# RDS Single-AZ with Read Replica + Basic Monitoring


> **Name:** Shloka Kamdar
> **Date: 04/12/2025**
> **Trainer:** Vishal Shinde


## Objective

The objective of this assignment is to deploy a secure MySQL RDS database in Single-AZ mode, create a same-region read replica to demonstrate read scaling, enable backup/monitoring features, configure CloudWatch alarms linked with SNS for alerts, and validate connectivity using a private EC2 instance.

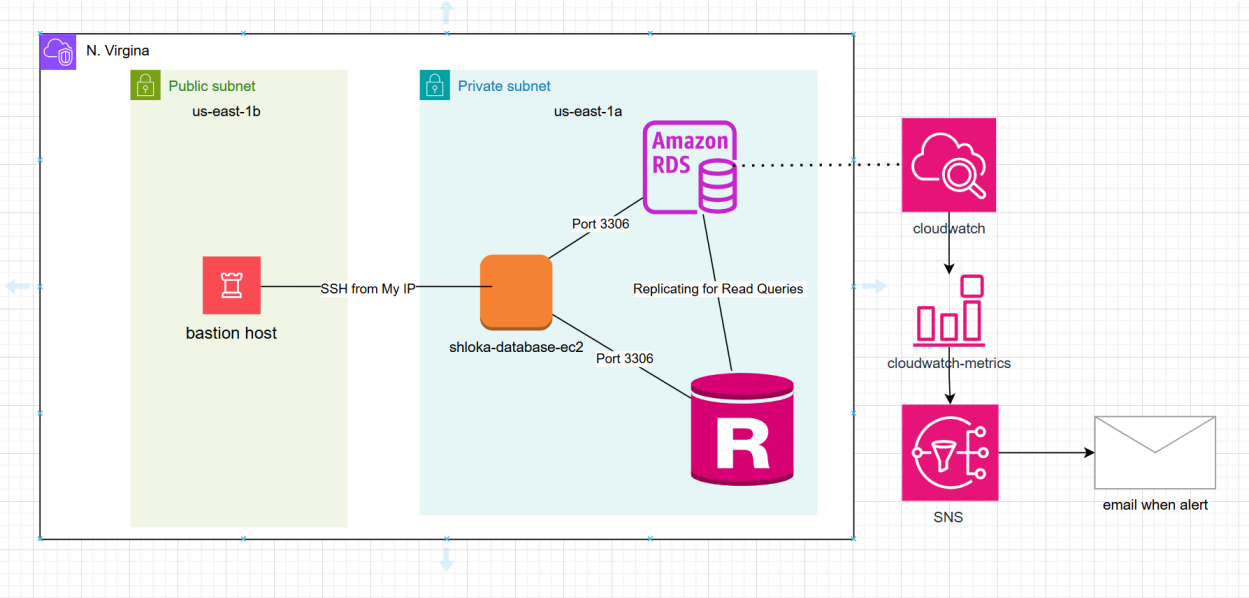
## Architecture Overview

I deployed a database architecture entirely inside a private VPC with no public access. The flow:

- **Primary RDS MySQL 8.0** (Single-AZ) in private subnets
- **Read Replica** inside same VPC/subnet group
- **Private EC2 instance** inside RDS private subnet to test connectivity
- **Bastion Host** for SSH access into private EC2
- **Custom Parameter Group** for `slow_query_log` and `long_query_time`
- **CloudWatch Alarm** monitoring RDS CPU > 70% for 5 minutes
- **SNS Topic** sending email alerts
- **EC2 → RDS Security Group routing enabled** for MySQL port 3306

This architecture follows production-style private DB design principles.

## Architecture Diagram




# STEP-BY-STEP IMPLEMENTATION

## Step 0 - PREREQUISITES

Before deploying the RDS Primary + Read Replica architecture, the following foundational infrastructure was created:

### VPC Details

- **VPC Name:** `shloka-database-vpc`
- **CIDR Block:** `10.0.0.0/16`
- **Subnets:**
  - Public Subnet: `10.0.1.0/24` (us-east-1a)
  - Private Subnet: `10.0.2.0/24` (us-east-1a)
- **Internet Gateway:** `shloka-database-igw`
- **NAT Gateway:** `shloka-database-nat` (regional NAT)

- **Route Tables:**
  - Public Route Table → IGW Route
  - Private Route Table → NAT Route


VPC resource map (initial)

## EC2 Details

- **Public EC2 (Bastion Host):**
  - AMI: Amazon Linux 2
  - Subnet: Public subnet
  - SG: Allows SSH (22) from my IP only
- **Private EC2 (DB Client):**
  - AMI: Amazon Linux 2
  - Subnet: RDS Private Subnet
  - SG: Allows SSH from Bastion SG
  - Installed tools: MariaDB client for DB connectivity

These components form the base network that allows RDS to operate securely inside private subnets, while EC2 instances provide controlled connectivity for administration and testing.

# Step 1: Create RDS MySQL Primary Instance (Single-AZ)

**What was done:**

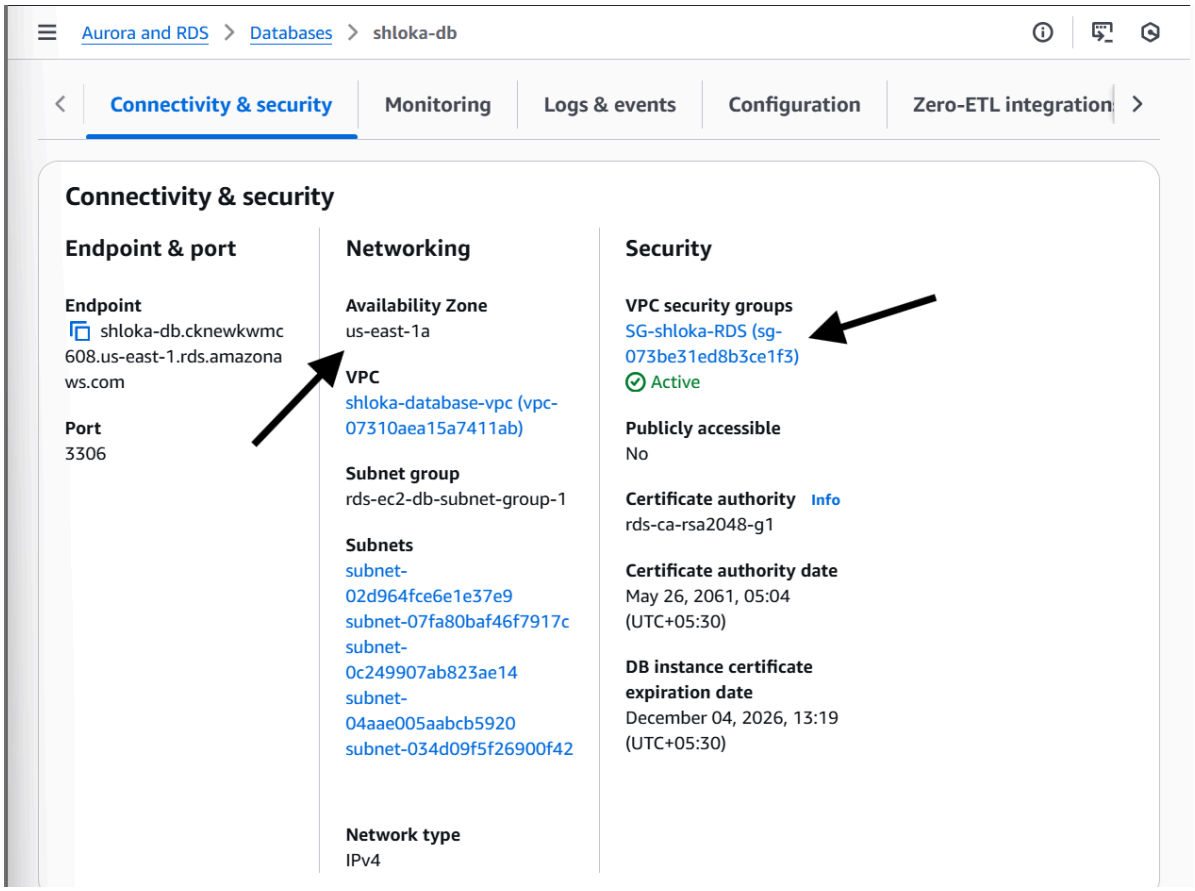
Deployed a secure MySQL 8.0 Single-AZ RDS instance with storage, backups, monitoring.

**Why was it done:**

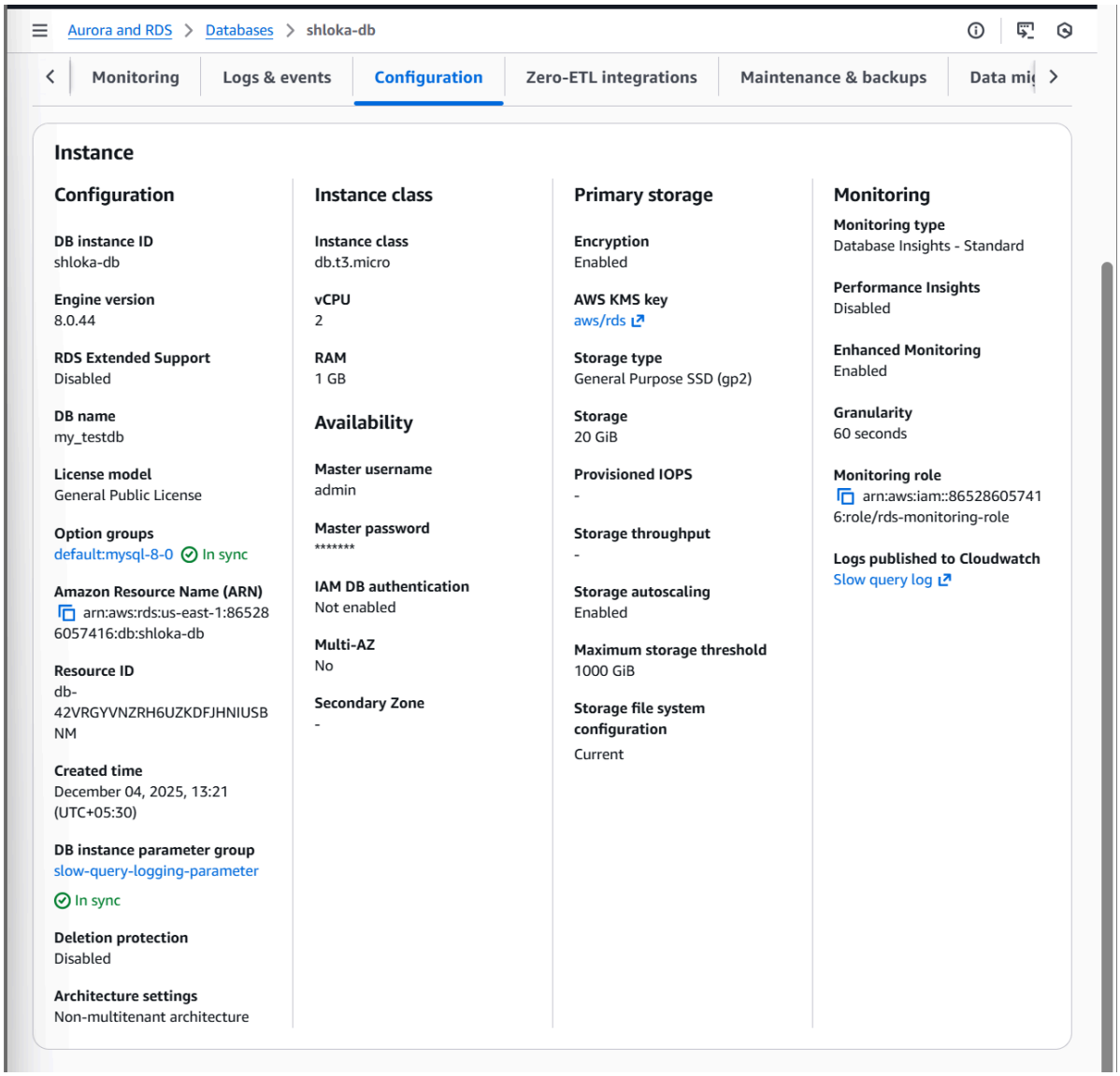
To host a production-style primary database that supports read replication.

**How was it done:**

- RDS → Create Database
- Engine: MySQL 8.0
- Instance class: db.t3.micro
- Storage: 20GB gp3
- No public access
- Select DB Subnet Group
- Backup retention: 1 day (changed from 3 days to 1 day due to free tier limitations)
- Enable Enhanced Monitoring
- DB Name: `my_testdb`




RDS created in us-east-1a (same as private instance)  and custom SG attached  * refer SG further




more RDS configuration details as per the requirements

# Step 2: Create Read Replica

**What was done:**

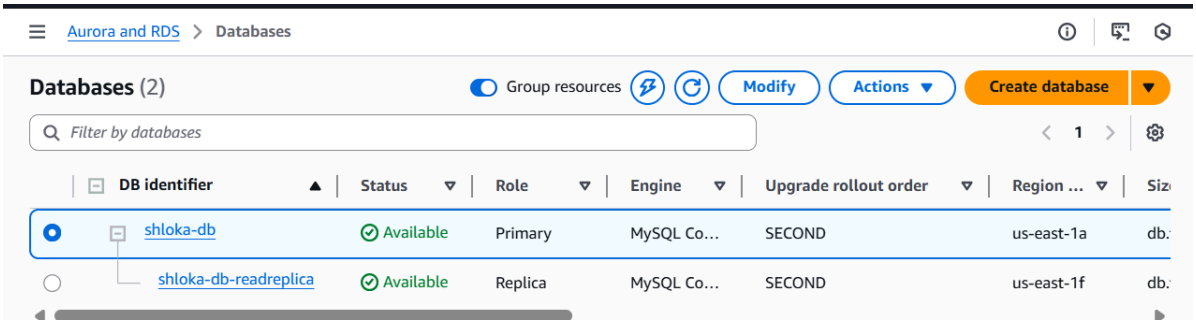
Created an RDS MySQL read replica inside the same VPC and region.

**Why was it done:**

To demonstrate horizontal read scaling and replication.

**How was it done:**

- Select primary DB → Actions → Create Read Replica
- Choose same VPC
- Same instance class
- Keep default replication settings




Read Replica in different AZ - AWS spreads replicas across different AZs for high availability + disaster resilience.

---

# Step 3: Create & Apply Custom Parameter Group

**What was done:**

Created MySQL parameter group with performance monitoring settings.

**Why was it done:**

To enable slow query logging and capture long-running queries.

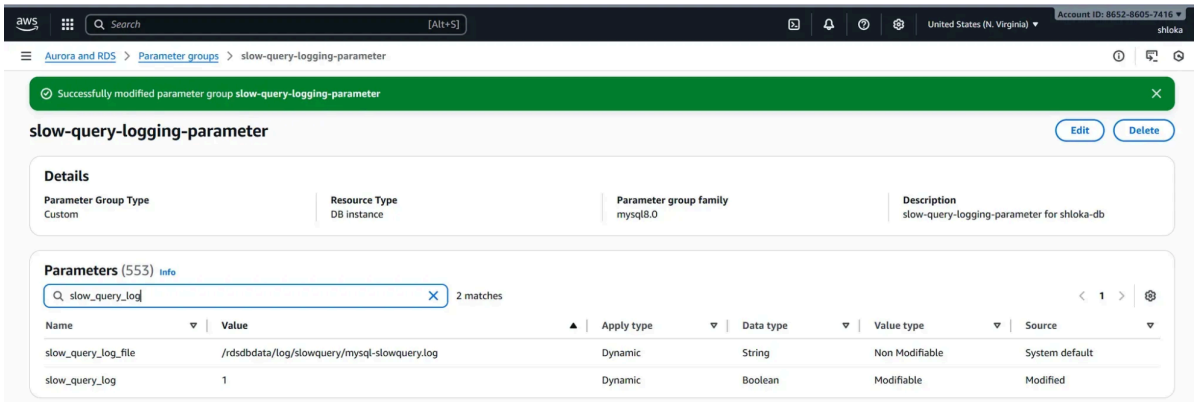
Enabling slow query logging helps identify inefficient SQL queries that exceed the defined execution time threshold. These logs highlight performance issues such as missing indexes, full table scans, poorly structured joins, or suboptimal schema design.

By analyzing slow queries alongside CPU metrics, administrators can determine whether performance degradation is caused by query load or resource limitations. Slow query logs are an essential tool in production environments for diagnosing bottlenecks and guiding database optimization efforts.

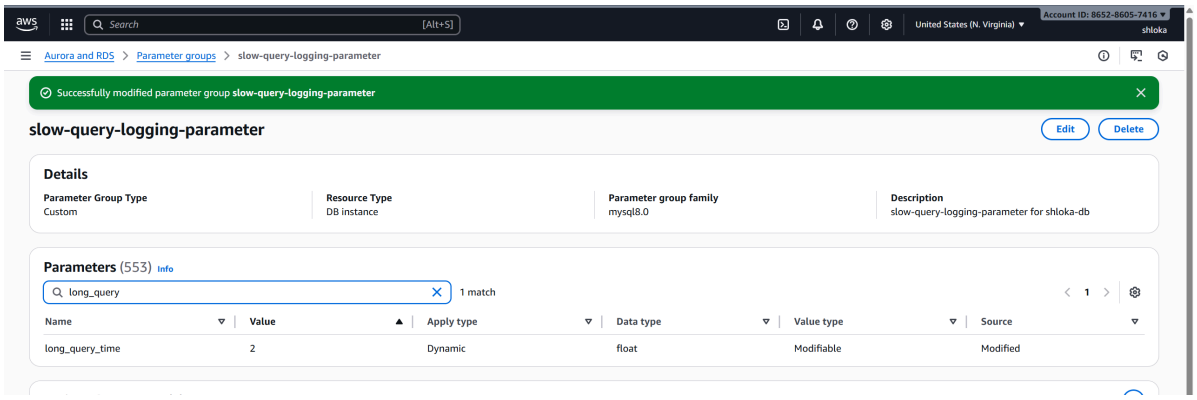
**How was it done:**

- RDS → Parameter Groups → Create

- Engine: MySQL 8.0
- Edit parameters:
  - `slow_query_log = 1`
  - `long_query_time = 2`
- Attach param group to primary RDS
- **Reboot required** → rebooted RDS instance




slow_query_log




long_query_long

---

# Step 4: Configure Correct Subnet Group + Networking Fixes

### What was done:

Adjusted subnet groups + NAT routing so EC2 could reach RDS.

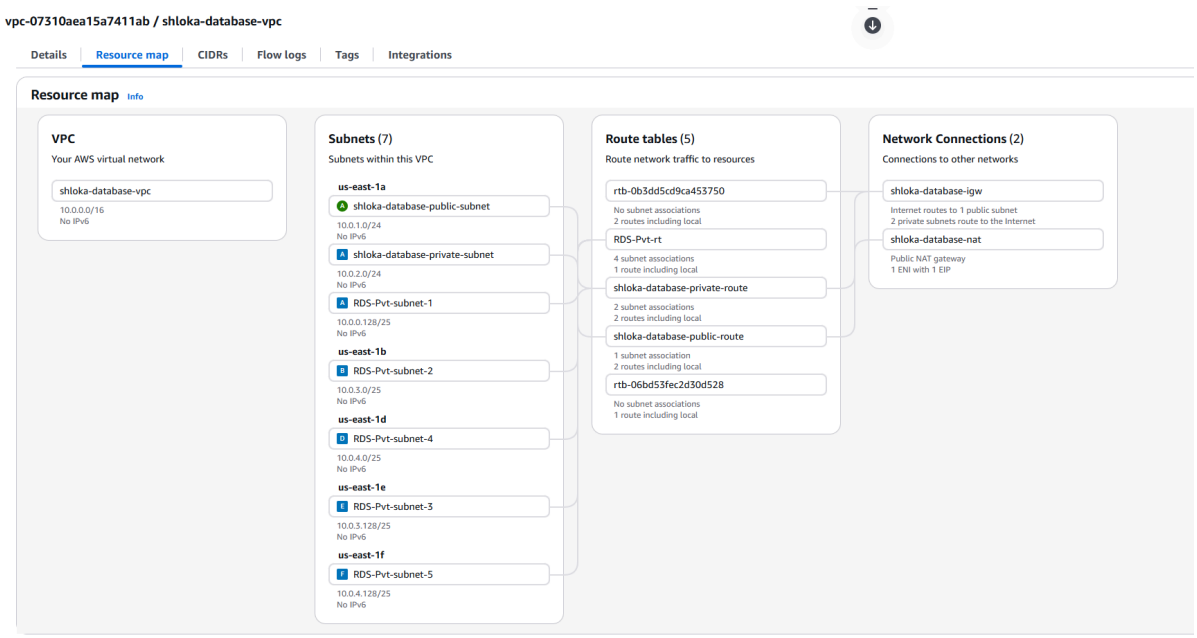
**Why was it done:**

Originally, EC2 couldn't connect to RDS because it wasn't in the same DB Subnet Group + subnet routing wasn’t configured.

**How was it done:**

- Added RDS’s private subnet to the **private route table**
- Added route → NAT Gateway
- Verified EC2 + RDS inside same VPC
- Launched a NEW EC2 inside **RDS private subnet**

(This step fixed all TIMEOUT / connectivity errors.)




updated VPC + subnets

---

# Step 5: Configure Security Groups (Final Correct Structure)

**What was done: :**

Rebuilt SG structure cleanly; removed AWS auto-generated conflicting SGs.

**Why was it done:**

RDS had multiple SGs, one of which blocked inbound 3306, causing silent failures.

**How was it done:**

Created 3 SGs:

| Name | Inbound | Outbound |
|---|---|---|
| SG-BASTION | SSH → My IP | All |
| SG-EC2-PRIVATE | SSH → SG-BASTION | All |
| SG-RDS | 3306 → SG-EC2-PRIVATE | All |

Attached **only SG-RDS** to both Primary & Replica, and removed the default ones

---

## Step 6: Testing Primary DB Connectivity

**What was  done:**

Tested RDS connectivity from EC2.

**Why was it done:**

To validate security group + subnet + routing configuration.

**How was it done:**

Installed MySQL client:

```
sudo yum install mariadb -y
```

Then:

```
mysql -h <primary-endpoint> -u admin -p
```

**Successfully logged in.**

---

## Step 7: Create Table + Test Write Operations

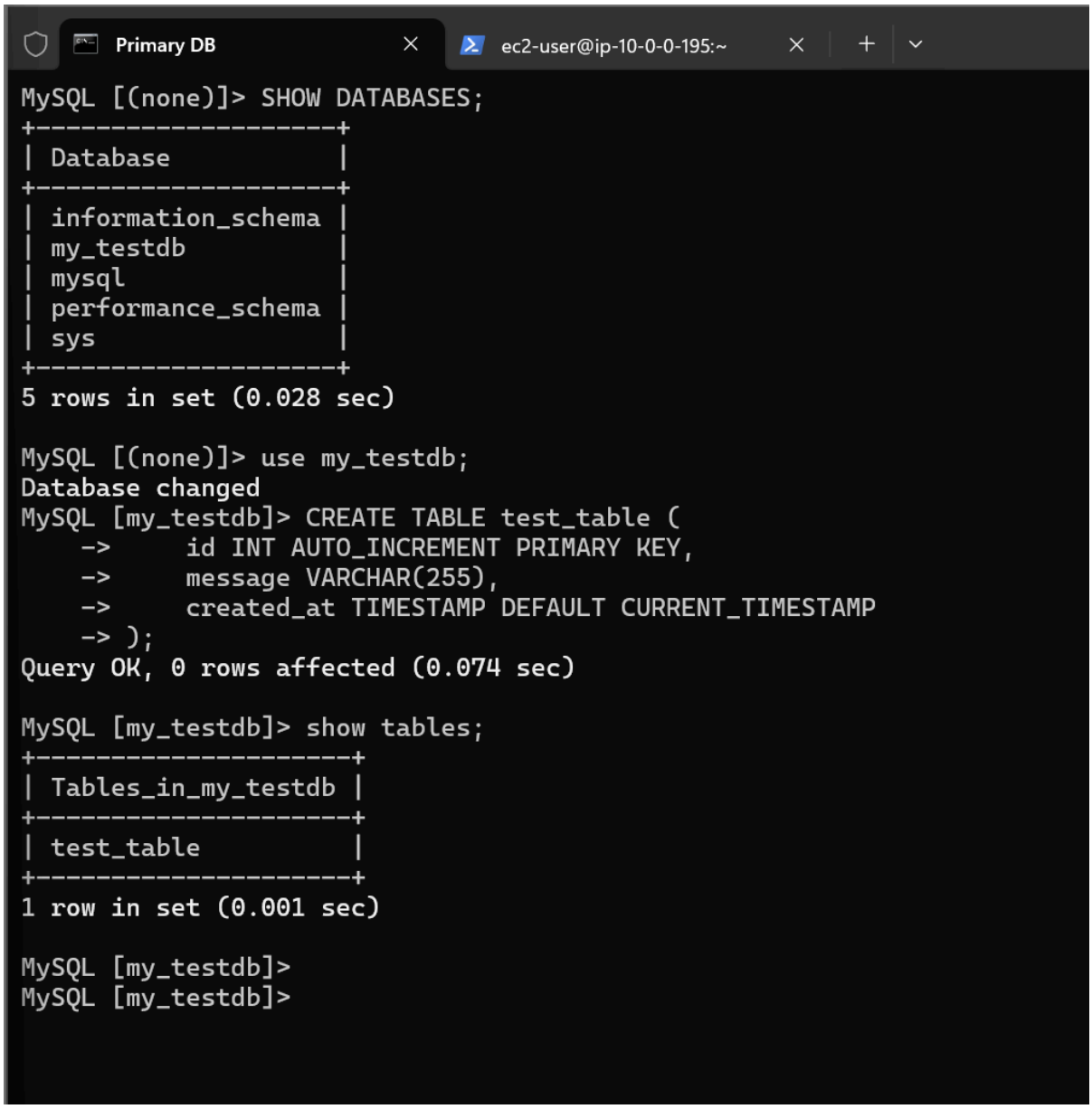
**What was done:**

Created test table and inserted data.

**Why was it done:**

Required to validate replication to read replica.

**How was it done:**




Verifying writing operations in Primary DB

## Step 8: Configure CloudWatch CPU Alarm + SNS

**What was done:**

Created CPU alarm with SNS notification.

**Why was it done:**

To demonstrate monitoring + alerting configuration.

**How It Was Done:**

CloudWatch → Alarms → Create Alarm

- Metric: CPUUtilization
- Threshold: Greater than 70% for 5 minutes
- Action: SNS topic (email subscription confirmed)
- Confirmed subscription from email sent by AWS

---

## Step 9: Trigger CPU Alarm via Load Testing

**What was done:**

Generated artificial CPU load to trigger alarm.

**Why was it done:**

To validate monitoring and SNS delivery.

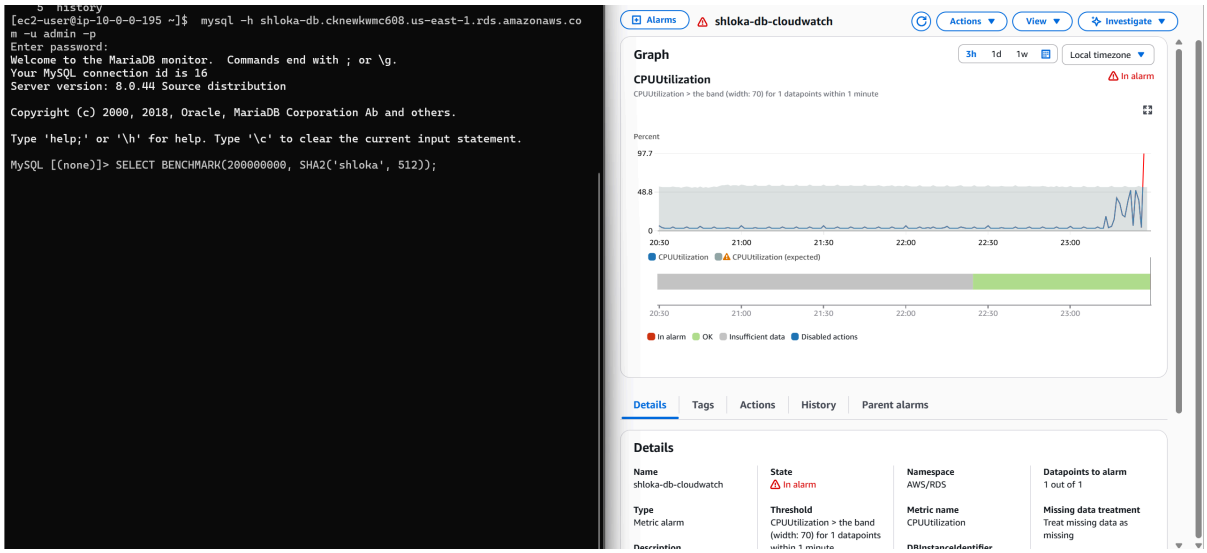
**How It Was Done:**

Ran CPU-heavy BENCHMARK queries:

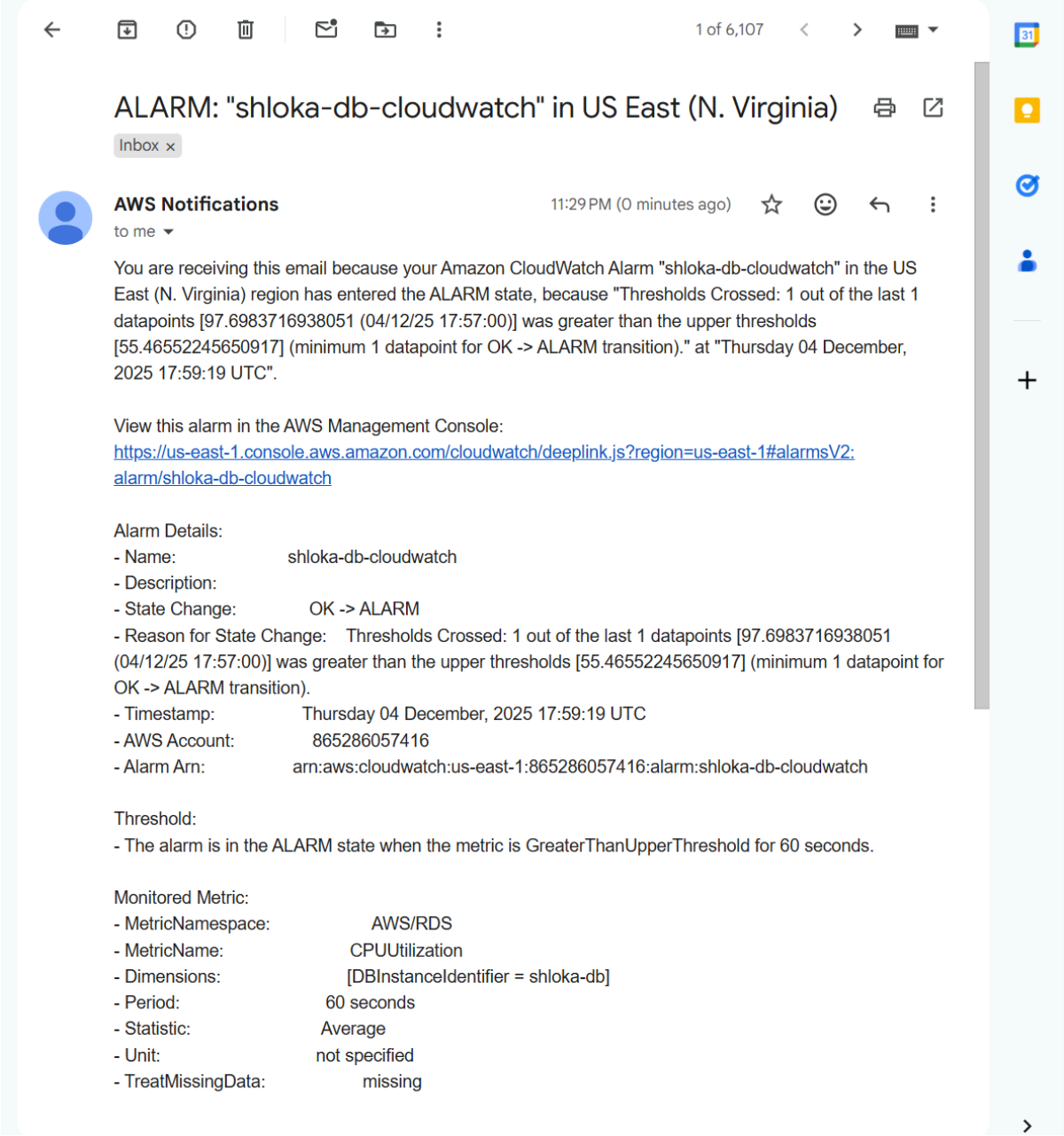
```
SELECT BENCHMARK(200000000, SHA2('shloka', 512));
```

In multiple terminals (2) to sustain >70% CPU for 5 mins.

**SNS email received.**




Increasing the database load and verifying load in real time




AWS cloudwatch email

# Step 10: Test Read Replica Connectivity

**What was done:**

Connected to read replica.

**Why was it done:**

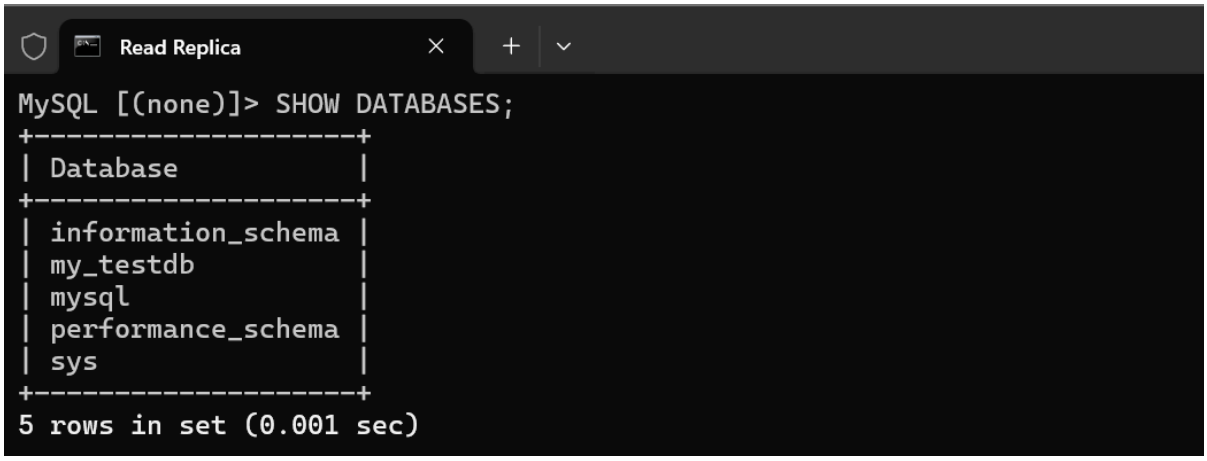
To verify replication and read-only access.

**How It Was Done:**

```
mysql -h <replica-endpoint> -u admin -p
```

```
USE my_testdb;
SELECT * FROM test_table;
```

→ Data successfully replicated.




Verifying read queries on read replica - SUCCESS

Attempt write:


Attempting to write on read replica - FAIL

Output:

```
ERROR 1290: server is read-only
```

## Step 11 — Resource Clean-Up

**What Was Done:**

All AWS resources created for this assignment were cleaned up in the correct order to avoid unnecessary charges and to ensure no dependent services were left running.

**Why It Was Done:**

AWS resources such as RDS instances, replicas, NAT Gateways, and EC2 instances incur ongoing costs. Cleaning up in the proper sequence prevents errors, avoids being billed for unused infrastructure, and maintains a clean environment.

**How It Was Done:**

1. **Delete Read Replica**
    - RDS → Databases → Select Read Replica → Actions → Delete
    - Confirm snapshot preference
    - Wait for deletion to complete
2. **Delete Primary RDS Instance**
    - RDS → Databases → Select Primary DB → Actions → Delete
    - Skip final snapshot
    - Confirm deletion
3. **Delete RDS Parameter Group**
4. **Delete RDS Subnet Group**
5. **Terminate EC2 Instances**
6. **Delete Security Groups**
7. **Delete NAT Gateway**
8. **Delete Internet Gateway**

9. **Delete Route Tables**
10. **Delete Subnets**
11. **Delete VPC**

# TESTING & VALIDATION

✔️ EC2 → Primary RDS connectivity successful

✔️ Table created + write query successful

✔️ Read replica shows same data

✔️ Replica blocks write queries

✔️ Replication lag = 0

✔️ Slow query log enabled (post reboot)

✔️ CloudWatch alarm triggered

✔️ SNS alert email received

---

# CHALLENGES FACED & SOLUTIONS

## Challenge 1: MySQL TIMED OUT — couldn't connect

**Reason:** EC2 was NOT in the same RDS subnet group, SGs were conflicting.

**Fix:** Launched EC2 directly inside **RDS-Pvt-subnet**, cleaned SGs, ensured single SG for RDS.

---

## Challenge 2: TIMEOUT despite fixed SGs

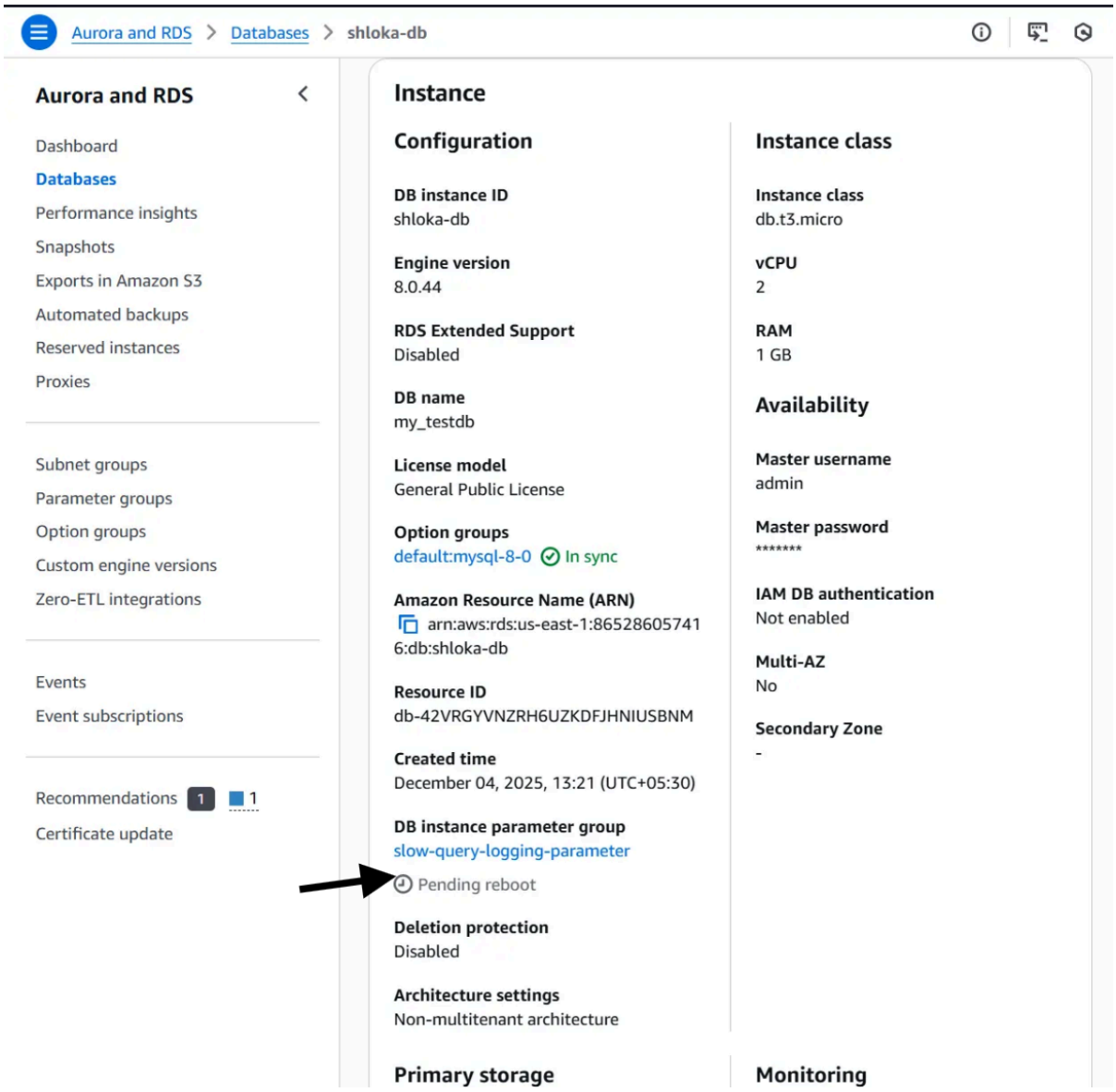
**Reason:** Subnet routing incomplete — private subnet not routed to NAT Gateway.

**Fix:** Added subnet to private route table + NAT → connectivity restored.

---

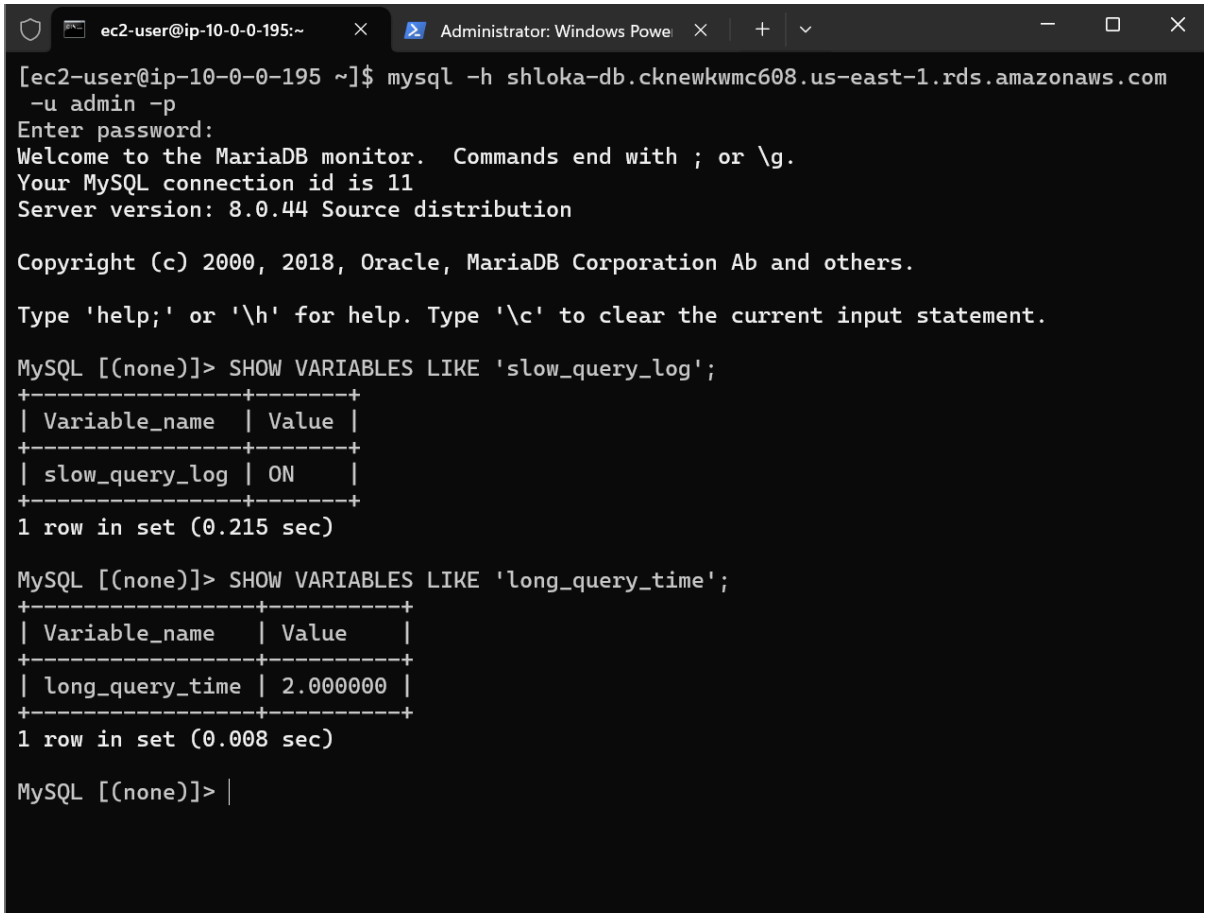
## Challenge 3: Parameter Group not applying despite it being attached

**Reason:** DB reboot required.




reboot required

**Fix:** Rebooted RDS → values applied successfully.




result after reboot

### Challenge 4: CPU Alarm not triggering

**Reason:** Single benchmark wasn't enough to sustain 70% for 5 minutes.

**Fix:** Ran multiple heavy benchmark queries in parallel.

# CONCLUSION

This assignment gave me real production-level experience with AWS RDS, subnet groups, replication, monitoring, and debugging complex connectivity issues. I learned how to design private-only architectures, structure correct security groups, troubleshoot DNS & connectivity problems, and validate replication and monitoring workflows.

## Learning Outcomes Achieved

- Deploying RDS MySQL in Single-AZ
- Creating and validating read replicas
- Applying parameter groups and rebooting for static params
- Designing subnet groups and private routing
- Configuring SG-to-SG communication
- Implementing CloudWatch alarms + SNS alerts
- Testing replication and read-only behavior
- Troubleshooting authentication, routing, and subnet issues
- Running CPU load tests
- Producing complete documentation with proof