## RDS TASK

1. ***Create MariaDB DB on EC2.***

## Task Title

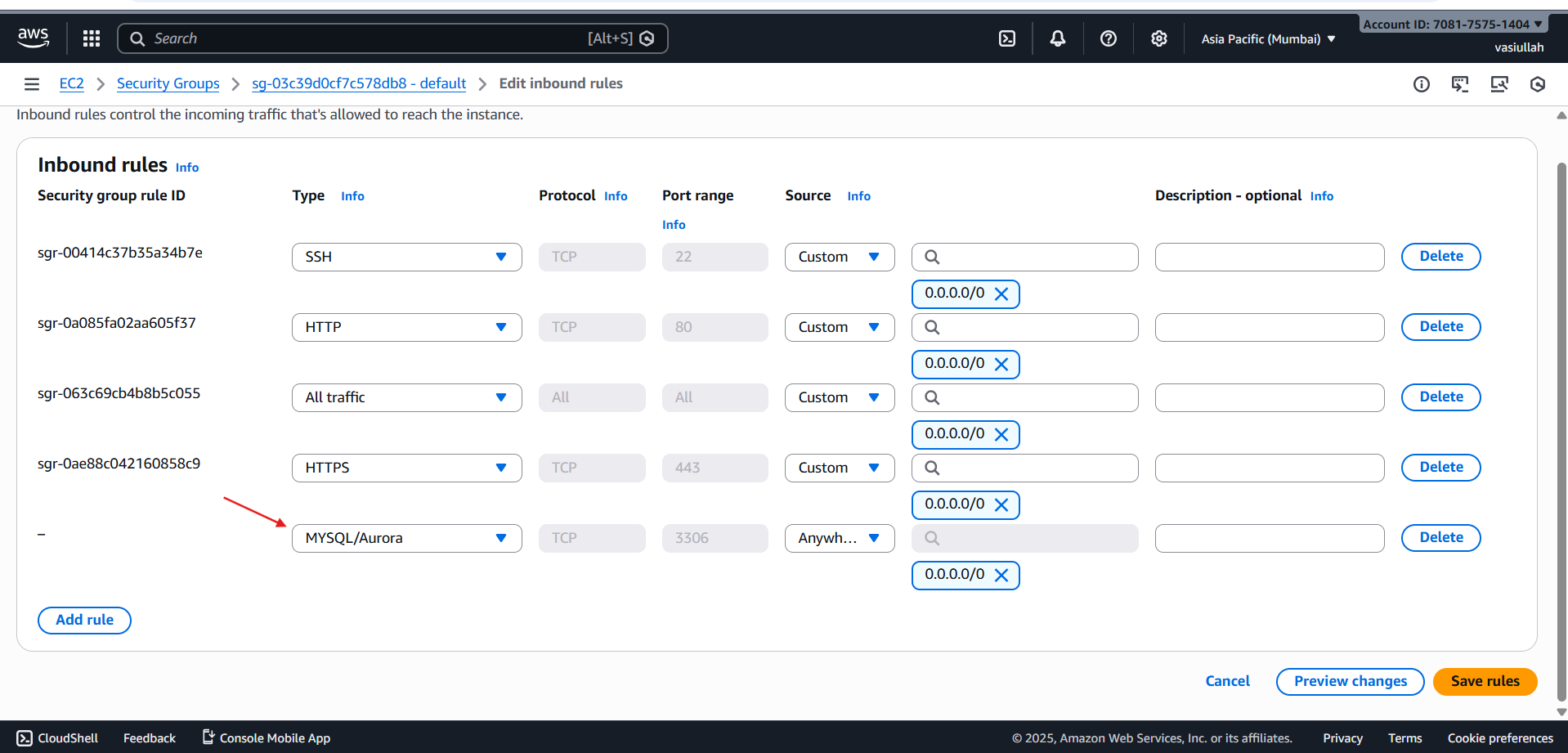
* Create MariaDB DB on EC2

## Objective

* To install the MariaDB server, configure security group rules for remote access, and automate the creation of a specific database and user via a SQL setup script.

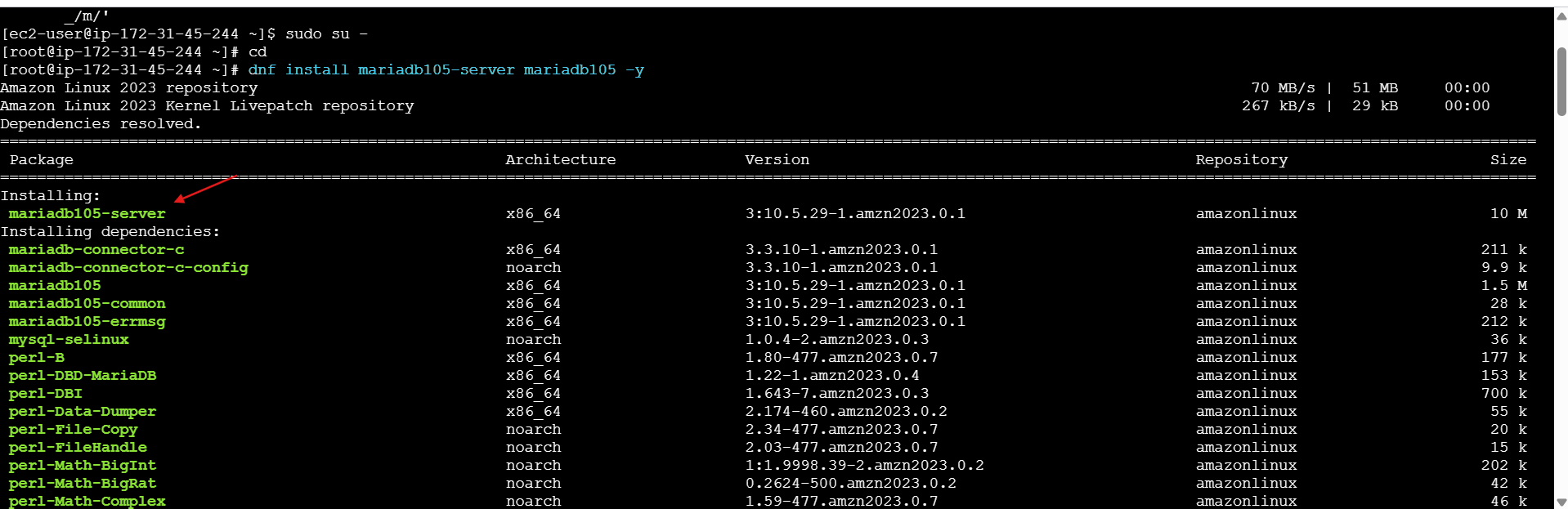
# Prerequisites

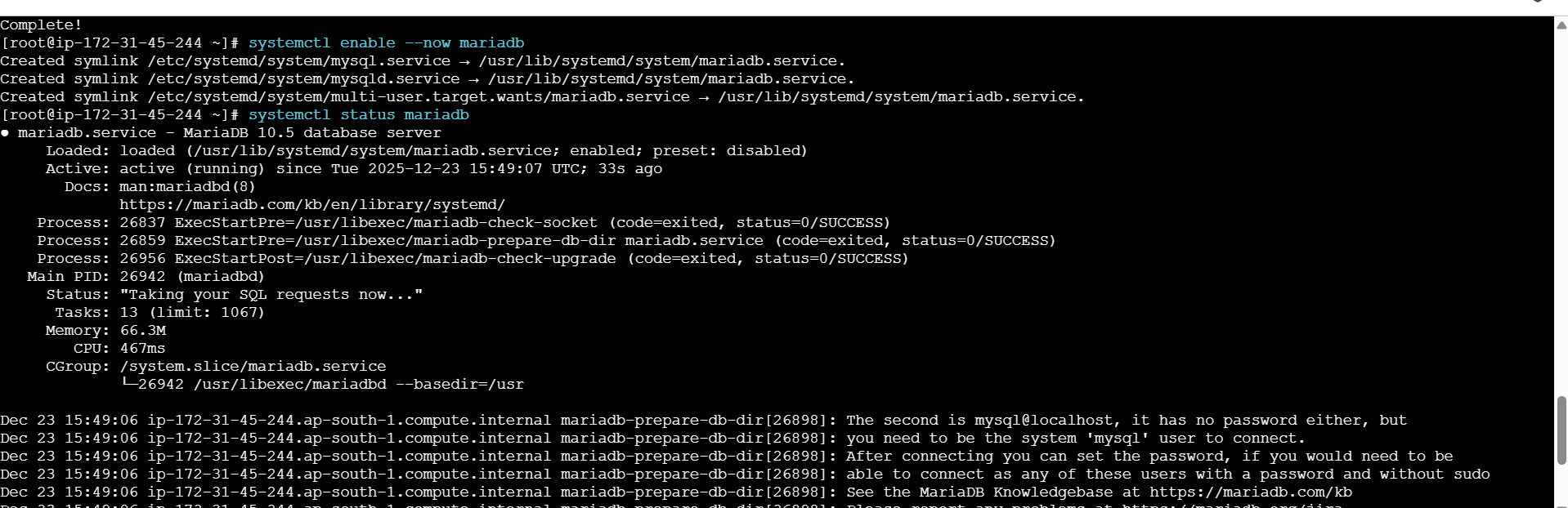
* AWS EC2 Instance: Running Amazon Linux 2023.
* Root/Sudo Access: Ability to switch to the root user for system-level changes.
* Network Access: Permissions to modify EC2 Security Group inbound rules.

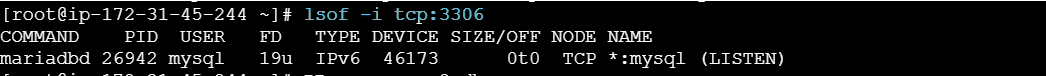
******

## Step-by-Step Implementation

* Install MariaDB Server: Use the dnf package manager to install the MariaDB 10.5 server and client tools. dnf install mariadb105-server mariadb105 -y
* Enable and Start Service: Initialize the database service and set it to start automatically on boot. systemctl enable --now mariadb

******

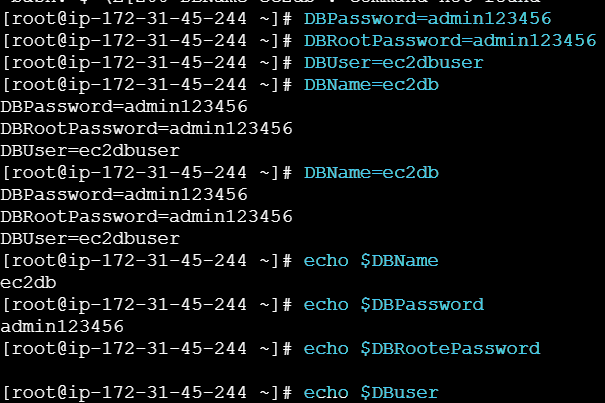
******

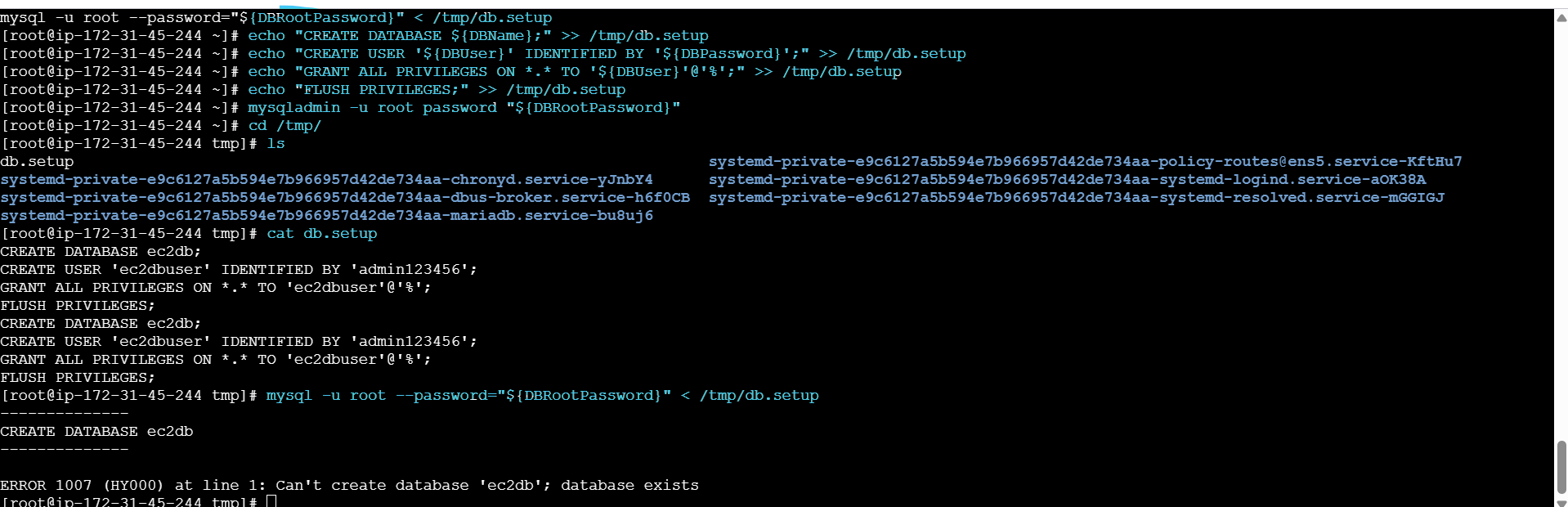
******

* DBName=ec2db
* DBUser=ec2dbuser
* DBPassword=admin123456

## Validation Steps

* Service Status: Run systemctl status mariadb to confirm the service is "active (running)".
* Port Monitoring: Run lsof -i tcp:3306 to verify that mariadbd is successfully listening on the MySQL port.
* Script Verification: Use cat /tmp/db.setup to ensure the SQL commands were generated with the correct variable expansion.

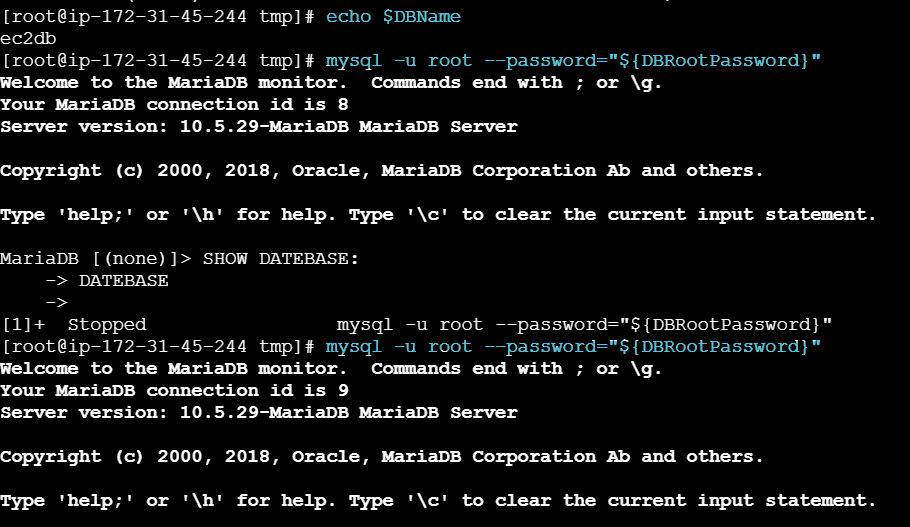
******

******

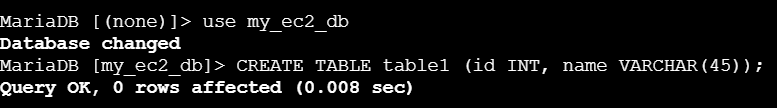
## Conclusion

* The MariaDB 10.5 server is successfully installed and running on the EC2 instance. While the automated script functions correctly, the "Database exists" error indicates that the implementation should be made more idempotent for future deployments to avoid execution breaks.

1. ***Insert some dummy data.***

******

******

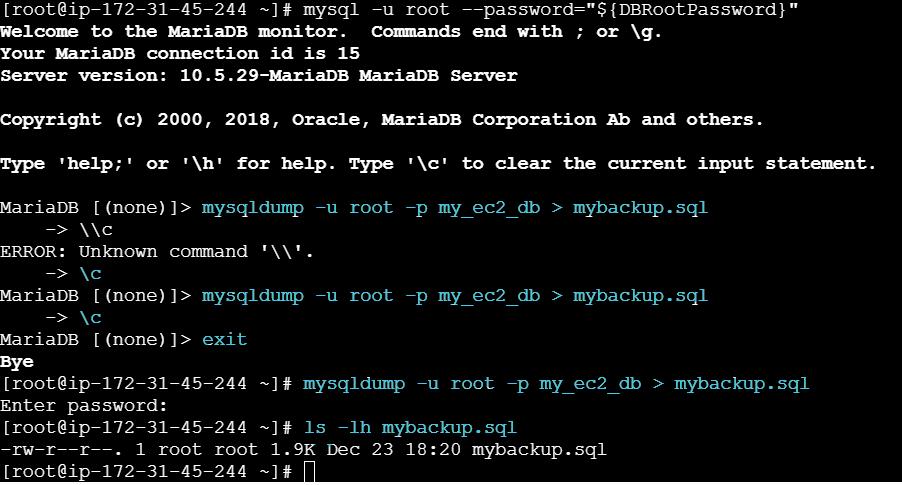
******

1. ***Take the backup of dummy data on EC2.***

* To deploy a MariaDB database server on an AWS EC2 instance, automate user and database creation, and perform a data backup using mysqldump.
* **Network Permissions:** Inbound rules allowing traffic on TCP port 3306.
* Create a setup script at /tmp/db.setup containing SQL commands to create the database and user.
* Manually create an additional database: CREATE DATABASE my\_ec2\_db;.
* Create a table within the new database: CREATE TABLE table1 (id INT, name VARCHAR(45));.

## Database Backup:

* Execute a backup of the target database: mysqldump -u root -p my\_ec2\_db > mybackup.sql.

******

* Backup Verification: Confirm the backup file exists and has size: ls -lh mybackup.sql

### Conclusion

* The MariaDB environment is successfully provisioned and secured via AWS Security Groups. By utilizing mysqldump, we have established a baseline for data persistence and recovery.

1. ***Launch MariaDB RDS instance.***

# Task Title

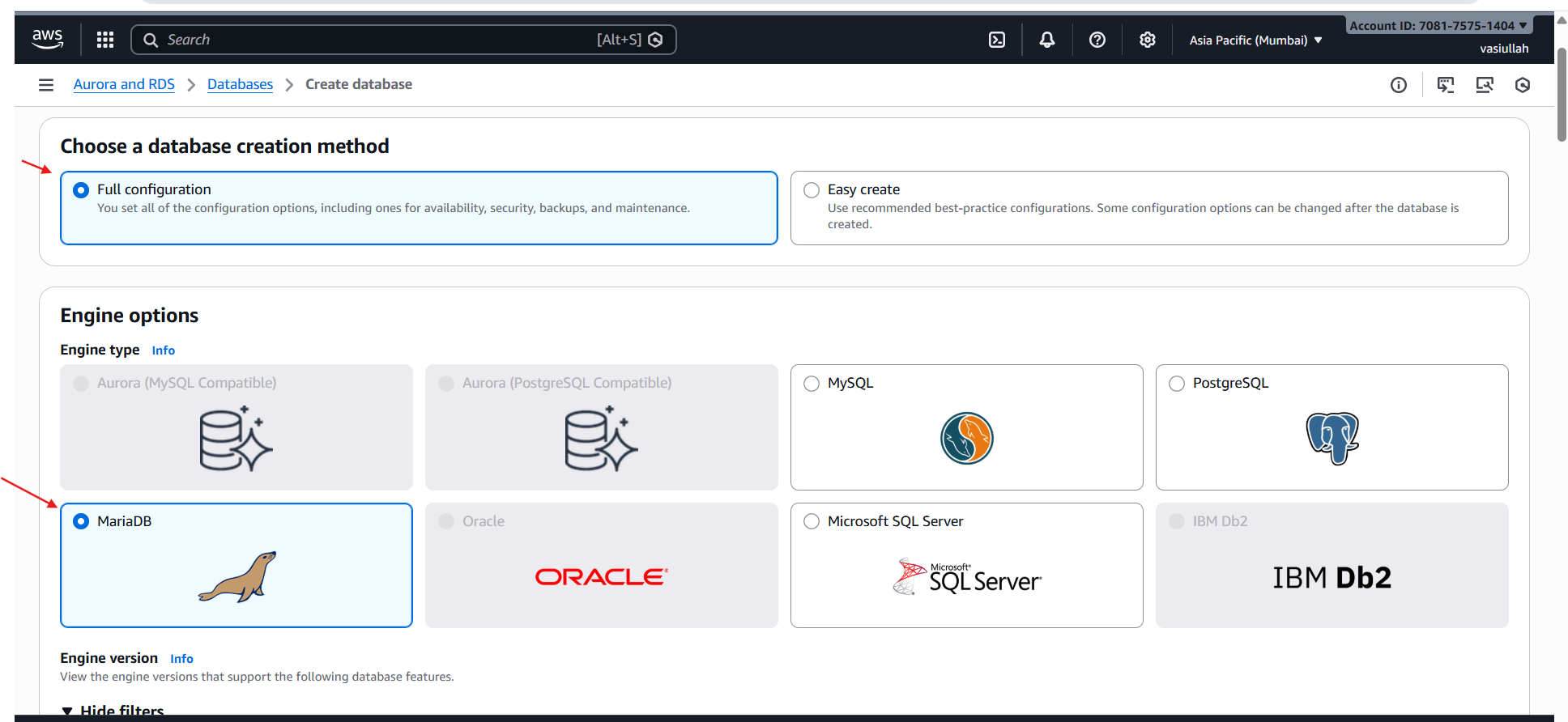
* Deploying a Private MariaDB RDS Instance with EC2 Connectivity

# Objective

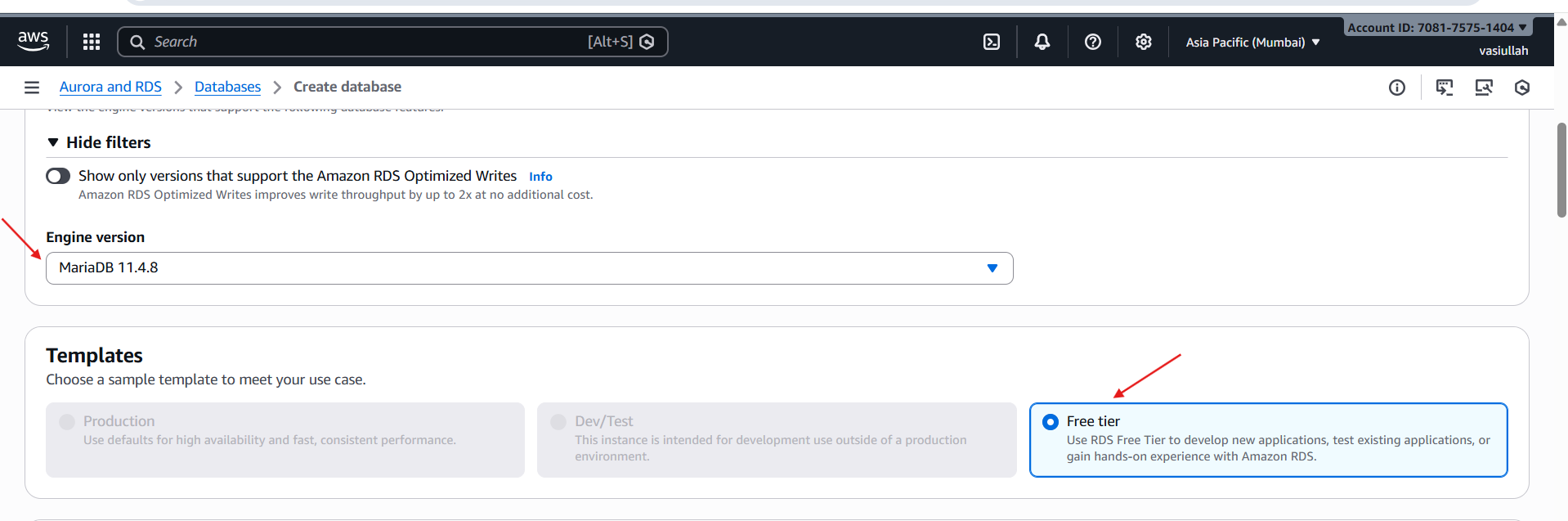
* To launch a secure, private MariaDB RDS instance using the AWS Free Tier, manage credentials via AWS Secrets Manager, and establish automatic connectivity to an existing EC2 compute

# Prerequisites

* **AWS Account**: Active account with permissions for RDS and Secrets Manager.
* **EC2 Instance**: An existing instance (e.g., i-0477c29c5aa8b8f5) to act as the application server.
* **VPC**: A Virtual Private Cloud configured for RDS deployment.

******

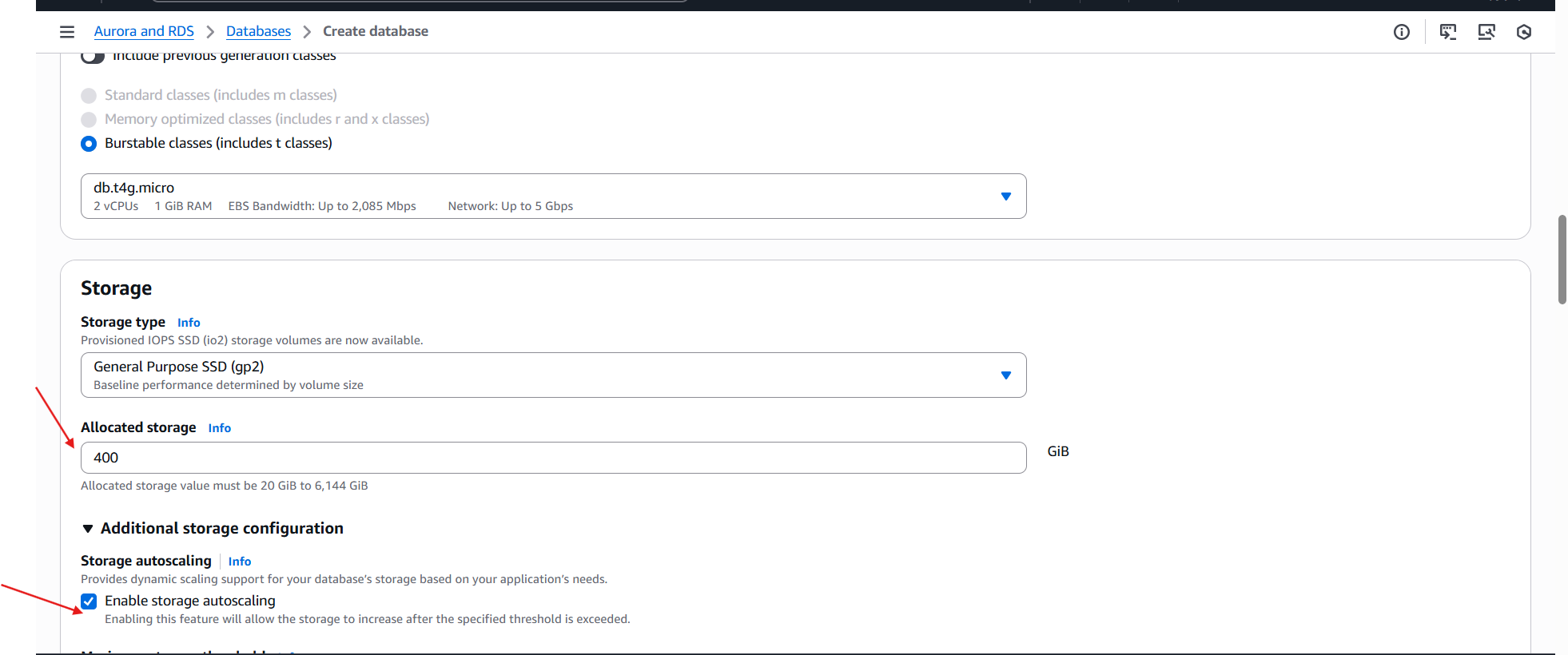
* Engine Options: Chose MariaDB as the database engine.
* Engine Version: Selected version 11.4.8.
* Template: Picked the Free tier template to minimize costs.

******

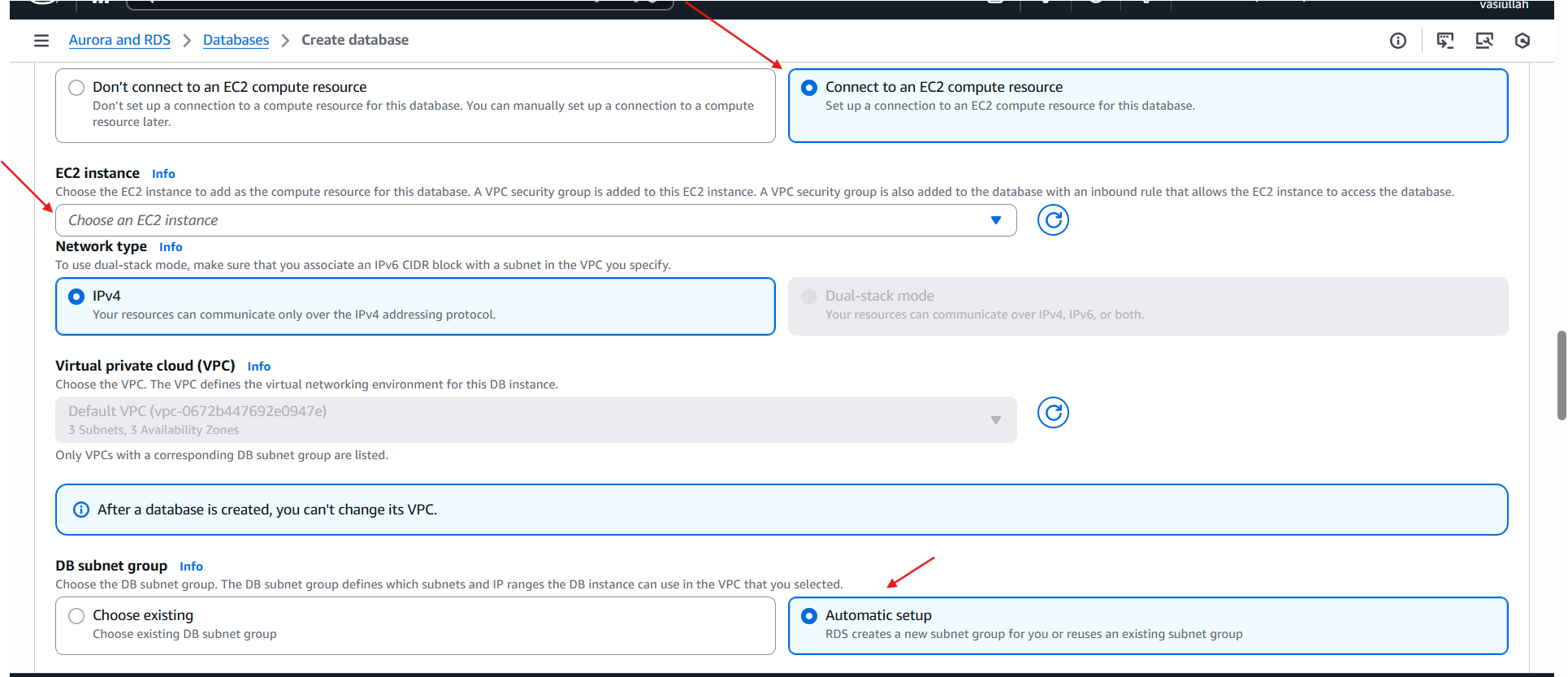
* Instance Identifier: Named the instance database-1.
* Credentials: Selected Managed by AWS Secrets Manager.
* Automation: AWS automatically generated and stored the master username and password in Secrets Manager using the default aws/secretsmanager encryption key.

******

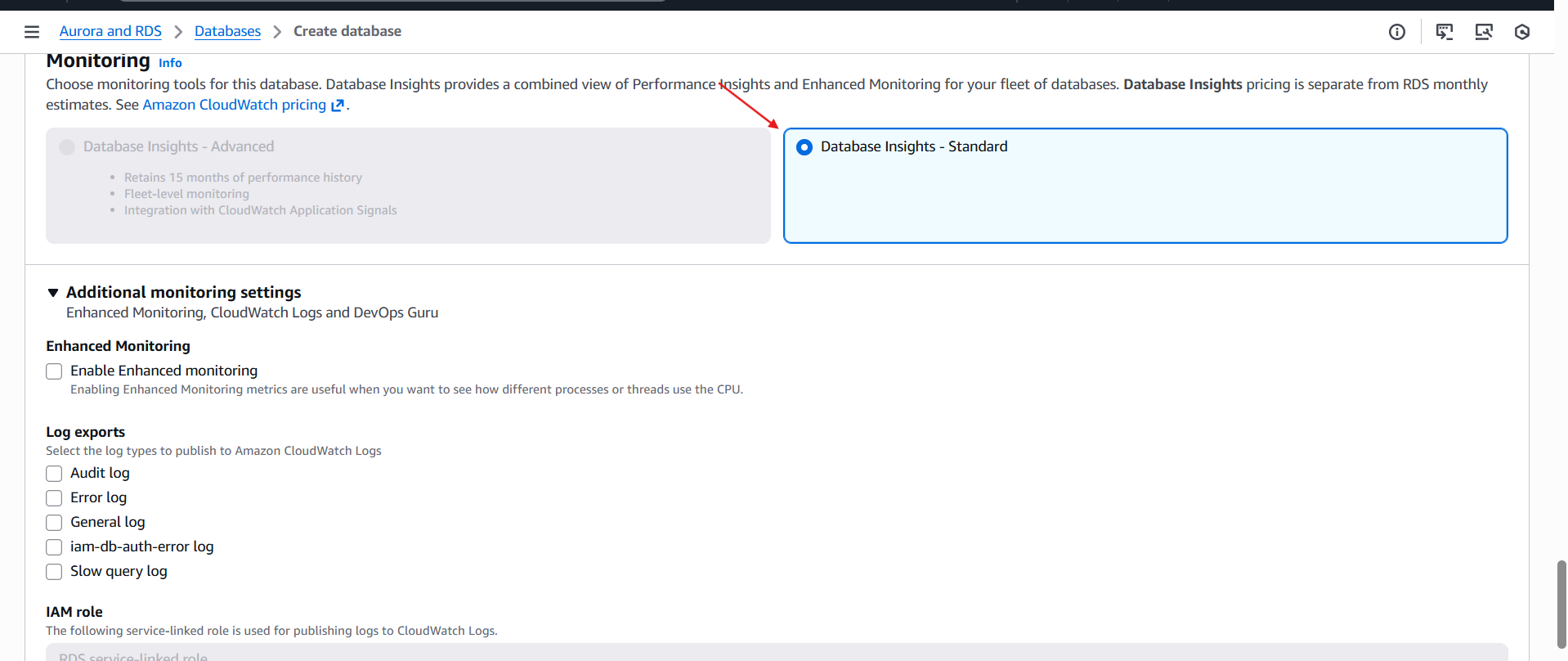
* Instance Class: Selected db.t4g.micro (Free Tier eligible).
* Storage Type: Chose General Purpose SSD (gp2).
* Allocation: Set to 20 GB default. Scalability: Enabled Storage autoscaling to allow the volume to grow dynamically as data increases.

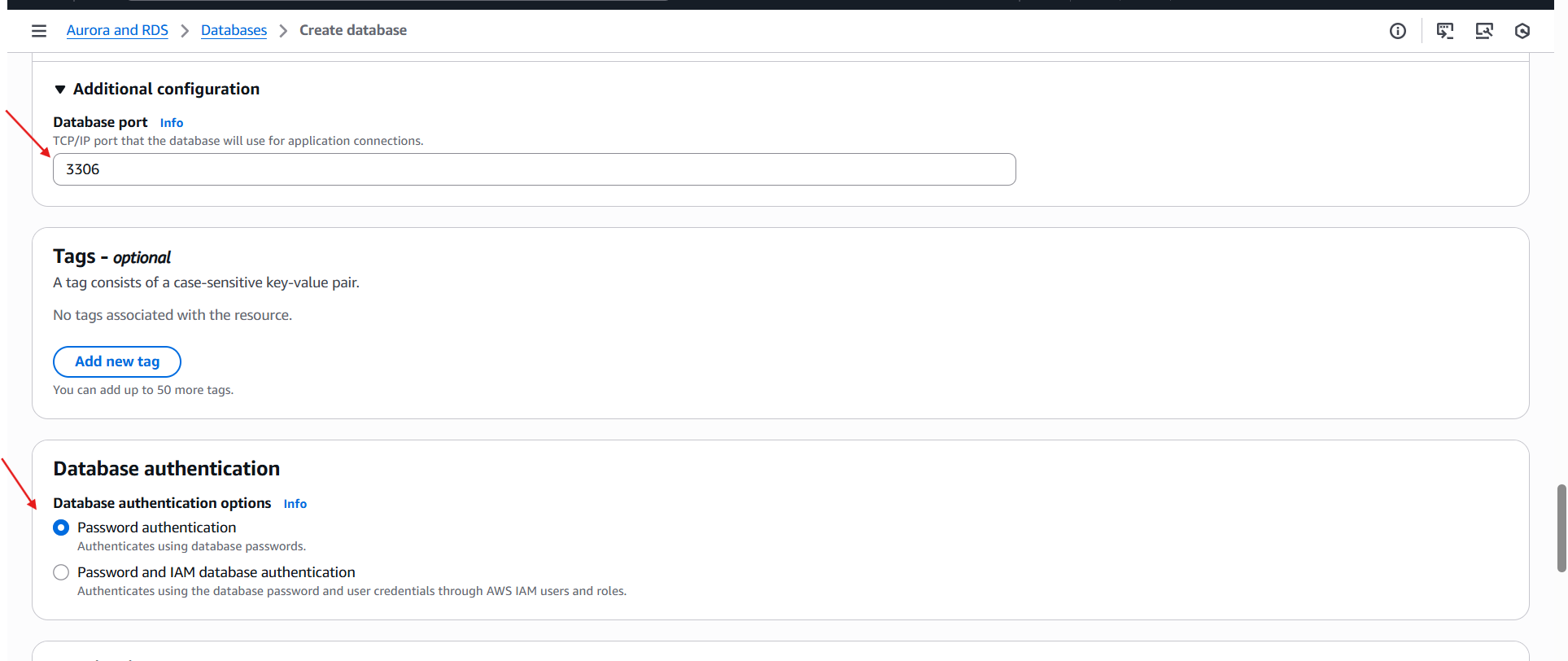
******

* Instance Status: Confirm the DB status shows as Available in the RDS Console.
* Connectivity Check: Verify the Endpoint and Port (3306) are displayed and reachable from the attached EC2 instance.
* Security Review: Confirm Public accessibility is set to No, ensuring the instance remains private.

******

* Monitoring: Check that Database Insights – Standard is active for performance tracking.

******

******

******

# Conclusion

* The MariaDB RDS instance was successfully deployed in a private subnet within the us-east-1a availability zone. By leveraging AWS Secrets Manager and the EC2 auto-connect feature, the deployment follows security best practices while reducing manual networking configuration.

1. ***Migrate database from EC2 to RDS.***

## Task Title

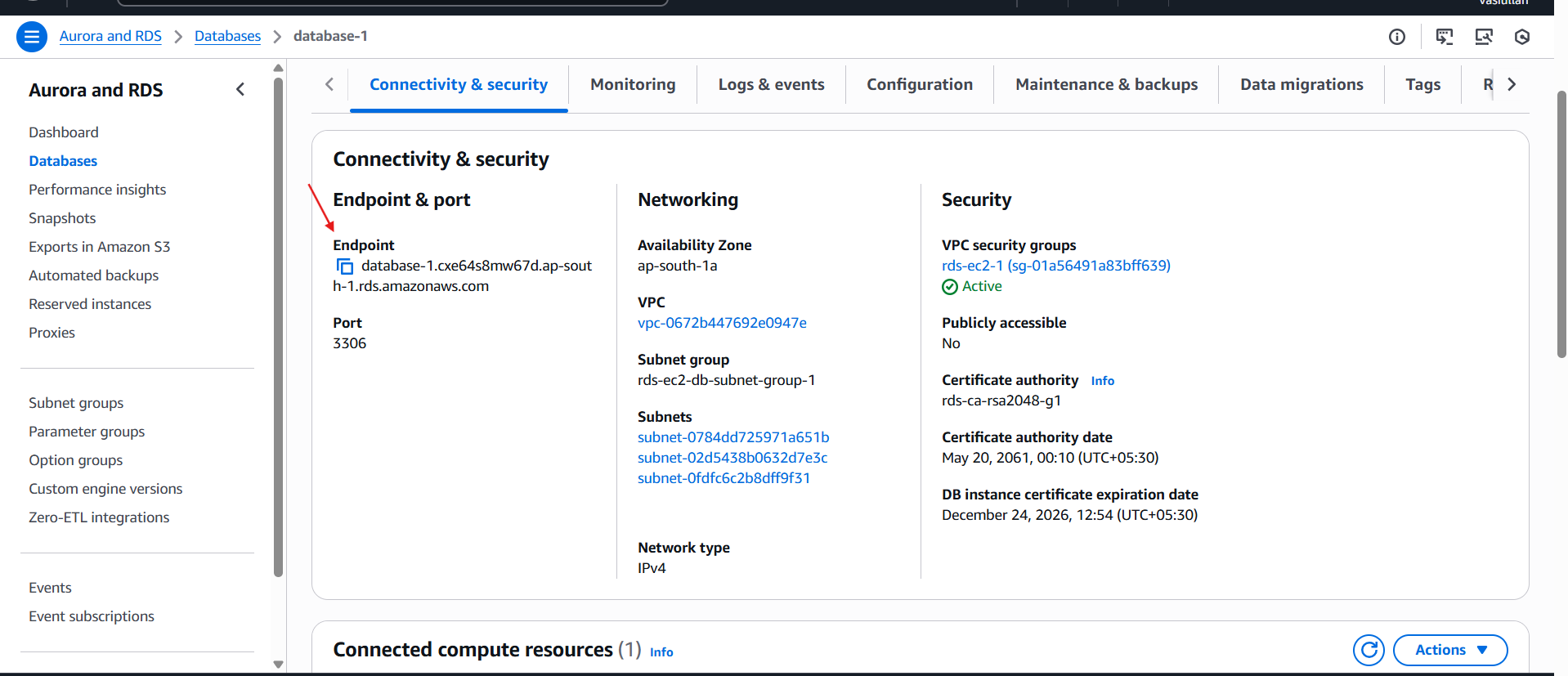
* Migrating a MariaDB Database from EC2 to an AWS RDS Instance

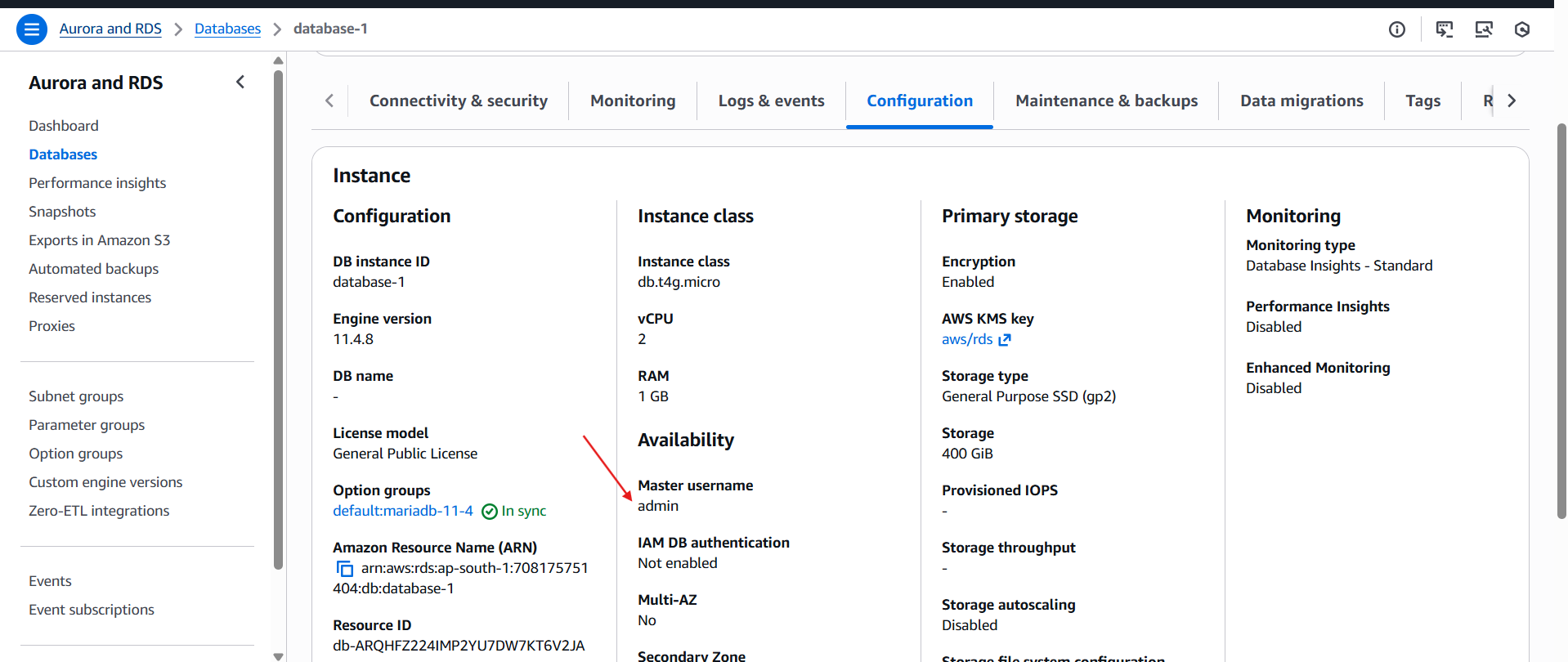
# Objective

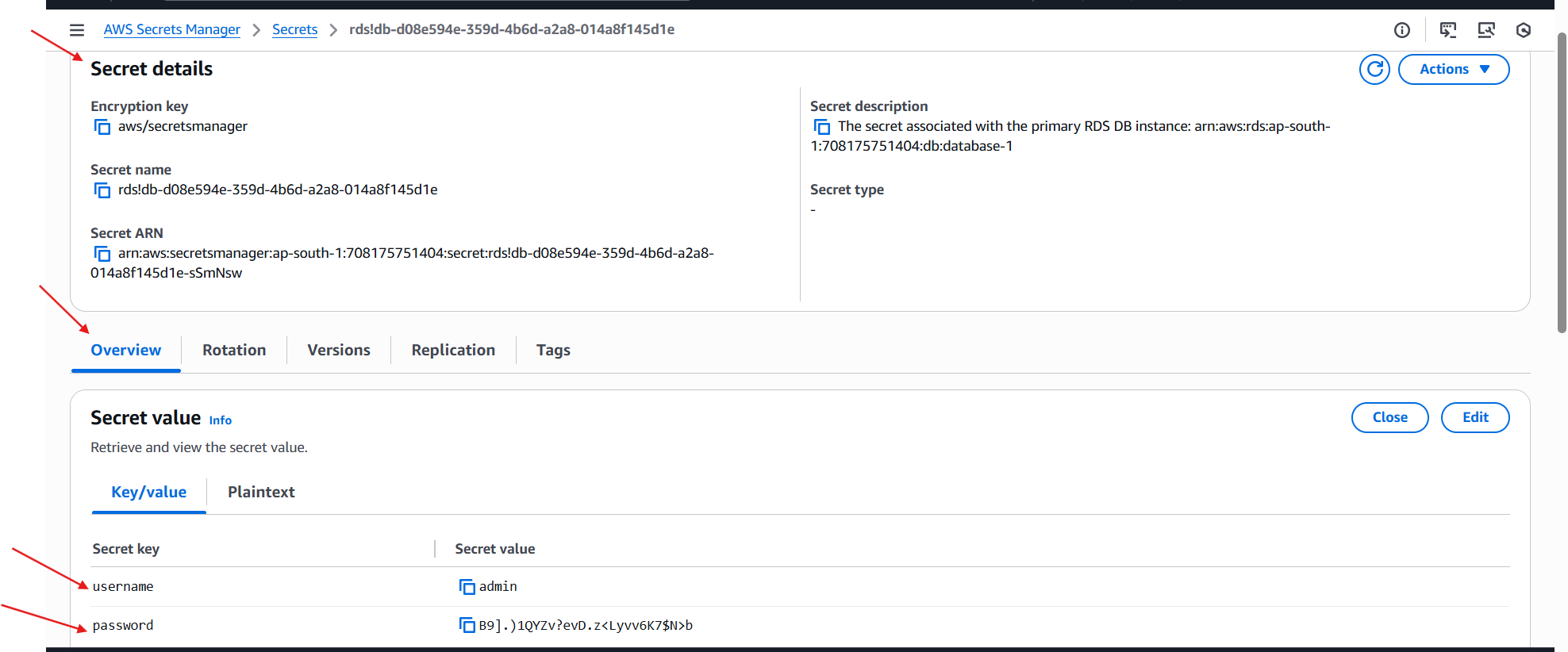
* To successfully export a database from a local MariaDB installation on an EC2 instance and restore it into a managed AWS RDS MariaDB instance, ensuring data integrity and connectivity.

# Prerequisites

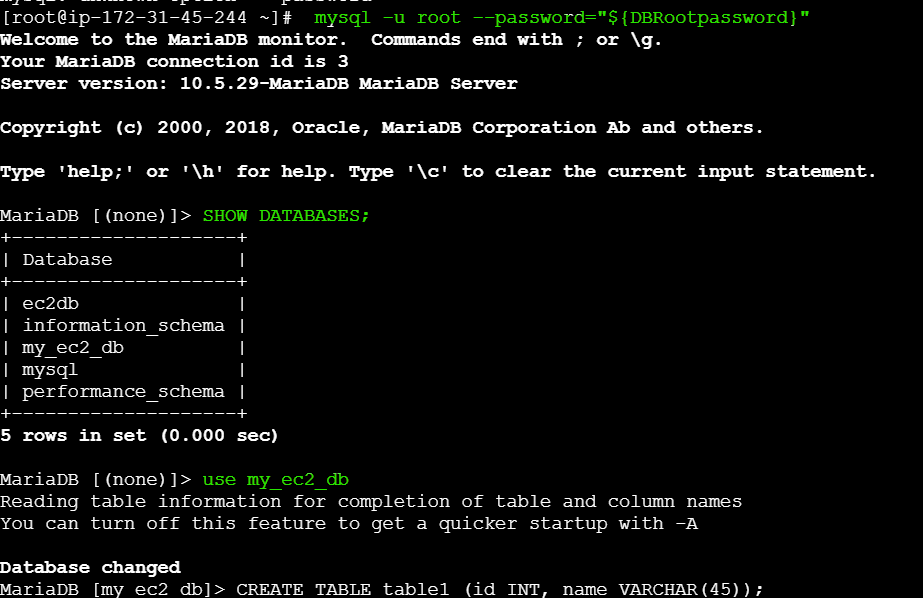
* **Active EC2 Instance**: Must contain the source MariaDB database and a valid backup file (e.g., /mybackup.sql).
* **Provisioned RDS Instance**: A MariaDB RDS instance (e.g., database-2) in an "Available" state.
* **Connectivity**: The EC2 instance must have network access to the RDS endpoint on port 3306.
* **Credentials**: Access to the RDS Master Username and Password, managed via AWS Secrets Manager.
* Access the RDS Console to identify the **Endpoint URL** and confirm the port is **3306**.
* Verify the **Master Username** (e.g., admin) under the Configuration tab.

******

******

******

* Log in to the local MariaDB on EC2: mysql -u root --password="${DBRootPassword}".
* Confirm the database and table contents:
* SHOW DATABASES;
* USE my\_ec2\_db;
* SELECT \* FROM table1;
* **Connect to RDS**: Log in to the RDS instance directly: mysql -h <RDS-Endpoint> -u admin -p
* **Verify Database Selection**: Switch to the migrated database: USE my\_ec2\_db;
* **Verify Schema**: Run SHOW TABLES; to ensure all tables were created.
* **Verify Data Integrity**: Execute SELECT \* FROM table1; and compare the output with the original EC2 data to confirm they are identical.

******

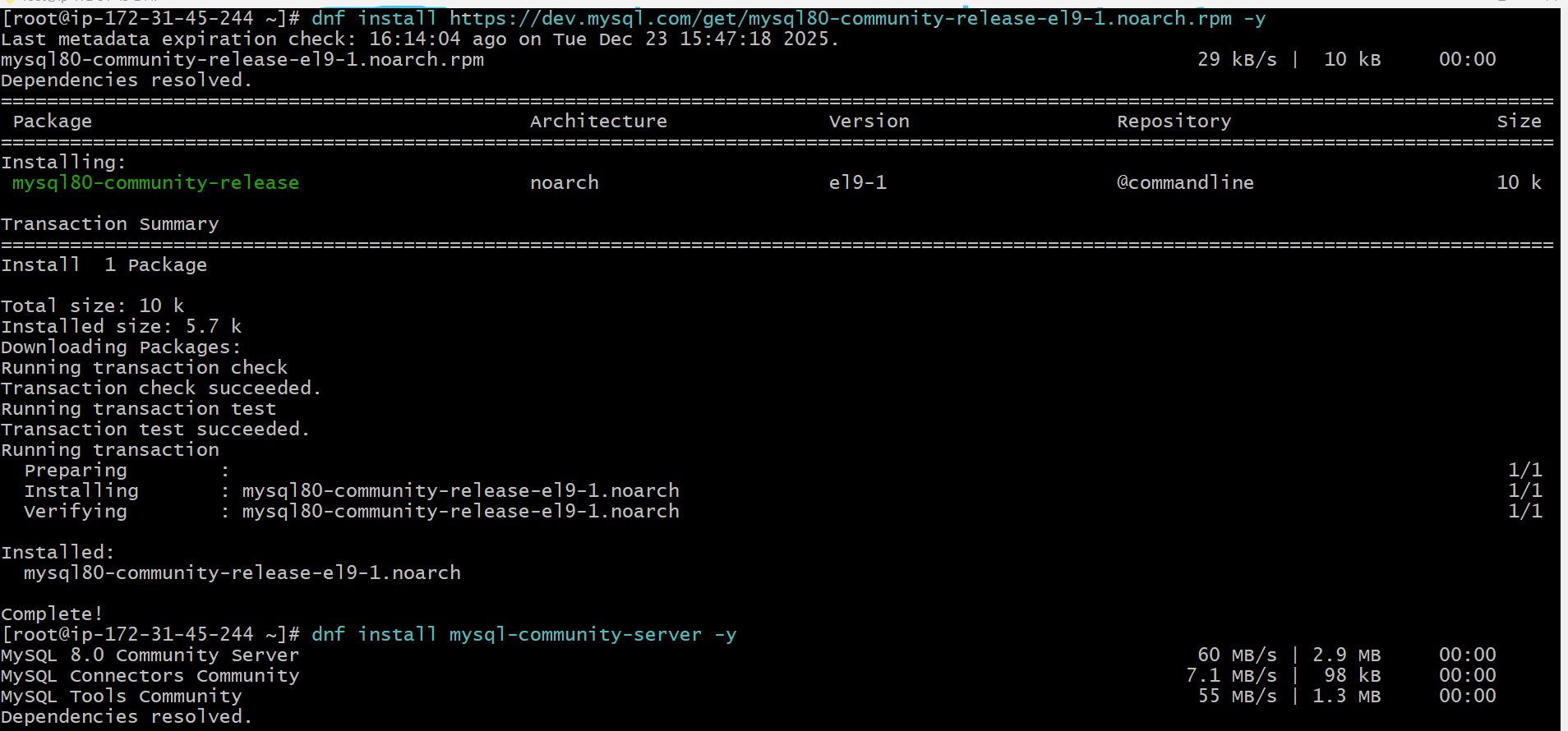
# 

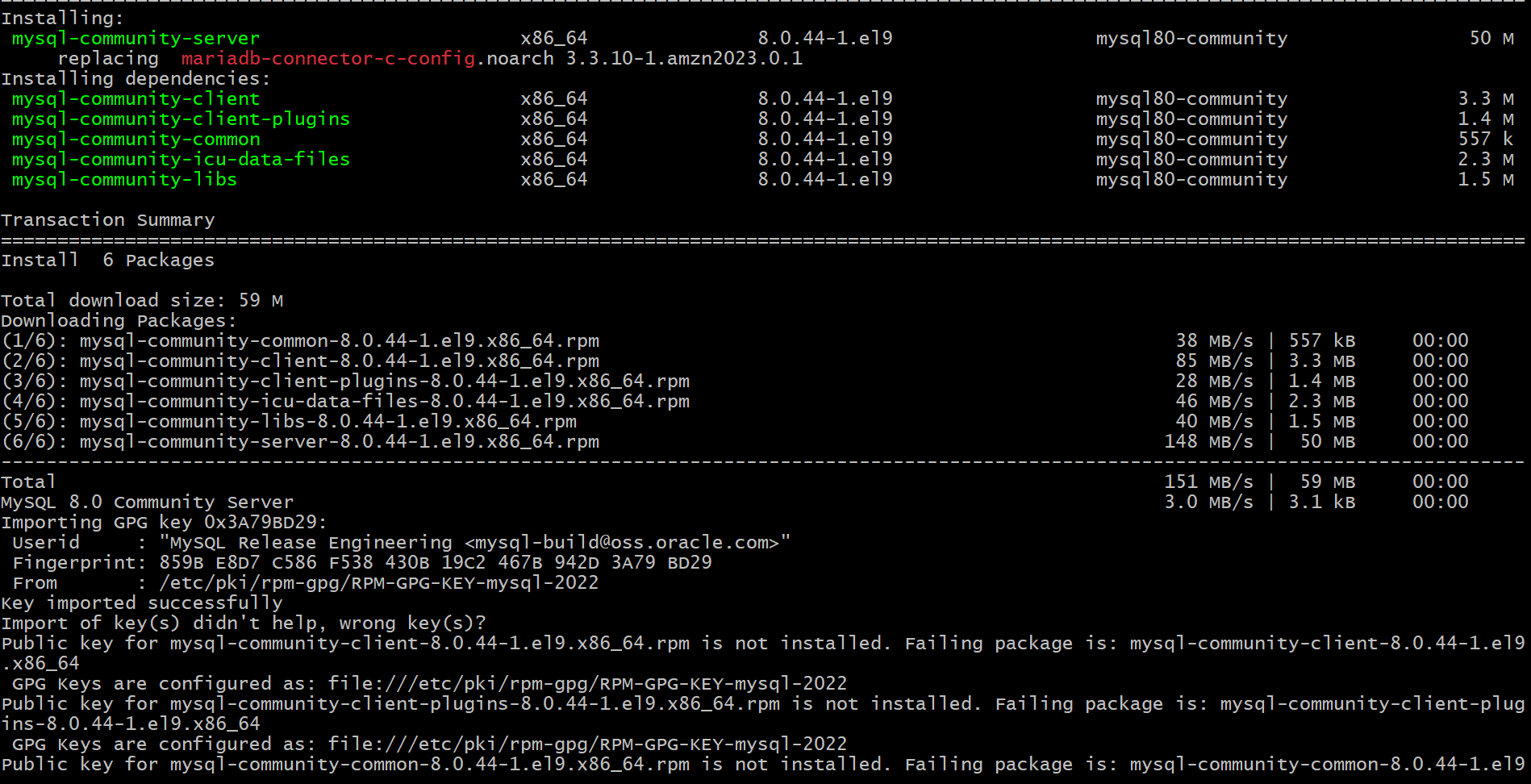
# Conclusion

* The migration from EC2 to RDS was completed successfully, as evidenced by the consistent query results across both environments. Transitioning to RDS provides the database with enhanced availability, automated backups, and simplified scaling compared to the standalone EC2 installation.

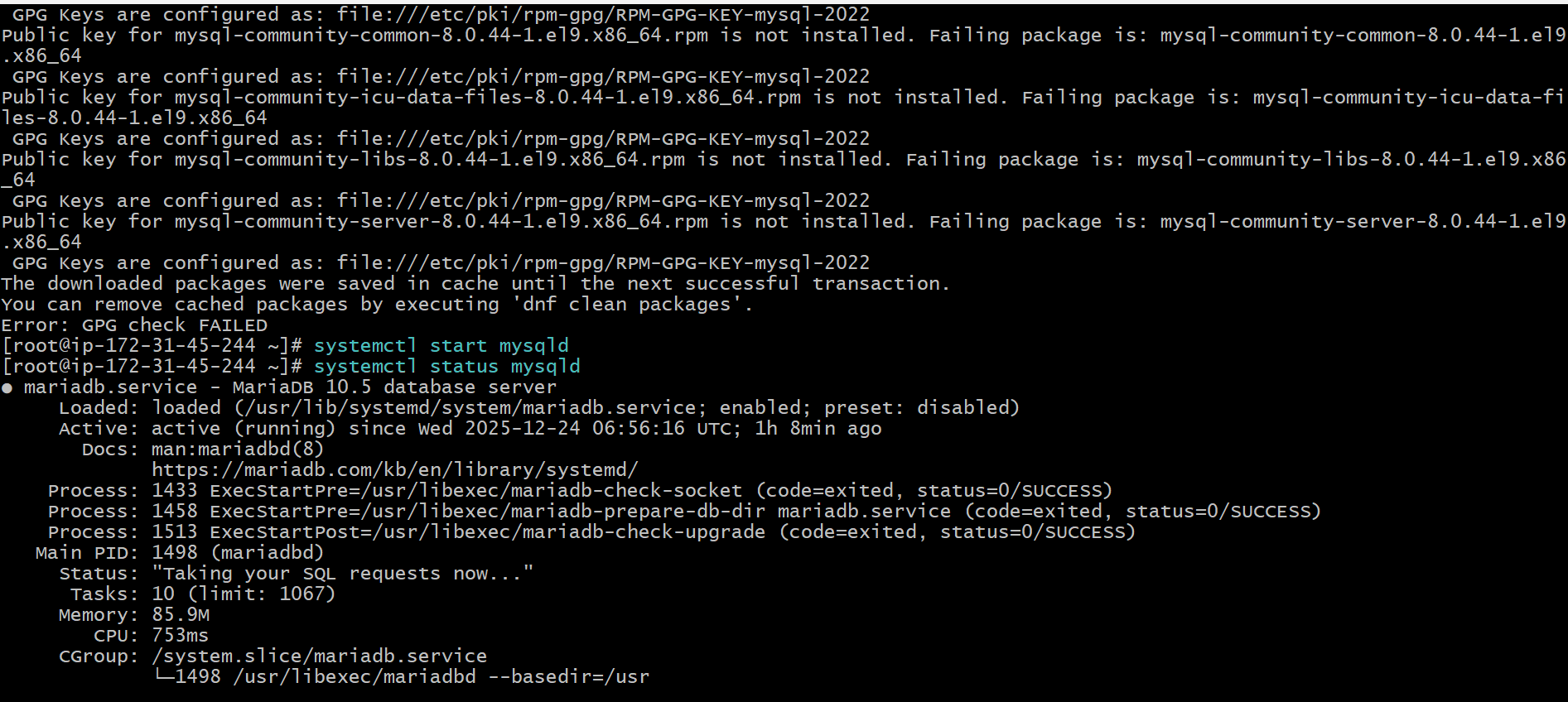
1. ***Install MySQL DB on EC2.***

* To transition an EC2 instance from MariaDB-based libraries to a full MySQL 8.0 Community Server environment, including the repository setup, service initialization, and verification.
* Run the following command: dnf install https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm -y
* Run the command: dnf install mysql-community-server -y

******

******

* Start the service: systemctl start mysqld
* **Check Service Status**: Run systemctl status mysqld to confirm the process is **active (running)**.
* **Port Check**: (Optional) Use lsof -i tcp:3306 to ensure the MySQL daemon is listening on the default port.

******

# Conclusion

* The EC2 instance has been successfully migrated from a default MariaDB-centric configuration to a formal MySQL 8.0 Community Server environment. The service is currently operational and ready for database and user creation.

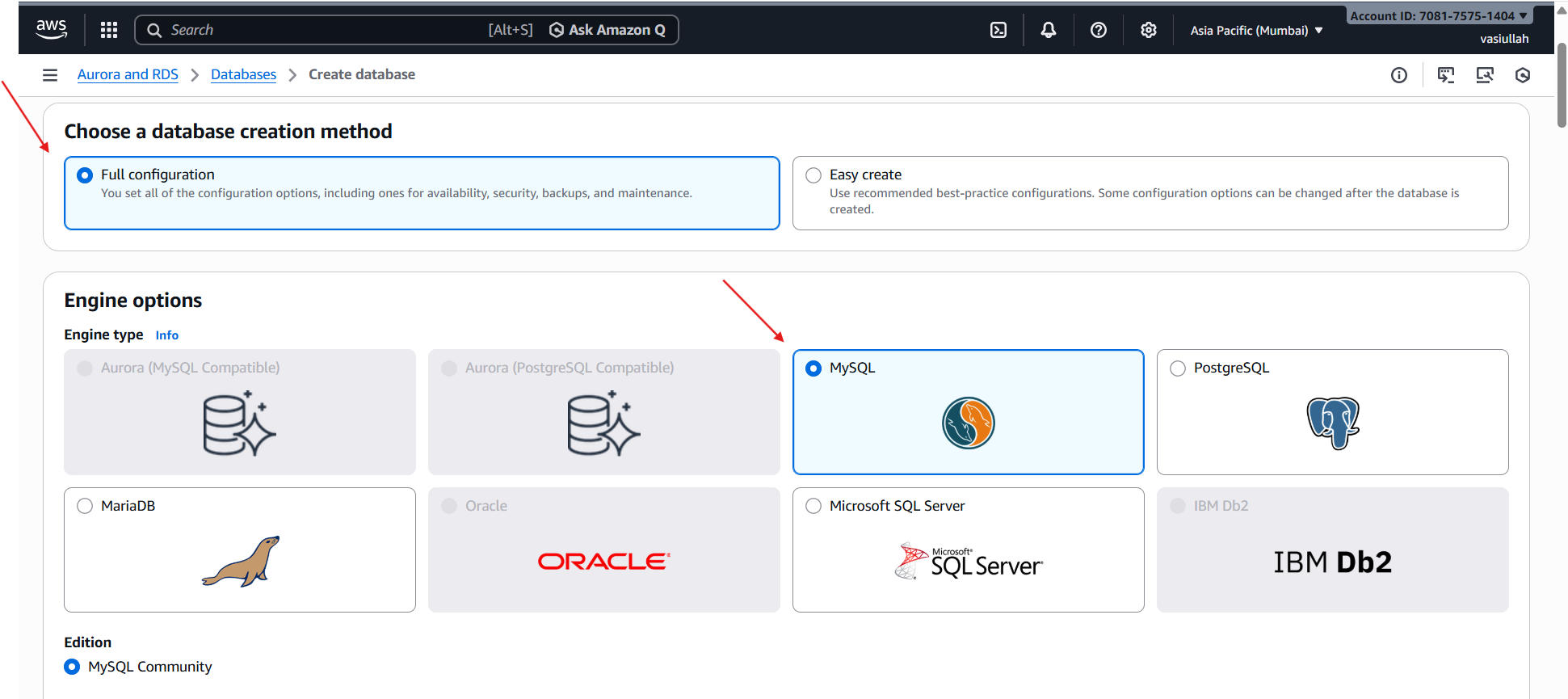
1. ***Launch MySQL RDS image.***

## Task Title

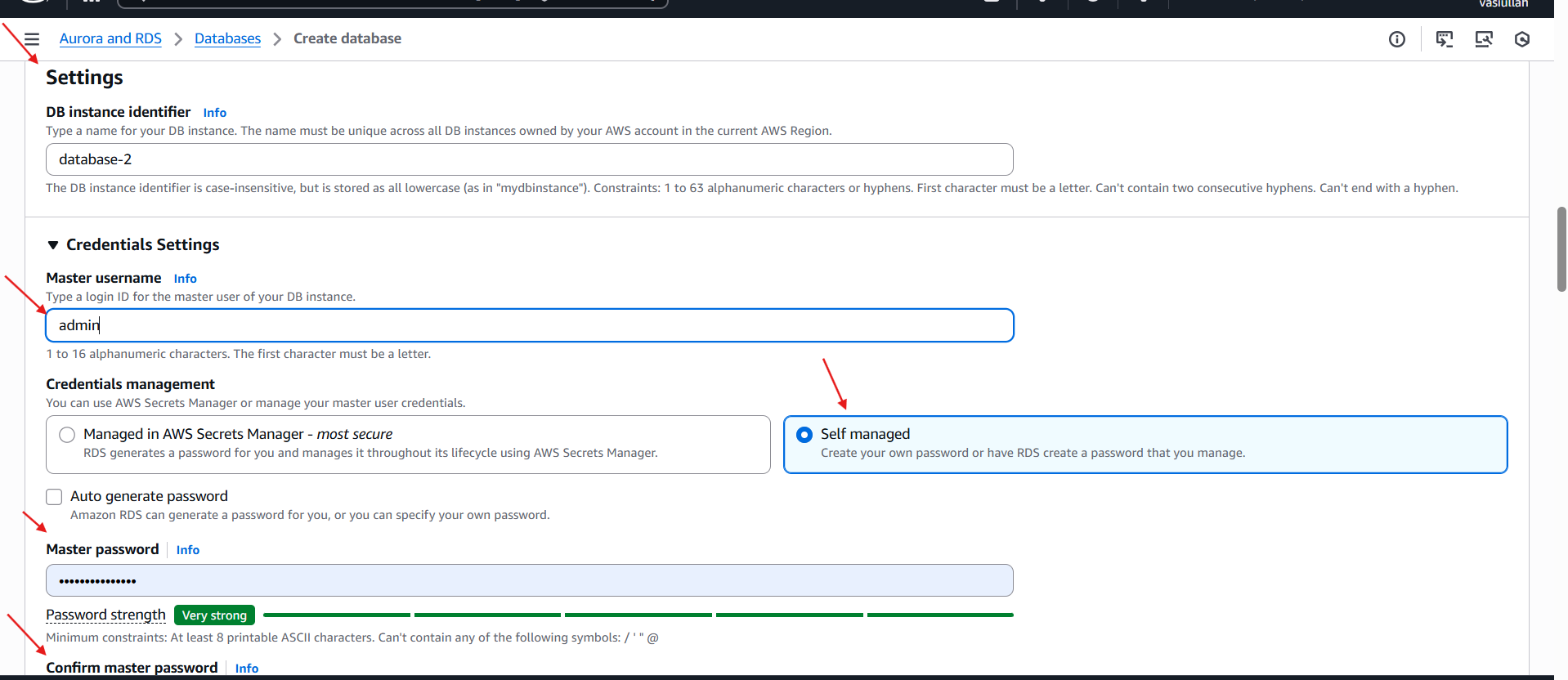
* Launching a Private MySQL RDS Instance with Automated EC2 Connectivity

# Objective

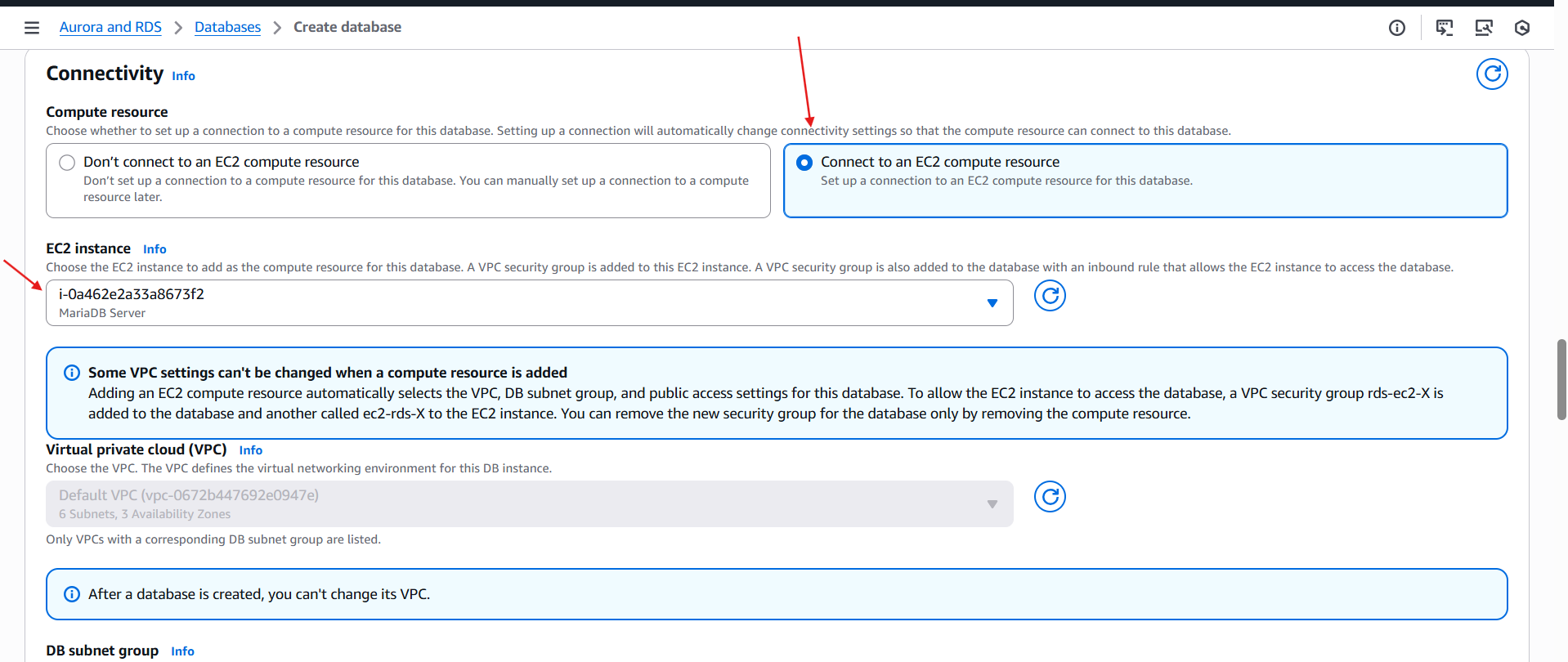
* To provision a managed MySQL Community Edition database using Amazon RDS, implement self-managed credential security, and establish private network connectivity to a specific EC2 compute resource.
* **AWS Account**: Access to the RDS console and EC2 management.
* **Method**: Chose **Standard Create** (Full configuration) to allow for manual customization of all parameters.
* **Engine**: Selected the **MySQL** engine.
* **Edition**: Opted for **MySQL Community**.

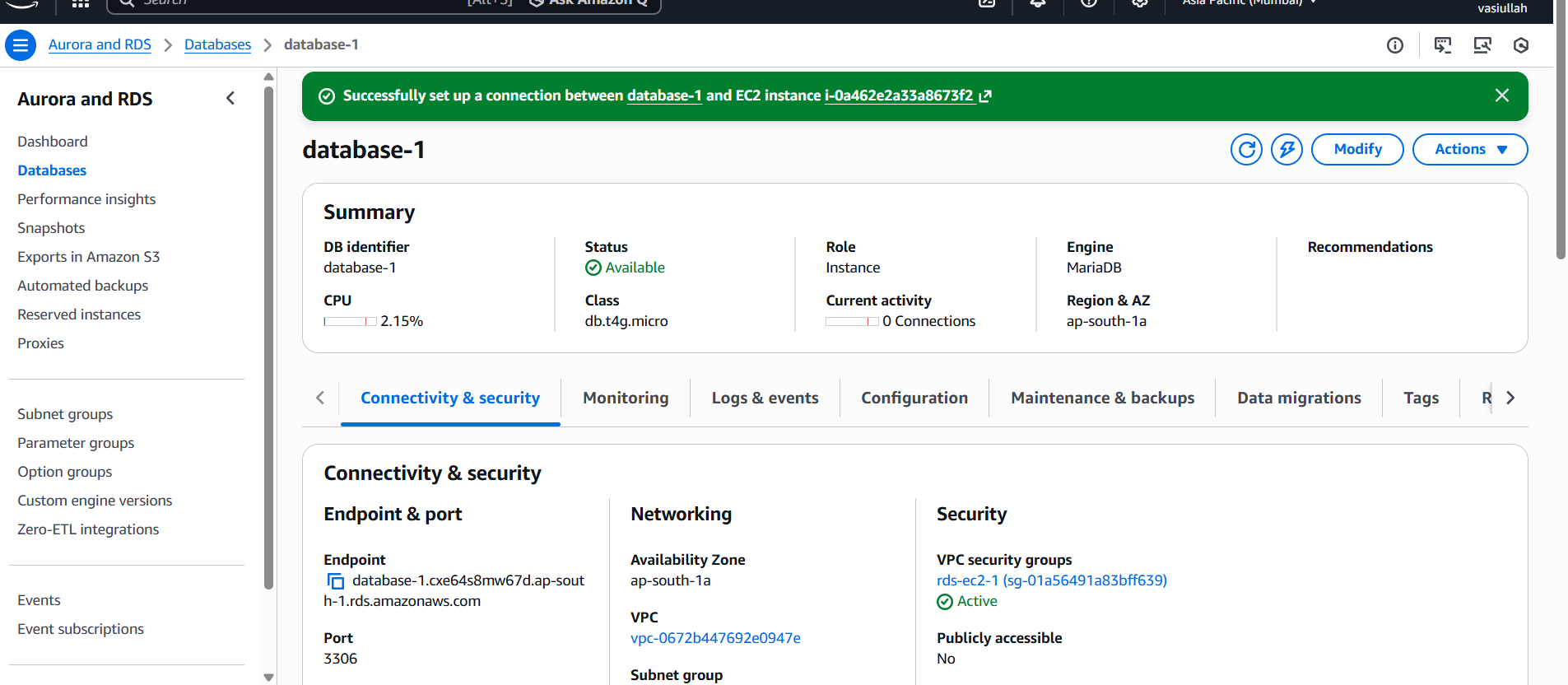
******

* Instance ID: Set the DB instance identifier to database-1.
* Username: Used the default master username admin.
* Security Model: Selected Self-managed credentials rather than AWS Secrets Manager.

******

* Privacy Check: Ensure Public accessibility is set to No, confirming the database is only reachable from the EC2 instance via the private network.
* Port Verification: Verify the instance is actively listening on Port 3306.

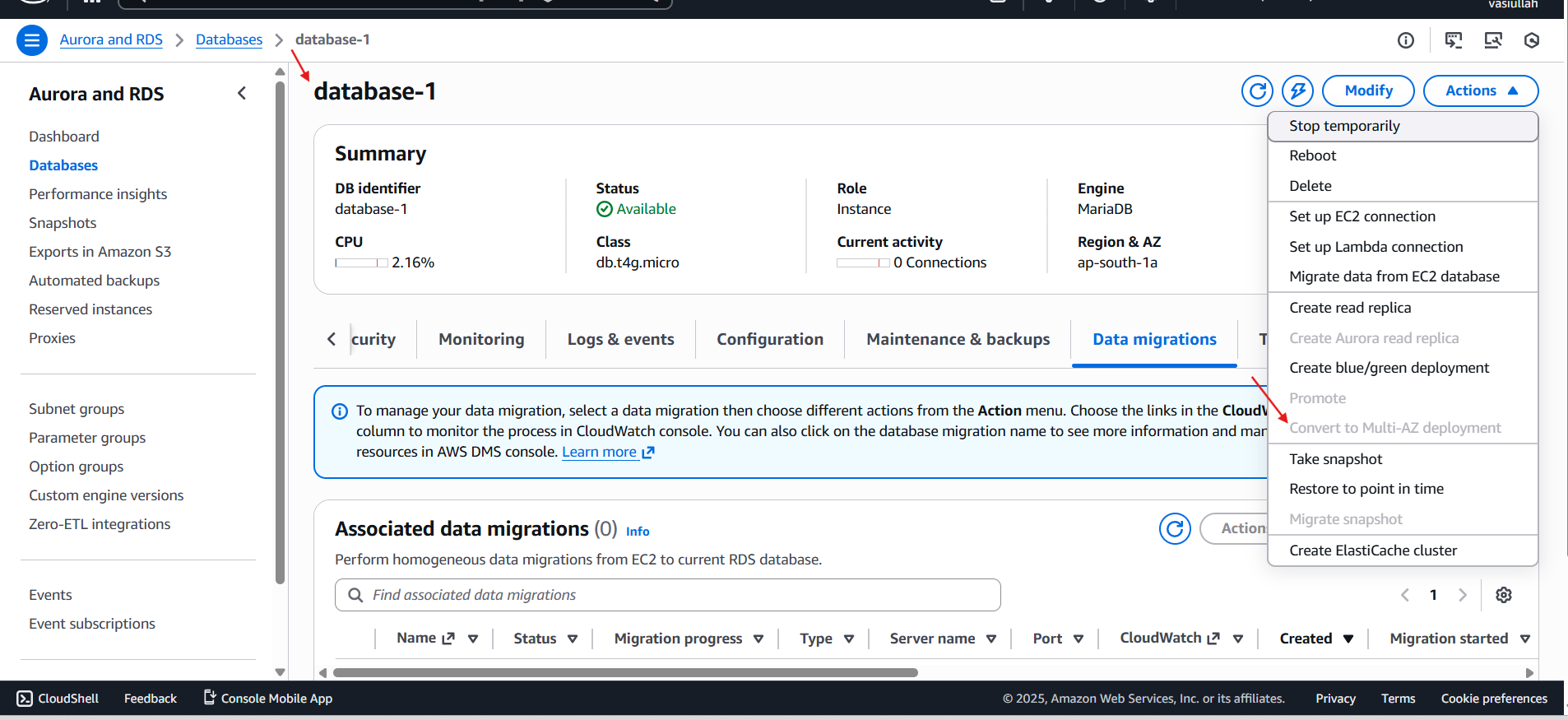
******

******

# Conclusion

* The MySQL RDS instance is successfully running and integrated with the application server. By using the automated connectivity feature, the deployment avoided common manual networking errors, resulting in a secure, private database environment accessible only by the designated EC2 resource.

1. ***Configure Multi-AZ.***

******

1. ***Take backup of DB and restore the DB.***

## Task Title

* Creating a Manual RDS Database Snapshot for MariaDB

# Objective

* To manually trigger and configure a point-in-time backup (snapshot) of a MariaDB RDS instance via the AWS Management Console to ensure data persistence and recovery readiness

# Prerequisites

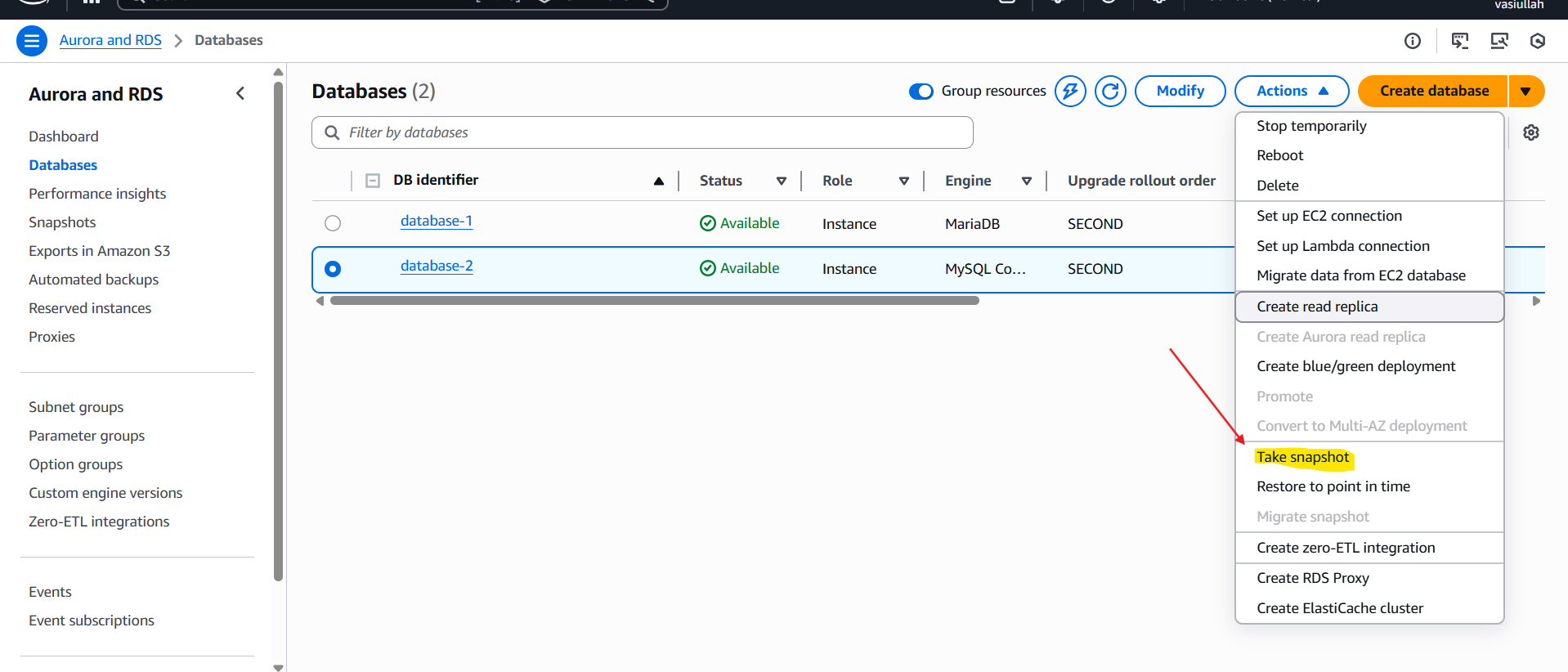
* **Active RDS Instance**: A running MariaDB instance, such as database-2.
* **Console Access**: Permissions to view RDS instances and manage snapshots.

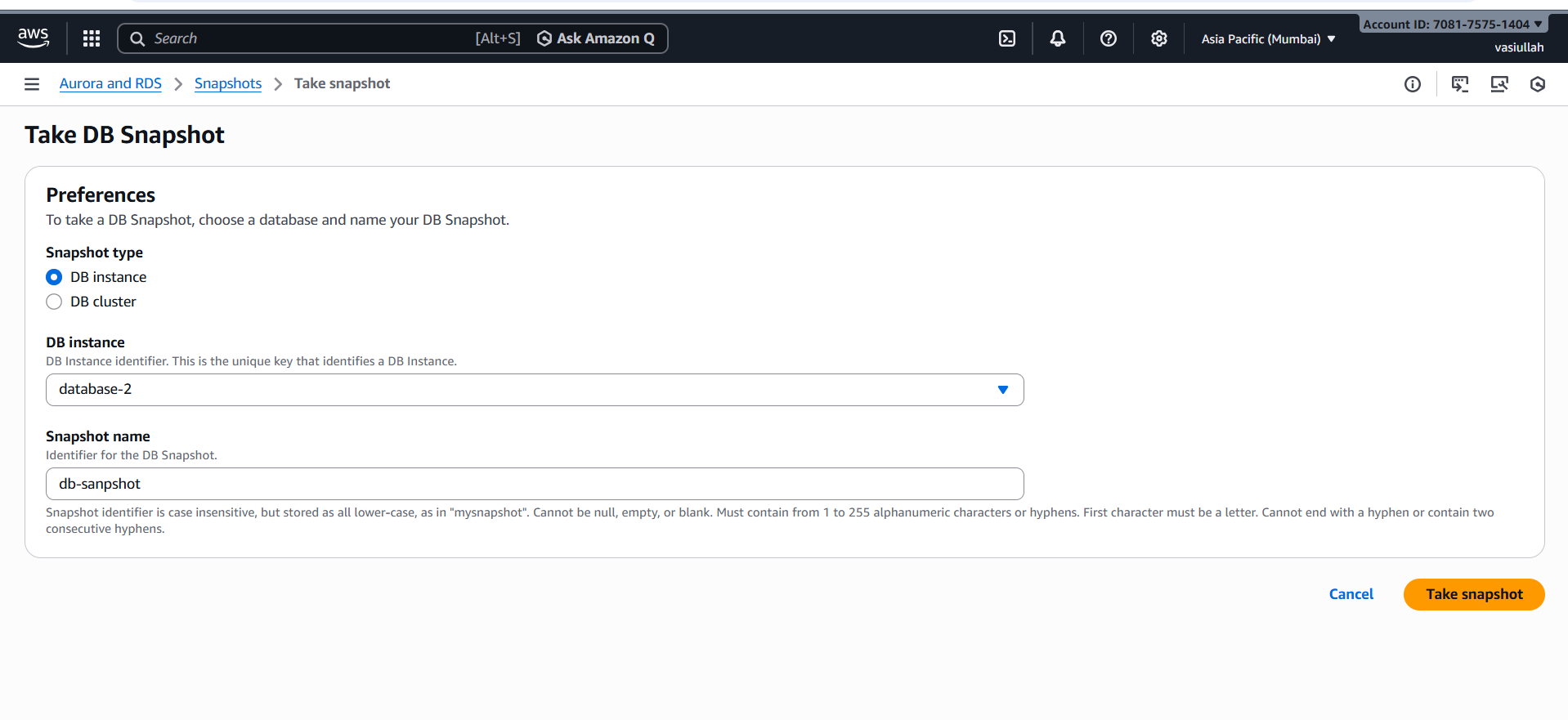
# Step-by-Step Implementation

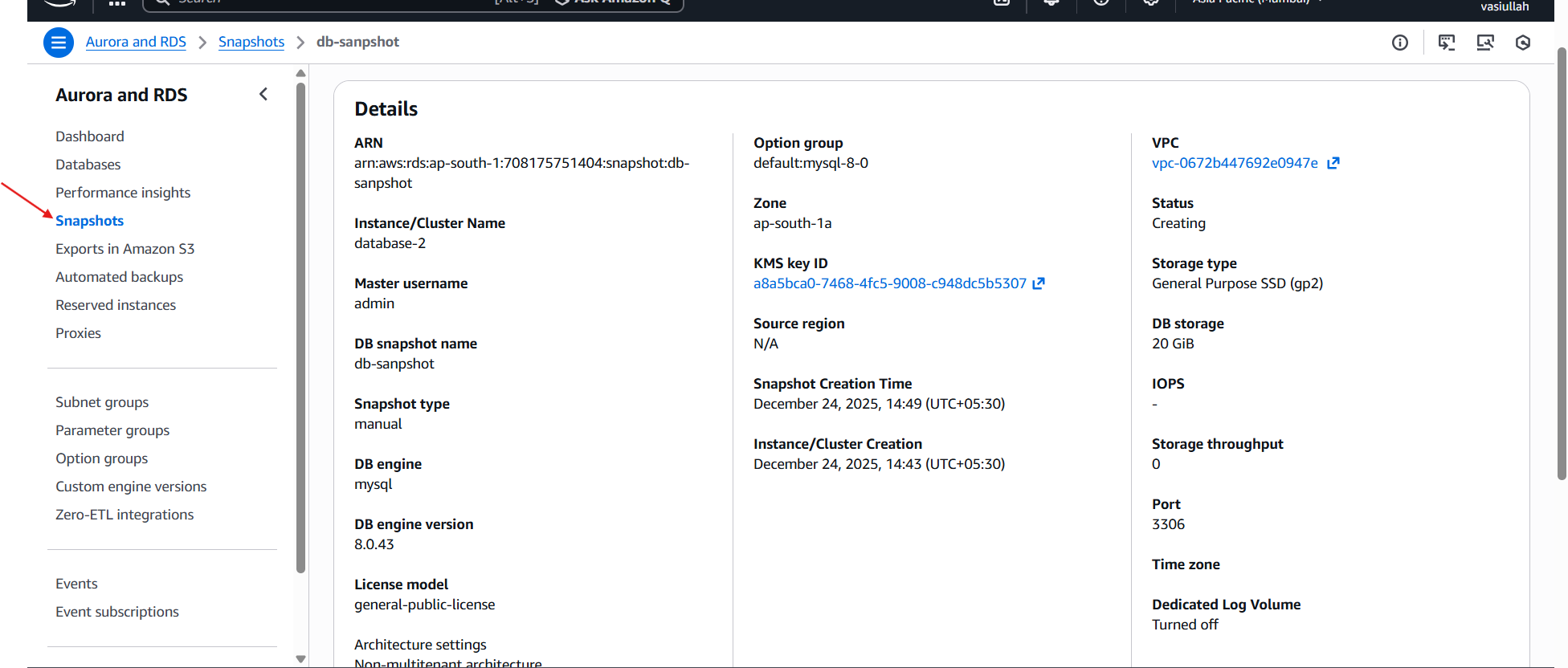
*  Navigate to the **AWS RDS Console** and select the target instance: database-2.
*  Click the **Actions** dropdown menu and select **Take snapshot**.
*  Note: A system banner may appear confirming existing EC2-to-DB connectivity, which is independent of the snapshot process.

### Validation Steps

* **Status Verification**: Check the **Status** column in the Snapshots dashboard; it should initially show **Creating**.
* **Type Confirmation**: Ensure the snapshot type is listed as **manual**, distinguishing it from automated system backups.
* **Metadata Check**: Confirm the **Instance source** matches the intended database (database-2).

******

******

******

# Conclusion

* The manual snapshot request for the MariaDB instance was successfully initiated. This provides a reliable recovery point that will remain in the account until manually deleted, offering a higher level of data security than automated snapshots alone.
* confirms the snapshot creation request was successfully initiated

***10.Create read replica.***

# Task Title

* Configuring an RDS Read Replica for MySQL Community Edition

# Objective

* To offload read traffic from a primary MySQL RDS instance by creating a synchronized read-only replica, ensuring high availability and improved performance for read-heavy applications.

# Prerequisites

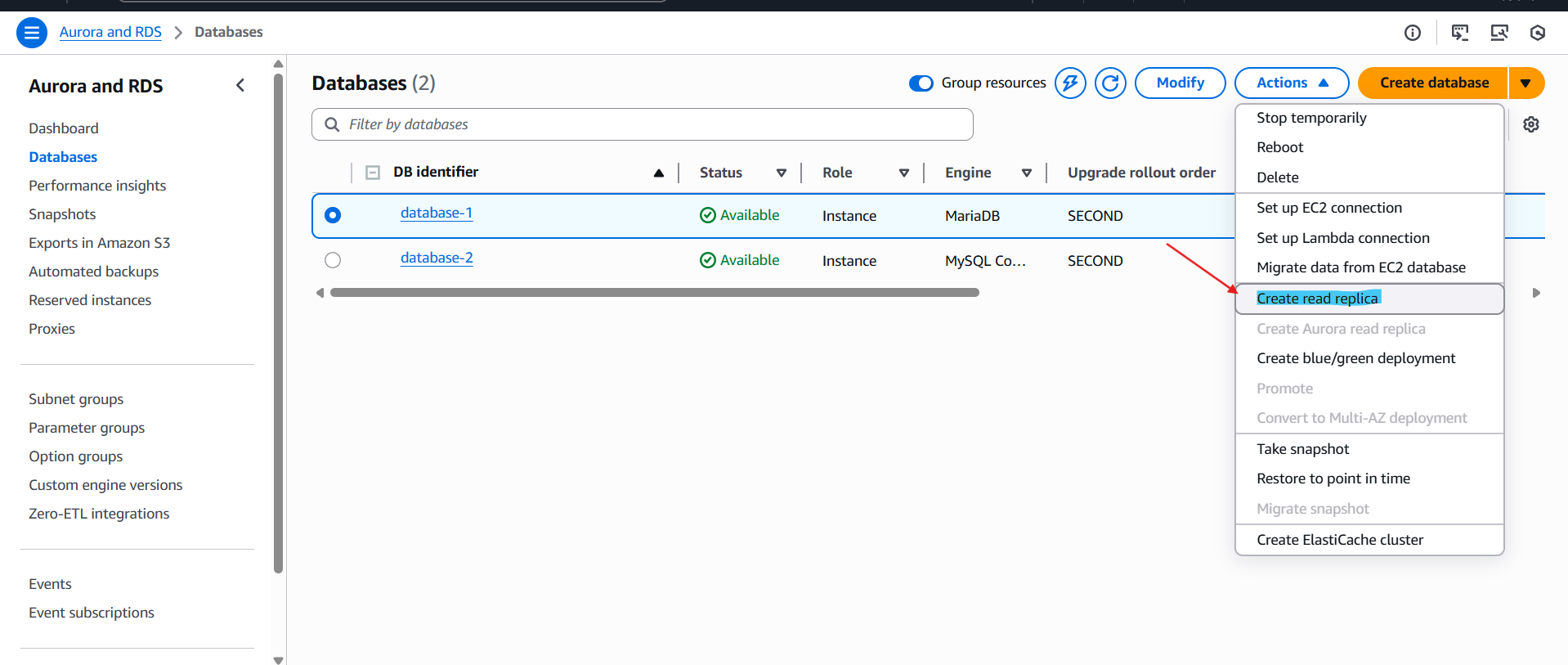
* Primary RDS Instance: An active, running MySQL instance (e.g., database-1).
* Storage and Backups: Automated backups must be enabled on the primary instance (a requirement for replication).
* AWS Service Quotas: Sufficient instance limits within the AWS account to provision additional resources.

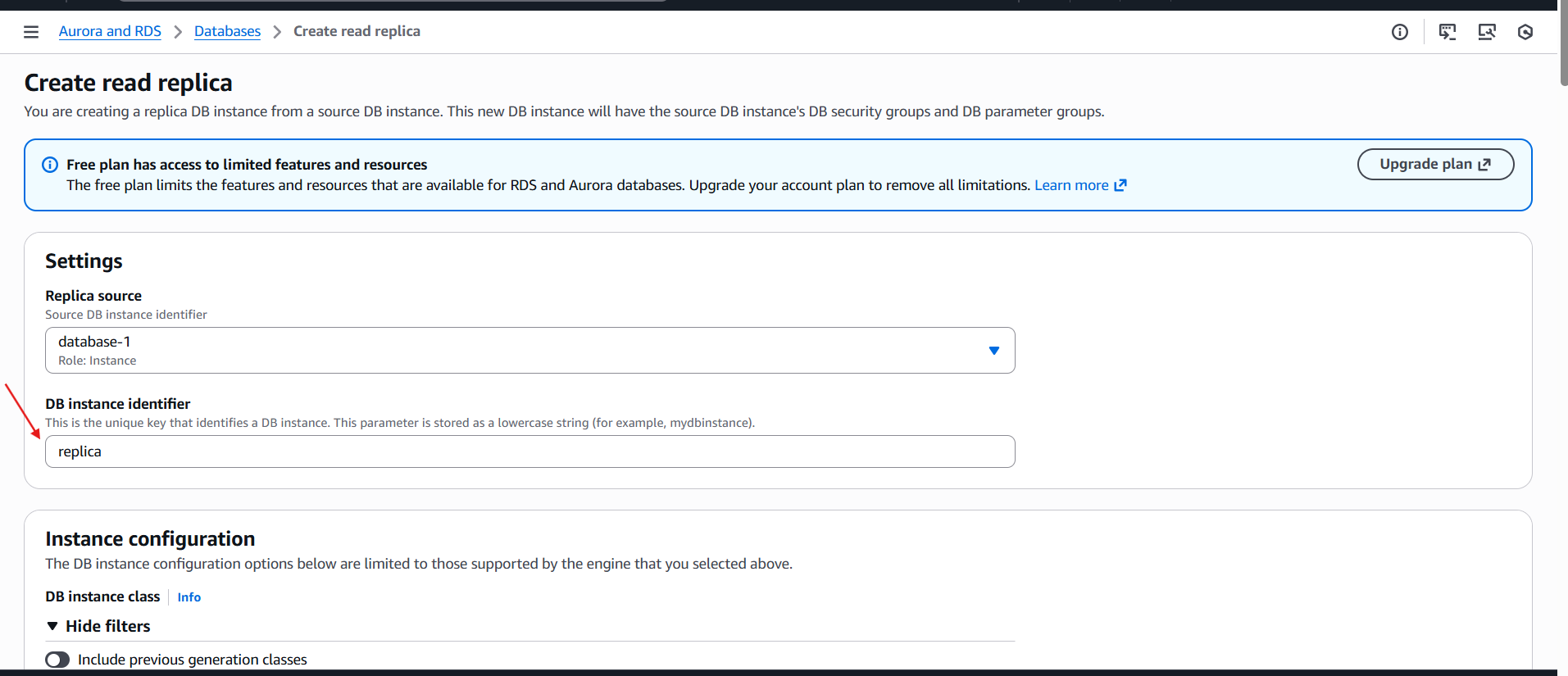
# Initiation via RDS Console

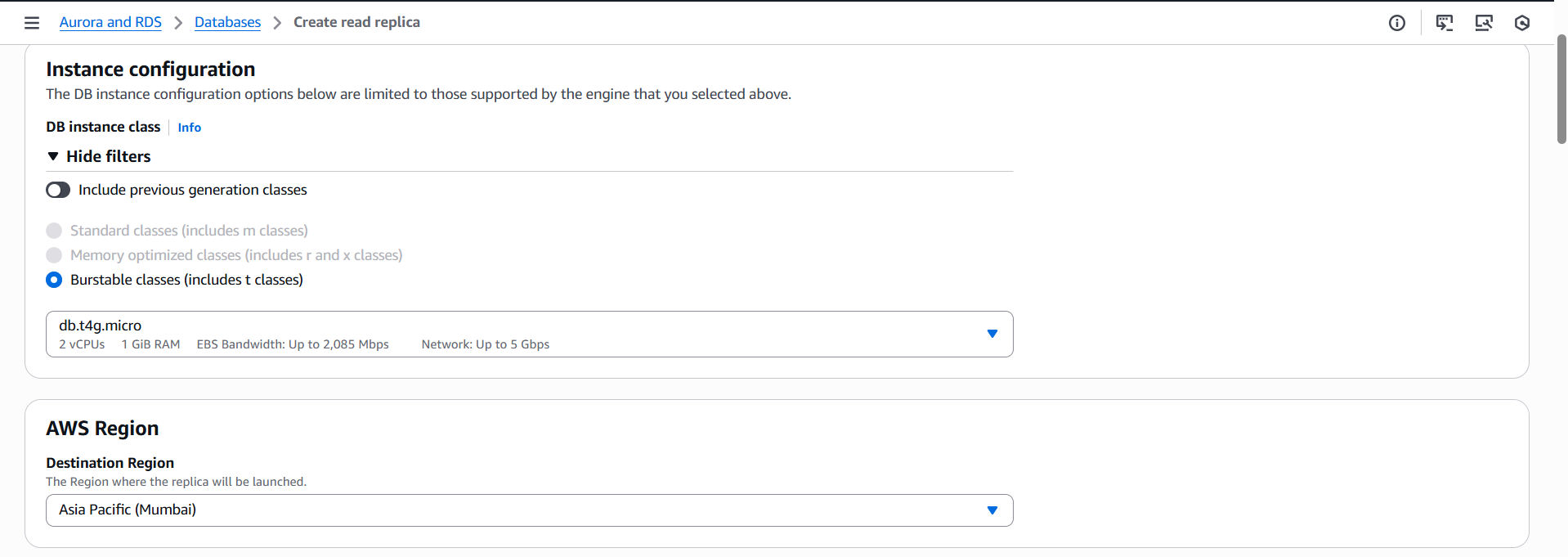
* Navigate to the RDS Databases list and select the primary instance: database-1.
* Open the Actions dropdown menu and select Create read replica.

# Replica Identification and Region

* Replica Source: Verified as database-1.
* DB Instance Identifier: Assigned the name replica1.

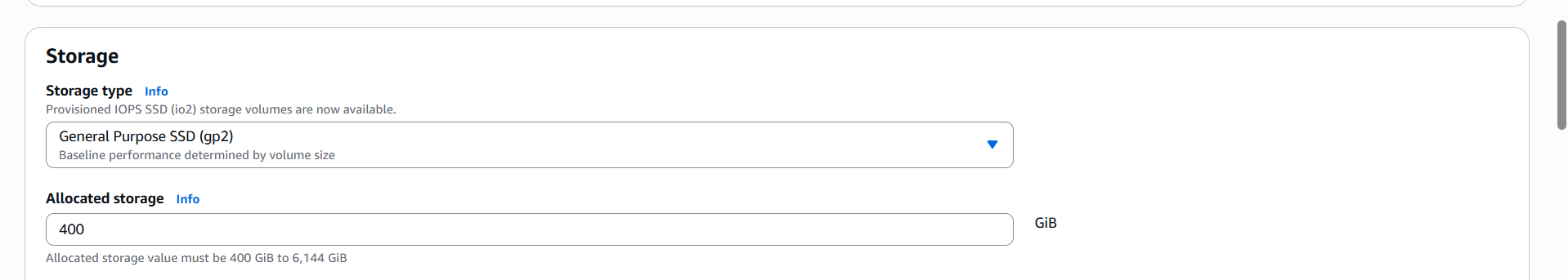
******

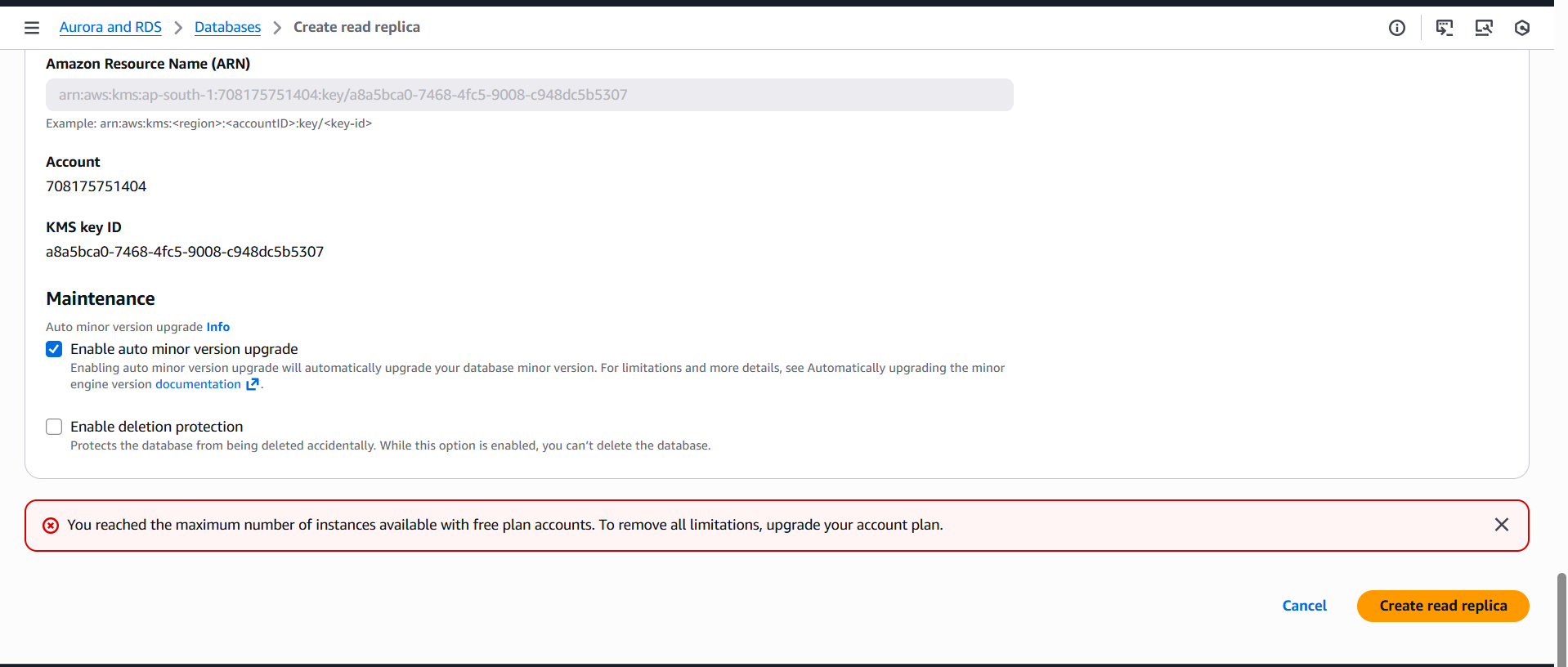
******

******

# Validation Steps

* Configuration Review: Review the summary page to ensure instance class and storage match the primary database requirements.
* Final Submission: Click the Create read replica button to trigger the AWS provisioning process.

******

******

# Conclusion

* While the configuration steps were followed correctly, the Read Replica could not be created due to AWS Free Tier limitations. This highlights the importance of monitoring service quotas when managing multiple RDS instances (database-1 and database-2) simultaneously, as each replica adds to the total instance count.
* AWS is blocking creation because free plan limits have been exceeded.