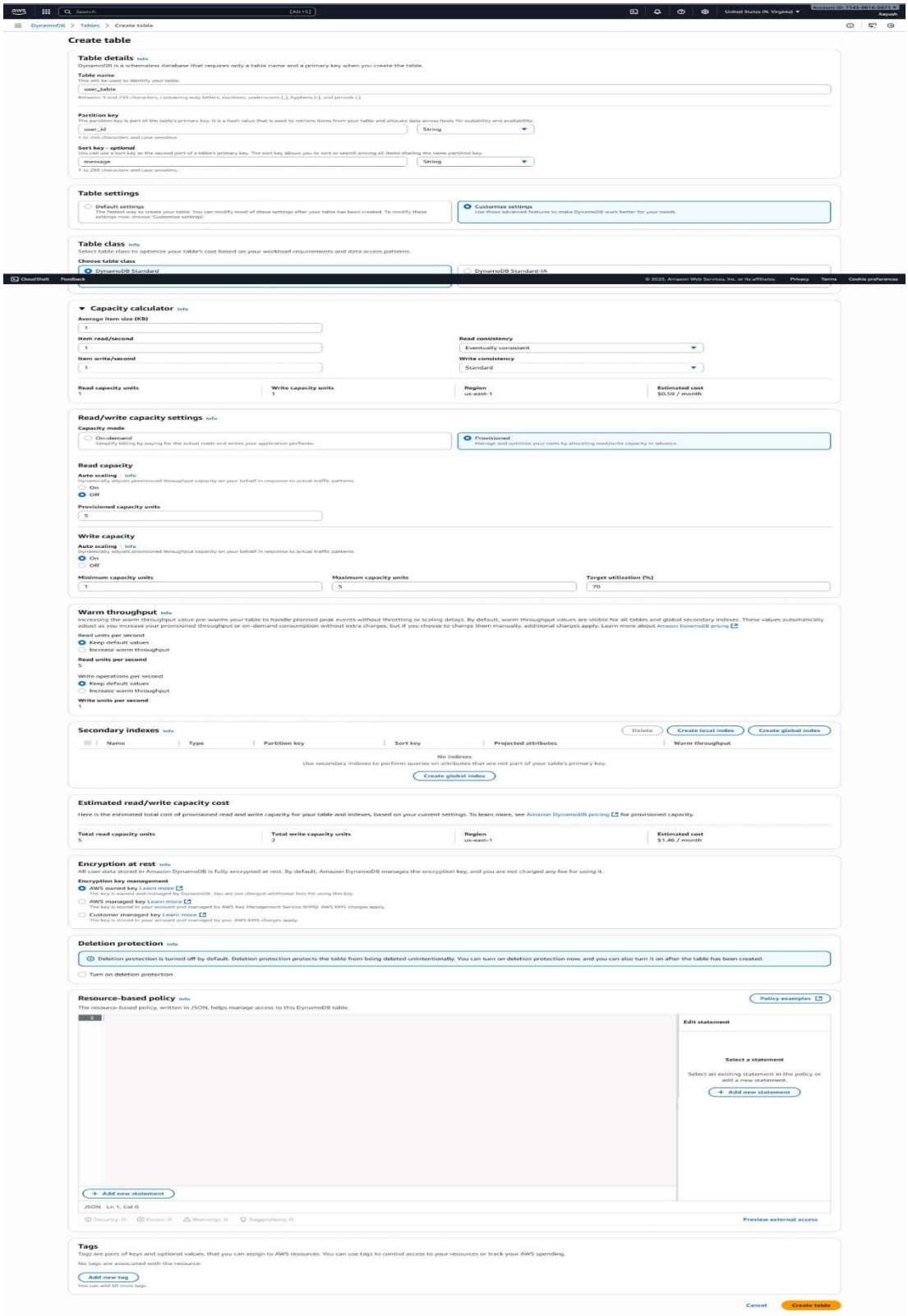
5. Primary Key Options

- **Partition Key (hash key):** Ensures data distribution.
- **Partition Key + Sort Key (composite key):** Allows range queries and grouping.

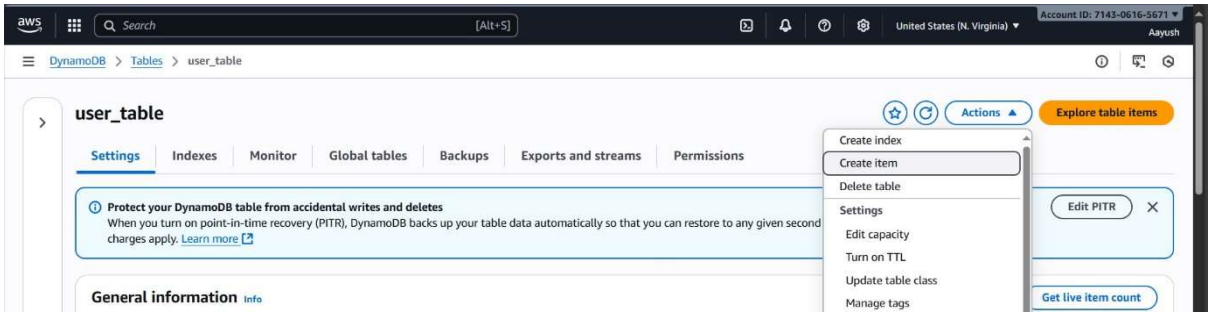
6. Indexes for Queries

- **Global Secondary Index (GSI):** Query on non-primary attributes.
- **Local Secondary Index (LSI):** Query with same partition key but different sort key.

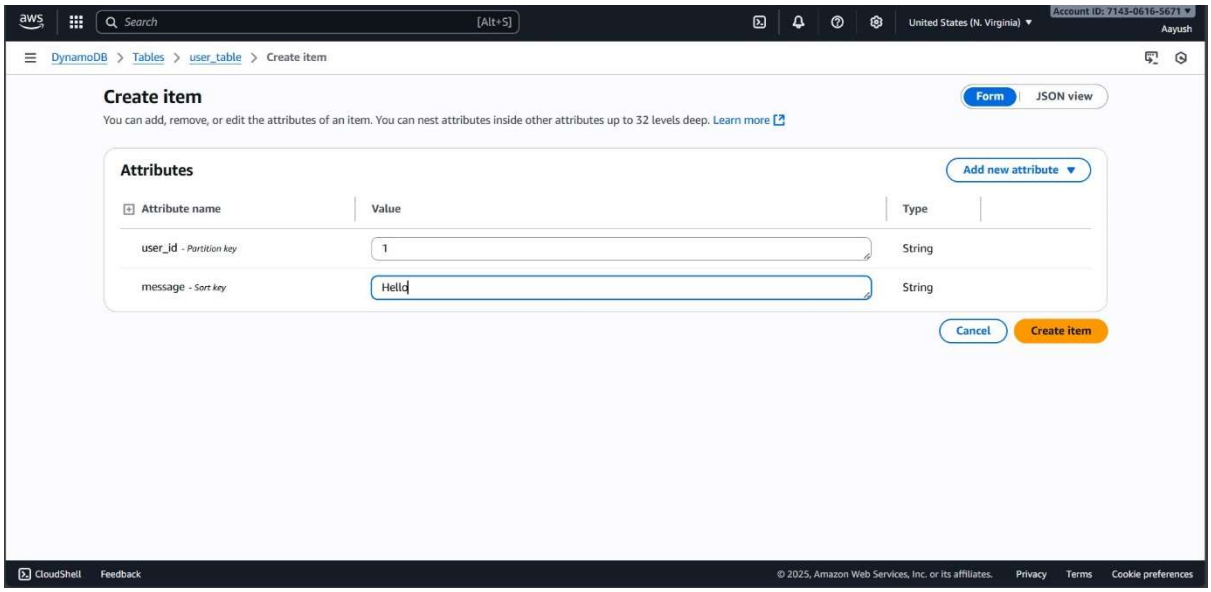
Step 1: - Open AWS Console and search for DynamoDB and click on Create table and insert detail as shown in picture (for demo only).



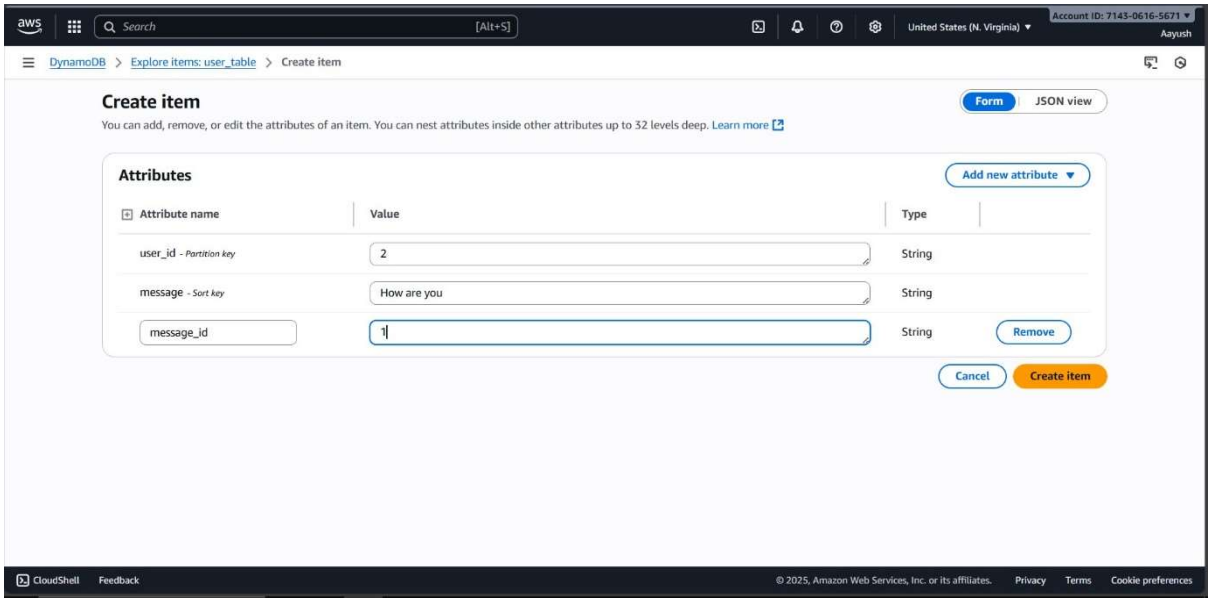
Step 2: - Now goto DynamoDB dashboard and on table name and then click on Action button then click on Create item.



Step 3: - Now enter value and you can add a new attribute then click on Create item.



Step 4: - Adding multiple custom item in table.



Step 5: - Now you can add parameters and click on scan and then click on qurey to run.