

Exercise–13 Stress Testing a Linux EC2 Instance (CPU Load Test)

Objective:

To generate high CPU load on an Amazon EC2 Linux instance using the stress-ng tool and observe CPU utilization in CloudWatch.

Step 1 — Launch a Linux EC2 Instance

1. Log in to the AWS Console.
2. Go to **EC2 → Instances → Launch Instance**.
3. Name: **StressTest-EC2**
4. AMI: **Amazon Linux 2023 (Free Tier eligible)**
5. Instance type: **t2.micro / t3.micro**
6. Key pair: Select or create a .pem key.
7. Security Group:
 - Allow **SSH (22)** from My IP.
 - Allow **HTTP (80)** from anywhere (optional).
8. Launch the instance.

Step 2 — Connect to the EC2 Instance

Use Windows PowerShell

Navigate to the folder where pem file is located and type the following command:

```
ssh -i "your-key.pem" ec2-user@<public-ip>
```

Step 3 — Install Apache (Optional, to verify the instance is working)

```
sudo yum install httpd -y
```

```
sudo systemctl start httpd
```

```
sudo systemctl enable httpd
```

Create a test web page:

```
echo "<h1>EC2 Stress Test Demo — $(hostname)</h1>" | sudo tee  
/var/www/html/index.html
```

STEP 4 — Install and Run the Stress Testing Tool (stress-ng)

(For Amazon Linux 2023 EC2 Instances)

Move to the ec2-user home directory

(It is recommended to run stress-ng from the home folder)

```
cd ~
```

Install stress-ng

Amazon Linux 2023 includes stress-ng directly through dnf, so install it using:

```
sudo dnf install stress-ng -y
```

Run a CPU Stress Test

Generate high CPU load using 4 CPU workers for 2 minutes:

```
stress-ng --cpu 4 --timeout 120
```

Expected Result

- Terminal becomes busy during the 120-second load test
- After completion, you will see:

successful run completed in 120.02s

- CPU usage will spike to 90–100% (you can check using top)

Step 5 — Run Stress Test (CPU Load Generation)

Move to home directory to avoid permission issues:

```
cd ~
```

Run CPU stress for 2 minutes using 4 CPU workers:

```
stress-ng --cpu 4 --timeout 120
```

You should see messages like:

```
stress-ng: info: dispatching hogs: 4 cpu
```

```
stress-ng: info: successful run completed in 120.02s
```

This will increase CPU usage to ~100% for the duration.

Step 6 — Observe CPU Utilization in CloudWatch

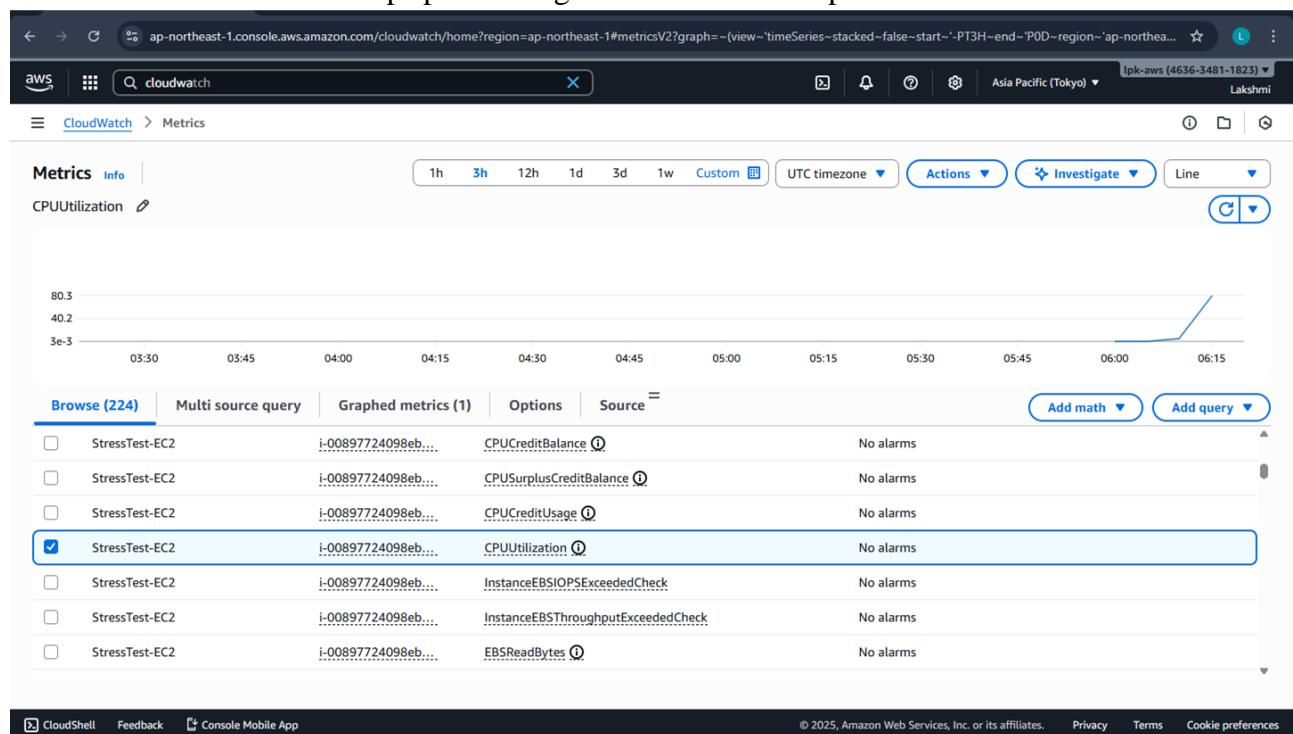
Go to CloudWatch → Metrics → EC2 → Per-Instance Metrics

Select:

CPUUtilization → InstanceId → your instance

View graph in 1-minute or 5-minute intervals.

You should see a sharp spike during the 2-minute stress period.



Step 7 — Stop or Terminate the Instance

To avoid charges:

Option A — Stop the instance (safe):

Actions → Instance State → Stop

Option B — Terminate (permanently delete):

Actions → Instance State → Terminate

Expected Output

- CPU Utilization in CloudWatch should reach 90–100%.
- Stress test completes without errors.
- Students understand how CPU load affects EC2 metrics.

Exercise–17: Flask Web App with Registration & Login Using AWS RDS (MySQL) (Full Deployment on AWS EC2 + AWS RDS)

To design and deploy a Python Flask web application on an AWS EC2 instance that allows:

1. User Registration – store Name and Password in AWS RDS MySQL database.
2. User Login – validate user credentials from the RDS database.
3. Welcome Page – display a message on successful login.

This exercise demonstrates a complete cloud-based web application workflow using Flask + MySQL (RDS).

System Architecture

Flow: Browser → Flask App (on EC2) → MySQL Database (RDS)

- Frontend: HTML templates (registration form, login form, welcome page)
- Backend: Python Flask application running on EC2
- Database: AWS RDS – MySQL engine
- Hosting: Linux EC2 instance
- Web Server: Flask’s built-in development server (Apache/NGINX not used in this exercise)

Folder Structure (on EC2 instance)

Create the following structure inside the EC2 instance:

FlaskLoginApp/

 └── app.py

 └── db_config.py

 └── templates/

 └── register.html

 └── login.html

 └── success.html

 └── error.html

 └── welcome.html

From EC2Create Database and Table in RDS MySQL

Where:

The SQL commands are executed from the EC2 instance, by connecting to the RDS MySQL database using the MySQL client.

When:

After the RDS MySQL instance status becomes “Available” and after allowing EC2 security group access in RDS inbound rules.

Create:

- A database: flaskdb
- A table: users

SQL: Create Database and Table

```
CREATE DATABASE flaskdb;
```

```
USE flaskdb;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL
);
```

- id – unique identifier for each user (auto-increment primary key).
- name – username (cannot be NULL).
- password – password (cannot be NULL).

Note: For learning purposes, we store plain text. In real systems, passwords must be **hashed**.

Flask Application Code (Backend)

6.1 db_config.py

This file stores the **RDS MySQL connection details**.

```
# db_config.py
```

```
db_config = {
    'host': 'your-rds-endpoint.amazonaws.com',
    # e.g., flaskdb-instance.xxxxxx.ap-south-1.rds.amazonaws.com
    'user': 'admin',           # master username created in RDS
    'password': 'YourPasswordHere', # master password created in RDS
    'database': 'flaskdb'        # database name created in MySQL
}
```

Replace host, user, and password with the actual values from the RDS instance.

6.2 app.py

This is the **main Flask application**.

```
# app.py
```

```
from flask import Flask, request, render_template
import mysql.connector
from db_config import db_config

app = Flask(__name__)

# Function to create a new database connection
def get_connection():
    return mysql.connector.connect(**db_config)

@app.route('/')
def home():
    return render_template('register.html')

@app.route('/register', methods=['POST'])
def register():
    name = request.form['uname']
    password = request.form['pwd']

    conn = get_connection()
    cur = conn.cursor()

    sql = "INSERT INTO users (name, password) VALUES (%s, %s)"
    cur.execute(sql, (name, password))
    conn.commit()

    cur.close()
    conn.close()

    # Use HTML template for success page
    return render_template('success.html')

@app.route('/login')
def login():
    return render_template('login.html')

@app.route('/validate', methods=['POST'])
def validate():
    name = request.form['uname']
    password = request.form['pwd']

    conn = get_connection()
    cur = conn.cursor()

    sql = "SELECT * FROM users WHERE name = %s AND password = %s"
    cur.execute(sql, (name, password))
    user = cur.fetchone()

    cur.close()
    conn.close()

    if user:
        # Successful login -> welcome page
        return render_template('welcome.html', username=name)
    else:
```

```

# Invalid login -> error page
return render_template('error.html')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=True)

```

7. HTML Templates (Frontend)

All HTML files go inside the templates/ folder.

7.1 templates/register.html

```

<!DOCTYPE html>
<html>
<head>
    <title>User Registration</title>
</head>
<body>
    <h2>User Registration</h2>

    <form action="/register" method="post">
        <label>Name:</label>
        <input type="text" name="uname" required><br><br>

        <label>Password:</label>
        <input type="password" name="pwd" required><br><br>

        <button type="submit">Register</button>
    </form>

    <br>
    <a href="/login">Already registered? Click here to Login</a>
</body>
</html>

```

7.2 templates/login.html

```

<!DOCTYPE html>
<html>
<head>
    <title>User Login</title>
</head>
<body>
    <h2>User Login</h2>

    <form action="/validate" method="post">
        <label>Name:</label>
        <input type="text" name="uname" required><br><br>

        <label>Password:</label>
        <input type="password" name="pwd" required><br><br>

```

```
<button type="submit">Login</button>
</form>

<br>
<a href="/">New user? Click here to Register</a>
</body>
</html>
```

7.3 templates/welcome.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome</title>
</head>
<body>
    <h2>Welcome, {{ username }}!</h2>
    <p>You have successfully logged in.</p>
</body>
</html>
```

7.4 templates/success.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Registration Successful</title>
</head>
<body>
    <h3>Registration Successful!</h3>
    <a href="/login">Click here to Login</a>
</body>
</html>
```

7.5 templates/error.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Login Error</title>
</head>
<body>
    <h3>Invalid Username or Password</h3>
    <a href="/login">Try Again</a>
</body>
</html>
```

Deployment Steps –

PHASE 1 – Launch EC2 Linux Instance

- Open AWS Management Console → EC2 → Launch instance.
- Configuration:
 - Name: FlaskLoginServer
 - AMI: Amazon Linux 2023 (Free Tier eligible)
 - Instance type: t3.micro (Free Tier)
 - Key pair: create or select existing
 - Network: Default VPC and any public subnet
 - Auto-assign Public IP: Enable
- Security Group for EC2 – Inbound rules:

Type	Port	Source	Purpose
SSH	22	My IP	To connect via SSH/EC2 Connect
Custom TCP	5000	0.0.0.0/0	To test Flask app in browser

- Outbound: Allow all traffic.
- Click Launch instance.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, AWS Global View, Events, Instances (selected), Images, and Elastic Block Store. The main area displays a table titled 'Instances (1/1)'. It shows one instance named 'FlaskLoginServer' with the ID 'i-056b8bfd7c3fe8f'. The instance is listed as 'Running' with a 't3.micro' instance type. Below the table, under the heading 'i-056b8bfd7c3fe8f (FlaskLoginServer)', there are sections for 'Inbound rules' and 'Outbound rules'. Under 'Inbound rules', two rules are listed: one for port 22 (TCP) from 0.0.0.0/0 and another for port 5000 (TCP) from 0.0.0.0/0. Both rules are associated with the security group 'launch-wizard-3'. The 'Outbound rules' section is currently collapsed.

PHASE 2 – Connect to EC2 and Install Dependencies

- Go to EC2 → Instances → Select instance → Connect.
- Choose EC2 Instance Connect (browser-based SSH) → Connect.
Or open PowerShell on windows and connect using ssh command
- Update packages:
`sudo yum update -y`
- Check Python version (Amazon Linux 2023 has Python 3):
`python3 --version`

- Install pip (if not already installed):


```
sudo dnf install python3-pip -y
```
- Install Flask and MySQL Connector for Python:


```
pip3 install flask
pip3 install mysql-connector-python
```

PHASE 3 – Create Flask Project Structure on EC2

- Create project folder:


```
mkdir FlaskLoginApp
cd FlaskLoginApp
```
- Create templates folder:


```
mkdir templates
```
- Create the Python and HTML files using nano:


```
nano db_config.py → paste code for db_config.py
nano app.py → paste code for app.py
nano templates/register.html → paste register page
nano templates/login.html → paste login page
nano templates/welcome.html → paste welcome page
```

Save each file using: CTRL + O → Enter → CTRL + X

```
ec2-user@ip-172-31-10-125:~/FlaskLoginApp$ ls -R
.:
app.py  db_config.py  templates

./templates:
error.html  login.html  register.html  success.html  welcome.html
[ec2-user@ip-172-31-10-125 FlaskLoginApp]$ |
```

PHASE 4 – Create RDS MySQL Instance

- Go to AWS Console → RDS → Databases → Create database.
 - Engine type: MySQL
 - Templates: Free tier
 - DB instance identifier: flaskdb-instance
 - Master username: admin (example)
 - Master password: <your-password>
 - DB instance class: db.t3.micro (Free tier)
 - Storage: e.g. **20 GB** (Free tier)
 - Connectivity:
 - Connect to an EC2 compute resource → YES
 - Select the same EC2 instance (or its security group)
 - (AWS automatically performs the following:
 - Selects the same VPC as EC2
 - Creates/associates an RDS security group
 - Adds inbound rule: MySQL (3306) from EC2 security group)
 - Public access: No
 - Click Create database.
- Wait until the status becomes “Available”.

PHASE 5 – Verify RDS Security Group (Auto-configured)

- Go to AWS Console → RDS → Databases → flaskdb-instance.
- Open Connectivity & security section.
- Under VPC security groups, click the linked security group name.
- In Inbound rules, verify that the following rule exists:

Type	Port	Source
MySQL / Aurora	3306	EC2 Security Group

5. If the rule exists, EC2 to RDS connectivity is correctly configured.

The screenshot shows the AWS RDS console for the 'flaskdb-instance' database. The 'Connectivity & security' tab is active. Key details shown include:

- Endpoint:** flaskdb-instance.cf82gsqee06w.ap-northeast-1.rds.amazonaws.com
- Port:** 3306
- Networking:**
 - Availability Zone: ap-northeast-1c
 - VPC: vpc-0ec766fa292708a26
 - Subnet group: rds-ec2-db-subnet-group-1
 - Subnets: subnet-05d55dc648c910f7b, subnet-05c2ab105b410200
- Security:**
 - VPC security groups: rds-ec2-1 (sg-0a08c950cc1db26ef) - Active
 - Publicly accessible: No
 - Certificate authority: rds-ca-2048-g1
 - Certificate authority date: Mon, 26 Dec 2024, 04:14:07 UTC (05:12)

PHASE 6 – Create Database and Table in RDS from EC2

- From the EC2 terminal, install MySQL client (MariaDB client):


```
sudo dnf install mariadb105 -y
# if this fails, try: sudo dnf install mariadb -y
```
- Connect to the RDS MySQL instance from EC2:


```
mysql -h <RDS-endpoint> -u admin -p
Example: mysql -h flaskdb-instance.xxxxxx.ap-south-1.rds.amazonaws.com -u admin -p
```
- Inside the MySQL prompt, run following commands:

```
CREATE DATABASE flaskdb;
```

```
USE flaskdb;
```

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL
```

- Exit MySQL: exit;

```
[ec2-user@ip-172-31-10-125 FlaskLoginApp]$ mysql -h flaskdb-instance.cf82gsqee06w.ap-northeast-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 8.0.43 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.

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

MySQL [(none)]> USE flaskdb;
Database changed
MySQL [flaskdb]> CREATE TABLE users (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(50) NOT NULL,
    ->     password VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.030 sec)
```

PHASE 7 – Configure Flask to Use RDS and Run Application

- Open db_config.py on EC2 and update with **correct RDS details**:

```
nano db_config.py
```

Ensure:

```
db_config = {
    'host': 'your-rds-endpoint.amazonaws.com',
    'user': 'admin',
    'password': 'YourPasswordHere',
    'database': 'flaskdb'
}
```

Save and exit.
- Start the Flask app:

```
cd ~/FlaskLoginApp
python3 app.py
```

You should see: * Running on http://0.0.0.0:5000

PHASE 8 – Test the Application from Browser

- In your local system browser, open: http://<EC2-public-IP>:5000

Example: http://13.232.122.81:5000
You should see: Registration Page (default route /)
- After registration → success message with link to /login
- Login Page → if correct credentials → Welcome Page

Check that data is stored in flaskdb.users table in RDS. - SELECT * FROM users;

```
ec2-user@ip-172-31-10-125:~| + | - | X
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 35
Server version: 8.0.43 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.

MySQL [(none)]> use flaskdb;
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
MySQL [flaskdb]> show tables;
+-----+
| Tables_in_flaskdb |
+-----+
| users           |
+-----+
1 row in set (0.001 sec)

MySQL [flaskdb]> select * from users;
+----+-----+-----+
| id | name | password |
+----+-----+-----+
| 1  | lpk  | 1234   |
+----+-----+-----+
1 row in set (0.001 sec)

MySQL [flaskdb]> |
```

Expected Output

1. Registration Page
 - o User enters Name and Password.
 - o On submit, data is inserted into users table in RDS MySQL.
2. Login Page
 - o User enters same Name and Password.
 - o Flask executes SELECT query on users table to check match.
3. Welcome Page
 - o For valid credentials:

“Welcome, <username>! You have successfully logged in.”

- o For invalid credentials:

“Invalid Username or Password” message with link to try again.

Result

A **fully working cloud-based registration and login system** was successfully deployed using:

- **Flask** as the backend web framework
- **HTML** templates for frontend pages
- **MySQL** database hosted on **AWS RDS**
- **AWS EC2** instance as the application server

This exercise demonstrates a **complete end-to-end mini project** on AWS using **Flask + MySQL (RDS)**.

Sample Viva Questions

1. What is **Amazon RDS**? Why do we use it instead of installing MySQL directly on EC2?
2. What is the difference between **EC2** and **RDS**?
3. How does **Flask** connect to **MySQL** in this exercise?
4. What is a **POST request**? Where is it used in this application?
5. Why must the **EC2 security group** be allowed inside **RDS inbound rules**?

AWS RDS databases are private by default and do not allow any incoming connections unless explicitly permitted by security group rules.

Since the Flask application runs on EC2, it must be allowed to connect to the RDS MySQL database.

Therefore, the EC2 security group must be added to the RDS inbound rules to allow:

- Port: 3306 (MySQL)
- Source: EC2 Security Group

This ensures that only the EC2 instance running the application can access the database.

6. What is the purpose of db_config.py?
7. What is **SQL Injection**? How does using parameterized queries (%s) help prevent it?

SQL Injection:

A technique where malicious SQL code is entered as user input.

In this application, it is prevented using parameterized queries (%s).

8. What is the difference between **Registration workflow** and **Login workflow** in this app?
9. Why do we use host='0.0.0.0' in app.run() on EC2?

By default, Flask runs on 127.0.0.1 (localhost), which means the application is accessible **only from inside the EC2 instance**.

When we set: app.run(host='0.0.0.0', port=5000)

it tells Flask to **listen on all network interfaces** of the EC2 instance.

This allows:

- Access from the **browser on our local system**
- Access using the **EC2 public IP address**

What happens if we don't use 0.0.0.0?

- Flask listens only on localhost
- Application works **inside EC2**
- Browser access using http://<EC2-public-IP>:5000 fails

In this exercise

- Flask runs on EC2

- Users access it from outside EC2
- Therefore, Flask must listen on all interfaces using 0.0.0.0

We use host='0.0.0.0' so that the Flask application running on EC2 is accessible from external systems using the EC2 public IP.

10. Why is debug=True not recommended in production?

DATE: 28-12-25

Exercise-18: Creating and Operating a NoSQL Key-Value Database using Amazon

DynamoDB

To create an Amazon DynamoDB table and perform CRUD operations (Create, Read, Update, Delete) using the AWS Management Console, and understand Partition Key, Sort Key, Query vs Scan, and table cleanup.

Phase 1: Create DynamoDB Table

Create a DynamoDB table to store Student Records.

Store items like: USN, Name, Semester, Course, Attendance, IA1Marks

Step 1: Open DynamoDB

- Login to AWS Console
- Search → type DynamoDB → open Amazon DynamoDB

Step 2: Create a Table

- Left menu → Tables
- Click Create table
- Fill details:

Table details

- Table name: MCA_StudentLabInternals
- Partition key (PK): USN (Type: String)
- Sort key (SK): CourseCode (Type: String)
(Enable sort key option by setting sort key)

Why this design?

- One student can have multiple courses → Sort key separates records per course.
- This creates a composite primary key (PK + SK).

Step 3: Table settings

- Under Table settings, choose:
Default settings (recommended for lab)
- Ensure Capacity mode is: On-demand (default already selected)

Step 4: Create

- Click Create table
- Wait until table status becomes Active

The screenshot shows the AWS DynamoDB 'Tables' page. A green success message at the top states: 'The MCA_StudentLabInternals table was created successfully.' Below this, the 'Tables (1/1)' section is displayed. A single table row for 'MCA_StudentLabInternals' is listed, showing its status as 'Active', partition key as 'USN (\$)', and sort key as 'CourseCode (\$)'. The table has 0 items and 0 indexes. The 'Actions' button is visible next to the table name.

Phase 2: Insert Items (Create Data)

Step 5: Open Table and Add Items

- Click the table: MCA_StudentLabInternals
- Click: Explore table items
- Click Create item

Step 6: Add Item 1 (Student 1 - Course 1)

- You will see attributes:
 - USN (String)

- CourseCode (String)

Enter:

- USN = 1MS24MCA001
- CourseCode = CCL301

Now add additional attributes (click Add new attribute):

- Name (String) = Arun
- Semester (Number) = 3
- Attendance (Number) = 86
- IA1Marks (Number) = 18

Click Create item

Step 7: Add Item 2 (Same student - another course)

Repeat Create item with:

- USN = 1MS24MCA001
- CourseCode = DBS301
- Name = Arun
- Semester = 3
- Attendance = 88
- IA1Marks = 20

Step 8: Add Item 3 (Another student)

Create item:

- USN = 1MS24MCA002
- CourseCode = CCL301
- Name = Chitra
- Semester = 3
- Attendance = 74
- IA1Marks = 14

You should now see 3 items in the table list.

The screenshot shows the Amazon DynamoDB Item Explorer interface. On the left, there's a sidebar with navigation links like 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', etc. The main area shows a table named 'Table - MCA_StudentLabInternals'. A message at the top right says 'Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCU consumed: 2'. Below this, a table titled 'Table: MCA_StudentLabInternals - Items returned (3)' lists three items:

	USN (String)	CourseCode (String)	Attendance	IA1Marks	Name
<input type="checkbox"/>	1MS24MCA001	CCL301	86	18	Arun
<input type="checkbox"/>	1MS24MCA001	DBS301	88	20	Arun
<input type="checkbox"/>	1MS24MCA002	CCL301	74	14	Chitra

At the bottom, there are links for 'CloudShell', 'Feedback', 'Console Mobile App', and various system status indicators.

Phase 3: Read Data (Get / Query / Scan)

Step 9: Get a single item (exact PK + SK)

- In “Explore table items”, use search/filter:
- Use USN = 1MS24MC001 and CourseCode = CCL301
- Open the item → verify all attributes

Key concept:

To uniquely identify an item in a PK+SK table, you must provide **both**.

Step 10: Query – fetch all courses for one student

- Click Run query
- Set Partition key condition: USN equals 1MS24MCA001
- Run

Expected output:

- Items for Arun in both CCL301 and DBS301

Key concept:

Query works only with Partition Key (and optional Sort Key conditions). It is efficient.

Step 11: Scan (Not for large tables)

- Click Scan
- Run scan without filters

Expected output:

- All items (Arun + Chitra)

Key concept:

Scan reads the entire table → slow/costly for big tables.

The screenshot shows the AWS DynamoDB console with the 'Explore Items' feature for the 'MCA_StudentLabInternals' table. The table contains two items:

USN	CourseCode	Attendance	IA1Marks	Name	Semester
1MS24MC001	CCL301	86	18	Arun	3
1MS24MC001	DBS301	88	20	Arun	3

Phase 4: Update Data

Step 12: Update IA1 marks of Chitra for CCL301

- Open item where:
 - USN = 1MS24MCA002
 - CourseCode = CCL301

- Click Edit
- Change: IA1Marks from 14 to 16
- Click Save changes

Verify updated value appears.

Phase 5: Delete Data

Step 13: Delete one item (Arun's DBS301 record)

- Select item:
 - USN = 1MS24MCA001
 - CourseCode = DBS301
- Click **Delete**
- Confirm delete

Now total items should reduce by 1.

The screenshot shows the AWS DynamoDB console interface. On the left, there is a navigation sidebar with options like Dashboard, Tables, Explore items (which is selected), PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a DAX section with Clusters, Subnet groups, Parameter groups, and Events. The main area shows a table named 'Tables (1)' with one entry: 'MCA_StudentLabInternal'. A success message at the top says 'Selected items have been deleted successfully.' Below the table, the 'Scan or query items' section is visible, showing 'Scan' selected. It includes fields for 'Select a table or index' (set to 'Table - MCA_StudentLabInternals') and 'Select attribute projection' (set to 'All attributes'). A 'Completed' message below the table indicates 'Items returned: 2'. At the bottom, there is a table titled 'Table: MCA_StudentLabInternals - Items returned (1)' with columns: USN (String), CourseCode (String), Attendance, IA1Marks, Name, and Semester. The table shows two rows of data.

Phase 6: Cleanup (Must Do to avoid charges)

Step 15: Delete the Table

- DynamoDB → Tables
- Select MCA_StudentLabInternals
- Click Delete
- Type confirmation (if asked) and delete

Ensure table disappears from list.

List tables | Amazon DynamoDB

ap-northeast-1.console.aws.amazon.com/dynamodbv2/home?region=ap-northeast-1#tables

DynamoDB > Tables

The request to delete the "MCA_StudentLabInternals" table has been submitted successfully.

Tables (1/1) [Info](#)

Last updated January 2, 2026, 09:44 (UTC+5:30)

Actions [Delete](#) [Create table](#)

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection
MCA_StudentLabInternals	Deleting	-	-	0	0	Off

Find tables Filter by tag Any tag key Filter by tag value Any tag value

1

DynamoDB Tables Explore items PartQL editor Backups Exports to S3 Imports from S3 Integrations Reserved capacity

Lakshmi Asia Pacific (Tokyo) lpk-aws (4636-3481-1823)

This screenshot shows the AWS DynamoDB console interface. A prominent green success message at the top states: 'The request to delete the "MCA_StudentLabInternals" table has been submitted successfully.' Below this, the 'Tables' section displays one table named 'MCA_StudentLabInternals'. The table row is highlighted with a blue selection bar. The 'Status' column shows 'Deleting'. The 'Deletion protection' column is set to 'Off'. The top navigation bar includes links for 'Dashboard', 'Tables', and other features like 'Explore items' and 'Exports to S3'. The top right corner shows the user's name 'Lakshmi' and the region 'Asia Pacific (Tokyo)'.

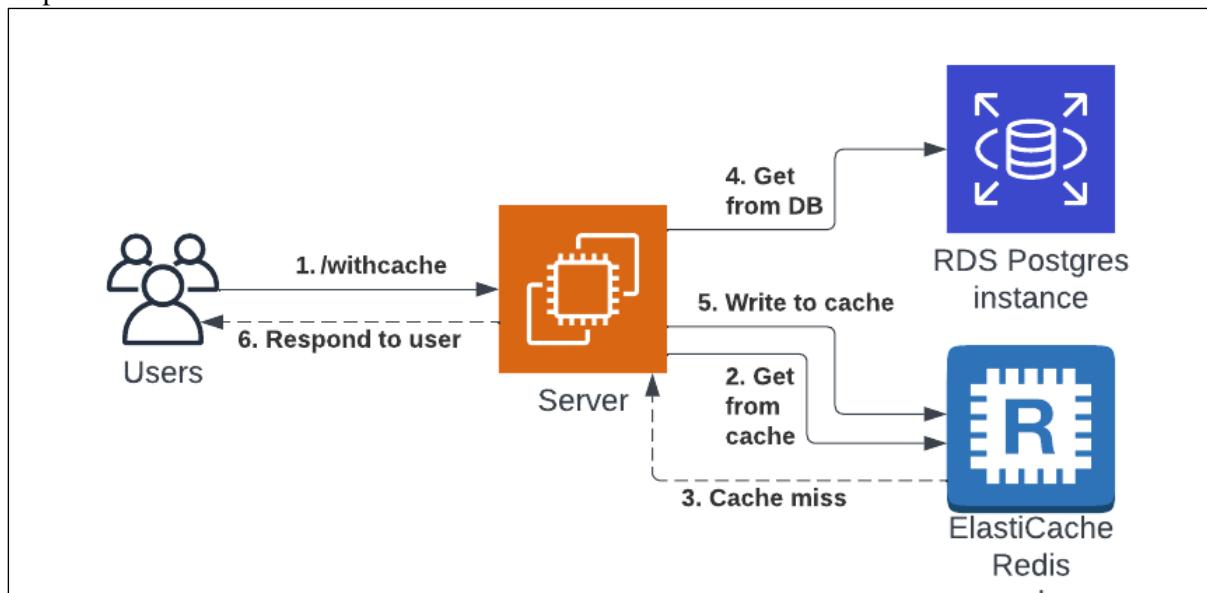
DATE: 03-12-25

Exercise–19: ElastiCache (Redis) as an In-Memory Cache

To implement Amazon ElastiCache (Redis) as an in-memory caching service by deploying a Redis cluster and performing basic cache operations from an EC2 instance.

Application-Level Data Flow (Logical)

- Step 1: User requests data
- Step 2: Application checks ElastiCache (Redis)
- Step 3: If data exists → return from cache (FAST)
- Step 4: If data does not exist → fetch from RDS
- Step 5: Store fetched data in ElastiCache
- Step 6: Return data to user



This exercise provides **exposure** to an industry-used in-memory data store.

Redis improves performance by serving frequently accessed data from memory and is typically used **in front of databases** in real-world applications.

Feature	RDS	ElastiCache
Storage	Disk	RAM
Data lifetime	Permanent	Temporary
Access speed	Slower	Very fast
TTL support	No	Yes

Phase-1 → EC2 creation + valkey-cli installation

Phase-2 → Redis cache creation

Phase-3 → Connect EC2 → Redis engine and run commands

Phase-1: Launch EC2 Client using Amazon Linux 2023 (for Valkey / Redis Client)

Purpose of Phase-1

To create an EC2 instance using **Amazon Linux 2023** that will act as a **client machine** to connect to Amazon ElastiCache (Redis OSS) using **valkey-cli**.

Step 1: Select AWS Region

- Choose ONE region and stick to it for the entire exercise
ap-south-1 (Mumbai) (*recommended*)
- Ensure ElastiCache and EC2 will be in the SAME region

Step 2: Launch EC2 Instance

1. Go to **AWS Console → EC2**
2. Click **Launch instance**

Step 3: Configure Instance Basics

- Name: RedisClient-AL2023
- AMI: Select Amazon Linux 2023 AMI
- Instance Type: Select t3.micro (Free Tier eligible)
- Key Pair: Select an existing key pair OR create a new one (RSA, .pem)
- Network & Security
 - Network
 - VPC: Default VPC
 - Subnet: No preference
 - Auto-assign public IP: Enabled
- Security Group (VERY IMPORTANT)
Create a new security group:
 - Security Group Name: SG-RedisClient
 - Description: Security group for Redis client EC2Inbound Rules

Type	Port	Source
SSH	22	Anywhere 0.0.0.0/0

Do NOT add Redis (6379) here (The client does not listen on 6379)
Outbound rules: Allow all (default)

Step 5: Storage

- Keep default

Step 6: Launch Instance

- Click Launch instance
- Wait until Instance State = Running

Step 7: Connect to EC2

1. EC2 → Instances → select RedisClient-AL2023
 2. Click Connect
 3. Choose EC2 Instance Connect
 4. Click Connect
- You are now inside the EC2 terminal.

Step 8: Install Valkey Client

Run the following commands:

```
sudo dnf update -y  
sudo dnf install -y valkey
```

Verify installation:

```
valkey-cli --version
```

Expected Output (example)

valkey-cli 8.x.x

This confirms the EC2 instance is ready as a Redis-compatible client.

Note: why install valkey and not redis??

Amazon Linux 2023 does not provide redis-cli directly.

AWS now provides **Valkey**, which is fully Redis-protocol compatible.

Hence, we use **valkey-cli** to connect to ElastiCache Redis.

```
Installing:
valkey           x86_64      8.0.6-3.amzn2023.0.2      amazonlinux      1.6 M
Transaction Summary
Install 1 Package

Total download size: 1.6 M
Installed size: 5.8 M
Downloading Packages:
valkey-8.0.6-3.amzn2023.0.2.x86_64.rpm          18 MB/s | 1.6 MB   00:00
Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing       :
  Running scriptlet: valkey-8.0.6-3.amzn2023.0.2.x86_64      1/1
  Installing      : valkey-8.0.6-3.amzn2023.0.2.x86_64      1/1
  Running scriptlet: valkey-8.0.6-3.amzn2023.0.2.x86_64      1/1
  Verifying       : valkey-8.0.6-3.amzn2023.0.2.x86_64      1/1

Installed:
  valkey-8.0.6-3.amzn2023.0.2.x86_64

Complete!
[ec2-user@ip-172-31-26-115 ~]$ valkey-cli --version
valkey-cli 8.0.6
[ec2-user@ip-172-31-26-115 ~]$
```

Phase-2: Create Amazon ElastiCache (Redis OSS) Cluster

Purpose of Phase-2

Purpose of Phase-2
To create an Amazon ElastiCache Redis OSS cluster that will act as an in-memory cache, accessible from the EC2 client created in Phase-1.

Step 1: Confirm AWS Region

- Ensure you are in the **same region** used in Phase-1
EC2 and ElastiCache must be in the **SAME** region and VPC

Step 2: Open ElastiCache Console

1. AWS Console → **Services**
 2. Select ElastiCache
 3. Click Create cache

Step 3: Select Cache Engine

- ### • Engine: Redis OSS

Step 4: Choose Deployment Settings

- Deployment option: Node-based cluster
 - Creation method: Easy create

Easy create is used to avoid advanced production settings

Step 5: Select Configuration

- Configuration: Demo

This automatically selects:

- A small, low-cost node (e.g., cache.t4g.micro)
- Suitable for labs and practice

Step 6: Provide Cluster Information

- Cache name: lab-redis
- Description: Optional (lab-redis)

Step 7: Configure Network and Subnet Group

Network

- Network type: IPv4

Subnet Group

- Select: Create a new subnet group
 - Subnet group name: redis-subnet-group
 - VPC: Default VPC
 - Subnets: Leave AWS auto-selected subnets unchanged

No manual subnet selection required.

Step 8: Configure Security

Security Group

- Create or select a security group:
 - Name: SG-RedisCache

Inbound Rule for Redis

Add one inbound rule to SG-RedisCache:

Type	Port	Source
Custom TCP	6379	SG-RedisClient

This allows only the EC2 client to access Redis.

Step 9: Authentication

- Authentication: Disabled

Step 10: Create Cache

1. Review all settings
2. Click Create
3. Wait until Status = Available

This may take a few minutes.

Step 11: Note the Redis Endpoint

1. Click on the cache name lab-redis
2. Copy the Configuration Endpoint
 - Example: lab-redis.xxxxxx.cache.amazonaws.com

This endpoint will be used in **Phase-3** to connect from EC2.

Note:

We have created an in-memory Redis cache using Amazon ElastiCache.
It runs inside the AWS VPC and does not have a public IP.
Only our EC2 client is allowed to access it using port 6379.

Cluster details

Cluster name	Description	Node type	Status
lab-redis	Easy created demo cluster on 2026-01-02T04:22:35.397Z	cache.t4g.micro	Available
Engine	Redis	Global datastore	Global datastore role
Update status	Up to date	Cluster mode	Number of nodes
Data tiering	Disabled	Enabled	1
Encryption at rest	Enabled	Multi-AZ	Encryption in transit
Configuration endpoint	clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379	Parameter group default.redis7.cluster.on	Enabled
Data migration	No active migrations	Primary endpoint -	Transit encryption mode Required
		Reader endpoint -	ARN arn:aws:elasticache:ap-northeast-1:463634811823:replicationgroup:lab-redis

Phase-3: Connect EC2 to ElastiCache Redis and Execute Cache Commands

Purpose of Phase-3

To connect the EC2 client (Amazon Linux 2023) to the ElastiCache Redis OSS cluster using valkey-cli and demonstrate basic in-memory cache operations.

Pre-checks - Before connecting, confirm:

- EC2 and ElastiCache are in the same AWS region
- EC2 security group = SG-RedisClient
- Redis security group = SG-RedisCache
- Redis security group allows port 6379 from SG-RedisClient
- Redis Status = Available
- You have copied the Primary Endpoint

Step 1: Connect to EC2

1. AWS Console → EC2
2. Select instance RedisClient-AL2023
3. Click Connect
4. Choose EC2 Instance Connect
5. Click Connect

You are now inside the EC2 terminal.

Step 2: Verify Valkey Client

Run: valkey-cli --version

Expected Output (example)

valkey-cli 8.x.x

This confirms the EC2 is ready to act as a Redis-compatible client.

Step 3: Connect to ElastiCache Redis

Use the Configuration Endpoint from Phase-2.

```
valkey-cli -h <Configuration_ENDPOINT> -p 6379
```

Example: valkey-cli -h lab-redis.xxxxxx.cache.amazonaws.com -p 6379 --tls -c

Expected Result

You should see a prompt like:

```
lab-redis.xxxxxx.cache.amazonaws.com:6379>
```

This means the connection is successful.

Step 4: Execute Redis / Valkey Commands

These commands simulate what an application does internally.

Test Connectivity

PING

Expected Output

PONG

Store and Retrieve Data (SET / GET)

SET course "Cloud Computing"

GET course

Expected Output

"Cloud Computing"

Demonstrates **key-value storage in memory**.

Counter Example (INCR)

INCR visits

INCR visits

GET visits

Expected Output

"2"

NOTE: Common real-world use - page views, hit counters.

4.4 Set Data with Expiry (TTL)

SET notice "Results Published" EX 60

TTL notice

GET notice

Expected Output

- TTL shows a value ≤ 60
- GET returns:

"Results Published"

Shows **temporary cache data**.

4.5 Verify Automatic Expiry

Wait ~60 seconds, then run:

GET notice

Expected Output

(nil)

Confirms data is **automatically removed from memory**.

4.6 Delete Data Manually

DEL course

GET course

Expected Output

(nil)

```

Installed:
valkey-8.0.6-3.amzn2023.0.2.x86_64

Complete!
[ec2-user@ip-172-31-26-115 ~]$ valkey-cli --version
valkey-cli 8.0.6
[ec2-user@ip-172-31-26-115 ~]$ valkey-cli -h clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com -p 6379 --tls -c
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> ping
PONG
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> SET course "Cloud Computing"
OK
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET course
"Cloud Computing"
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> INCR visits
(integer) 1
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> INCR visits
(integer) 2
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET visits
"2"
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> SET notice "Results Published" EX 60
OK
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> TTL notice
(integer) 60
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET notice
"Results Published"
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET notice
"Results Published"
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET notice
"Results Published"
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> GET notice
(nil)
clustercfg.lab-redis.wyijj2.apne1.cache.amazonaws.com:6379> |
```

i-09843e8d2701a3948 (RedisClient-AL2023)

PublicIPs: 35.74.248.50 PrivateIPs: 172.31.26.115

[CloudShell](#) [Feedback](#) [Console Mobile App](#)

© 2026, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Step 5: Exit Redis Client

EXIT

Note:

“The application first checks Redis.

If data is present, it is returned immediately from memory.

If not present, the application fetches data from the database and stores it in Redis with a TTL.
Redis automatically removes the data after expiry.”

Phase-3 Outcome

- EC2 successfully connected to ElastiCache Redis
- In-memory key–value operations verified
- Temporary storage and TTL behavior observed
- Redis used as a **cache**, not as a primary database

Common Errors & Quick Fix

Problem	Reason	Fix
Connection timeout	Wrong SG or region	Check SG-RedisCache inbound rule
Command hangs	Redis not Available	Wait & retry
(nil) output	Key expired	Expected behavior

Clean-Up (Mandatory)

Step 1: Delete Redis Cluster

- ElastiCache → Select lab-redis → Delete
- Disable snapshots

Step 2: Terminate EC2 Instance

- EC2 → Instances → Terminate RedisClient-EC2

Step 3: Optional

- Delete unused security groups

Redis Commands Explanation

- **SET** course "Cloud Computing"
- **GET** course
- **EXPIRE** course 30
- **TTL** course
- The above commands simulate application caching behavior.
The **SET** command represents storing frequently accessed data in cache.
The **EXPIRE** command shows that cached data is temporary and stored in memory.
After expiry, the application would fetch fresh data from the database again.

Exercise—22: CloudFormation – Launch EC2 (Amazon Linux 2023) + Install Apache using UserData

Create an EC2 instance using AWS CloudFormation (YAML template) and automatically install Apache web server (httpd) using UserData, then verify the website from a browser.

Pre-requisites

- Region set to **Asia Pacific (Mumbai) – ap-south-1**
- Basic knowledge of EC2 and Security Groups

Part A — Create Key Pair (One-time setup)

Step A1: Open EC2 Key Pairs

1. AWS Console → Search **EC2**
2. Left menu → **Key Pairs** (under “Network & Security”)
3. Click **Create key pair**

Step A2: Create key pair

- Name: pemkeypair (any name is fine)
- Key pair type: **RSA**
- Private key file format:
 - **.pem** (recommended)

Click **Create key pair**

A file will download like: pemkeypair.pem

Note: CloudFormation uses **key pair NAME** (pemkeypair), not the file name.

Part B — Create CloudFormation Template File (Amazon Linux 2023 + Apache)

Step B1: Create YAML file on your computer

1. Open **Notepad**

2. Paste the full YAML template given below

3. Save as: **ec2-apache-al2023.yaml**

- Save type: **All files**
- Encoding: UTF-8 (if asked)

Full CloudFormation Template (Amazon Linux 2023)

Important:

- Do **not** add .pem anywhere.
- You will select Key Pair from dropdown during stack creation.

```
AWSTemplateFormatVersion: "2010-09-09"
```

```
Description: Launch EC2 (Amazon Linux 2023) and install Apache (httpd) using UserData
```

Parameters:

KeyName:

```
Type: AWS::EC2::KeyPair::KeyName
```

```
Description: Select an existing EC2 Key Pair to enable SSH access
```

Resources:

WebServerSG:

```
Type: AWS::EC2::SecurityGroup
```

Properties:

```
GroupDescription: Allow SSH (22) and HTTP (80)
```

SecurityGroupIngress:

```
- IpProtocol: tcp
```

```
  FromPort: 22
```

```
  ToPort: 22
```

```
  CidrIp: 0.0.0.0/0
```

```
- IpProtocol: tcp
```

```
  FromPort: 80
```

```
  ToPort: 80
```

```
  CidrIp: 0.0.0.0/0
```

```
WebServerInstance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType: t3.micro
    KeyName: !Ref KeyName
    SecurityGroups:
      - !Ref WebServerSG

    # Amazon Linux 2023 AMI for Mumbai (ap-south-1)
    ImageId: ami-02b49a24cfb95941c

  UserData:
    Fn::Base64: |
      #!/bin/bash
      dnf update -y
      dnf install -y httpd
      systemctl enable httpd
      systemctl start httpd
      echo "<h1>Apache Installed via CloudFormation UserData (Amazon Linux 2023)!</h1>" >
      /var/www/html/index.html

  Outputs:
    InstanceId:
      Description: EC2 Instance ID
      Value: !Ref WebServerInstance

    WebsiteURL:
      Description: Apache Website URL
      Value: !Sub "http://${WebServerInstance.PublicDnsName}"
```

Part C — Create CloudFormation Stack (Console Steps)

Step C1: Open CloudFormation

1. AWS Console → Search **CloudFormation**
2. Click **Stacks**
3. Click **Create stack** → **With new resources (standard)**

Step C2: Prepare template (select correct options)

1. Under **Prepare template**:
Select **Choose an existing template**
2. Under **Template source**:
Select **Upload a template file**
3. Click **Choose file** → select ec2-apache-al2023.yaml
4. Click **Next**

Part D — Specify Stack Details

Step D1: Stack name

- Stack name: EC2-Apache-AL2023

Click **Next**

Step D2: Parameters

- Under **KeyName**, select your key pair name from dropdown
Example: pemkeypair

Click **Next**

Part E — Configure Stack Options (keep default)

1. Leave everything as default
2. Click **Next**

Part F — Review and Create

1. Scroll down
2. Click **Create stack**

Part G — Monitor Stack Creation

1. Wait for Stack status to become:
CREATE_COMPLETE
2. Open the stack → click **Outputs** tab
3. Copy **WebsiteURL**
4. Paste URL in browser

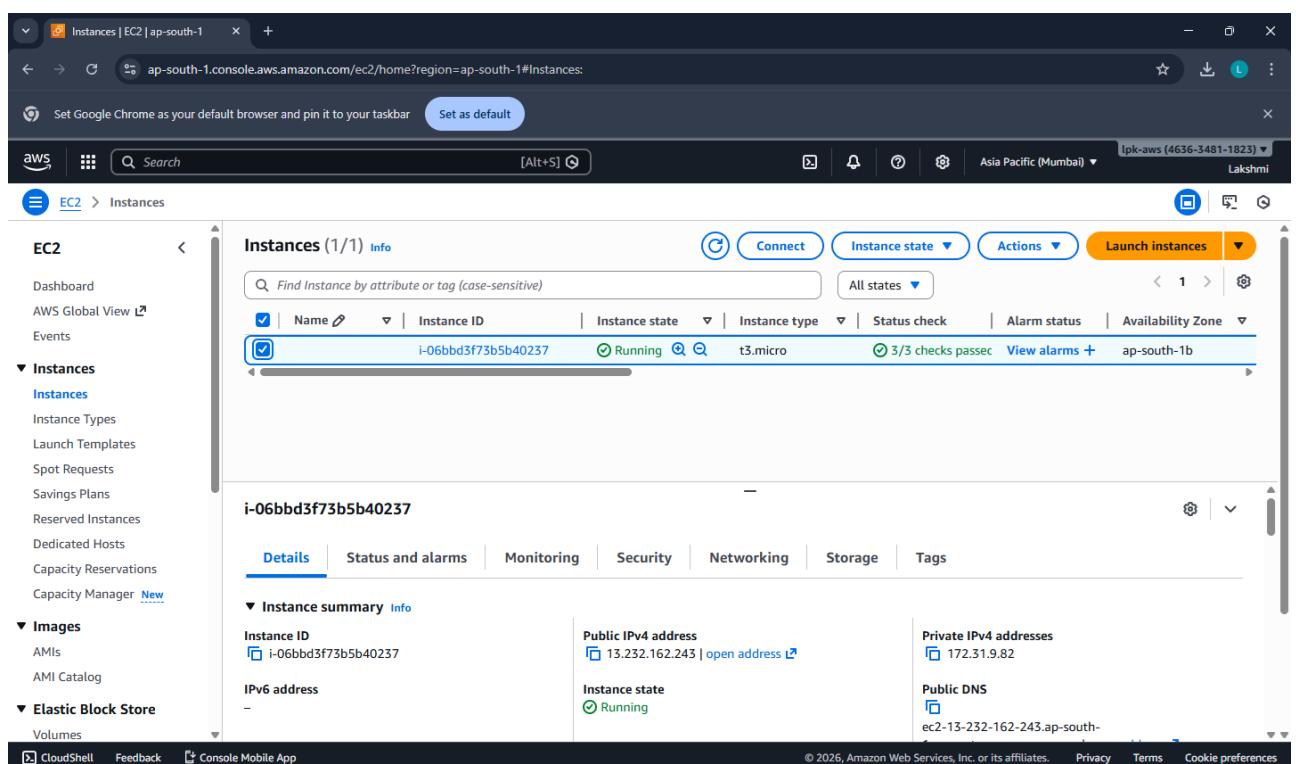
Expected Output in browser:

Apache Installed via CloudFormation UserData (Amazon Linux 2023)!



Part H — Verify in EC2 (Optional)

1. Go to **EC2 → Instances**
2. Instance should be running
3. Security group should allow:
 - SSH 22
 - HTTP 80



Troubleshooting (Common Errors)

1) Website not opening

Check:

- EC2 instance status checks are **2/2 passed**
- Security group has **port 80 open**
- Wait 2–3 minutes (UserData takes time)

2) Stack went to ROLLBACK

Go to stack → **Events tab** → check the failure reason

Most common reason:

- Wrong Key Pair selected / key pair does not exist

Part I — Clean-up

To avoid charges:

1. CloudFormation → Stacks
2. Select EC2-Apache-AL2023
3. Click **Delete**
4. Confirm

This deletes EC2 + Security Group automatically.

DATE: 17-12-25

Exercise—23: EC2 + S3 (Static Content Pulled from S3 using CloudFormation)

Create an **S3 bucket** using CloudFormation.

Upload a static HTML file (index.html) to S3.

Launch **EC2 (Amazon Linux 2023) + Apache** using CloudFormation.

EC2 will **pull index.html from S3** and host it via Apache.

Pre-requisites

- Region: **Mumbai (ap-south-1)**
- A Key Pair exists (example: pemkeypair)

PART 0 — One-time: Create Key Pair

EC2 → Key Pairs → Create key pair

- Name: pemkeypair
- Format: .pem
Download it and keep safe.

PART 1 — Stack-1: Create S3 Bucket (CloudFormation)

Step 1.1: Create S3 template file

Create a file: **stack1-s3-bucket.yaml** and paste:

AWSTemplateFormatVersion: "2010-09-09"

Description: Create an S3 bucket to store website content (index.html)

Resources:

WebsiteBucket:

Type: AWS::S3::Bucket

Outputs:

BucketName:

Description: S3 Bucket Name (use this to upload index.html)

Value: !Ref WebsiteBucket

Step 1.2: Create stack

CloudFormation → Stacks → Create stack → With new resources

- Choose an existing template
- Upload a template file → stack1-s3-bucket.yaml
- Stack name: S3-Website-Bucket
- Create stack

Step 1.3: Copy bucket name

Open stack → **Outputs** → copy BucketName

PART 2 — Upload Static Content to S3 (Manual)

Step 2.1: Create index.html on your PC

Create a file named **index.html** with this content:

```
<!DOCTYPE html>

<html>
  <head>
    <title>EC2 + S3 Demo</title>
  </head>
  <body>
    <h1>Hello! This page was pulled from S3 to EC2 automatically.</h1>
    <p>Deployed using CloudFormation + UserData</p>
  </body>
</html>
```

Step 2.2: Upload to S3 bucket

S3 → Buckets → open your bucket (from Output) → Upload

- Upload **index.html**
- Keep it at **root** (no folder)
- Upload

Now S3 has: s3://<your-bucket-name>/index.html

PART 3 — Stack-2: Launch EC2 + Apache + Pull from S3

This stack will:

- Create IAM Role (permission to read the bucket)
- Create Security Group (22, 80)
- Launch EC2 (Amazon Linux 2023)
- Install Apache
- Copy index.html from S3 to /var/www/html/index.html

Step 3.1: Create EC2 template file

Create a file: **stack2-ec2-pull-from-s3.yaml** and paste:

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Launch EC2 (Amazon Linux 2023), install Apache, and pull index.html from S3

Parameters:
KeyName:
Type: AWS::EC2::KeyPair::KeyName
Description: Select an existing EC2 Key Pair to enable SSH access

BucketName:
Type: String
Description: Enter the S3 bucket name created in Stack-1 (Output)

SubnetId:
Type: AWS::EC2::Subnet::Id
Description: Select a PUBLIC subnet (default VPC public subnet recommended)

VpcId:
Type: AWS::EC2::VPC::Id
Description: Select the VPC (default VPC recommended)

Resources:
WebServerRole:
Type: AWS::IAM::Role
Properties:
AssumeRolePolicyDocument:
Version: "2012-10-17"
Statement:
- Effect: Allow
Principal:
Service: ec2.amazonaws.com
Action: sts:AssumeRole
Policies:
- PolicyName: S3ReadOnlyForWebsiteBucket
PolicyDocument:
```

```
Version: "2012-10-17"
Statement:
  - Effect: Allow
    Action:
      - s3:GetObject
      - s3>ListBucket
    Resource:
      - !Sub "arn:aws:s3:::${BucketName}"
      - !Sub "arn:aws:s3:::${BucketName}/*"
```

```
WebServerInstanceProfile:
  Type: AWS::IAM::InstanceProfile
  Properties:
    Roles:
      - !Ref WebServerRole
```

```
WebServerSG:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Allow SSH (22) and HTTP (80)
    VpcId: !Ref VpcId
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
```

```
WebServerInstance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType: t2.micro
    KeyName: !Ref KeyName
    SubnetId: !Ref SubnetId
    SecurityGroupIds:
      - !Ref WebServerSG
    IamInstanceProfile: !Ref WebServerInstanceProfile
```

```
# Amazon Linux 2023 AMI for Mumbai (ap-south-1)
ImageId: ami-02b49a24cfb95941c
```

```
UserData:
  Fn::Base64: !Sub |
    #!/bin/bash
    dnf update -y
    dnf install -y httpd
    systemctl enable httpd
    systemctl start httpd
```

```
# pull index.html from S3 to Apache web root
aws s3 cp s3://${BucketName}/index.html /var/www/html/index.html

systemctl restart httpd
```

Outputs:

WebsiteURL:

Description: Open this URL in browser

Value: !Sub "http://\${WebServerInstance.PublicDnsName}"

Step 3.2: Create stack

CloudFormation → Stacks → Create stack

- Upload template → stack2-ec2-pull-from-s3.yaml
- Stack name: EC2-Pull-S3-Website
- Parameters:
 - **KeyName**: select your key pair (e.g., pemkeypair)
 - **BucketName**: paste bucket name from Stack-1 Output
 - **VpcId**: select **default VPC**
 - **SubnetId**: select a **PUBLIC subnet** in default VPC
(usually the subnet name shows “public” or you can pick any default subnet that gives public IP; default subnets generally work)

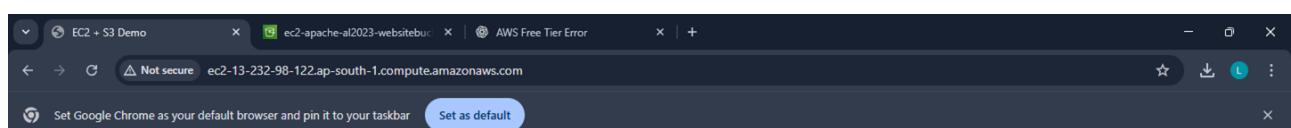
Create stack → wait for **CREATE_COMPLETE**

PART 4 — Verify Output

Stack-2 → **Outputs** → copy **WebsiteURL** → open in browser.



“Hello! This page was pulled from S3 to EC2 automatically...”



Hello! This page was pulled from S3 to EC2 automatically.

Deployed using CloudFormation + UserData

Troubleshooting (common)

1) Website not opening

- Wait 1–2 minutes (UserData takes time)
- Ensure Security Group has **HTTP 80 open**
- Ensure you selected a **public subnet** (needs internet to reach S3)

2) Stack-2 fails with S3 access error

- BucketName typed wrong
- index.html not uploaded at root of bucket

Clean-up

Delete Stack-2 first

CloudFormation → select EC2-Pull-S3-Website → Delete

Empty the S3 bucket

S3 → bucket → delete index.html (empty bucket)

Delete Stack-1

CloudFormation → select S3-Website-Bucket → Delete

Viva-ready questions

1. What is the purpose of **UserData** in EC2?
2. Why do we need an **IAM Role + Instance Profile**?
3. Why is S3 bucket not made public in this lab?
4. What happens when a CloudFormation stack is deleted?
5. Which ports are required for web hosting and SSH?

DATE: 19-12-25

Exercise—24: AWS Lambda: Input Processing, Business Logic Execution, and CloudWatch Logging

Create one Lambda function that:

1. prints a welcome message
2. reads JSON input event
3. performs business logic (grade calculation)
4. writes logs → view them in CloudWatch Logs

Yes — this is the **first combined Lambda lab exercise** (basics + input + logic + logging). Below is a

complete step-by-step procedure using AWS Console.

Create the Lambda Function

Step 1: Open Lambda

AWS Console → Search Lambda → Open AWS Lambda

Step 2: Create function

Click Create function

- Select - Author from scratch
- Function name: StudentGradeLogger
- Runtime: Python 3.11
- Architecture: x86_64 (default)

Step 3: Permissions (Execution Role)

Under "Permissions"

- Choose: Create a new role with basic Lambda permissions

This automatically gives permission to write logs to CloudWatch.

Click Create function

Add Code (Business Logic + Logs)

Step 4: Paste this code

In Code tab → lambda_function.py → paste and Deploy

```
import json

def lambda_handler(event, context):
    # Welcome message
    print("Lambda invoked successfully")

    # Read input from event
    student_name = event["StudentName"]
    marks = event["Marks"]

    # Business logic: grade calculation
    if marks >= 75:
        result = "Pass"
        grade = "A"
    else:
        result = "Fail"
        grade = "F"

    # Log output
    print("Student:", student_name)
    print("Marks:", marks)
    print("Grade:", grade)
    print("Result:", result)

    # Return response
    return {
        "statusCode": 200,
        "body": json.dumps({
            "StudentName": student_name,
            "Marks": marks,
            "Grade": grade,
            "Result": result
        })
    }
```

```
    })  
}
```

Click Deploy.

Test with Input Events (JSON)

Step 5: Create a test event (Valid case)

Go to Test (top right) → Configure test event

- Event name: ValidInput
- Paste this JSON:

```
{  
  "StudentName": "Anita",  
  "Marks": 86  
}
```

Click Save

Step 6: Run test

Click Test

Expected response (sample):

- statusCode: 200
- body will include Grade A and Result Pass

The screenshot shows the AWS Lambda function configuration page for 'StudentGradeLogger'. At the top, there are two green success notifications: one about updating the function and another about saving a test event. Below these, a log output window displays the execution log for a test event, showing the input parameters (StudentName: Anita, Marks: 86) and the resulting output (Grade: A, Result: Pass). The log also includes Lambda metadata like RequestId and Duration. At the bottom, there's a 'Test event' section where the user can create or edit a test event. The 'Test' button is highlighted in yellow. To the right, there's a sidebar with a tutorial for creating a simple web app, which includes steps to build a Lambda function that outputs a webpage and invoke it through its URL.

Step 7: Test invalid input (Missing Marks)

Create another test event:

- Event name: MissingMarks

```
{  
  "StudentName": "Rahul"  
}
```

Run Test

Expected:

- statusCode: 400
- message: Missing 'Marks'

Step 8: Test invalid marks range

Event name: InvalidMarks

```
{
  "StudentName": "Priya",
  "Marks": 150
}
```

Expected:

- statusCode: 400
- message: Marks must be 0–100

The screenshot shows the AWS Lambda function details page for 'StudentGradeLogger'. A green success message box displays two items: 'Successfully updated the function StudentGradeLogger.' and 'The test event "MissingMarks" was successfully saved.' Below this, the function configuration section shows 'Duration' (4.52 ms) and 'Resources configured' (128 MB). The 'Log output' section contains a log entry for a failed request due to an invalid mark value:

```

START RequestId: 8dc012f6-5f98-404e-b564-5454767e982f Version: $LATEST
Lambda Invoked successfully
[ERROR] KeyError: 'Marks'
Traceback (most recent call last):
  File "/var/task/lambda_function.py", line 9, in lambda_handler
    marks = event["Marks"]
END RequestId: 8dc012f6-5f98-404e-b564-5454767e982f
REPORT RequestId: 8dc012f6-5f98-404e-b564-5454767e982f Duration: 4.52 ms Billed Duration: 5 ms Memory Size: 128 MB Max Memory Used: 34 MB
  
```

View Logs in CloudWatch

Step 9: Open logs from Lambda directly

On the Lambda function page:

- Go to **Monitor** tab
- Click **View CloudWatch logs**

You will see:

- Log group: /aws/lambda/StudentGradeLogger
- Open latest **log stream**
- You should see all print() outputs:
 - “Lambda invoked successfully!”
 - “Received event...”
 - grade/result logs
 - error logs for invalid tests

Cleanup (Avoid charges, keep account clean)

Lambda itself usually costs nothing in small use, but cleanup is good practice:

1. Lambda → Functions → select StudentGradeLogger → **Delete**

2. CloudWatch → Log groups → find /aws/lambda/StudentGradeLogger → **Delete**
3. IAM role created (optional):
 - IAM → Roles → search role name created for Lambda → delete (only if not used elsewhere)

DATE: 24-12-25

Exercise-25: Mini Project: Event-Driven Notification System using AWS Lambda and Amazon SNS. Simulating real-time alerts from events using serverless computing.

- An event in JSON format is given as input to an AWS Lambda function.
- The Lambda function processes the event and generates a notification message.
- The message is published to an Amazon SNS topic.
- SNS sends the notification to the subscribed email address.
- The execution details are verified using CloudWatch Logs.

Create SNS Topic + Email Subscription

Step 1: Open SNS

AWS Console → Search SNS → Open Simple Notification Service

Step 2: Create a Topic

SNS → Topics → Create topic

- Type: Standard
- Name: NotificationTopic

Click Create topic

Step 3: Copy Topic ARN

Open the created topic → copy Topic ARN

(You will paste it into Lambda code later)

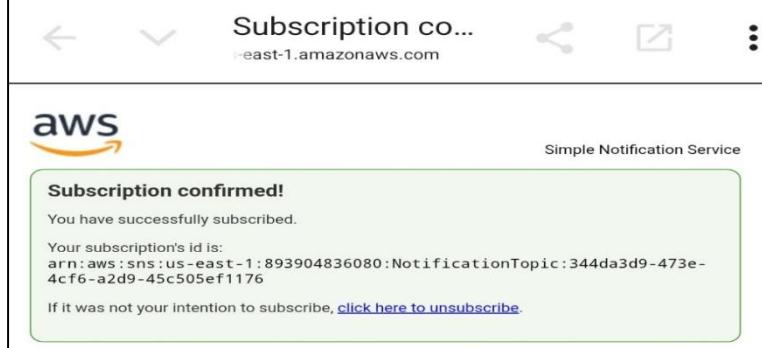
Step 4: Create an Email Subscription

Inside the same topic:

Click Create subscription

- Protocol: Email
- Endpoint: (your email id)

Click Create subscription



Step 5: Confirm Subscription

Open your email inbox → find AWS SNS confirmation mail → click Confirm subscription
→ Status in SNS should become Confirmed.

Create Lambda Function

Step 6: Open Lambda

AWS Console → Search Lambda → Open AWS Lambda

Step 7: Create function

Click Create function

- Select: Author from scratch
 - Function name: NotificationSimulator
 - Runtime: Python 3.11
 - Permissions: Create a new role with basic Lambda permissions
- Click Create function

Add Lambda Code (with SNS Publish)

Step 8: Paste code in lambda_function.py

Lambda → Code tab → open lambda_function.py → remove existing code → paste:

Replace <SNS_TOPIC_ARN> with your copied Topic ARN.

```
import json
import boto3

sns = boto3.client('sns')

TOPIC_ARN = "<SNS_TOPIC_ARN>"

def lambda_handler(event, context):
    print("Notification Simulator invoked")

    event_type = event["eventType"]
    user = event["user"]

    if event_type == "ORDER_PLACED":
        message = f"Hi {user}, your order is placed successfully."
        priority = "NORMAL"

    elif event_type == "PAYMENT_FAILED":
        message = f"Hi {user}, your payment has failed. Please retry."
        priority = "HIGH"

    elif event_type == "LOW_ATTENDANCE":
        message = f"Hi {user}, your attendance is low. Please take action."
        priority = "HIGH"

    else:
        message = f"Hi {user}, unknown event received."
        priority = "LOW"

    print("Event Type:", event_type)
    print("Message:", message)
    print("Priority:", priority)

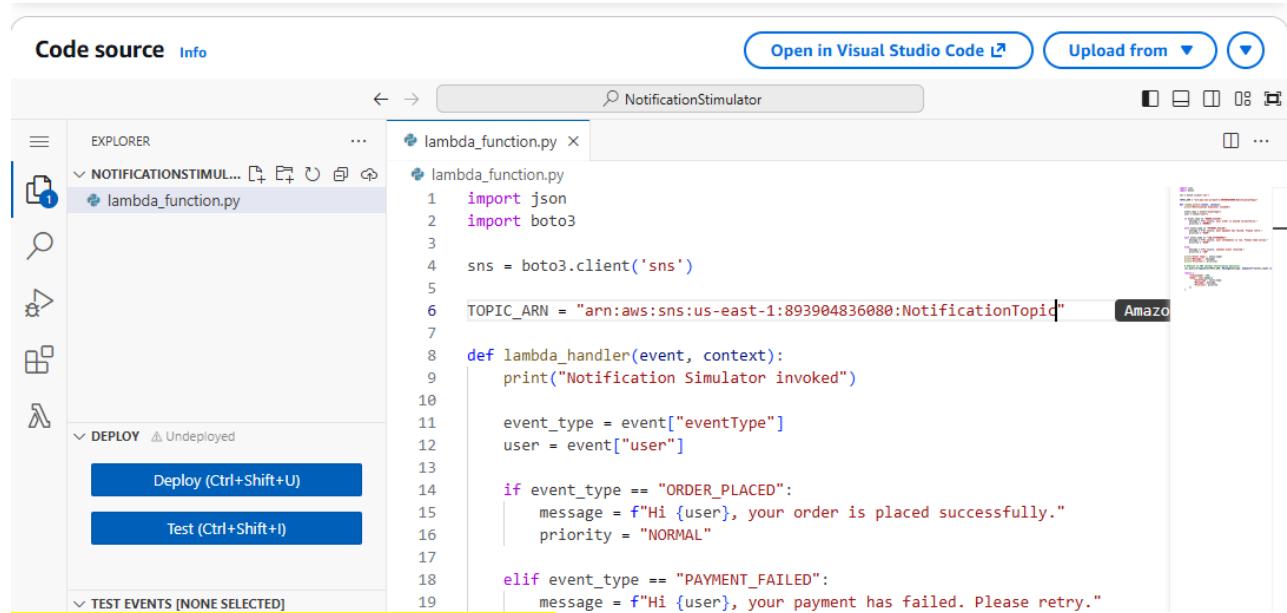
    # Publish to SNS (Actual notification delivery)
    sns.publish(TopicArn=TOPIC_ARN, Message=message, Subject=f"{event_type} [{priority}]")

    return {
```

```
}
```

Click Deploy

Wait for “Successfully updated function...”



```
lambda_function.py
1 import json
2 import boto3
3
4 sns = boto3.client('sns')
5
6 TOPIC_ARN = "arn:aws:sns:us-east-1:893904836080:NotificationTopic"
7
8 def lambda_handler(event, context):
9     print("Notification Simulator invoked")
10
11     event_type = event["eventType"]
12     user = event["user"]
13
14     if event_type == "ORDER_PLACED":
15         message = f"Hi {user}, your order is placed successfully."
16         priority = "NORMAL"
17
18     elif event_type == "PAYMENT_FAILED":
19         message = f"Hi {user}, your payment has failed. Please retry."
```

Give Lambda Permission to Publish to SNS

Step 10: Open Lambda Execution Role

Lambda → Configuration → Permissions

Click the Role name (execution role link)

Step 11: Attach SNS Publish Policy

IAM Role page → Add permissions → Attach policies

Attach: AmazonSNSFullAccess

Click Add permissions

Now Lambda can publish to SNS.

Test the Mini Project

Step 12: Create a test event

Lambda → Test tab → Create new test event

Event name: PaymentFailed

Paste:

```
{
    "eventType": "PAYMENT_FAILED",
    "user": "Anita"
}
```

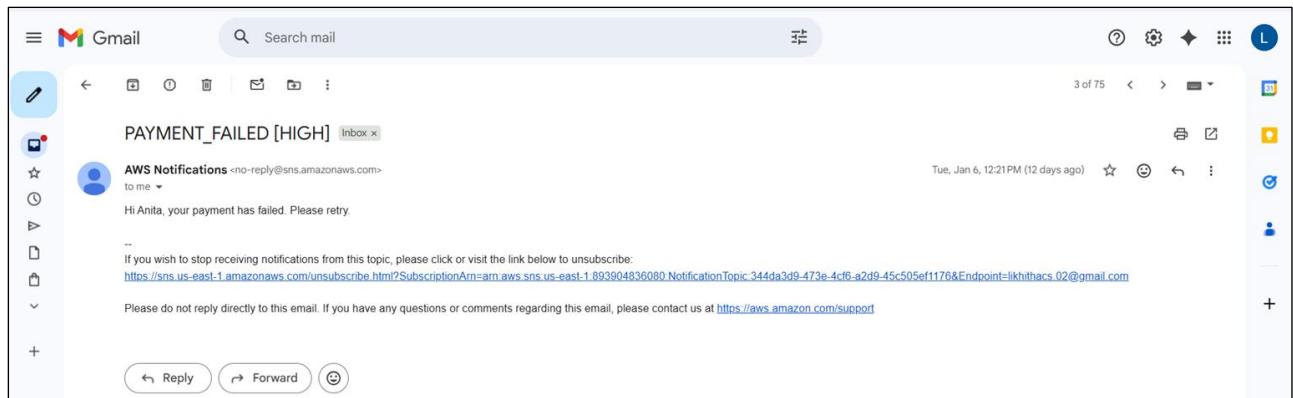
Click Save

Step 13: Run Test

Click Test

Expected:

- Execution: **Succeeded**
- Email should arrive within a few seconds:
Subject like: PAYMENT_FAILED [HIGH]
Message: “Hi Anita, your payment has failed. Please retry.”



Verify CloudWatch Logs

Step 14: View logs

Lambda → Monitor → View CloudWatch logs

Open latest log stream.

Verify these prints exist:

- Notification Simulator invoked
- Event Type:
- Message:
- Priority:

Log events			Actions ▾	Start tailing	Create metric filter		
Filter events - press enter to search			1m	1h	Local timezone ▾	Display ▾	⚙️
▶	Timestamp	Message					
		No older events at this moment. Retry					
▶	2026-01-06T12:21:42.316+05:30	INIT_START Runtime Version: python:3.11.v109 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:49f733259c7ce7e0dee..					
▶	2026-01-06T12:21:42.752+05:30	START RequestId: 28d7ec8b-b40b-47d0-a2e8-e8c2030ca088 Version: \$LATEST					
▶	2026-01-06T12:21:42.753+05:30	Notification Simulator invoked					
▶	2026-01-06T12:21:42.753+05:30	Event Type: PAYMENT FAILED					
▶	2026-01-06T12:21:42.753+05:30	Message: Hi Anita, your payment has failed. Please retry.					
▶	2026-01-06T12:21:42.753+05:30	Priority: HIGH					
▶	2026-01-06T12:21:43.015+05:30	END RequestId: 28d7ec8b-b40b-47d0-a2e8-e8c2030ca088					
▶	2026-01-06T12:21:43.015+05:30	REPORT RequestId: 28d7ec8b-b40b-47d0-a2e8-e8c2030ca088 Duration: 261.14 ms Billed Duration: 694 ms Memory Size: 128 MB ..					
		No newer events at this moment. Auto retry paused. Resume					

Expected Outputs

1. Lambda test status: Succeeded
2. Email received from SNS with correct message
3. CloudWatch logs showing event type and generated message

Cleanup

1. Delete Lambda function: NotificationSimulator
2. SNS → delete topic NotificationTopic (subscriptions removed automatically)
3. CloudWatch → delete log group for Lambda
4. (Optional) Delete IAM role if not used elsewhere

DATE: 26-12-25

Exercise–26: Analyze a CSV File in S3 Using Amazon Athena (No Server)

Upload a small CSV to **S3**, then use Athena to run SQL queries like count, average, max, group by.

Part A — Create Sample CSV (on your PC)

1. Open **Notepad**
2. Paste this data and save as: **students.csv**

```
StudentID,Name,Dept,Marks,Result
```

```
101,Anita,MCA,85,Pass  
102,Ravi,MCA,72,Pass  
103,Meera,MBA,64,Pass  
104,John,MCA,35,Fail  
105,Sneha,MBA,91,Pass  
106,Arun,MCA,49,Fail  
107,Kiran,MBA,58,Pass  
108,Divya,MCA,77,Pass
```

Part B — Create S3 Bucket and Upload CSV

1. Go to **AWS Console → S3**
2. Click **Create bucket**
3. Bucket name: athena-lab-<yourname>-<number> (must be globally unique)
4. Keep defaults → Click **Create bucket**
5. Open the bucket → Click **Upload**
6. Upload **students.csv**
7. Click **Upload**

Now your CSV is in S3.

The screenshot shows the AWS S3 'Upload: status' page. At the top, there's a summary section with a note: 'After you navigate away from this page, the following information is no longer available.' Below this is a 'Summary' section with two main status boxes: 'Succeeded' (1 file, 215.0 B (100.00%)) and 'Failed' (0 files, 0 B (0%)). Underneath is a 'Files and folders' tab, which displays a table with one row: 'students.csv' (text/csv, 215.0 B, Succeeded). The table has columns for Name, Folder, Type, Size, Status, and Error.

Part C — Open Athena and Set Query Result Location (IMPORTANT)

1. Go to AWS Console → Amazon Athena
2. If it shows “Get started” / “Query editor”, open it.
3. It will ask to set a **Query result location**
4. Click the link/button like **Settings / Manage / Edit**
5. Set query result location to something like:
 - o s3://your-bucket-name/athena-results/
6. Click **Save**

This is required so Athena can store query outputs.

The screenshot shows the Amazon SageMaker Unified Studio 'Query in Amazon SageMaker Unified Studio' interface. On the left, there's a 'Data' panel with 'Data source' set to 'AwsDataCatalog'. The main area is titled 'Query 1' and contains a single line of code: '1'. The bottom of the screen shows a standard Windows taskbar with various icons.

Part D — Create a Database in Athena

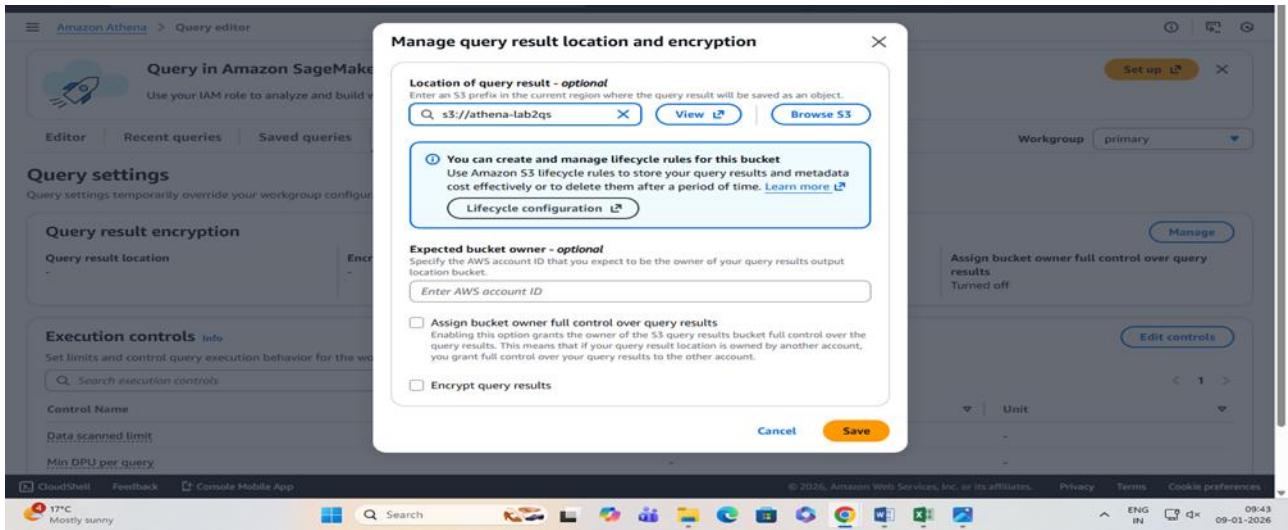
In Athena query editor, run:

`CREATE DATABASE labdb;`

Then on the left side, select:

- **Data source:** AwsDataCatalog

- **Database:** labdb



Part E — Create a Table for the CSV in S3

Run this (replace YOUR_BUCKET_NAME with your actual bucket name):

```
CREATE EXTERNAL TABLE IF NOT EXISTS students (
```

```
    StudentID int,
```

```
    Name string,
```

```
    Dept string,
```

```
    Marks int,
```

```
    Result string
```

```
)
```

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
```

```
WITH SERDEPROPERTIES (
```

```
    'separatorChar' = ',',
```

```
    'quoteChar' = "",
```

```
    'escapeChar' = '\\'
```

```
)
```

```
STORED AS TEXTFILE
```

```
LOCATION 's3://YOUR_BUCKET_NAME/'
```

```
TBLPROPERTIES ('skip.header.line.count'=1);
```

This tells Athena: "My CSV is in S3, treat it like a table."

Part F — Run Simple Analysis Queries (Core Part)

1) View all rows

SELECT * FROM students;

Expected: 8 records.

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Tables and views' sidebar lists a single table 'students'. The main area contains the SQL query: 'SELECT * FROM students;'. Below the query, the 'Run' button is highlighted. To the right, the 'Query results' tab is selected, showing the output of the query. The results table has columns: #, studentid, name, dept, marks, and result. There are 8 rows of data. At the bottom of the results table, it says 'Time in queue: 54 ms Run time: 375 ms Data scanned: 0.21 KB'. The bottom of the screen shows the Windows taskbar with various icons.

#	studentid	name	dept	marks	result
1	101	Anita	MCA	85	Pass
2	102	Ravi	MCA	72	Pass
3	103	Meera	MBA	64	Pass
4	104	John	MCA	35	Fail
5	105	Sneha	MBA	91	Pass
6	106	Arun	MCA	49	Fail
7	107	Kiran	MBA	58	Pass
8	108	Divya	MCA	77	Pass

2) Total students

SELECT COUNT(*) AS total_students FROM students;

Expected: 8

3) Pass vs Fail count

SELECT Result, COUNT(*) AS cnt

FROM students

GROUP BY Result;

Expected (based on data): Pass = 6, Fail = 2

4) Average marks overall

SELECT AVG(Marks) AS avg_marks

FROM students;

5) Department-wise average marks

SELECT Dept, AVG(Marks) AS avg_marks

FROM students

GROUP BY Dept;

6) Top scorer

SELECT Name, Dept, Marks

FROM students

ORDER BY Marks DESC

LIMIT 1;

Expected: Sneha (91)

7) List failed students

```
SELECT StudentID, Name, Dept, Marks
```

```
FROM students
```

```
WHERE Result = 'Fail';
```

Expected: John, Arun

The screenshot shows the Amazon Athena Query Editor interface. On the left, the sidebar displays the database 'labdb' and a table 'students'. The main area contains a SQL query: 'SELECT StudentID, Name, Dept, Marks FROM students WHERE Result = 'Fail';'. Below the query, the 'Query results' tab is selected, showing a table with two rows of data: StudentID 104 (Name: John, Dept: MCA, Marks: 35) and StudentID 106 (Name: Arun, Dept: MCA, Marks: 49). The results are completed, with a run time of 438 ms and data scanned of 1.11 KB.

Cleanup (to avoid any charges)

1. In Athena, you can keep DB/table (no cost by itself), but clean S3:
2. Go to **S3 bucket**
3. Delete:
 - o students.csv
 - o athena-results/ folder contents (query outputs)
4. Delete the bucket (must be empty to delete)

Cost Note (Simple)

- **S3 storage:** tiny (almost negligible for this)
- **Athena:** charges based on **data scanned**; with this tiny CSV it's usually minimal, but **always delete outputs and bucket** after lab.