



## **DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS**

### **LABORATORY MANUAL**

**III Semester**

**Batch:2024-26**

**Name: LIKHITHA B K**

**USN: 1MS24MC049**

**Course: Cloud Computing**

**Course code: 24MCASS3**

**Course Credits: 0 : 1: 2**

---

**RAMAIAH INSTITUTE OF TECHNOLOGY**

(Autonomous Institute, Affiliated to VTU)

Accredited by National Board of Accreditation & NAAC with 'A+' Grade, MSR  
Nagar, MSRIT Post, Bangalore-560054  
[www.msrit.edu](http://www.msrit.edu)

## Table of Lab Programs

| <b>Sl.<br/>No.</b> | <b>Programs/Exercise/Topic.</b>   |
|--------------------|---|
| 1.                 | AWS Account Setup and Configuration. AWS Console Overview. Enable MFA. Create AWS budget alert.   |
| 2.                 | AWS Identity Access Management (IAM) User and Group creation. Enable AWS IAM MFA. Create an AWS Account Alias (for Alternate Sign-in URL)   |
| 3.                 | Amazon S3 – Introduction, Bucket Creation and upload objects (files).   |
| 4.                 | Amazon S3 – Static Website Hosting (Multi-Page website), Versioning, Cross-Region Replication rule.   |
| 5.                 | Overview of EC2.<br>To Launch a Windows EC2 Instance and Connect via RDP Client.  |
| 6.                 | Launch a Linux EC2 Instance and Connect using SSH through PowerShell and PuTTY on Windows machine.<br>Launch a Linux EC2 Instance and Connect using SSH through Linux terminal on Linux machine.<br>Launch a Linux EC2 Instance and Connect through EC2 connect option on the console.          |
| 7.                 | Hosting a static website on EC2 instance:<br>- Manual Installation of Apache (httpd) Web Server on EC2 for Static Website Hosting.<br>- Launching an EC2 Instance with User Data Script to Automatically Install Apache and Host a Static Website.  |
| 8.                 | Custom AMI<br>- Create a Custom AMI from a Working EC2 Instance.<br>- Launch an EC2 Instance using a Custom AMI.<br>- Delete the custom AMI [Deregister and delete snapshot]  |
| 9.                 | <b>Mini-Project:</b> Hosting a Multi-Page Website using EC2 and S3<br>Submission: 11-11-2025, Marks: 5  |
| 10.                | Creation and Configuration of a Custom AWS <b>Virtual Private Cloud (VPC)</b> .<br>[Including Public and Private Subnets, Internet Gateway, NAT Gateway, Route Tables]  |
| 11.                | Launching and Configuring EC2 Instances for <b>Web Server in Public Subnet</b> and <b>Database Server in Private Subnet</b> Using NAT Gateway for Outbound Access.  |
| 12.                | Deploying and Testing a <b>Load-Balanced Web Application</b><br>By Creating a Web Server, Building a Custom AMI, Configuring a Target Group, Launching an Application Load Balancer (ALB), Registering the Instance in the Target Group, and Testing the ALB DNS.                               |
| 13.                | <b>Stress Testing</b> a Linux EC2 Instance (CPU Load Test)  |
| 14.                | <b>Mini Project:</b><br>Deploying a Load-Balanced Web Application using <b>Application Load Balancer (ALB)</b> , <b>Auto Scaling Group (ASG)</b> , Custom AMI, and Target Group on AWS with CloudWatch Alarms & Stress Testing for Auto Scaling Validation.<br>Submission: 22-11-2025, Marks: 5 |

|     |  |
|-----|--|
| 15. | Hosting a WordPress Content Management Website on Amazon <b>Lightsail</b> . (only demo)  |
| 16. | Building a Basic Python Flask Web Application — Fundamentals of Web Requests & Form Handling ( <b>Pre-Requisite for AWS RDS Connectivity Lab</b> ) |
| 17. | Flask Web App with Registration & Login Using AWS <b>RDS</b> (MySQL)<br>(Full Deployment on AWS EC2 + AWS RDS)                                     |
| 18. | Creating and Operating a NoSQL Key-Value Database using Amazon <b>DynamoDB</b>   |
| 19. | <b>ElastiCache</b> (Redis) as an In-Memory Cache   |
| 20. | Deploy a simple Flask application on AWS <b>Elastic Beanstalk</b> and verify it using the public URL.  |
| 21. | Deploy a Flask App on AWS <b>Elastic Beanstalk</b> and Perform CRUD on <b>DynamoDB</b> (Using AWS SDK/boto3)                                       |
| 22. | <b>CloudFormation</b> – Launch EC2 (Amazon Linux 2023) + Install Apache using UserData   |
| 23. | Static Content Pulled from S3 using <b>CloudFormation</b> [EC2 + S3]   |
| 24. | AWS <b>Lambda</b> : Input Processing, Business Logic Execution, and CloudWatch Logging.  |
| 25. | Event-Driven Notification System using <b>AWS Lambda</b> and Amazon <b>SNS</b>   |
| 26. | Analyze data in a CSV File stored in S3 Using <b>Amazon Athena</b> (No Server)   |
| 27. | Smart Sensor Monitoring System using <b>AWS IoT Core</b> . Cloud-Based IoT Data Ingestion and Processing using AWS                                 |
| 28. | Accelerate an S3 Static Website Using <b>Amazon CloudFront</b> (CDN)   |
|     | <b>Mini Project</b> – Global Static Website Delivery using S3 + CloudFront with Cache Update   |
|     | <b>Mini Project</b> – Deploying a Containerized App on AWS using <b>Amazon EKS</b>   |

**Date: 8-10-2025**

**Exercise 1:** AWS Account Setup and Configuration. AWS Console Overview. Enable MFA. Create AWS budget alert.

AWS Console overview / AWS Home page/ AWS Dash board

**Widgets** – default view/ Add or remove widgets - small panels on the dashboard showing metrics or shortcuts; users can add or remove them as needed.

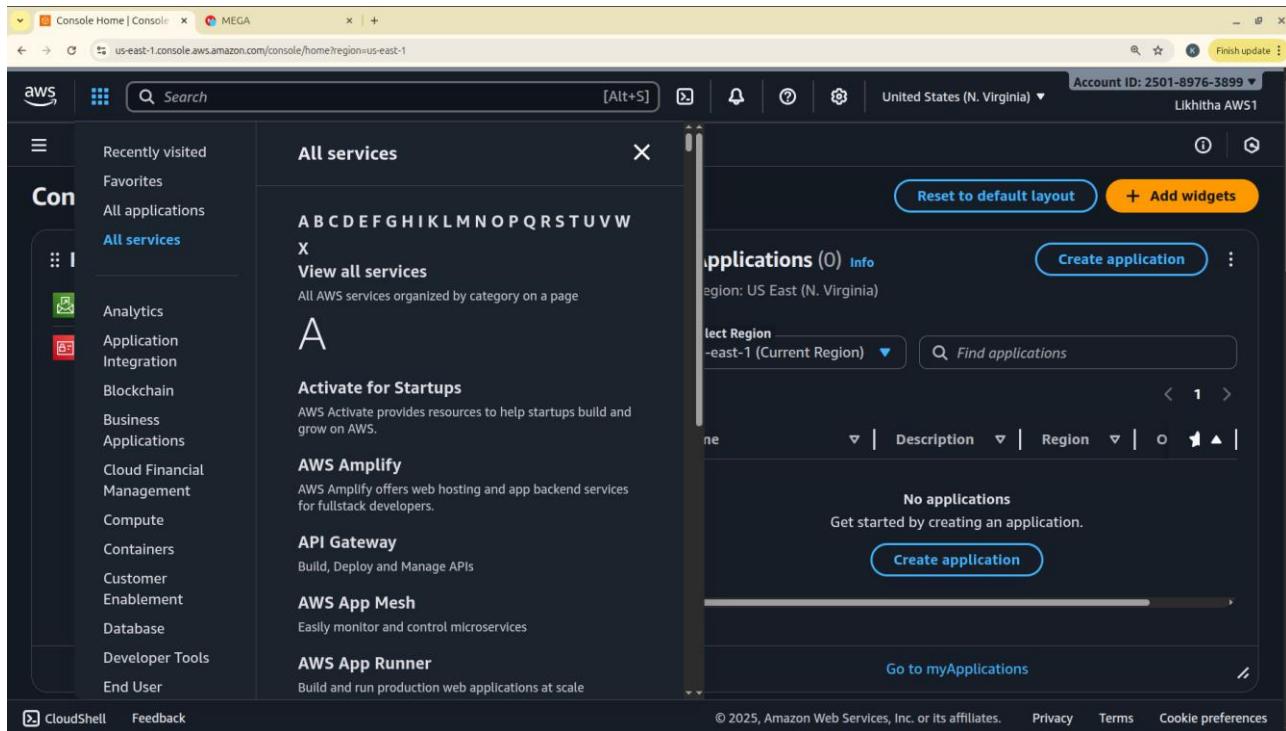
The screenshot shows the AWS Console Home page. At the top, there's a search bar and navigation links. On the left, a sidebar lists 'Recently visited' services: Billing and Cost Management and IAM. Below this is a 'View all services' link. The main content area is titled 'Console Home'. It features a 'Reset to default layout' button and an '+ Add widgets' button. The 'Applications' section is displayed, showing '0' applications. It includes a 'Create application' button, a 'Select Region' dropdown set to 'us-east-1 (Current Region)', and a 'Find applications' search bar. A message says 'No applications' and 'Get started by creating an application.' At the bottom, there are links for 'CloudShell', 'Feedback', 'Go to myApplications', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates.' followed by 'Privacy', 'Terms', and 'Cookie preferences'.

**Region** – Specifies the geographical data center location where your AWS resources are deployed.

The screenshot shows the AWS Console Home page. At the top right, the account ID is displayed as 2501-8976-3899 and the user name as Likhitha AWS1. Below the header, there is a search bar and a 'Recently visited' section with links to 'Billing and Cost Management' and 'IAM'. On the right side, a large dropdown menu is open, listing AWS Regions categorized by continent:

| Name          | Region         |
|---------------|----------------|
| United States |                |
| N. Virginia   | us-east-1      |
| Ohio          | us-east-2      |
| N. California | us-west-1      |
| Oregon        | us-west-2      |
| Asia Pacific  |                |
| Mumbai        | ap-south-1     |
| Osaka         | ap-northeast-3 |
| Seoul         | ap-northeast-2 |
| Singapore     | ap-southeast-1 |
| Sydney        | ap-southeast-2 |
| Tokyo         | ap-northeast-1 |
| Canada        |                |
| Central       | ca-central-1   |
| Europe        |                |

At the bottom of the dropdown, there are links for 'Manage Regions' and 'Manage Local Zones'. The 'Go to myApplications' link is also visible at the bottom right of the dropdown.

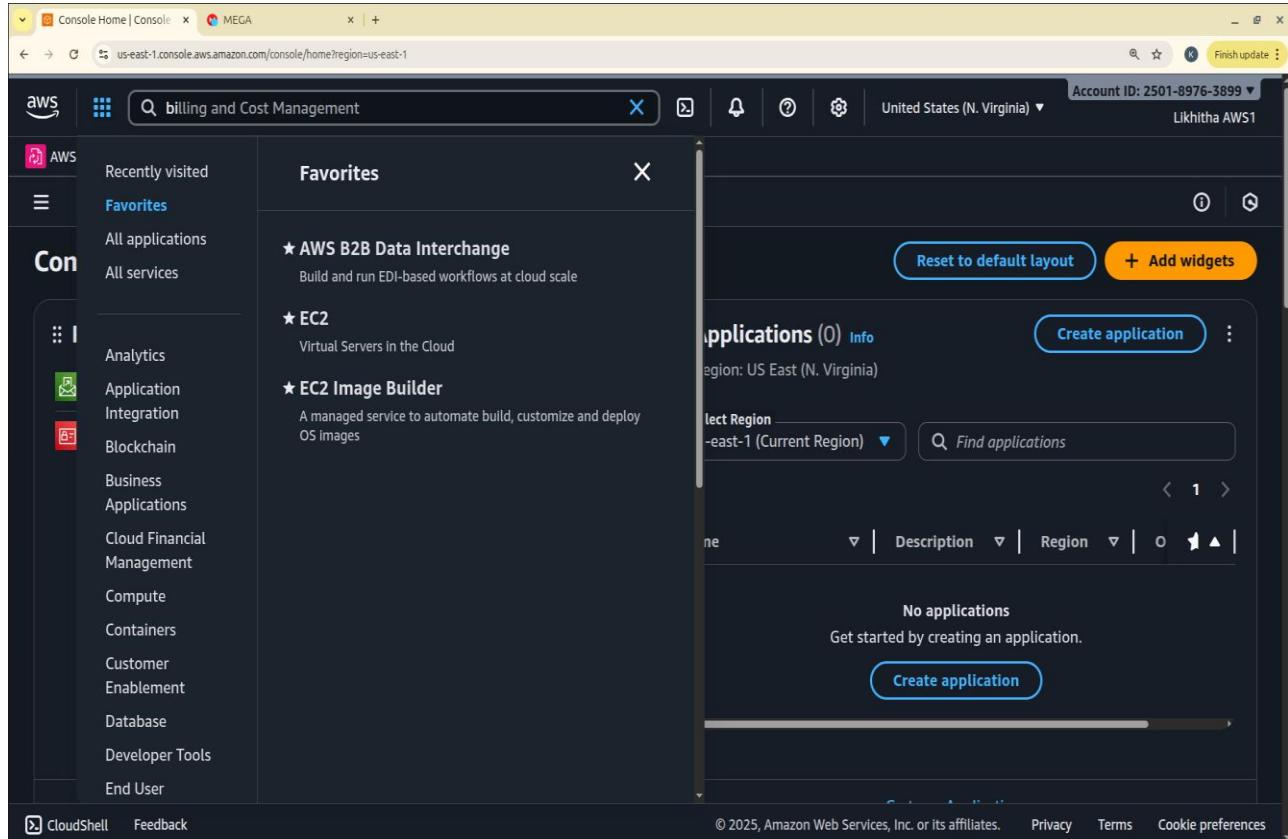


**Services** – A categorized list of all AWS offerings such as Compute, Storage, Database, etc.

**Search bar** – A quick-access bar to search and pin frequently used services for faster access.



pin the most used services to console by clicking on star next to the service name.



## Enable MFA

### Notes

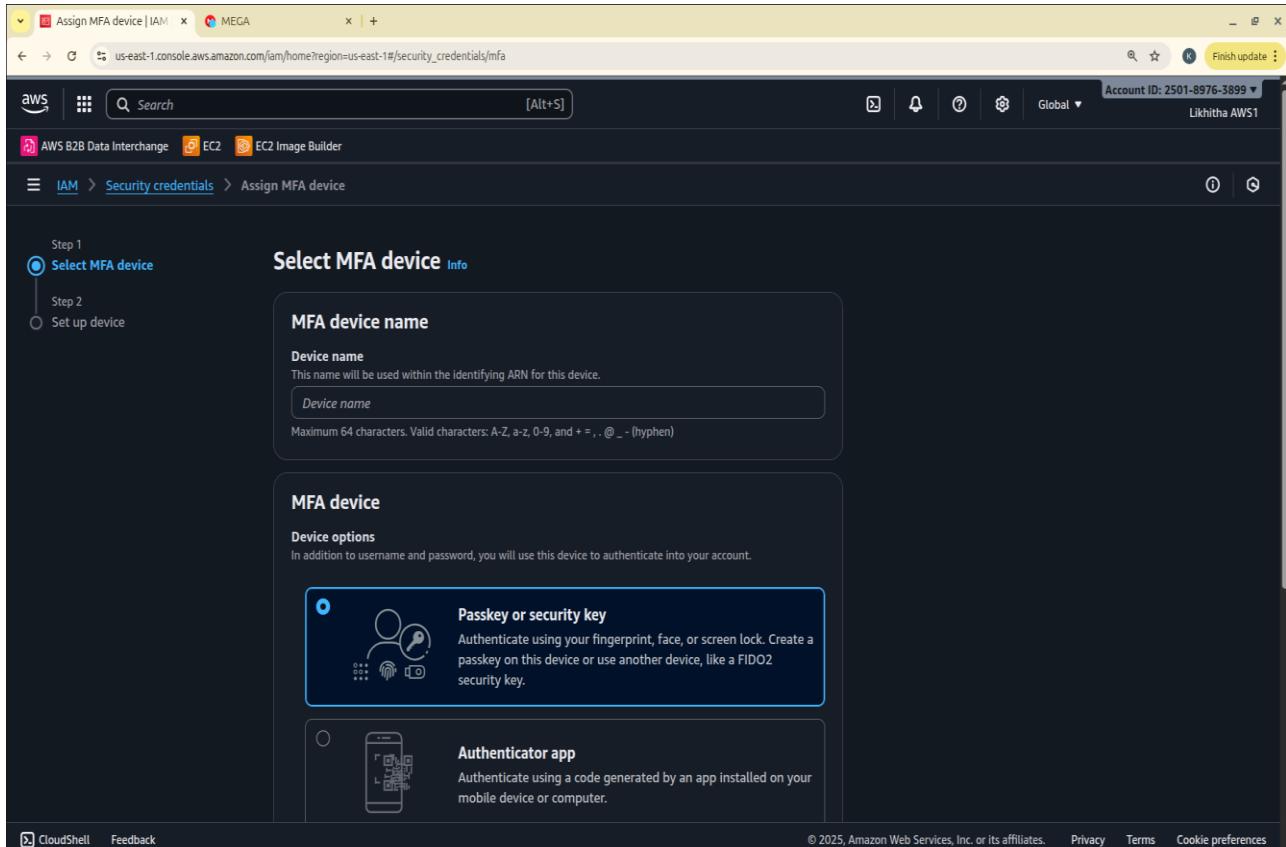
- Make sure your phone is unlocked, Bluetooth is on, and it uses a screen lock (fingerprint/PIN).

### **Option 1: Add a Passkey for Easier Login**

#### Step1: Go to Security Credentials

- **Sign in to AWS console.**
- Go to **your username** → **Security credentials**.
- Under **Multi-factor authentication (MFA)** click “**Assign MFA device.**”
- Choose “**Passkeys and security keys**” → **Next**.
- On the next screen choose “**Phone or tablet**”.
- AWS will show a **browser pop-up** asking to use a device.
  1. Select **your phone** (or “Use another device” if it prompts).
- Look at your phone — you should get a “**Use passkey**” or **biometric prompt**.

- Approve using **fingerprint or phone PIN**.
- Back in AWS, click **Finish**. The passkey is now your MFA method.



Next time you sign in, just choose “**Sign in with a passkey**” → **approve on phone**.

## Step 2: Test the Login

- Sign out of AWS.
- Go to the login page.
- Choose “Sign in with a passkey” → Select your phone.
- Approve the prompt on your phone — you should be signed in without any MFA codes.

## Option 2: Authenticator App

This uses a 6-digit code from Google Authenticator / Authy / Microsoft Authenticator.

1. In **Security credentials**, click “**Assign MFA device**.”
2. Select “**Authenticator app**” → **Next**.
3. A QR code appears.
4. Open your authenticator app on your phone → **Add account** → **Scan QR code**.

5. The app shows a 6-digit code.
6. Enter that code back in AWS → **Assign MFA**. You'll use the 6-digit code from the app each time you log in.

### Create AWS budget alert

Allows to create a simple budget and to send alarms to registered email.

Example: if you are close to or exceeding your designated budget.

By setting a budget you can monitor budget threshold from the start.

Creating a budget

Step 1:

In the search bar type budgets and under the search results:

Select ‘Budgets’ from the Features group, which is essentially a feature of Billing and Cost Management service.

Step 2: After the ‘Budgets’ page loads

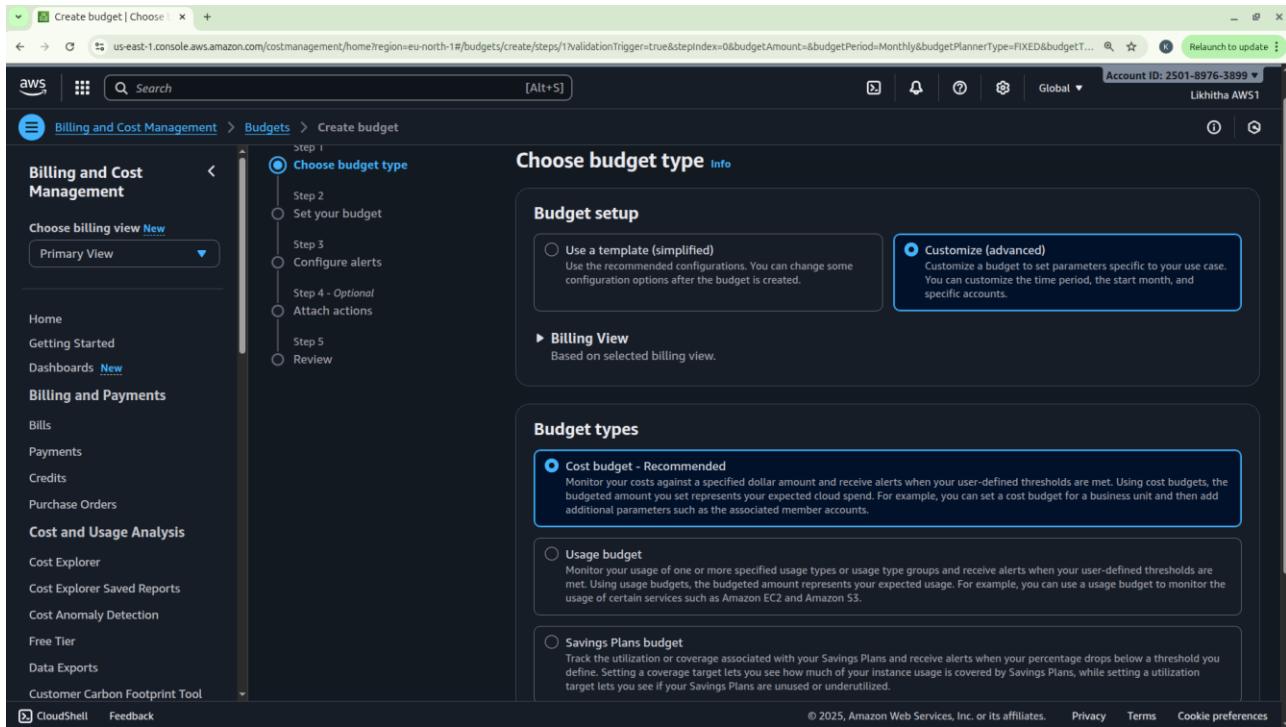
Click on **Create Budget** button

| Name     | Thresholds | Health status | Billing View | Budget  | Amount... | Forecast |
|----------|------------|---------------|--------------|---------|-----------|----------|
| MyBudget | OK         | Healthy       | Primary View | \$20.00 | \$0.00    |          |

Under Budget setup select ‘Customize’ option

Under Budget types select ‘Cost budget’ option

Click on **Next** button.



Step 3: Set your budget page loads and fill in the details: MyBudget, Monthly, Recurring budget, set Month and Year, select Fixed – Budgeting method - Enter your budgeted amount – 20.00, select ‘All AWS services’ Scope options. Advanced options – leave it on default.

Click on **Next** button.

The screenshot shows the 'Create budget' wizard in the AWS Cost Management console. The current step is 'Advanced options'. The left sidebar shows the navigation path: Billing and Cost Management > Budgets > Create budget. The main content area includes sections for tracking cost from specific services, advanced options (with a note about charge types), aggregate costs by unblended costs, and optional tags. A 'Budget preview' section on the right shows a chart of actual costs from October 2024 to July 2025, and an 'Alerts' section indicates no alerts are configured.

#### Step 4: Configure alerts

Click on **Add an alert threshold**

Under Set alert threshold

Set Threshold: 80 and Trigger: Actual

Email recipients: enter your email id.

Click on **Next** button.

The screenshot shows the 'Create budget' wizard in the AWS Billing and Cost Management console. The left sidebar shows navigation options like Home, Getting Started, Dashboards, Billing and Payments, Cost and Usage Analysis, and Customer Carbon Footprint Tool. The main content area is titled 'Create budget' and shows the configuration for a new budget:

- Threshold:** When should this alert be triggered? Set to 80% of budgeted amount.
- Trigger:** How should this alert be triggered? Set to Actual.
- Email recipients:** An email address 'likhitabkgowda75@gmail.com' is listed as a recipient.
- Notification preferences:** Options for Amazon SNS Alerts and AWS Chatbot Alerts are shown.
- Budget preview:** A line chart showing actual costs from Oct 2024 to Jul 2025 compared to the budget.
- Alerts:** No alerts are currently configured.

At the bottom, there are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' being highlighted in yellow.

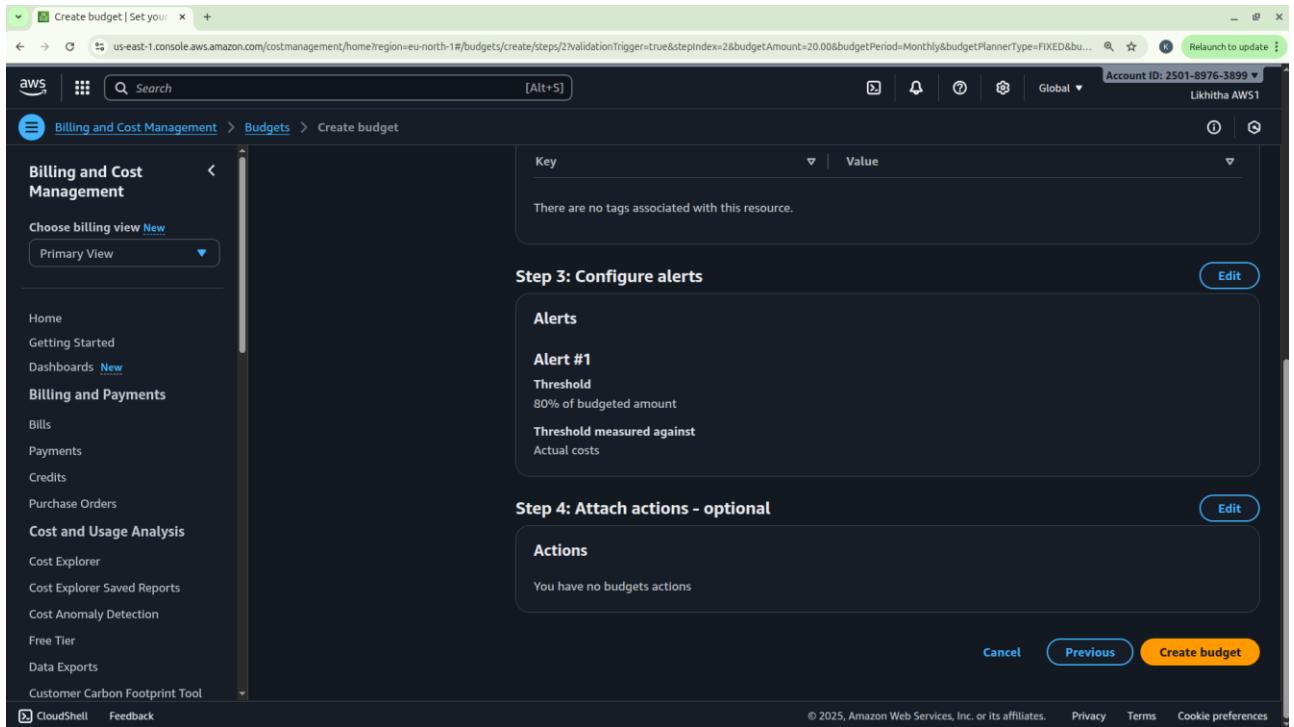
Step 5: Under Attach actions – leave it on default

Click on **Next** button.

The screenshot shows the 'Create budget' wizard in the AWS Billing and Cost Management console. The current step is 'Using budgets actions'. The left sidebar shows navigation options like Home, Getting Started, Dashboards, Billing and Payments, Cost and Usage Analysis, and Customer Carbon Footprint Tool. The main content area has two sections: 'Using budgets actions' and 'Alert #1 (0 actions attached)'. The 'Using budgets actions' section includes a diagram of three boxes connected by arrows, a 'What is a budget action?' section, and a 'How to get started?' section. The 'Alert #1' section shows a threshold of 80% measured against Actual Costs, with email recipients set to likhitabkgowda75@gmail.com and Amazon SNS Not configured. An 'Add action' button is present. At the bottom are 'Cancel', 'Previous', and 'Next' buttons. To the right, there's a 'Budget preview' section showing a line chart from Oct 2024 to Jul 2025, an 'Alerts' section with a message about actual cost being above 80%, and links to AWS Cost Explorer and CloudShell.

Step 6: In review page – check and review all the options are set to what is desired.

Click on **Create budget** button.



Now, the budget has been created.

The screenshot shows the AWS Budgets Overview page. The left sidebar lists various services under 'Billing and Cost Management'. The main area displays a table titled 'Budgets (1)'. The table has columns for Name, Thresholds, Health status, Billing View, Budget, Amount ..., and Forecast. One row is shown for 'MyBudget', which is marked as 'OK' and 'Healthy'.

| Name     | Thresholds | Health status | Billing View | Budget  | Amount ... | Forecast |
|----------|------------|---------------|--------------|---------|------------|----------|
| MyBudget | OK         | Healthy       | Primary View | \$20.00 | \$0.00     |          |

**Date: 9-10-2025**

**Exercise 2:** AWS Identity Access Management (IAM) User and Group creation. Enable AWS IAM MFA. Create an AWS Account Alias (for Alternate Sign-in URL)

AWS Identity and Access Management (IAM) is a security service that helps you control who can access your AWS resources and what actions they can perform. It is a global AWS service. It allows you to securely manage users, groups, roles, and permissions in your AWS account.

| Concept   | Description  |
|-----------|--|
| Root User | The account owner who created the AWS account. Has full access and should be used only for account setup.                                      |
| Group     | A collection of IAM users that share the same permissions. For example, a “Developers” group or “Students” group.                              |
| User      | A person or application that interacts with AWS (e.g., student1, admin, developer). Each user has its own username, password, and access keys. |
| Policy    | A JSON document that defines what actions are allowed or denied (e.g., “allow S3 read access”).  |
| Role      | A set of permissions that can be temporarily assumed by a user, service, or application — often used by EC2 instances or Lambda functions.     |

### **STEPS for AWS IAM User Group Creation**

#### **Step 1: Sign in to AWS Console**

Log in to your AWS Management Console using an administrator account.  
From the Services menu, search for IAM and open it.

#### **Step 2: Open Groups Section**

- In the IAM dashboard, look at the left sidebar.
- Click on “User groups” → then click “Create group”.

#### **Step 3: Name the Group**

- Enter a Group name (example: Developers, Admins, Students, etc.).
- Group names must be unique within your account.

#### **Step 4: Attach Permissions Policies**

- You can attach IAM policies to define what members of the group can do.  
Select the following:
  - AdministratorAccess → Full access to all services.
  - IAMUserChangePassword

#### **Step 5: Review and Create**

- Review all details (group name + permissions).
- Click “Create group” to finalize.

#### **Group Successfully Created**

- The new group now appears in the IAM dashboard.

- Any user added to this group automatically inherits all permissions attached to the group.

The screenshot shows the AWS IAM User Groups page. On the left, there's a navigation sidebar with 'Identity and Access Management (IAM)' selected. Under 'Access management', 'User groups' is also selected. The main area displays a table titled 'User groups (1)'. The table has columns for Group name, Users, Permissions, and Creation time. One group, 'admin', is listed with 1 user, 'Defined' permissions, and created '12 days ago'.

| Group name | Users | Permissions | Creation time |
|------------|-------|-------------|---------------|
| admin      | 1     | Defined     | 12 days ago   |

## **STEPS for AWS IAM User Creation**

### **Step 1: Sign in to AWS Management Console**

- Login to the AWS Management Console using your root user credentials.
- In the search bar, type IAM and open IAM service.

### **Step 2: Navigate to Users Section**

- In the IAM dashboard's left panel (info panel), click on Users.
- Then click on the “Add users” button.

### **Step 3: Set User Details**

- Enter a user name.
- Checkbox – ‘Provide user access to the AWS Management Console’.
- Select ‘I want to create an IAM user’ option

### **Step 4: Set Permissions**

You can grant permissions to the new user in three ways:

- Add user to group** – Assign predefined permission groups.
- Copy permissions from an existing user.
- Attach existing policies directly.

Recommended: Use IAM **groups** to manage permissions easily.

### **Step 5: Set User Details and Tags (Optional)**

- Add tags like Department: MCA or Role: Faculty/Student for easy identification.

- Click Next to review.

### Step 6: Review and Create User

- Review all settings (user name, permissions, tags).
- Click Create user.

### Step 7: Save Credentials

- After creation, AWS displays:
  - Console sign-in URL
  - Username
  - Password

**NOTE:** Download or copy these credentials immediately — they cannot be retrieved later.

### Step 8: Test the User Login

- Visit the IAM login URL provided (unique for your AWS account).
- Log in with the newly created username and password.
- Verify access and permissions.

The screenshot shows the AWS IAM Users page. The left sidebar navigation includes 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'Users' selected), 'Roles', 'Policies', 'Identity providers', 'Account settings', 'Root access management', 'Access reports' (with 'Access Analyzer' selected), and 'Resource analysis New'. The main content area is titled 'Users (1) Info' and contains a table with one row for 'Likhitha'. The table columns are 'User name' (Likhitha), 'Path' (/), 'Groups' (1), 'Last activity' (Pass...), 'MFA' (none), and 'Password' (12 day). There are 'Delete' and 'Create user' buttons at the top right of the table.

### Enable AWS IAM MFA

#### Step 1: Sign in to AWS Console as root user

#### Step 2: Open IAM Dashboard

- In the search bar, type IAM and select IAM (Identity and Access Management).

- From the left navigation pane, select Users.

### **Step 3: Select a User**

- Click the user name for whom you want to enable MFA.
- This opens the User Summary page.

### **Step 4: Go to Security Credentials Tab**

- Click the Security credentials tab.
- Scroll down to the section “Multi-factor authentication (MFA)”.

### **Step 5: Assign MFA Device**

- Click “Assign MFA device”.
- Choose the MFA type:
  1. Virtual MFA device (e.g., Google Authenticator, Authy — most common)
  2. Security key (hardware-based, e.g., YubiKey)
  3. Authenticator app on phone

### **Step 6: Configure Virtual MFA**

- If you select Virtual MFA device:
  1. Open your Google Authenticator or Authy app on your phone.
  2. Scan the QR code shown on the AWS screen.
  3. The app starts generating 6-digit codes.

### **Step 7: Verify MFA**

- Enter two consecutive codes from your app in the verification fields.
- Click “Assign MFA”.

### **Step 8: Confirm Setup**

- You will see a green checkmark confirming MFA is successfully assigned.
- The user now requires MFA each time they sign in.

The screenshot shows the AWS IAM User Details page for a user named 'Likhitha'. The top navigation bar includes the account ID (2501-8976-3899) and the name 'Likhitha AWS1'. The left sidebar has a 'Search IAM' bar and sections for 'Dashboard', 'Access management' (User groups, Users, Roles, Policies, Identity providers, Account settings, Root access management), and 'Access reports' (Access Analyzer, Resource analysis). The main content area is titled 'Likhitha Info' and contains a 'Summary' section with details like ARN (arn:aws:iam::250189763899:user/Likhitha), Created (October 13, 2025, 12:20 (UTC+05:30)), Console access (Enabled with MFA), and Last console sign-in (12 days ago). Below this is a 'Security credentials' tab, which is selected, showing a 'Console sign-in' section with a 'Console sign-in link' (https://likhitha-75.signin.aws.amazon.com/console) and a 'Console password' (Updated 12 days ago (2025-10-13 12:20 GMT+5:30)). A 'Last console sign-in' entry is also present. At the bottom of the page are links for CloudShell, Feedback, and copyright information (© 2025, Amazon Web Services, Inc. or its affiliates.).

### Create an AWS Account Alias (for Alternate Sign-in URL)

As an IAM user you can sign in using the default URL or create an account alias for it.

An Account Alias gives your AWS account a name instead of using the long numeric Account ID in your sign-in URL.

This makes it easier for IAM users to remember and log in.

#### Step 1:

- Sign in to the AWS Management Console using root user or an IAM user with administrative privileges.
- In the search bar, type IAM, and open the IAM service.

#### Step 2:

- In the left navigation pane, scroll down and select Dashboard.

#### Step 3:

- Under the “AWS Account” section, find “Account Alias”.
- Click on “Create” (or “Edit” if one already exists).

#### Step 4:

- In the pop-up box, enter your preferred alias name.
- Click “Create alias”.

#### Step 5:

- Once created, you'll see a new Sign-in URL displayed.

Screenshot of the AWS IAM Dashboard (Global) showing security recommendations, IAM resources, and quick links.

**Security recommendations:**

- Root user has MFA: Having multi-factor authentication (MFA) for the root user improves security for this account.
- Root user has no active access keys: Using access keys attached to an IAM user instead of the root user improves security.

**IAM resources:**

| User groups | Users | Roles | Policies | Identity provider s |
|-------------|-------|-------|----------|---------------------|
| 1           | 1     | 3     | 0        | 0                   |

**AWS Account:**

- Account ID: 250189763899
- Account Alias: likhitha-75 [Edit](#) | [Delete](#)
- Sign-in URL for IAM users in this account: <https://likhitha-75.sigin.aws.amazon.com/console>

**Quick Links:**

- [My security credentials](#)
- Manage your access keys, multi-factor authentication (MFA) and other

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

**Date: 14-10-2025**

**Exercise-3:** Amazon S3 – Introduction, Bucket Creation and upload objects (files).

Amazon S3 (Simple Storage Service) is a fully managed, object-based storage service offered by AWS. It allows you to store and retrieve unlimited amounts of data from anywhere on the web.

Unlike traditional file systems that store data in folders, S3 stores data as objects inside buckets. Each object has:

4. Data (the actual file),
5. Metadata (information about the file),
6. Unique key (its name within the bucket).

You can store unlimited data in Amazon S3.

However, each AWS account can create up to 100 buckets by default (can be increased by request).

Each object (file) can be up to 5 TB (terabytes) in size.

Single upload limit: 5 GB (may vary), but larger files can be uploaded using Multipart Upload.

Amazon S3 is designed for:

- Durability: 99.99999999% (11 nines) – this means your data is extremely safe.
- Availability: 99.99% uptime – your data is almost always accessible.
- S3 automatically stores multiple copies of your data across multiple Availability Zones in a region.

Storage classes: S3 offers different storage classes depending on cost and frequency of access.

| Storage Class           | Description  |
|-------------------------|--|
| S3 Standard             | For frequently accessed data (default).                              |
| S3 Intelligent-Tiering  | Automatically moves data between frequent - infrequent access tiers. |
| S3 Standard-IA          | For infrequently accessed data but still quickly available.          |
| S3 Glacier              | For archival storage (low cost, slower retrieval).                   |
| S3 Glacier Deep Archive | For long-term archival (lowest cost).                                |

Each object can be accessed through a unique URL.  
S3 is commonly used for:

Backup and archival

Static website hosting

Application data storage

Media content delivery

| Feature                | Description  |
|------------------------|--|
| Buckets                | Top-level containers for storing objects.                    |
| Objects                | Actual files stored in S3 (e.g., images, HTML, PDFs).        |
| Region                 | Each bucket is created in a specific AWS region.             |
| Versioning             | Maintains multiple versions of the same object for recovery. |
| Static Website Hosting | Allows you to host HTML pages directly from an S3 bucket.    |

**Steps to Create an S3 Bucket and Upload an Image**

### **Step 1: Open the S3 Service**

- Sign in to your AWS Management Console.
- In the search bar, type S3 and click Amazon S3.

### **Step 2: Create a New Bucket**

- Click “Create bucket”.
- Bucket type: General purpose
- Bucket name: Enter a name (Bucket names must be globally unique and lowercase.)
- AWS Region: Choose the region nearest to you.
- Scroll down and uncheck:
  - “Block all public access” → Uncheck this (for viewing file publicly).
  - Confirm the warning checkbox.
- Keep all other settings as default.
- Click Create bucket.

### **Step 3: Upload an Image File**

1. Click on the newly created bucket name.
2. Click Upload → Add files.
3. Choose any image file from your computer.
4. Scroll down and click Upload.

### **Step 4: Make the Object Public**

By default, S3 objects are private. To view them in a browser, make the file public.

1. After upload, go to the Objects tab inside your bucket.
2. Select the uploaded file.
3. Click Actions → Make public using ACL (or Object actions → Make public, depending on console version).
4. Confirm.

The screenshot shows the AWS S3 console interface. At the top, there's a green success message: "Successfully edited public access". Below it, a section titled "Make public: status" indicates "1 object, 19.0 B" were successfully edited. A note says "After you navigate away from this page, the following information is no longer available." The "Failed to edit public access" tab is selected, showing "0 objects failed to edit". The bottom of the screen includes standard AWS links like CloudShell, Feedback, and copyright information.

## Step 5: Copy the Object URL

1. Open the object again by clicking its name.
2. Scroll down to the **Object URL** section.
3. Copy this URL and open it in a new browser tab.

The image is displayed directly from the S3 bucket — meaning AWS S3 is serving that object over HTTP.

The screenshot shows a browser window with the URL "aws-source33.s3.eu-north-1.amazonaws.com/s3.txt" in the address bar. The page content is the string "bnhgvgfreuytgbuoelj".

**Date: 16-10-2025, 23-10-2025**

**Exercise-4:** Amazon S3 – Static Website Hosting (Multi-Page website), Versioning, Cross-Region Replication rule.

### Amazon S3 Static Website Hosting

Amazon S3 can host a static website – [a website consisting of only HTML, CSS, JavaScript, images, etc. – no server-side scripting like PHP or Python].

When you enable “Static Website Hosting,” your S3 bucket acts like a web server, and AWS provides a public website URL to access it.

You can create a multi-page static website (e.g., index.html, about.html, contact.html) and upload it to S3. Links within these pages allow users to navigate between them just like a normal website.

### Steps to Create a Multi-Page Static Website on S3

#### **Step 1: Create an S3 Bucket**

- Open the AWS Management Console → Navigate to S3.
- Click Create bucket.
- Select Bucket type: General purpose
- Enter a unique bucket name (e.g., my-static-web-demo).
- Select ACLs enabled under Object Ownership section.
- Uncheck “Block all public access” → acknowledge.
- Click Create bucket.

The screenshot shows the AWS S3 console interface. The left sidebar includes options for General purpose buckets, Directory buckets, Table buckets, Vector buckets, Access Grants, Access Points (General Purpose Buckets, FSx file systems), Access Points (Directory Buckets), Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, and Block Public Access settings for this account. The Storage Lens section is also visible. The main content area displays the 'General purpose buckets' list. The table shows the following data:

| Name                         | AWS Region                       | Creation date                          |
|------------------------------|----------------------------------|--|
| amzn-s3-chotu-static-website | Europe (Stockholm) eu-north-1    | October 15, 2025, 12:23:35 (UTC+05:30) |
| aws-destination3             | Asia Pacific (Mumbai) ap-south-1 | October 28, 2025, 12:24:21 (UTC+05:30) |
| aws-source33                 | Europe (Stockholm) eu-north-1    | October 28, 2025, 12:22:22 (UTC+05:30) |

On the right side, there are two callout boxes: 'Account snapshot' (Updated daily) and 'External access summary - new' (Updated daily). The 'Account snapshot' box provides visibility into storage usage and activity trends. The 'External access summary' box helps identify bucket permissions that allow public access or access from other AWS accounts.

## **Step 2:** Prepare Website Files

Before uploading, organize your files in a folder structure as follows:

my-website/

```
|  
|   └── index.html  
|   └── about.html  
|   └── contact.html  
|   └── error.html  
└── images/  
    └── banner.jpg
```

Each HTML file should include navigation links.

## **Step 3:** Upload Website Files

- Open your S3 bucket → Click Upload.
- Add all files and folders (HTML, CSS, JS, images).
- Click Upload to store them in S3.

## **Step 4:** Enable Static Website Hosting

- Go to the **Properties** tab of the bucket.
- Scroll down to Static website hosting → Click Edit.
- Choose Enable and select ‘Host a static website’.
- Set:
  - Index document: index.html
  - Error document: error.html
- Click Save changes.

## **Step 5:** Make Files Public (Bucket Policy)

By default, your files are private. To make them public:

- Go to the **Permissions** tab → Bucket Policy → Edit.
- Paste the following policy (replace bucket name):

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "PublicReadGetObject",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": "s3:GetObject",  
      "Resource": "arn:aws:s3:::my-static-web-demo/*"  
    }  
  ]}
```

}

- Save the changes.

| Name         | Type   | Last modified                          | Size    | Storage class |
|--------------|--------|--|---------|---------------|
| about.html   | html   | October 28, 2025, 11:27:46 (UTC+05:30) | 1.1 KB  | Standard      |
| contact.html | html   | October 28, 2025, 11:27:46 (UTC+05:30) | 1.9 KB  | Standard      |
| error.html   | html   | October 28, 2025, 11:27:45 (UTC+05:30) | 976.0 B | Standard      |
| images/      | Folder | -                                      | -       | -             |
| index.html   | html   | October 28, 2025, 11:27:47 (UTC+05:30) | 1.2 KB  | Standard      |

## Step 6: Access Your Website

- Go to the Properties tab → Scroll to Static website hosting.
- Copy the Bucket Website Endpoint URL.
- Paste it into your browser — your homepage (index.html) should appear.

**Static website hosting**  
Use this bucket to host a website or redirect requests. [Learn more](#)

**We recommend using AWS Amplify Hosting for static website hosting**  
Deploy a fast, secure, and reliable website quickly with AWS Amplify Hosting. Learn more about [Amplify Hosting](#) or [View your existing Amplify apps](#)

**S3 static website hosting**  
Enabled

**Hosting type**  
Bucket hosting

**Bucket website endpoint**  
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)  
 <http://amzn-s3-chotu-static-website.s3-website.eu-north-1.amazonaws.com>

- Use the header links to navigate between pages (About, Contact, etc.).

## **When Will the Error Page Be Shown?**

If a user enters a wrong URL or tries to access a file that doesn't exist (e.g., /abc.html), Amazon S3 automatically displays the file you set as the **Error document** (error.html).

## **Amazon S3 Versioning**

Versioning allows you to keep multiple versions of an object in a bucket.

If a file is accidentally deleted or overwritten, you can recover the previous version.  
Each version gets a unique version ID.

### **Steps to Enable Versioning**

- Go to your S3 bucket
- Open the Properties tab
- Scroll to Bucket Versioning
- Click Edit → Enable
- Click Save changes

Now whenever you upload a file with the same name, S3 will keep both versions.

**Note:** You can view versions by clicking "List versions" in the bucket objects page.

### **To Restore or Delete a Specific Version**

- Click the object name → Versions
- Select the desired version → Download / Delete  
(Deleting only adds a delete marker — older versions are still stored.)

## **Cross-Region Replication (CRR) – is a rule created on S3**

CRR automatically copies objects from one S3 bucket (source) to another (destination) in a different AWS Region.

Used for disaster recovery, compliance, or low-latency access in another region.

Requires Versioning to be enabled on both buckets.

### **Steps to Set Up CRR**

**NOTE:** Enable Versioning on both:

Source bucket  
Destination bucket

**Step 1:** Choose a different region before you create Destination bucket  
Create Destination Bucket first.

**Step 2:** Create Source Bucket

Give replication permission:  
Source bucket → Management tab → Replication rules → Create rule

### Step 3: Create Replication Rule page

Enter a Replication rule name  
Status: Enabled

Source bucket section:

Choose a rule scope: select “Apply to all objects in the bucket”

Destination:

Select “Choose a bucket in this account”  
Bucket name: Select the destination bucket

IAM role:

Select “Create new role”

[An IAM role gives Amazon S3 permission to replicate objects from your source bucket to your destination bucket.

S3 replication won’t work unless you assign (or create) a role with the right permissions.

IAM Role: Either create a new role automatically or choose an existing one]

### Step 4: Save

Any new objects uploaded to the source bucket will automatically replicate to the destination region.

The screenshot shows the AWS S3 console interface for the 'aws-source33' bucket. The left sidebar includes sections for General purpose buckets, Storage Lens, and AWS Marketplace. The main content area has tabs for Objects, Properties, Permissions, Metrics, Management (which is selected), and Access Points. Under the Management tab, there are two main sections: 'Lifecycle configuration' and 'Replication rules'. The 'Lifecycle configuration' section is currently empty, with a button to 'Create lifecycle rule'. The 'Replication rules' section shows one rule named 'aws-replication', which is configured to replicate objects to the 'aws-source33' bucket in the 'eu-north-1' region. The rule includes fields for Replication rule name, Status, Destination bucket, Destination Region, Priority, Scope, Storage class, Replica owner, Replication Time Control, KMS-encrypted objects (SSE-KMS or DSSE-KMS), and Replica modification sync.

**Note:** Replication is not retroactive — only new uploads after enabling CRR are copied.

**Reminder – Resource cleanup – to release/delete/terminate the resources created.**

**Date: 29-10-2025**

**Exercise-5:** Overview of EC2, To Launch a Windows EC2 Instance and Connect via RDP Client

**Amazon Elastic Compute Cloud (EC2)** is a core AWS Compute service that lets you run virtual servers (instances) in the cloud.

Amazon EC2 is an Infrastructure as a Service (IaaS) offering from AWS.

It allows you to launch virtual machines to host applications and manage them remotely – wherever you are in the world.

**Key concepts:**

**Instance** - A virtual machine running in the AWS cloud.

**AMI (Amazon Machine Image)** - A pre-configured template that includes: OS (Linux, Windows, etc.), Application software, other configurations.

**Instance Type** - Defines hardware power:

| Family            | Example     | Use Case                   |
|-------------------|-------------|----------------------------|
| General Purpose   | t2.micro    | Basic web apps             |
| Compute Optimized | c5.large    | High-performance computing |
| Memory Optimized  | r5.large    | Databases, analytics       |
| Storage Optimized | i3.large    | Data warehousing           |
| GPU Instances     | g4dn.xlarge | ML/AI, graphics            |

**EBS (Elastic Block Store)** - Persistent storage for your EC2 instance. Acts like a hard drive — data remains even after instance stops. Types: SSD, HDD, etc.

**Security Groups** - Virtual firewalls controlling inbound and outbound traffic.

Example: Allow HTTP (port 80), SSH (port 22), HTTPS (port 443)

**Key Pair** - Used for secure login (SSH for Linux, RDP for Windows). Consists of a public key (stored in AWS) and private key (.pem) that you download.

**Common Ways to Access EC2**

- SSH (Linux instances)
- RDP (Windows instances) - Use Remote Desktop with Administrator password.
- User Data Script: Run automation commands during instance launch.

**EC2 Use Cases**

- Hosting static or dynamic websites
- Deploying web servers (Apache/Nginx)
- Running applications, APIs, or databases
- Machine Learning model hosting

- Batch processing jobs

### Pricing Models

- On-Demand: Pay per hour/second; flexible.
- Reserved Instances: 1–3 year commitment; cheaper.
- Spot Instances: Unused capacity; up to 90% cheaper.
- Free Tier: t2.micro or t3.micro free for 6 months.

### Lifecycle of an Instance

| Step      | Description                                     |
|-----------|---|
| Launch    | Choose AMI, type, key, security group           |
| Running   | Accessible and operational                      |
| Stop      | Instance paused, EBS persists                   |
| Start     | Boot again from same EBS                        |
| Terminate | Deleted permanently, data lost unless backed up |

### Two types of IPv4 addresses

When you launch an EC2 instance, AWS automatically assigns two types of IPv4 addresses depending on your network settings (VPC, subnet, etc.):

#### 1. Private IPv4 Address

- Purpose: Used for internal communication within the same VPC (Virtual Private Cloud).
- Assigned Automatically: Yes, by AWS from the subnet's private IP range.
- Visibility: Not accessible from the Internet.
- Use Case:
  - Instance-to-instance communication inside AWS (e.g., web server ↔ database server).
  - Internal services that do not need public internet access.
- Persistence: The private IP remains attached to the instance until it is terminated.

#### 2. Public IPv4 Address

- Purpose: Used for communication over the Internet.
- Assigned Automatically:
  - Yes, if your subnet is public (i.e., auto-assign public IP enabled).
  - No, if it's a private subnet.
- Visibility: Accessible from the Internet.
- Use Case:
  - Accessing the instance via SSH or HTTP from your local system.
  - Hosting web applications publicly.
- Persistence:
  - The public IP changes each time you stop/start the instance.
  - To make it permanent, you can assign an Elastic IP (static public IP).

**Remote Desktop Protocol [RDP]**, is a secure communication protocol developed by Microsoft that allows a user to connect to and control another computer remotely. It establishes an encrypted channel to transmit keyboard and mouse inputs from the client to the remote computer and send screen information back to the client, providing a graphical user interface for remote access.

An **RDP client** is the software or app that you use to make this remote connection. It connects to a remote Windows server or Windows EC2 instance that is running an RDP server (which listens on port **3389**). It is pre-installed in Windows.

### **Steps to Launch a Windows EC2 Instance and Connect via RDP Client**

#### **Step 1: Sign in to AWS Management Console**

Select the **Region** closest to you.

#### **Step 2: Open the EC2 Dashboard**

In the AWS Console, search for **EC2** in the search bar.

Click on EC2 → Instances → Launch Instance.

Under Name and Tags, give your instance a name.

#### **Step 3: Choose an Amazon Machine Image (AMI)**

Under Application and OS Images (Amazon Machine Image) → click Browse more AMIs or choose: Microsoft Windows Server 2022 Base (Free tier eligible).

#### **Step 4: Choose Instance Type**

Choose t3.micro (Free-tier eligible).

#### **Step 5: Configure Key Pair**

Under Key pair (login), choose an existing key pair or create a new one.

If creating a new key pair:

- Choose type: RSA
- Format: .pem
- Download and save it safely — it's required to decrypt your Windows password later.

#### **Step 6: Configure Network Settings**

Leave default VPC and Subnet settings.

Under Firewall (security group) →

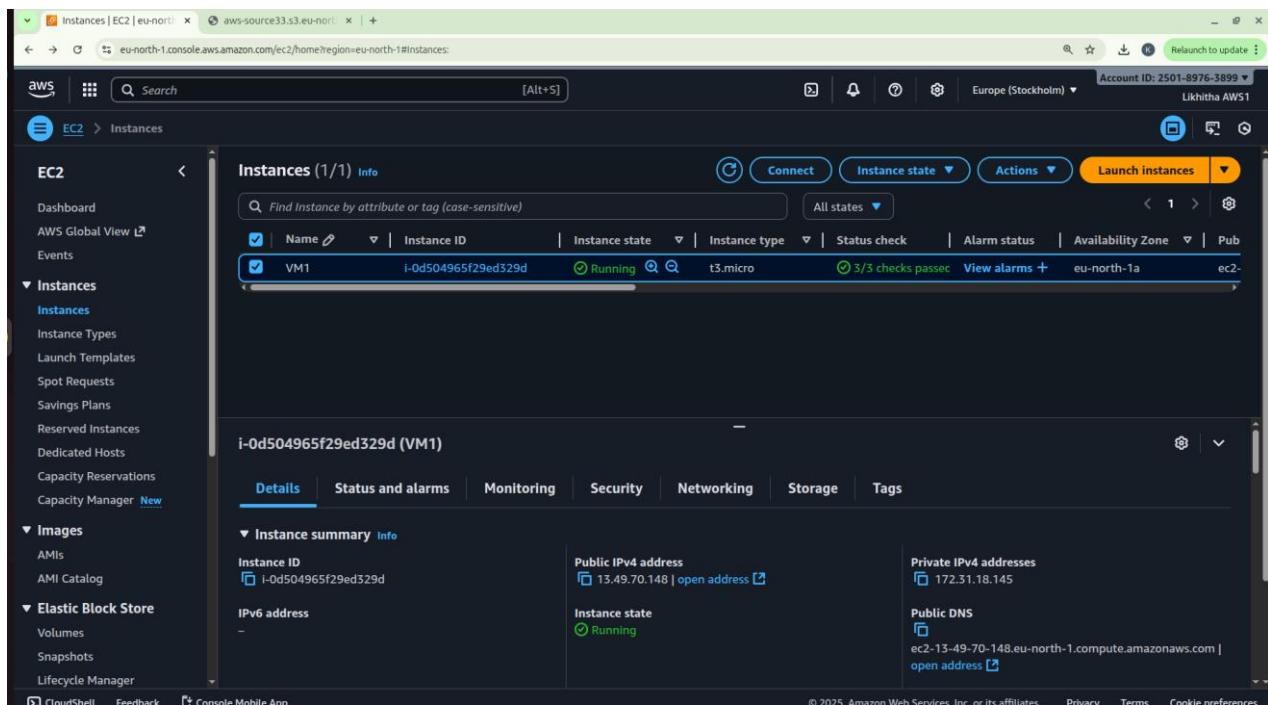
- Select Create security group.
- Allow RDP (port 3389) access from My IP (for better security) or anywhere (0.0.0.0/0).

#### **Step 7: Launch the Instance**

Review all configurations.

Click Launch Instance.

Wait until the Instance state changes to Running and Status check = 3/3 passed.



### Note:

Wait approximately 5 minutes after instance launch to allow AWS to fully initialize the instance and make the Administrator password available.

When a Windows EC2 instance is first launched, AWS needs a few minutes to:

Initialize the instance, Attach the root volume, Generate the Windows Administrator password (encrypted using your key pair).

### Step 8: Get the Administrator Password

Select your running instance → click Connect → choose RDP Client tab.

Click Get Password (you must wait about 4 minutes after launch).

Upload your .pem key file and click Decrypt Password.

Copy the Public IPv4 address and Administrator Password shown.

### Step 9: Connect Using RDP Client

On Windows system:

1. Open Remote Desktop Connection (from Start Menu).
2. Enter your instance's Public IPv4 address.
3. Click Connect → Enter:
  - o Username: Administrator
  - o Password: (*the decrypted password from AWS*)
4. Click OK → accept the certificate → the remote Windows desktop opens!

### Step 10: Verify Connection

- You should now see a Windows Server desktop running inside your local window.
- You can open File Explorer, browse settings, or install software (within the free-tier limits).

**Note:**

- Always **Stop** (not Terminate) the instance when not in use to avoid charges.
- RDP uses port 3389, so ensure it's open in the security group.
- Avoid sharing your decrypted password or key pair.

The screenshot shows the AWS EC2 Instances page for the ap-northeast-1 region. A green notification bar at the top indicates "Successfully initiated stopping of i-00cb05ee4705286a1". The main table lists one instance:

| Name             | Instance ID         | Instance state | Instance type | Status check      | Alarm status  | Availability Zone | Public IPv4 DNS |
|------------------|---------------------|----------------|---------------|-------------------|---------------|-------------------|-----------------|
| amz-ec2-insta... | i-00cb05ee4705286a1 | Stopped        | t3.micro      | 3/3 checks passed | View alarms + | ap-northeast-1d   | -               |

Below the table, the instance details for "i-00cb05ee4705286a1 (amz-ec2-instance)" are shown. The "Details" tab is selected. Key information includes:

- Instance ID: i-00cb05ee4705286a1
- Public IPv4 address: -
- Instance state: Stopped
- Private IP DNS name (IPv4 only): ip-172-31-20-198.ap-northeast-1.compute.internal
- Public DNS: -

An "Actions" dropdown menu is open, showing options like "Stop", "Start", "Reboot", "Termination protection", "Launch instances", and "Copy AMI".

An "AWS Lambda" icon is visible in the top right corner of the browser window.

A "Remote Desktop Connection" window is overlaid on the bottom right, showing the connection details for the instance:

- Computer: 13.158.67.242
- User name: Administrator
- Show Options button
- Connect button
- Help button

**Date: 30-10-2025**

**Exercise-6:** Launch a Linux EC2 Instance and Connect using SSH through PowerShell/Linux Terminal and PuTTY on Windows.

**Note: Choosing the Correct Key Pair Format**

While creating a Key Pair, you are asked to select the Private Key File Format — either .pem or .ppk. The correct choice depends on the operating system and the method you will use to connect to your EC2 instance.

| Scenario   | Key File Format | Explanation   |
|--|-----------------|---|
| Using PuTTY on Windows                               | .ppk            | The .ppk file is specific to the GUI based PuTTY application, a popular SSH client for Windows system.  |
| Using PowerShell on Windows, Linux terminal on Linux | .pem            | The .pem file is the default AWS key format used by the OpenSSH client available in PowerShell (Windows), Linux, and macOS terminals. Used for CLI. |

**Steps to Launch a Linux EC2 Instance and Connect using SSH through PowerShell/Linux**

**Step 1: Sign in to AWS Management Console**

select the nearest AWS Region.

**Step 2: Open EC2 Service**

In the search bar, type EC2 and click EC2 Dashboard.  
Select Instances → Launch Instance.

**Step 3: Configure Instance Details**

Under Name and Tags, give your instance a name, e.g., *Linux-SSH-Demo*.  
Under Application and OS Images (AMI) → select Amazon Linux 2 AMI (Free tier eligible).  
Under Instance type, choose t3.micro (Free-tier eligible).

**Step 4: Create or Select a Key Pair**

Under Key pair (login) → choose Create new key pair.

Choose:

- Key pair type: RSA
- Private key file format: .pem (for SSH via PowerShell/Linux)

Click Create key pair → the .pem file will automatically download.

**Save it securely on your local machine (you'll need it for SSH login).**

**Step 5: Configure Network Settings**

Under Network settings, leave the defaults.

In Firewall (security group) → select Create security group.

Ensure SSH (port 22) is allowed:

- Type: SSH
- Protocol: TCP

- Port Range: 22
- Source: My IP (recommended for security) or Anywhere (0.0.0.0/0).

### **Step 6: Launch the Instance**

Review the configuration → click Launch Instance.

Wait for the Instance State to show Running and Status Checks: 3/3 passed.

### **Step 7: Get the Public IP Address**

Select your instance → In the summary section →

Copy the Public IPv4 address — you'll use this to connect.

### **Step 8: Connect using SSH**

#### **(A) On Windows using PowerShell**

- Open **PowerShell** (search “PowerShell” from the Start menu).
- Navigate to the folder where your .pem file is saved:  
`cd "C:\Users\<YourName>\Downloads"`
- Connect using the SSH command:  
`ssh -i "keyfile.pem" ec2-user@<Public-IP-address>`
- When prompted, type **yes** to continue connecting.
- You'll now be logged into your EC2 Linux terminal.

#### **(B) On Linux Terminal (Ubuntu / macOS)**

- Open **Terminal**.
- Navigate to the directory where your .pem file is stored.
- Set proper permission for the key file:  
`chmod 400 keyfile.pem`
- Connect to the instance:  
`ssh -i keyfile.pem ec2-user@<Public-IP-address>`
- Type **yes** when prompted.
- You will be connected to your EC2 instance remotely.

### **Step 9: Verify Connection**

- Once connected, the command prompt will appear as:  
`[ec2-user@ip-172-31-xx-xx ~]$`
- Try a few commands:  
`uname -a` and `sudo yum update -y` to confirm access.

### **Step 10: Stop Instance after Use**

Go to the EC2 console.

Select your instance → Instance State → Stop Instance (to avoid charges).

The screenshot shows a CloudShell session with two tabs open. The left tab displays the AWS CloudFormation console, specifically the 'Instances | EC2 | eu-north-1' section. It lists three running instances in the 'eu-north-1a' availability zone, each with a Public DNS name starting with 'ec2-13-62-49-91.eu-north-1.compute.amazonaws.com'. The right tab is a Windows PowerShell window titled 'ec2-user@ip-172-31-16-50:~'. The user is attempting to SSH into another EC2 instance at '13.62.49.91' using a previously saved key ('03.pem'). The PowerShell session includes a warning about host fingerprint verification and a prompt for confirmation.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\MC049\Downloads> ssh -i "03.pem" ec2-user@13.62.49.91
ssh: Could not resolve hostname <13.62.49.91>. No such host is known.
PS C:\Users\MC049\Downloads> ssh -i "03.pem" ec2-user@13.62.49.91
The authenticity of host '13.62.49.91 (13.62.49.91)' can't be established.
ED25519 key fingerprint is SHA256:ge5zmb9jyFthPB56aB74tk37lgxC+ln35RvLiNGH/k.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '13.62.49.91' (ED25519) to the list of known hosts.

[ec2-user@ip-172-31-16-50 ~]$
```

## **Steps to Install PuTTY on Windows**

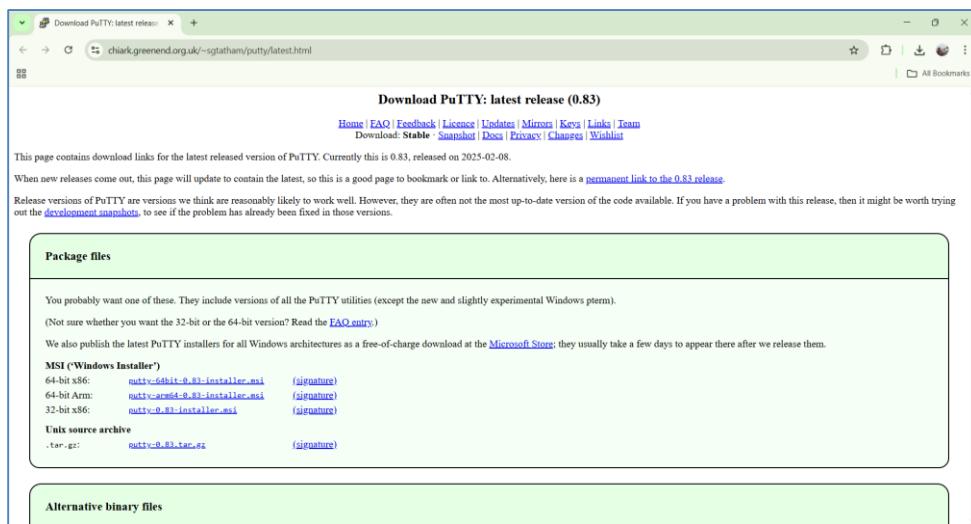
PuTTY is a client program for the SSH, Telnet and Rlogin network protocols. These protocols allow you to interact with a remote server as if you were sitting right in front of it.

It is primarily used in the Windows platform.

In an era where cloud computing and remote servers are the norm, being able to securely connect and interact with these servers is vital. It provides a secure and reliable way to connect to these remote systems. It supports a range of network protocols including the secure ones like SSH.

Use the correct, safe download link: Official download page: This is the only genuine PuTTY source.

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>



Under the **Package files** section, look for:

MSI ('Windows Installer')

64-bit x86: putty-64bit-0.83-installer.msi

Click the link: **putty-64bit-0.83-installer.msi**



Once it downloads, open the file and follow: Next → Next → Install → Finish

#### **After installation, you will have:**

- PuTTY – to connect to your EC2 instance via SSH
- PuTTYgen – to convert .pem to .ppk (if needed)
- Pageant – optional key manager

You can open PuTTY by typing **PuTTY** in the Windows search bar/start menu.

### **Steps to Launch a Linux EC2 Instance and Connect using PuTTY on Windows**

#### **Step 1: Sign in to AWS Management Console**

Select your nearest AWS Region.

#### **Step 2: Open the EC2 Service**

In the AWS Console search bar, type EC2 and select EC2 Dashboard.

Click Instances → Launch Instance.

#### **Step 3: Configure Instance Details**

Under Name and Tags, enter an instance name, e.g., *Linux-PuTTY-Demo*.

Under Application and OS Images (AMI) → choose Amazon Linux 2 AMI (Free tier eligible).

Under Instance Type, select t2.micro (Free tier eligible).

#### **Step 4: Create or Select a Key Pair**

Under Key pair (login) → select Create new key pair.

Choose the following:

- Key pair type: RSA
- Private key file format: .ppk (*for PuTTY on Windows*)

Click Create key pair → a .ppk file will download automatically.

**Save it securely — this file is required to connect later.**

#### **Step 5: Configure Network Settings**

Under Network settings, leave default VPC/Subnet settings.

Under Firewall (security group):

- Select Create security group.
- Ensure SSH (port 22) is allowed:
  - Type: SSH
  - Protocol: TCP
  - Port Range: 22
  - Source: anywhere.

#### **Step 6: Launch the Instance**

Review all configurations.

Click Launch Instance.

Wait until the Instance State changes to Running and Status Check = 3/3 passed.

#### **Step 7: Obtain the Public IP**

Select your instance → In the summary section →

Copy the Public IPv4 address or Public DNS (IPv4) — you'll use this to connect from PuTTY.

## **Step 8: Connect using PuTTY**

Open **PuTTY** on your Windows system.

In the **Host Name (or IP address)** field, enter: ec2-user@<Public-IP-address>

In the **Category** list on the left, expand: Connection → SSH → Auth → Credentials

Click **Browse** → locate and select your .ppk key file downloaded earlier.

Click **Open**.

When prompted, click **Accept** to trust the host.

A terminal window opens — you're now connected to your EC2 Linux instance!

## **Step 9: Verify Connection**

Once connected, your prompt should look like: [ec2-user@ip-172-31-xx-xx ~]\$

Try verifying: uname -a or update packages: sudo yum update -y

## **Step 10: Stop the Instance**

Return to the EC2 Dashboard.

Select your instance → click Instance State → Stop Instance.

This prevents charges when you're not using it.

### **Note**

- Use .ppk format key when connecting with **PuTTY (Windows)**.
- Use .pem format key when connecting with **PowerShell / Linux / macOS terminal**.
- Both keys serve the same purpose — secure authentication to your EC2 instance.

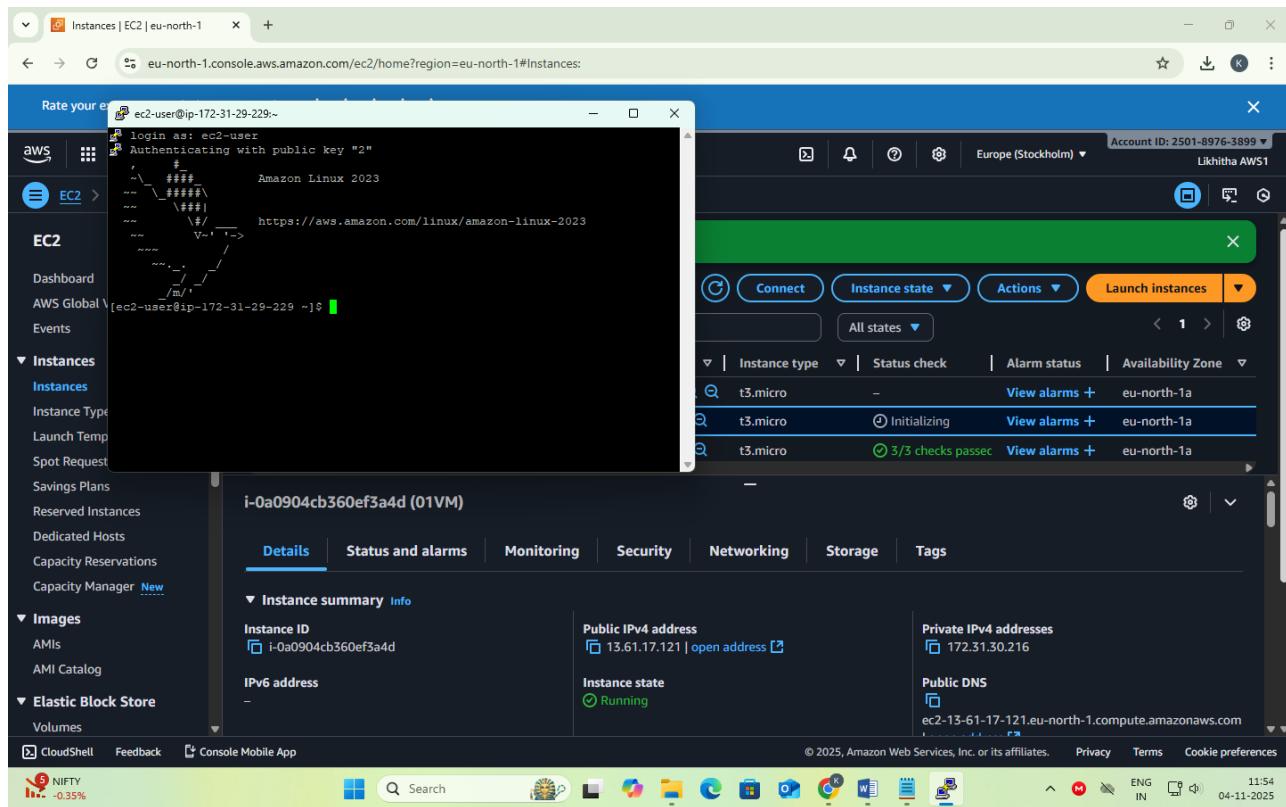
The message “**Permission denied (publickey)**” or “**Permissions are too open**” appears when the .pem key file or SSH configuration has incorrect permissions or mismatched ownership.

Run this command in your local terminal before connecting:

```
chmod 400 keypair.pem
```

Then connect again:

```
ssh -i "keypair.pem" ec2-user@<Public-IP>
```



**Date: 5-11-2025**

**Exercise-7:** Hosting a static website on EC2 instance.

- Manual Installation of Apache (httpd) Web Server on EC2 for Static Website Hosting.
- Launching an EC2 Instance with User Data Script to Automatically Install Apache and Host a Static Website.

**When you create an EC2 instance**, it's just a **bare machine** — It does not have the software to host a website yet. To host a website:

- You need a web server software → Apache (httpd)
- You put your website files (HTML, CSS, etc.) in a special folder (usually /var/www/html)
- Apache listens on port 80 (HTTP) or port 443 (HTTPS) for incoming connections

**Apache HTTP Server** (often called Apache or httpd) is a web server software.

It runs on a computer (like your EC2 instance) and delivers web pages (HTML, images, CSS, etc.) to users over the HTTP/HTTPS protocol.

So, whenever someone types your website's URL (like <http://your-ec2-ip/>), Apache receives that request and serves your webpage files from your server to the browser.

**httpd stands for **HTTP Daemon**.**

A *daemon* in Linux is a background service that keeps running to handle requests.

So, httpd is the background process that runs the Apache web server.

When you install Apache, you're really installing the **httpd service**.

**File Locations (default)** – Website files go into: /var/www/html

**Steps for Manual Installation of Apache (httpd) Web Server on EC2 for Static Website Hosting.**

### **Part 1 – Creating an EC2 Instance**

#### **Step 1: Sign in to AWS Management Console**

In the search bar, type EC2 and open the EC2 Dashboard.

#### **Step 2: Launch a New Instance**

Click "Launch Instance."

Give a name. (e.g., MyApacheServer)

**Step 3: Choose an Amazon Machine Image (AMI)**

Select Amazon Linux 2 AMI (Free tier eligible)

**Step 4: Choose Instance Type**

Choose t3.micro (Free tier eligible).

**Step 5: Create / Select Key Pair**

Under Key pair (login), choose:

Create new key pair → Give a name → File format = .pem

Download the key file → Keep it safe.

**Step 6: Configure Network Settings**

Click → Allow SSH traffic

→ Allow HTTP traffic from the internet. (This automatically opens port 80 for web access).

**Step 7: Launch Instance**

Review → Click **Launch Instance**.

Wait for the instance to start.

**Step 8: Connect to the Instance****Using powershell type the following commands:**

- Open PowerShell (search “PowerShell” from the Start menu).
- Navigate to the folder where your .pem file is saved:  
cd "C:\Users\<YourName>\Downloads"
- Connect using the SSH command:  
ssh -i "keyfile.pem" ec2-user@<Public-IP-address>  
ssh -i "web.pem" ec2-user@ 13.53.136.50
- When prompted, type yes to continue connecting.
- You'll now be logged into your EC2 Linux terminal.

**Part 2 – Manual Installation of Apache (httpd) Web Server****Step 1: Update the Packages**

sudo yum update -y

**Step 2: Install Apache (httpd)**

sudo yum install httpd -y

(*This installs the Apache Web Server package.*)

**Step 3: Start the Apache Service**

sudo systemctl start httpd

**Step 4: Enable Apache to Start on Boot**

```
sudo systemctl enable httpd
```

### Step 5: Check Apache Status

```
sudo systemctl status httpd [It should show active (running). Press q to exit status view.]
```

### Step 7: Test Apache Server

Copy your Public IPv4 address from the EC2 dashboard.

Paste it into a browser: http://

You should see the Apache Test Page

## Part 3 – Host a Static Website on Apache

### Step 1: Move to Web Root Folder

```
cd /var/www/html
```

### Step 2: Create your HTML file

```
sudo nano index.html
```

### Step 3: Add HTML content

Paste the following code:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to My Website</title>
</head>
<body style="text-align:center; background-color: #f0f0f0;">
<h1>Hello from Apache on AWS EC2!</h1>
<p>This is a static website hosted on Apache web server.</p>
</body>
</html>
```

**Press Ctrl + O, Enter, then Ctrl + X to save and exit.**

After Pressing Ctrl + O You'll see:

```
File Name to Write: index.html [Just press Enter to confirm saving with that name].
```

To Exit Nano: press Ctrl + X [This closes the Nano editor and will be back to the Linux prompt]

### Step 4: Restart Apache

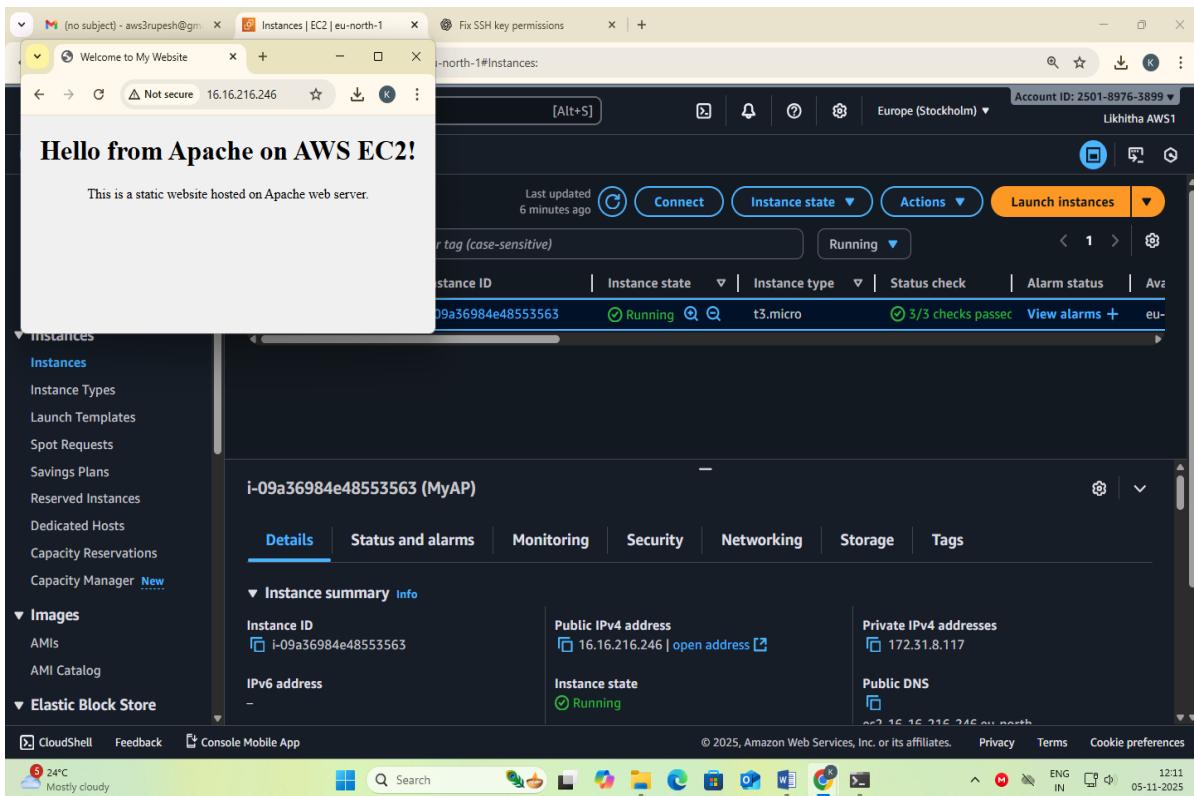
```
sudo systemctl restart httpd
```

### Step 5: View Website

- In browser: http://<your-ec2-public-ip>
- You'll see your custom HTML page.

### Optional: Add Multiple Pages

- Upload other pages like about.html, contact.html in the same directory.



## Steps for launching an EC2 Instance with User Data Script to Automatically Install Apache and Host a Static Website.

In this method, you don't manually install Apache or upload files after connecting. Instead, you write a **shell script** in the **User Data** section (during instance creation). When the EC2 instance starts for the first time, it automatically:

1. Installs Apache (httpd)
2. Starts the service
3. Creates an index.html webpage
4. Hosts your website immediately after launch

### Step 1: Sign in to AWS Management Console

Go to EC2 Dashboard → Click Launch Instance

### Step 2: Name and OS

Name: AutoApacheWebServer

AMI: Choose Amazon Linux 2 AMI (Free tier eligible)

Instance Type: t3.micro

### Step 3: Key Pair

Select existing key pair (or create new) → Format = .pem

### Step 4: Network Settings

Allow HTTP traffic from the Internet (port 80)  
Allow SSH traffic (port 22)

### Step 5: Add User Data Script

Scroll down to Advanced Details → User data box.

This script:

- Updates all packages
- Installs and starts Apache
- Creates a sample index.html webpage in /var/www/html

Paste the following shell script:

```
#!/bin/bash
# Update packages
yum update -y

# Install Apache Web Server
yum install -y httpd

# Start Apache
systemctl start httpd
systemctl enable httpd

# Create website content
echo "<html>
<head><title>Welcome to My Auto Web Server</title></head>
<body style='text-align:center; background-color:#e9f5ff;'>
<h1>Welcome to EC2 Instance</h1>
<h2>Apache Installed Automatically via User Data Script</h2>
<p>This is a static website hosted on EC2 using User Data automation.</p>
</body>
</html>" > /var/www/html/index.html
```

### Step 6: Launch the Instance

Click Launch Instance

Wait for the status → Running

### Step 7: Test Your Web Server

Copy the Public IPv4 address of your instance.

Paste it in your browser: <http://<your-public-ip>>

You should immediately see your webpage without logging into EC2!

The screenshot shows the AWS Management Console interface for the EC2 service. On the left, a sidebar menu lists various options under 'Instances' such as Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Capacity Manager. Below these are sections for 'Images' (AMIs, AMI Catalog) and 'Elastic Block Store'. A central search bar at the top contains the placeholder 'Search for tag (case-sensitive)'. The main content area displays a table of instances. One instance, 'Mywebserver' with ID i-08436bb99b800b911, is highlighted and selected. The instance details page is open, showing the following information:

| Details                                   | Status and alarms  | Monitoring  | Security | Networking | Storage | Tags |
|---|--|---|----------|------------|---------|------|
| <b>i-08436bb99b800b911 (Mywebserver)</b>  |  |   |          |            |         |      |
| <b>Instance ID</b><br>i-08436bb99b800b911 | <b>Public IPv4 address</b><br>16.170.108.68   <a href="#">open address</a> | <b>Private IPv4 addresses</b><br>172.31.4.72                            |          |            |         |      |
| <b>IPv6 address</b><br>-                  | <b>Instance state</b><br>Running   | <b>Public DNS</b><br>ec2-16-170-108-68.eu-north-1.compute.amazonaws.com |          |            |         |      |

At the bottom of the page, there are links for CloudShell, Feedback, and Console Mobile App. The status bar at the very bottom indicates the user's account ID (2501-8976-3899), region (Europe (Stockholm)), and session details (Likhitha AWS1). The date and time shown are 07-11-2025 09:07.

**Date: 7-11-2025**

**Exercise-8:** Create a Custom AMI from a Working EC2 Instance. Launch an EC2 Instance using a Custom AMI. Delete the custom AMI

**AMI (Amazon Machine Image)** is a pre-configured template that contains only Operating System (e.g., Amazon Linux, Ubuntu, Windows) needed to launch an EC2 instance.

When you launch a new EC2 instance, you select an AMI as the base image.

A **Custom AMI** is created from your *own running EC2 instance* after you've installed software, uploaded website files, or applied settings.

**Benefits:**

1. **Reusability** – Launch multiple identical servers quickly.
2. **Backup** – Acts as a snapshot of your configured instance.
3. **Auto Scaling** – Used by Auto Scaling Groups to create identical instances automatically.
4. **Disaster Recovery** – You can recreate your setup if the original instance fails.
5. **Time-saving** – No need to reinstall Apache or re-upload files each time.

A Custom AMI is like a master copy of your EC2 setup. It ensures your website or app can be duplicated instantly and consistently.

**Steps to Create a Custom AMI from a Working EC2 Instance**

To capture the current configuration — installed packages, website files, and settings — into a reusable Amazon Machine Image (AMI).

**Step 1: Select the running instance**

Go to EC2 → Instances.

Select the instance that already hosts your website.

Or create an instance (add user data for web hosting)

**Step 2: Create Image**

From the Actions → Image and templates → Create image.

**Step 3: Enter details**

Image name: MyWebsiteAMI

Description: AMI created from configured Apache website instance

Leave “No reboot” unchecked (so the filesystem is consistent).

**Step 4: Storage volumes**

The root volume will appear automatically; keep defaults unless you need more space.

## Step 5: Create image

Click **Create image**.

A confirmation message appears; note the **Image ID**.

## Step 6: Verify creation

In left panel → **AMIs** → refresh until **Status = Available**.

Your custom AMI is now saved in that region and can be used to launch identical webserver instances.

| Name | AMI ID                | Source            | Owner        |
|------|-----------------------|-------------------|--------------|
| AMII | ami-0ab0785e7af67bff6 | 250189763899/AMII | 250189763893 |

## Steps to Launch an EC2 Instance using a Custom AMI

To launch a new EC2 instance from a previously created Custom Amazon Machine Image (AMI) — containing your configured website and software.

### Step 1: Go to AMIs

Open the EC2 Service dashboard.

In the left navigation pane, click AMIs (under “Images”).

### Step 2: Select Your Custom AMI

Locate the AMI you created earlier (e.g., MyWebsiteAMI).

Ensure Status = Available.

### **Step 3: Launch Instance from AMI**

Select the AMI → click Launch instance from image.

### **Step 4: Configure Instance Details**

Name: WebServer-from-Custom-AMI

Instance type: t3.micro (Free Tier eligible)

Key pair: Choose an existing .pem key for SSH access.

Network settings:

VPC: Default

Subnet: Public

Security Group: Allow HTTP (80) and SSH (22)

### **Step 5: Storage (EBS Volume)**

Keep default root volume (e.g., 8 GB gp3).

### **Step 6: Review and Launch**

Click Launch instance.

Wait until the instance state becomes Running.

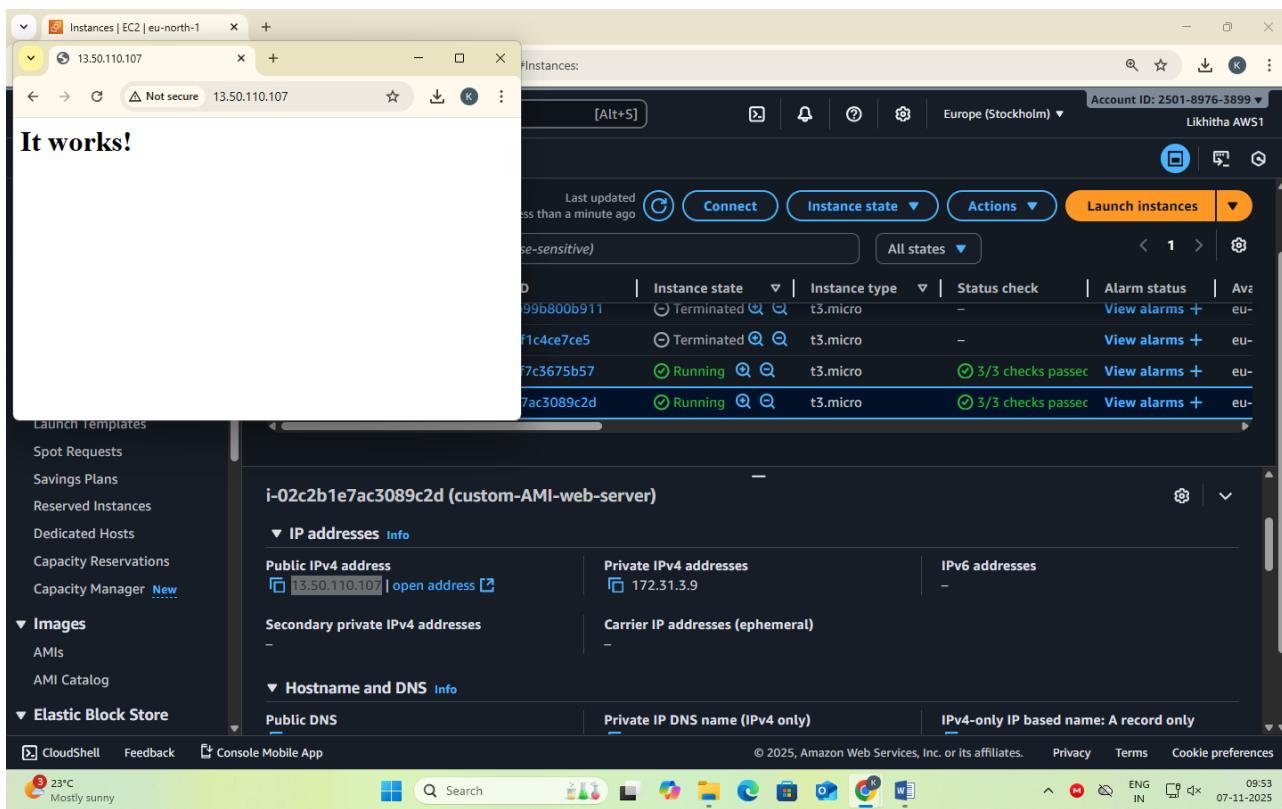
### **Step 7: Access the Website**

Copy the Public IPv4 address from the instance details.

Open in a browser → <http://<Public-IP>>

You should see your same website, confirming the custom AMI works.

A new EC2 instance is successfully launched using the Custom AMI, automatically containing the OS, Apache, configurations, and website files — no manual setup required.



## Steps to Delete the custom AMI

Deleting a **custom AMI** involves **deregistering** the image and then **deleting its associated EBS snapshot**.

When you deregister an AMI, it is removed from your account and can no longer be used to launch new instances.

However, the **snapshot** that was created along with the AMI still remains in your storage and **continues to incur charges, so you must delete it separately** to free up space and stop costs.

This ensures your AWS environment remains clean and cost-efficient.

### Step 1 – Open EC2 Dashboard

Sign in to the AWS Management Console.

Navigate to EC2 service.

In the left navigation pane, scroll down to Images → AMIs.

### Step 2 – Locate Your Custom AMI

In the Owned by me tab, you will see all AMIs you created (custom AMIs).

Select the AMI you want to delete.

You can identify it by Name or AMI ID.

## Step 3 – Deregister the AMI

Select the AMI → Click on Actions dropdown.

Choose Deregister AMI.

Confirm by clicking Deregister in the pop-up.

This removes the AMI record, but not the underlying snapshot(s).

The screenshot shows the AWS EC2 console interface. The left sidebar has 'EC2' selected under 'Images'. The main content area is titled 'Amazon Machine Images (AMIs)' and displays a message: 'No AMIs in this Region for: Owned by me.' Below this, it says 'You can use the filter to view Owned By Me, Private Images, Public Images, or Disabled Images.' There is a search bar and filter options for 'Name', 'AMI name', 'AMI ID', 'Source', and 'Owner'. At the bottom of the main area, there is a section titled 'Select an AMI'.

## **EXERCISE 9: Mini-Project Hosting a Multi-Page Website using EC2 and S3**

Amazon Web Services (AWS) offers cloud-based resources for hosting and managing web applications.

In this experiment, two AWS services are used together:

- **Amazon S3 (Simple Storage Service):** Used to store image files and make them publicly accessible via HTTPS URLs.
- **Amazon EC2 (Elastic Compute Cloud):** A virtual server used to host and serve the website's HTML and CSS files using **Apache** as the web server.

This setup demonstrates the concept of **decoupling compute and storage** — the website files are served from EC2 while the images are fetched from S3.

### **Steps:**

#### **1. Login to AWS Management Console.**

Navigate to the AWS dashboard using your account credentials.

#### **2. Create an S3 Bucket.**

- Go to **S3 Service** → **Create bucket**.
- Provide a unique bucket name (e.g., *my-website-images-123*).
- Select the region (e.g., *us-east-1*).
- Uncheck “Block all public access”.
- Click **Create bucket**.

#### **3. Upload Images to S3.**

- Upload images (cloud1.jpg, cloud2.jpg, cloud3.jpg).
- Use **Object actions** → **Make public using ACL** (or apply a bucket policy) to make the images publicly accessible.
- Note the **Object URLs** of each image.

The screenshot shows the AWS S3 console interface. The left sidebar lists various services under 'Amazon S3' such as General purpose buckets, Directory buckets, Table buckets, Vector buckets, Access Grants, Access Points (General Purpose Buckets, FSx file systems), Access Points (Directory Buckets), Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, and Storage Lens. The main content area is titled 's3-images-12 Info' and shows the 'Objects' tab selected. It displays three objects: 'cloud1.jpeg' (Type: jpeg, Last modified: November 8, 2025, 10:44:46 (UTC+05:30), Size: 6.6 KB, Storage class: Standard), 'cloud2.png' (Type: png, Last modified: November 8, 2025, 10:44:45 (UTC+05:30), Size: 10.4 KB, Storage class: Standard), and 'cloud3.jpeg' (Type: jpeg, Last modified: November 8, 2025, 10:44:44 (UTC+05:30), Size: 4.8 KB, Storage class: Standard). The top right corner shows the account ID: 2501-8976-3899, Europe (Stockholm), and the user name Likhitha AWS1.

#### 4. Launch an EC2 Instance.

- Open **EC2 Service** → **Launch instance**.
- Choose **Amazon Linux 2025 AMI** and instance type **t3. micro** (Free Tier).
- Create or select an existing key pair.
- Allow **HTTP (80)** and **SSH (22)** in the Security Group.
- In **Advanced details** → **User data**, paste your files or upload your files.

```
#!/bin/bash sudo yum -y  
update sudo yum -y install  
httpd sudo systemctl ena-  
ble httpd sudo systemctl  
start httpd
```

```
BUCKET="your-bucket-name"  
REGION="your-region"  
WEBROOT="/var/www/html"  
sudo rm -rf ${WEBROOT}/*
```

```
# ----- index.html -----
```

```
sudo tee ${WEBROOT}/index.html > /dev/null <<'EOF'  
<!DOCTYPE html><html><head><meta charset="UTF-8">  
<title>Home - Apache Hosted Website</title>  
<link rel="stylesheet" href="styles.css"></head>  
<body><h1>Website Hosted on Apache</h1>  
<p>This website is hosted on AWS EC2 using Apache and displays images from an  
S3 bucket.</p>  
<a href="/about.html">About</a> | <a href="/gallery.html">Gallery</a> | <a  
href="/contact.html">Contact</a>  
</body></html>
```

```
# ----- about.html -----
```

```
sudo tee ${WEBROOT}/about.html > /dev/null <<'EOF'  
<!DOCTYPE html><html><head><meta charset="UTF-8">  
<title>About</title><link rel="stylesheet" href="styles.css"></head>
```

```
<body><h1>About this Project</h1>
<p>This project demonstrates hosting a static website on AWS EC2 using Apache and S3.</p>
<ul><li>EC2 hosts web files</li><li>S3 stores images</li></ul>
<a href="/">Home</a></body></html>
```

```
# ----- gallery.html -----
```

```
sudo tee ${WEBROOT}/gallery.html > /dev/null <<EOF
<!DOCTYPE html><html><head><meta charset="UTF-8">
<title>Gallery</title><link rel="stylesheet" href="styles.css"></head>
<body><h1>Gallery</h1>
<p>Images loaded directly from S3:</p>



<a href="/">Home</a></body></html>
```

```
# ----- styles.css -----
```

```
sudo tee ${WEBROOT}/styles.css > /dev/null <<'EOF' body{font-
family:Arial;text-align:center;margin:0;background:#fafafa}
h1{color:#b22222;margin-top:20px} a{text-decoration:none;margin:0
10px;color:#b22222;font-weight:bold} img{margin:10px;border-
radius:8px}
```

```

sudo chown -R apache:apache "$WEBROOT" sudo chmod -R 755
"$WEBROOT"
sudo systemctl restart httpd

```

- Launch the 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), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, CloudShell, Feedback, and Console Mobile App. The main area displays a table titled 'Instances (1/3) Info' with three rows:

|                                     | Name            | Instance ID         | Instance state | Instance type | Status check | Alarm status                  | Availability Zone | Pub |
|-------------------------------------|-----------------|---------------------|----------------|---------------|--------------|-------------------------------|-------------------|-----|
| <input checked="" type="checkbox"/> | assign-01       | i-064b11ded047da83f | Stopped        | t3.micro      | -            | <a href="#">View alarms +</a> | eu-north-1a       | -   |
| <input type="checkbox"/>            | AMI Webserver   | i-033f23ef7c3675b57 | Stopped        | t3.micro      | -            | <a href="#">View alarms +</a> | eu-north-1c       | -   |
| <input type="checkbox"/>            | custom-AMI-w... | i-02c2b1e7ac3089c2d | Stopped        | t3.micro      | -            | <a href="#">View alarms +</a> | eu-north-1c       | -   |

Below the table, a specific instance is selected: 'i-064b11ded047da83f (assign-01)'. The 'Details' tab is active, showing the following information:

- Instance summary** (Info):
 

|  |   |                                      |
|--|---|--------------------------------------|
| Instance ID: i-064b11ded047da83f                                     | Public IPv4 address: -  | Private IP4 addresses: 172.31.17.136 |
| IPv6 address: -  | Instance state: Stopped   | Public DNS: -                        |
| Hostname type: IP name: ip-172-31-17-136.eu-north-1.compute.internal | Private IP DNS name (IPv4 only): ip-172-31-17-136.eu-north-1.compute.internal |                                      |

## 5. Access the Website.

- Once the instance state is “running”, copy the **Public IPv4 address**.
- Open a web browser and visit: <http://<PublicIPv4>>
- The home page will display links to *Home*, *About*, *Gallery*.
- The *Gallery* page loads the images directly from the S3 bucket via HTTPS links.

Instances | EC2 | eu-north | Home — Visual Stories | +

Not secure 56.228.80.190 Finish update

# VisualStories

# Where Stories Come Alive

A curated collection of moments, beautifully hosted on AWS infrastructure

Explore Gallery >

Cloud-Native S3 Storage Responsive

Instances | EC2 | eu-north | Gallery — Visual Stories | +

Not secure 56.228.80.190/gallery.html Finish update

# VisualStories

# Gallery

Images served directly from Amazon S3 storage

Powered by AWS Infrastructure

EC2 S3 Nginx

56.228.80.190/gallery.html

## **6. Verify Hosting.**

- Confirm that HTML pages are served by EC2 .
- Verify that all images load successfully from S3.
- If required, restart httpd using:  
`sudo systemctl restart httpd`

**Date: 12-11-2025**

**Exercise 10: Creation and Configuration of a Custom AWS Virtual Private Cloud (VPC)**

**[Including Public and Private Subnets, Internet Gateway, NAT Gateway, Route Tables]**

**Virtual Private Cloud [VPC]**

It is a logically isolated section of the AWS Cloud where you can launch your AWS resources (like EC2 instances, databases, etc.) in a customized, secure network environment — similar to having your own private data center inside AWS.

It is a virtual network dedicated to your AWS account.

It gives you complete control over your networking environment, including IP address ranges, subnets, route tables, and network gateways.

**Components of a VPC**

| Component                              | Description   |
|--|---|
| CIDR Block (IP Range)                  | The range of IP addresses for your VPC (e.g., 10.0.0.0/16).   |
| Subnets                                | Smaller divisions inside your VPC; can be Public (accessible from internet) or Private (internal only). |
| Internet Gateway (IGW)                 | Allows internet access for resources in public subnets.   |
| Route Tables                           | Define how traffic is directed between subnets and gateways.  |
| NAT Gateway / NAT Instance             | Enables instances in private subnets to connect to the internet without being exposed.                  |
| Security Groups                        | Virtual firewalls that control inbound/outbound traffic at the instance level.                          |
| Network ACLs<br>(Access Control Lists) | Additional firewall at the subnet level.  |
| VPC Peering                            | Connects two VPCs so they can communicate privately.  |

**Default VPC and Custom VPC**

**Characteristics of Default VPC**

When you first create an AWS account, AWS automatically creates a default VPC for you in each region.

| Feature  | Description   |
|----------|---|
| Best For | Beginners, quick testing, learning environments, or temporary setups. |

|                         |  |
|-------------------------|--|
| Created Automatically   | One Default VPC per AWS Region (created by AWS).                             |
| Ready to Use            | You can launch EC2 instances immediately — no setup needed.                  |
| CIDR Block              | Always uses 172.31.0.0/16.   |
| Subnets                 | One default subnet in each Availability Zone within the region.              |
| Internet Connectivity   | Each default subnet is a public subnet<br>(has a route to Internet Gateway). |
| Route Table             | Already configured to connect to the Internet Gateway.                       |
| Security Groups & NACLs | Default ones are automatically created and allow basic communication.        |

### Characteristics of Custom VPC

A Custom VPC is created manually by the user to have complete control over the network configuration.

| Feature               | Description  |
|-----------------------|--|
| Best For              | Production environments, enterprise setups, or multi-tier architectures. |
| Created Manually      | You define the VPC and its settings yourself.                            |
| CIDR Block            | You can choose your own IP range (e.g., 10.0.0.0/16).                    |
| Subnets               | You decide how many subnets, and whether they are public or private.     |
| Internet Connectivity | You attach your own Internet Gateway.                                    |
| Route Tables          | Must be created and configured manually.                                 |
| Security              | You can create custom Security Groups and NACLs as needed.               |

### Subnets

- Subdivisions inside a VPC, used to organize resources.
- Each subnet belongs to one Availability Zone.
- Two types:
  - **Public Subnet** → Connected to Internet Gateway; for web servers.
  - **Private Subnet** → No direct internet access; for databases or internal apps.
- CIDR examples:
  - Public: 10.0.1.0/24
  - Private: 10.0.2.0/24

## Internet Gateway (IGW)

- A gateway that connects your VPC to the Internet.
- Required for instances in a public subnet to receive internet traffic.
- Must be attached to the VPC and referenced in the route table.
- Supports bi-directional communication (inbound and outbound).

## NAT Gateway

- Enables outbound internet access for private subnet instances.
- Allows downloads and updates (e.g., OS patches) without exposing private IPs.
- Deployed inside a public subnet.
- Needs an Elastic IP for internet access.
- Traffic is one-way: private → internet only (not vice versa).

## Route Tables

- Define rules (routes) that determine where network traffic goes.
- Each subnet must be associated with one route table.
- Common routes:
  - For public subnet → 0.0.0.0/0 → Internet Gateway
  - For private subnet → 0.0.0.0/0 → NAT Gateway
- Ensures proper separation between public and private networks.

## Security Groups

- **Instance-level firewalls** controlling inbound and outbound traffic.
- Stateful: if inbound traffic is allowed, corresponding outbound is automatically allowed.
- Rules are based on protocol, port number, and source/destination.
- Example:
  - WebServer-SG: allows HTTP(80), HTTPS(443), SSH(22)
  - Database-SG: allows MySQL(3306) from WebServer-SG only

## Network ACL (Access Control List)

- **Subnet-level firewall**, acts as an additional layer of security.
- **Stateless**: inbound and outbound rules must be defined separately.
- Default NACL allows all traffic; custom NACLs can be restrictive.
- Used for fine-grained control or compliance environments.

## EC2 Instances

- Virtual machines running inside your subnets.
- Public subnet → hosts Web Server (Apache).
- Private subnet → hosts Database Server (MySQL).
- Public EC2s get a public IP; private ones use private IPs only.
- Controlled by their Security Groups and Route Tables.

## Elastic IP (EIP)

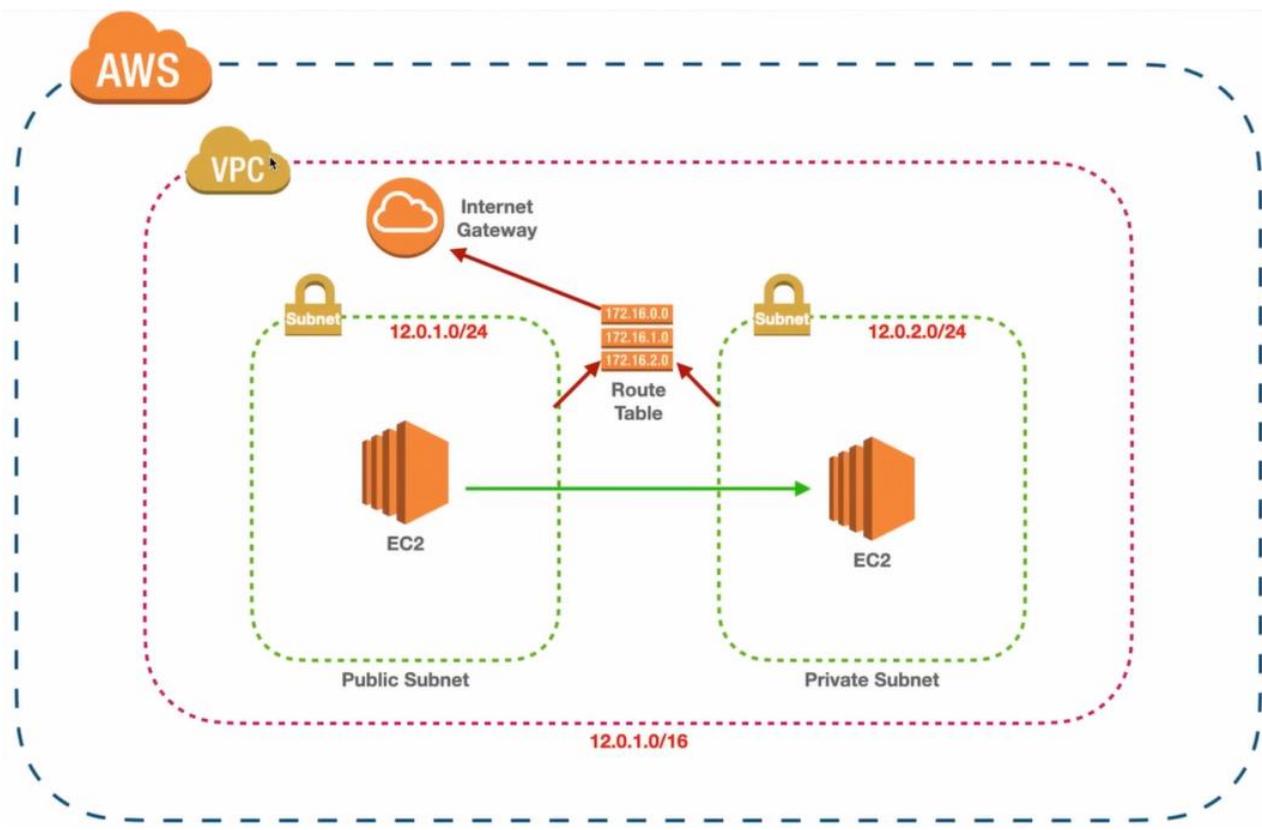
- A **static, public IPv4 address** that can be attached to an EC2 or NAT Gateway.
- Remains constant even if the instance is stopped or restarted.
- Useful for **NAT Gateways**, load balancers, or fixed server access.

| Tier          | Subnet         | Function                   | Internet Access           |
|---------------|----------------|----------------------------|---------------------------|
| Web Tier      | Public Subnet  | Hosts front-end web server | Yes (via IGW)             |
| Database Tier | Private Subnet | Hosts back-end database    | Outbound only (via NATGW) |

## Example: Two-Tier Architecture

Create a new VPC for a web application:

- One **Public Subnet** for web servers (via Internet Gateway)
- One **Private Subnet** for databases (via NAT Gateway)
- Custom route tables and tighter security rules.



## Objective

To design and configure a **Virtual Private Cloud (VPC)** in AWS with:

- One **Public Subnet** for web servers (via **Internet Gateway**)
- One **Private Subnet** for databases (via **NAT Gateway**)
- Separate **Route Tables** for public and private subnets

### Step 1 – Create a New VPC

- Open AWS Management Console → VPC.
- Click Create VPC.
- Choose VPC only option.
- Enter:
  - Name → MyWebAppVPC
  - IPv4 CIDR block → 10.0.0.0/16
  - Tenancy → Default
- Click Create VPC.

This allocates a private IP range for your virtual network.

The screenshot shows the AWS VPC console interface. The top navigation bar includes tabs for 'vpcs' and 'VPCs'. The main content area is titled 'Your VPCs (2)'. A table lists two VPCs: 'MyWebAppVPC' (VPC ID: `vpc-0c7417ef309efa367`, State: Available, Block Public: Off, IPv4 CIDR: 10.0.0.0/16) and a second VPC (VPC ID: `vpc-064b87dde6ec95cda`, State: Available, Block Public: Off, IPv4 CIDR: 172.31.0.0/16). The left sidebar contains a 'VPC dashboard' section and a 'Virtual private cloud' section with links for 'Your VPCs', 'Subnets', 'Route tables', 'Internet gateways', 'Egress-only internet gateways', 'DHCP option sets', 'Elastic IPs', 'Managed prefix lists', 'NAT gateways', 'Peering connections', and 'Route servers'. A 'Security' section is also present. At the bottom, there are links for 'CloudShell', 'Feedback', and 'Console Mobile App', along with a footer containing copyright information and navigation icons.

| Name        | VPC ID                             | State     | Block Public... | IPv4 CIDR     |
|-------------|------------------------------------|-----------|-----------------|---------------|
| -           | <code>vpc-064b87dde6ec95cda</code> | Available | Off             | 172.31.0.0/16 |
| MyWebAppVPC | <code>vpc-0c7417ef309efa367</code> | Available | Off             | 10.0.0.0/16   |

## **Step 2 – Create Subnets**

We'll create two subnets inside the VPC.

### **(a) Public Subnet**

- Go to Subnets → Create subnet.
- Select VPC: MyWebAppVPC.
- Choose Availability Zone A.
- Enter:
  - Name → PublicSubnet
  - IPv4 CIDR block → 10.0.1.0/24
- Click Create subnet.

### **(b) Private Subnet**

- Click Create subnet.
- Select same VPC.
- Choose Availability Zone B.
- Enter:
  - Name → PrivateSubnet
  - IPv4 CIDR block → 10.0.2.0/24
- Click Create subnet.

The screenshot shows the AWS VPC Subnets console. The left navigation pane is collapsed. The main area displays a table titled "Subnets (5) Info" with the following data:

| Name          | Subnet ID                 | State     | VPC                                 |
|---------------|---------------------------|-----------|-------------------------------------|
| -             | subnet-0cba195d4e7237eb9  | Available | vpc-064b87dde6ec95cda               |
| -             | subnet-016b8b7a03aacc189  | Available | vpc-064b87dde6ec95cda               |
| -             | subnet-0f3afe41d701151b5  | Available | vpc-064b87dde6ec95cda               |
| PublicSubnet  | subnet-09ef983bfdfccfb2e0 | Available | vpc-0c7417ef309efa367   MyWebAppVPC |
| PrivateSubnet | subnet-08f939cbaa6fa59a5  | Available | vpc-0c7417ef309efa367   MyWebAppVPC |

Below the table, a message says "Select a subnet". The bottom of the screen shows the AWS footer and various browser tabs.

### Step 3 – Create and Attach an Internet Gateway

- In the left navigation pane, scroll down and click on “Internet Gateways”.
- Internet Gateways → Create Internet Gateway.
  - Name: MyWebApp-IGW
- Click Create Internet Gateway

At this point, the IGW exists but is not yet connected to your VPC.

- Attach the Internet Gateway to Your VPC**
  - Select the IGW you just created (checkbox).
  - Click on the “Actions” drop-down.
  - Choose “Attach to VPC.”
  - From the list, select your **VPC name**, e.g. MyWebAppVPC.
  - Click “Attach Internet Gateway.”

Now the IGW is linked to your VPC and the public subnet can reach the internet.

The screenshot shows the AWS VPC Console with the URL [eu-north-1.console.aws.amazon.com/vpcconsole/home?region=eu-north-1#igws](https://eu-north-1.console.aws.amazon.com/vpcconsole/home?region=eu-north-1#igws). The left sidebar is expanded, showing the 'Virtual private cloud' section with 'Internet gateways' selected. The main area is titled 'Internet gateways (1/2)' and lists two entries:

| Name         | Internet gateway ID   | State    | VPC ID                              |
|--------------|-----------------------|----------|-------------------------------------|
| -            | igw-07b6af7fdc40af3b  | Attached | ypc-064b87dde6ec95cda               |
| MyWebApp-IGW | igw-097a3784f95a165ab | Attached | ypc-0c7417ef309efa367   MyWebAppVPC |

At the bottom of the main area, it says 'igw-097a3784f95a165ab / MyWebApp-IGW'. The bottom navigation bar includes links for CloudShell, Feedback, and Console Mobile App, along with standard browser controls and a footer with copyright information.

## Step 4 – Configure Route Table for IGW

### Public Route Table

- Go to Route Tables → Create route table.
  - Name → PublicRT
  - VPC → MyWebAppVPC
- Under Routes → Edit routes → Add route
  - Destination: 0.0.0.0/0
  - Target: Internet Gateway (MyWebApp-IGW)
- Under Subnet Associations → Edit subnet associations → Select PublicSubnet → Save.

Now the PublicSubnet has internet access.

The screenshot shows the AWS VPC Route Tables console. The left sidebar navigation includes 'VPC dashboard', 'AWS Global View' (with a 'Filter by VPC' dropdown), 'Virtual private cloud' (with 'Your VPCs', 'Subnets', and 'Route tables' selected), and 'Security' (with 'Network ACLs'). The main content area displays a table titled 'Route tables (1/3) Info'. The table has columns: Name, Route table ID, Explicit subnet associ..., Edge associations, and Main. There are three rows: one row with a minus sign and 'rtb-0884b0b70624ccc22', another row with a checkmark for 'PublicRT' and 'rtb-0249ad619ced750fd' associated with 'subnet-09ef983bfdccfb2...', and a third row with a minus sign and 'rtb-07c956189642edbbb'. A search bar at the top of the table says 'Find route tables by attribute or tag'. At the bottom of the table, it says 'rtb-0249ad619ced750fd / PublicRT'. The top right of the page shows 'Account ID: 2501-8976-3899', 'Europe (Stockholm)', and 'Likhitha AWS1'. The bottom of the screen shows the AWS navigation bar with links like CloudShell, Feedback, Console Mobile App, and various AWS service icons.

## Step 5 – Create a NAT Gateway

**NOTE: NAT Gateway must always be created in a **public** subnet**

- Go to NAT Gateways → Create NAT Gateway.
- Name → NAT-GW
- Choose:
  - Subnet → PublicSubnet
  - Elastic IP Allocation ID → Click Allocate Elastic IP
- Click Create NAT Gateway.

This allows instances in private subnet to access the internet (e.g., for updates) without being exposed.

NAT gateways (1) Info

| Name   | NAT gateway ID        | Connectivity... | State   | State message | Primary pub... |
|--------|-----------------------|-----------------|---------|---------------|----------------|
| NAT-GW | nat-03b757fbedb7edd5f | Public          | Pending | -             | -              |

Select a NAT gateway

**Note:** NAT Gateway is a paid resource.

## Step 6 – Configure Route Table for IGW

### Private Route Table

- Create another Route Table → Name PrivateRT.
- Under Routes → Edit routes → Add route
  - Destination: 0.0.0.0/0
  - Target: NAT Gateway (MyWebApp-NATGW)
- Under Subnet Associations → Select PrivateSubnet → Save.

PrivateSubnet traffic goes through the NAT Gateway.

The screenshot shows the AWS VPC Route Tables console. The main view displays the details of a route table named "rtb-0c090721a511e040c / PrivateRT". The "Details" section includes fields for Route table ID (rtb-0c090721a511e040c), Main (No), Owner ID (vpc-0c7417ef309efa367 | MyWebAppVPC), and Explicit subnet associations (subnet-08f939cbaa6fa59a5 / PrivateSubnet). The "Routes" tab is selected, showing two routes:

| Destination | Target                | Status | Propagated | Route Origin       |
|-------------|-----------------------|--------|------------|--------------------|
| 0.0.0.0/0   | nat-03b757fbedb7edd5f | Active | No         | Create Route       |
| 10.0.0.0/16 | local                 | Active | No         | Create Route Table |

The left sidebar navigation includes sections for VPC dashboard, AWS Global View, Virtual private cloud (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only internet gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections, Route servers), Security (Network ACLs), and CloudShell/Feedback/Console Mobile App.

### Note:

Each subnet in a VPC can be associated with **only one route table**.

If a subnet is not explicitly associated with a custom table, it will automatically use the **Main Route Table** created by default with the VPC.

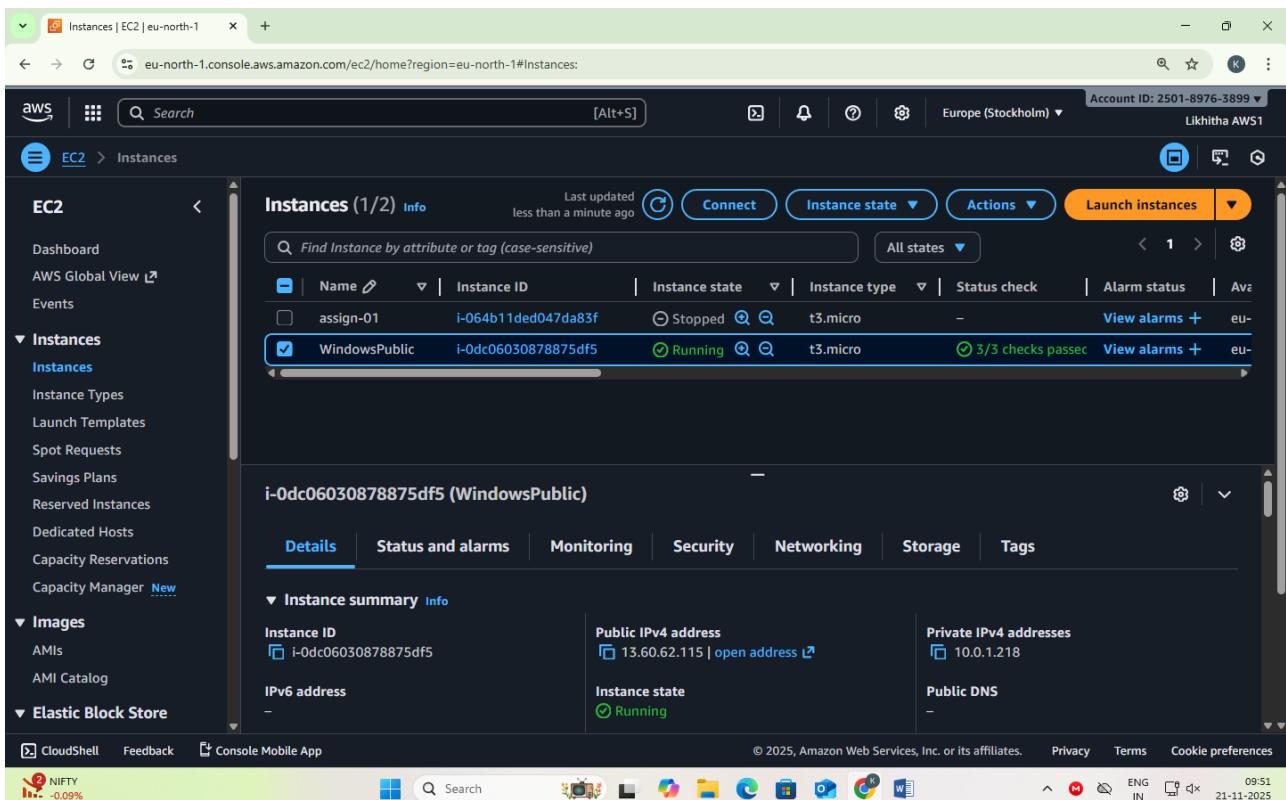
**DATE: 14-11-2025**

## **EXERCISE-11**

### **Launching and Configuring EC2 Instances for Web Server in Public Subnet and Database Server in Private Subnet Using NAT Gateway for Outbound Access**

#### **Step 1 — Launch Windows instance in Public Subnet**

- Go to EC2 → Launch Instance.
- Choose:
  - AMI → Windows Server (Free tier eligible)
  - Instance type → t3.micro
- Select:
  - VPC → MyWebAppVPC
  - Subnet → PublicSubnet
  - Auto assign Public IP → Enable
- Security Group:
  - Create WebInstance-SG
  - Allow:
    - RDP (3389) → Anywhere (for practice)
- Launch instance using key pair (.pem).



## Step 2 — Launch Windows instance in Private Subnet

- Click Launch Instance.
- Choose:
  - Windows Server AMI
- Select:
  - VPC → MyWebAppVPC
  - Subnet → PrivateSubnet
  - Auto assign Public IP → Disable
- Security Group:
  - Create DBInstance-SG
  - Allow RDP only from WebInstance-SG
- Launch instance.

The DBInstance is now **fully isolated** and cannot be accessed directly from the internet.

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays the 'Instances (1/3)' table. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. There are three rows:

| Name                  | Instance ID                | Instance state | Instance type   | Status check             | Alarm status         |
|-----------------------|----------------------------|----------------|-----------------|--------------------------|----------------------|
| assign-01             | i-064b11ded047da83f        | Stopped        | t3.micro        | -                        | View alarms +        |
| <b>WindowsPrivate</b> | <b>i-05d06ee1f2cf9fe3a</b> | <b>Running</b> | <b>t3.micro</b> | <b>3/3 checks passed</b> | <b>View alarms +</b> |
| WindowsPublic         | i-0dc06030878875df5        | Running        | t3.micro        | 3/3 checks passed        | View alarms +        |

Below the table, the details for the selected instance (i-05d06ee1f2cf9fe3a) are shown. The 'Details' tab is active. The 'Instance summary' section includes fields for Instance ID (i-05d06ee1f2cf9fe3a), Public IPv4 address (empty), Private IPv4 addresses (10.0.2.43), IPv6 address (empty), Instance state (Running), and Public DNS (empty).

## Step 3 — Connect to Public Windows Instance

1. Select WebInstance → Click Connect.
2. Choose RDP Client tab.
3. Click Get Password.

4. Upload your .pem key pair.
5. It shows the decrypted Windows Administrator password.
6. Copy the Public IP into a document for further use.

### Connect using RDP

- Open Remote Desktop Connection (mstsc).
- In the dialog:
  - Computer → Public IP of WebInstance
- Click Connect.
- Credentials dialog:
- Username → Administrator
- Password → Paste decrypted password
- Click OK.

You are now inside the public Windows instance.

### Step 4 — connect to private windows instance

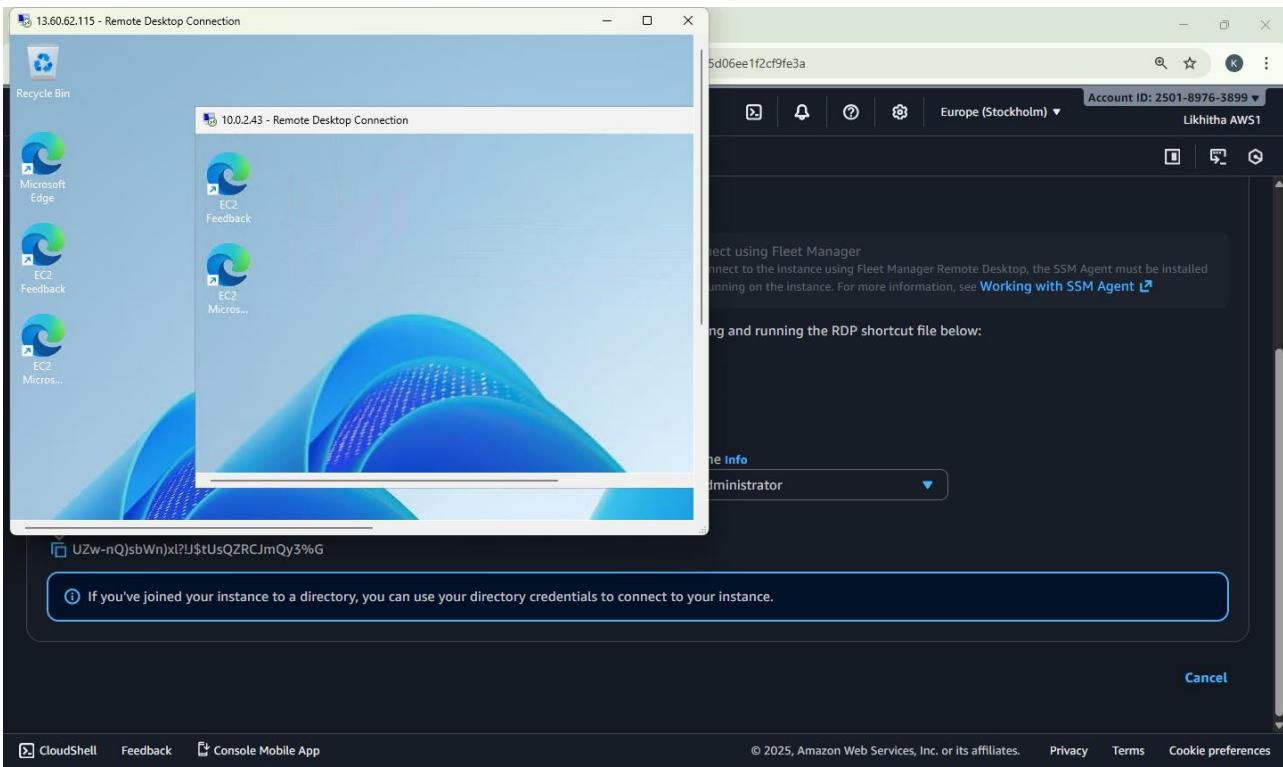
We cannot connect directly from your laptop → No Public IP.

We must connect **from within WebInstance** (jump server / bastion host).

### Remote Login from WebInstance → DBInstance

- While logged into WebInstance, open Remote Desktop Connection. [**mstsc**]
- Enter:
  - Computer → Private IP of DBInstance
- Click Connect.
- Credentials:
  - Username → Administrator
  - Password → Paste the decrypted password (same key pair)
- Click OK.

You are now inside the private Windows instance.



## Step 5 — testing NAT gateway connectivity (Verify Internet Access)

### From WebInstance (Public Subnet)

- Open a browser → Internet should work (through IGW).

### From DBInstance (Private Subnet)

- Open browser → Internet should also work (through NAT Gateway).
- Internet icon will show "Internet Access".

### Security Check

- DBInstance cannot receive inbound traffic from internet.
- Only outbound is allowed via NAT (safe for DB servers).

**Test:** From DBInstance (Private Subnet) → ping Google (Outbound Allowed)

### Steps

Connect to DBInstance (using RDP from WebInstance).

Open Command Prompt inside DBInstance.

Type: ping google.com

### Expected Result

- It will successfully ping google.com
- You will see replies like:  
Reply from 142.250.xxx.xxx: bytes=32 time=20ms TTL=115

### Why does this work?

- Outbound traffic is allowed from Private Subnet → NAT Gateway → Internet.
- NAT Gateway acts as a proxy for the private instance.
- So the private instance can reach internet,  
but internet cannot reach the private instance.

### What will NOT work?

From the Private DBInstance: ping <Your Own Public IP>  
or anyone trying to ping: ping <DBInstance Private IP>

Both will fail because:

- Inbound traffic is blocked
- DBInstance has no Public IP
- SG allows inbound only from WebInstance-SG

**Private instance → internet = YES (via NAT Gateway)**

**Internet → private instance = NO (fully blocked)**

Because NAT Gateway is **one-way** only.

### Elastic IP address

- A static IP address is a permanent address that doesn't change. You can manually configure a device to have a static IP address.
- An Elastic IP address is static and has to be used in a specific Region, it cannot be moved to a different Region.
- An Elastic IP address comes from Amazon's pool of IPv4 addresses.
- To use an Elastic IP address, you first allocate one to your account, and then associate it with your instance or a network interface.
- When you associate an Elastic IP address with an instance or its primary network interface, if the instance already has a public IPv4 address associated with it, that public IPv4 address is released back into Amazon's pool of public IPv4 addresses and the Elastic IP address is associated with the instance instead.
- You can disassociate an Elastic IP address from a resource, and then associate it with a different resource.
- A disassociated Elastic IP address remains allocated to your account until you explicitly release it.

- You are charged for all Elastic IP addresses in your account, regardless of whether they are associated or disassociated with an instance.
- Static IP addresses are especially important in cases where a device has to be quickly found over the internet on a permanent basis.
- Web Servers: A website must have one or more static IP addresses to be assigned to the domain always point to the correct server.

**DATE: 19-11-25**

## **Exercise–12 Deploying and Testing a Load-Balanced Web Application**

By Creating a Web Server, Building a Custom AMI, Configuring a Target Group, Launching an Application Load Balancer (ALB), Registering the Instance in the Target Group, and Testing the ALB DNS

### **Objective**

To deploy a scalable and highly available web application on AWS by configuring:

- EC2 web server
- Custom AMI
- Target Group
- Application Load Balancer (ALB)
- Testing the ALB DNS

### **Description for Each Component**

#### **EC2 Web Server**

**What:** A virtual Linux machine that hosts your web application files.

**Why:** It serves the actual website content that users access through the ALB.

#### **Custom AMI**

**What:** A snapshot/template of your configured EC2 instance (with Apache + website files).

**Why:** Ensures every new instance launched by the Auto Scaling Group has the same setup automatically.

#### **Target Group**

**What:** A collection of EC2 instances that the ALB sends traffic to.

**Why:** It allows the load balancer to forward requests only to healthy instances in the Auto Scaling Group.

#### **Application Load Balancer (ALB)**

**What:** A managed service that distributes incoming HTTP traffic across multiple EC2 instances.

**Why:** Ensures high availability and balanced traffic distribution for the web application.

#### **Security Group**

**What:** A virtual firewall controlling inbound/outbound traffic to EC2 instances and ALB.

**Why:** Ensures only required traffic (HTTP/SSH) is allowed for the application components.

#### **VPC + Subnets**

**What:** A private network environment where all resources run.

**Why:** Provides isolation, routing, and a structured network setup for public & private components.

## Launch Template

**What:** A reusable blueprint containing AMI, instance type, key pair, and security group settings.

**Why:** The ASG uses this template to launch identical EC2 instances.

## STEP 1 — Launch Base EC2 Instance

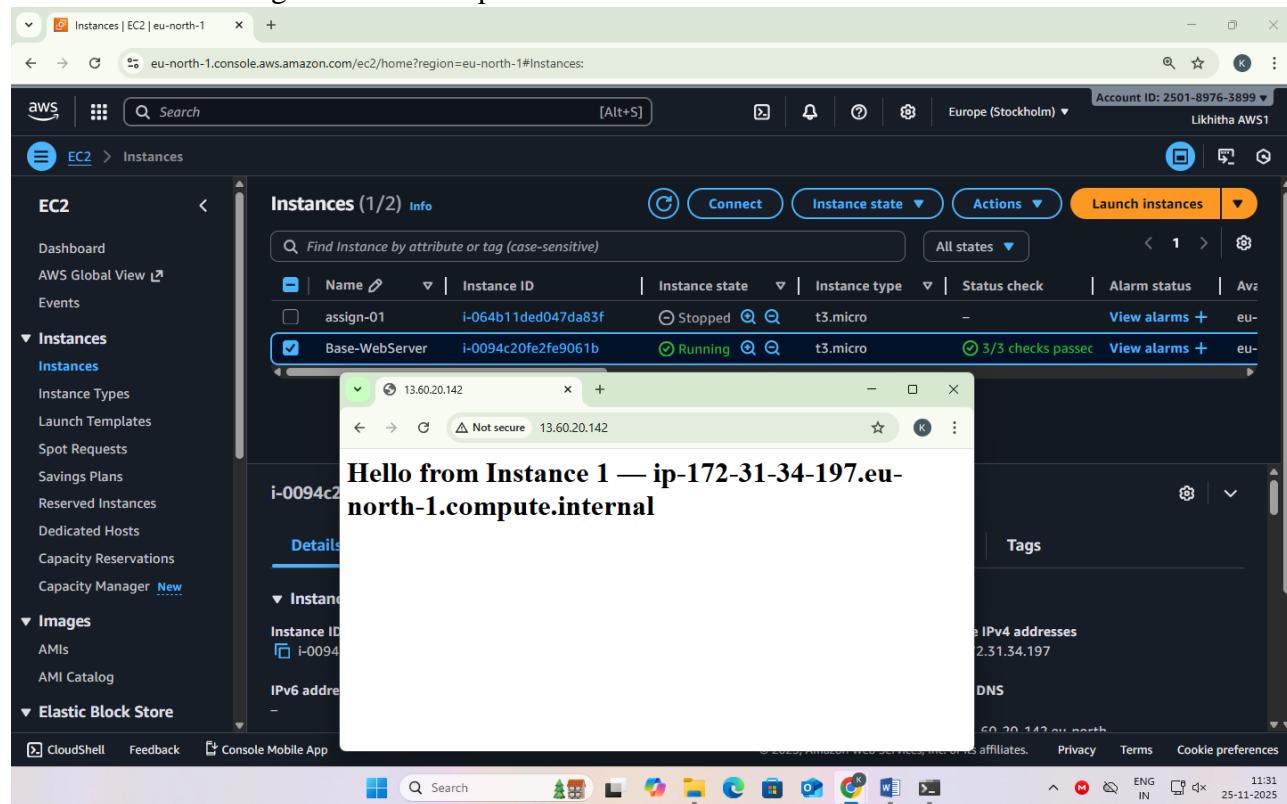
- Open AWS Console → EC2 → Launch Instance
- Name: Base-WebServer
- AMI: Amazon Linux 2
- Instance type: t3.micro
- Key Pair: your .pem file
- Security Group:
  - HTTP (80) → Anywhere
  - SSH (22) → My IP
- Launch the instance

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 of instances. One instance is selected: "Base-WebServer" (i-0094c20fe2fe9061b). The instance details show it's running, has a Public IPv4 address of 13.60.20.142, and a Private IPv4 address of 172.31.34.197. The Networking tab is active. At the bottom, there are links for CloudShell, Feedback, and Console Mobile App, along with a footer for 2025, Amazon Web Services, Inc. or its affiliates, and various system status icons.

## STEP 2 — Install Apache and Create Web Page

- SSH into the instance.
- Install Apache:
  - sudo yum install httpd -y
  - sudo systemctl start httpd
  - sudo systemctl enable httpd
- Move to Web Root Folder - cd /var/www/html
- Create a webpage with hostname:
  - echo "<h1>Hello from Instance 1 — \$(hostname)</h1>" | sudo tee /var/www/html/index.html

Test in browser using the instance's public IP.



## STEP 3 — Create Custom AMI

- Go to EC2 → Instances
- Select Base-WebServer
- Actions → Image and Templates → Create Image
- Name: WebServer-AMI
- Create
- Wait until AMI status = Available

This AMI now contains Apache + your index.html.

The screenshot shows the AWS EC2 AMI Management interface. On the left, a sidebar navigation includes EC2, Instances, Images (selected), and Elastic Block Store. The main area displays 'Amazon Machine Images (AMIs) (1/1)' with a single entry: 'WebServer-AMI'. The details page for this AMI is shown, with tabs for Details, Permissions, Storage, My AMI usage, AMI ancestry - new, and Tags. The 'Details' tab is active, showing the following information:

| AMI ID                | Image type       | Platform details      | Root device type    |
|-----------------------|------------------|-----------------------|---------------------|
| ami-0e05eb6baf071c1e4 | machine          | Linux/UNIX            | EBS                 |
| AMI name              | Owner account ID | Architecture          | Usage operation     |
| WebServer-AMI         | 250189763899     | x86_64                | RunInstances        |
| Root device name      | Status           | Source                | Virtualization type |
|                       | green            | ami-0e05eb6baf071c1e4 |                     |

## STEP 4 — Create Target Group

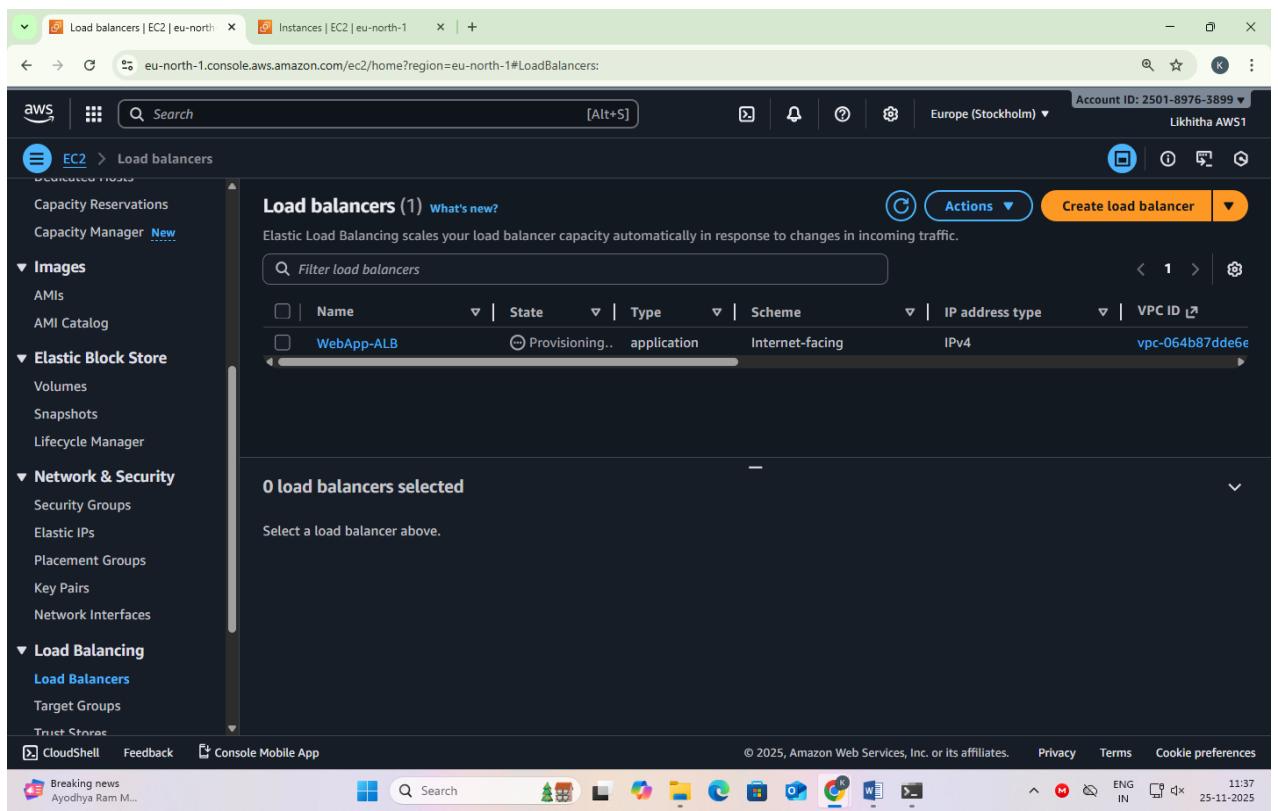
- EC2 → Target Groups → Create Target Group
- Target type: Instances
- Name: WebApp-TG
- Protocol: HTTP
- Port: 80
- Health Check Path: /
- Create (do not register instances manually)

**Purpose:** Target Group holds the list of EC2 instances behind the Load Balancer.

The screenshot shows the AWS EC2 Target groups page. The left sidebar navigation includes: Capacity Reservations, Capacity Manager, Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), and Load Balancing (Load Balancers, Target Groups). The 'Target Groups' section is currently selected. The main content area displays a table titled 'Target groups (1)'. The table has columns: Name, ARN, Port, Protocol, Target type, and Load b. One row is listed: 'WebApp-TG' with ARN 'arn:aws:elasticloadbalancing...', Port '80', Protocol 'HTTP', Target type 'Instance', and Load b 'Non'. A 'Actions' dropdown menu is open above the table. Below the table, a message says '0 target groups selected' and 'Select a target group above.' The bottom of the screen shows the AWS footer with links for CloudShell, Feedback, and Console Mobile App, along with copyright information and user details.

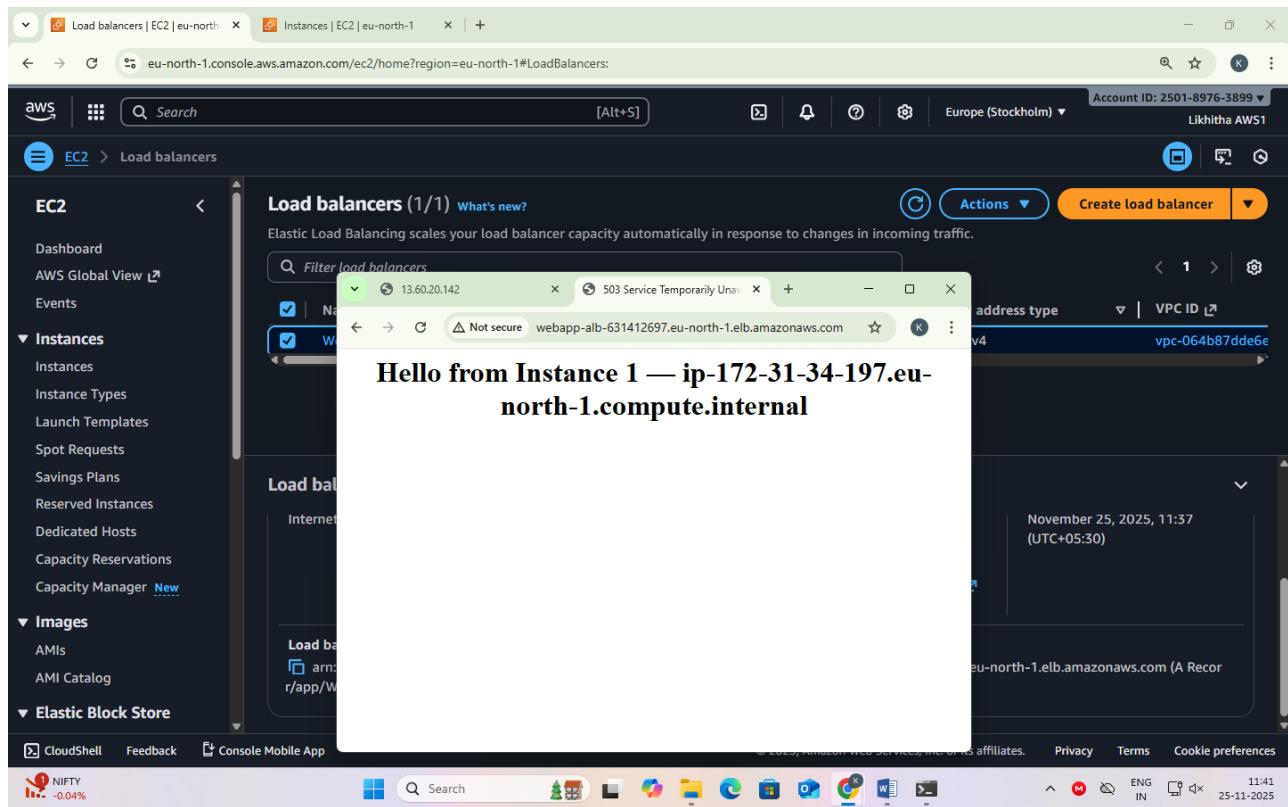
## STEP 5 — Create Application Load Balancer (ALB)

- EC2 → Load Balancers → Create Load Balancer
- Choose Application Load Balancer
- Name: WebApp-ALB
- Scheme: Internet-facing
- Listeners: HTTP (80)
- Select two public subnets
- Security Group: allow HTTP (port 80)
- Forward to Target Group → WebApp-TG
- Create



## STEP 6 — Test the Load Balancer Using ALB DNS

- Go to AWS Console → EC2 → Load Balancers
  - Select your ALB: ALB-WebServer
- Copy the ALB DNS Name
  - You will see something like:
  - WebApp-ALB-123456789.ap-south-1.elb.amazonaws.com
- Open a Browser and Paste the DNS Name
  - Enter: http://<your-alb-dns-name>
  - Example: http://WebApp-ALB-123456789.ap-south-1.elb.amazonaws.com
- View the Output
  - You should see the webpage you created earlier:
  - Hello from Instance 1 — ip-xx-xx-xx-xx
  - This confirms that: The ALB is working
  - The Target Group is routing traffic
  - The instance created from your custom AMI is serving the web page.



- Optional – Refresh Multiple Times
  - If you later use Auto Scaling with multiple instances:
  - Clicking Refresh will show different hostnames
  - This confirms load balancing across multiple servers

**DATE: 20-11-25**

## **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

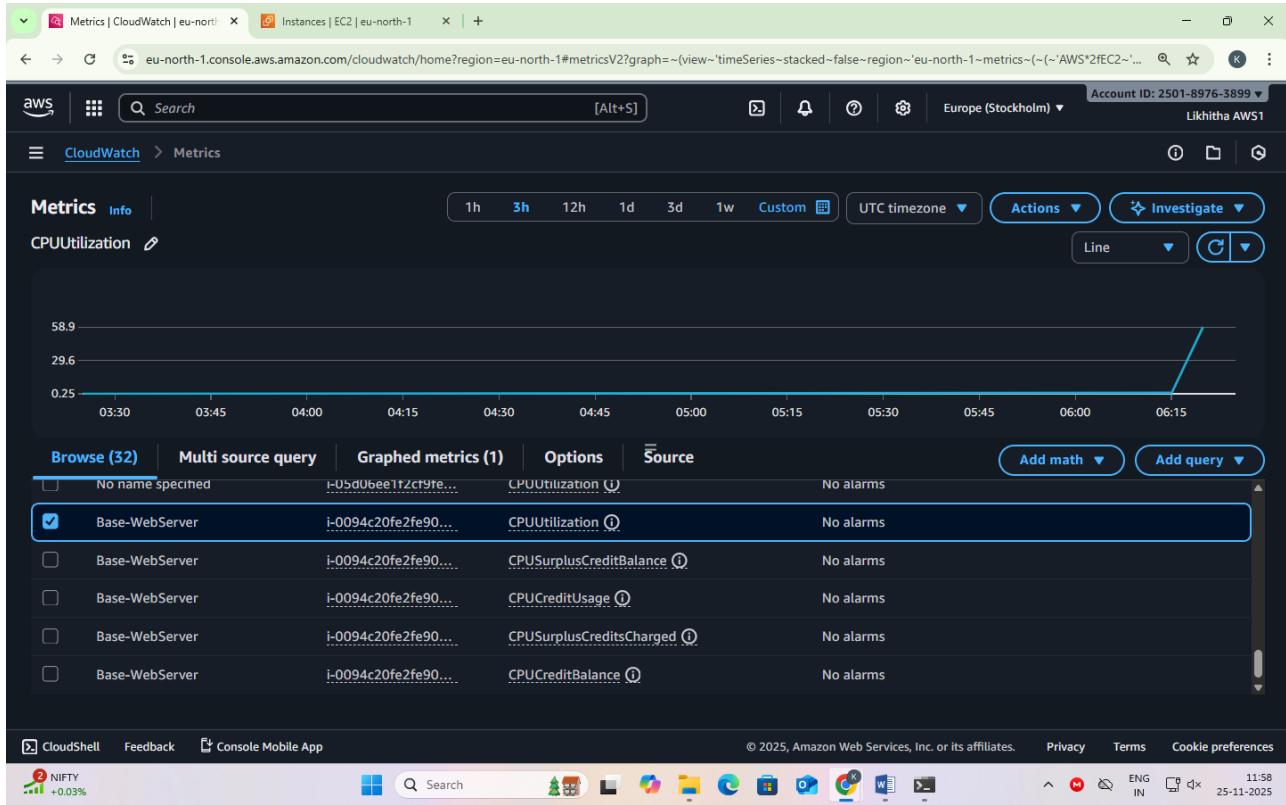
Go to CloudWatch → Metrics → EC2 → Per-Instance Metrics

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.

**DATE: 21-11-25**

### **Exercise–14 Mini Project –**

**Deploying a Load-Balanced Web Application using Application Load Balancer (ALB), Auto Scaling Group (ASG), Custom AMI, and Target Group on AWS with CloudWatch Alarms & Stress Testing for Auto Scaling Validation.**

#### **Auto Scaling Group (ASG)**

**What:** A service that automatically launches or terminates EC2 instances based on demand.

**Why:** Provides scalability so your application can handle variable traffic.

#### **CPU-Based Scaling Policies**

**What:** Rules that add or remove EC2 instances based on CPU usage thresholds.

**Why:** Ensures your application automatically scales up during heavy load and scales down to save cost.

#### **Stress Testing (using stress/stress-ng tool)**

**What:** A method to artificially increase CPU load on instances.

**Why:** Used to validate whether the Auto Scaling Group is scaling up/down correctly.

### **STEP 7A — Create Launch Template**

Go to EC2 → Launch Templates → Create launch template

Template Name

- Launch template name: LT- WebServer

Choose AMI (Custom AMI)

Under Application and OS Images (AMI):

- Click My AMIs
- Select your custom AMI: AMI-WebServer (the AMI you created earlier)

Instance Type

Under Instance type: Select t3.micro

Key Pair

Under Key pair (login): Select your existing key pair (pemfile1.pem)

Security Group

Under Network settings → Firewall (security groups):

- Choose Select existing security group
- Select a security group that allows:
  - SSH (Port 22)

- HTTP (Port 80)

Example: launch-wizard-1 (already has both rules)

Storage

Leave default: 8 GiB gp3 (root volume)

Create Template

Click Create launch template

**Launch Template is now ready to be used by the Auto Scaling Group.**

## **STEP 7B — Create Auto Scaling Group (ASG)**

(Using the Launch Template you created in Step 7A)

Open ASG Wizard

Go to EC2 → Auto Scaling Groups → Create Auto Scaling Group

Name and Choose Launch Template

Auto Scaling group name: ASG-WebServer

Launch template: Choose LT-WebServer

Click Next.

Select Network (VPC + Subnets)

Since we are using default VPC, select:

VPC: vpc-0c952a6cabfbe594b (Default VPC)

Subnets (select ANY 2)

Example: ap-south-1a and ap-south-1b

(These are public subnets in default VPC — good for web servers)

Click Next.

Attach Load Balancer

Under Load balancing options:

Select: Attach to an existing load balancer

Under Select load balancers to attach:

Choose: Choose from your load balancer target groups

Under dropdown:

Select your Target Group: TG-WebServer (or whatever name you created)

AWS will automatically show:

- Load balancer: ALB-WebServer
- Type: Application/HTTP

Click Next.

Configure Group Size

Set:

Desired capacity = 1

Minimum capacity = 1

Maximum capacity = 3

We are not adding scaling policies now (stress test comes later), so:

Select No scaling policies

Click Next.  
Skip Notifications and Tags

No changes.

Click Next.

Review and Create

Check:

- Launch template = LT-WebServer
- Load balancer = ALB-WebServer
- Target group = TG-WebServer
- Subnets = 2 selected
- Desired = 1 instance

Then click: Create Auto Scaling Group

## ASG Creation Done

### Check if ASG launched an instance.

Verify ASG Instance Creation

Go to EC2 → Auto Scaling Groups → ASG-WebServer

Open it and check: Under Activity tab

You should see: Successful — Launching a new EC2 instance

Under Instances tab: You should see an instance like: i-xxxxxxxxxxxxx (Running)

The screenshot shows the AWS EC2 Auto Scaling Groups page. The left sidebar is collapsed. The main area displays the 'Auto Scaling groups' section with one entry:

| Name          | Launch template/configuration  | Instances | Status | Desired capacity |
|---------------|--------------------------------|-----------|--------|------------------|
| ASG-WebServer | LT-WebServer   Version Default | 1         | -      | 1                |

Below the table, a message says '0 Auto Scaling groups selected'. At the bottom, there are links for CloudShell, Feedback, and Console Mobile App, along with copyright information and a footer with various icons and language settings.

This instance was launched automatically by the ASG using custom AMI through the Launch Template.

Now that ASG is created and one instance is running, we will add Auto Scaling Policies so the group can:

- Scale OUT (add more instances when CPU is high)
- Scale IN (remove instances when CPU is low)

## STEP 8 — Configure Auto Scaling Policies (Scale Out & Scale In)

### PART 1 – Create High-CPU Alarm (for Scale-OUT)

Open CloudWatch and start alarm

Go to CloudWatch → left menu Alarms → All alarms.

Click Create alarm.

Choose metric (CPU of ASG)

Click Select metric.

Navigate:

EC2 → By Auto Scaling Group (or By AutoScalingGroupName).

Click your group ASG-WebServer.

Select CPUUtilization.

Click Select metric.

Set condition (CPU > 60)

Statistic: Average

Period: 1 minute

Under Conditions:

Threshold type: Static

“Whenever CPUUtilization is...” → choose Greater

“than...” → 60

In Additional configuration, datapoints to alarm: 1 out of 1 (default is fine).

Click Next.

Skip actions (no SNS, no ASG here)

On Configure actions page, do NOT add anything.

Just click Next.

If a popup says “*No actions configured*”, click Proceed without actions.

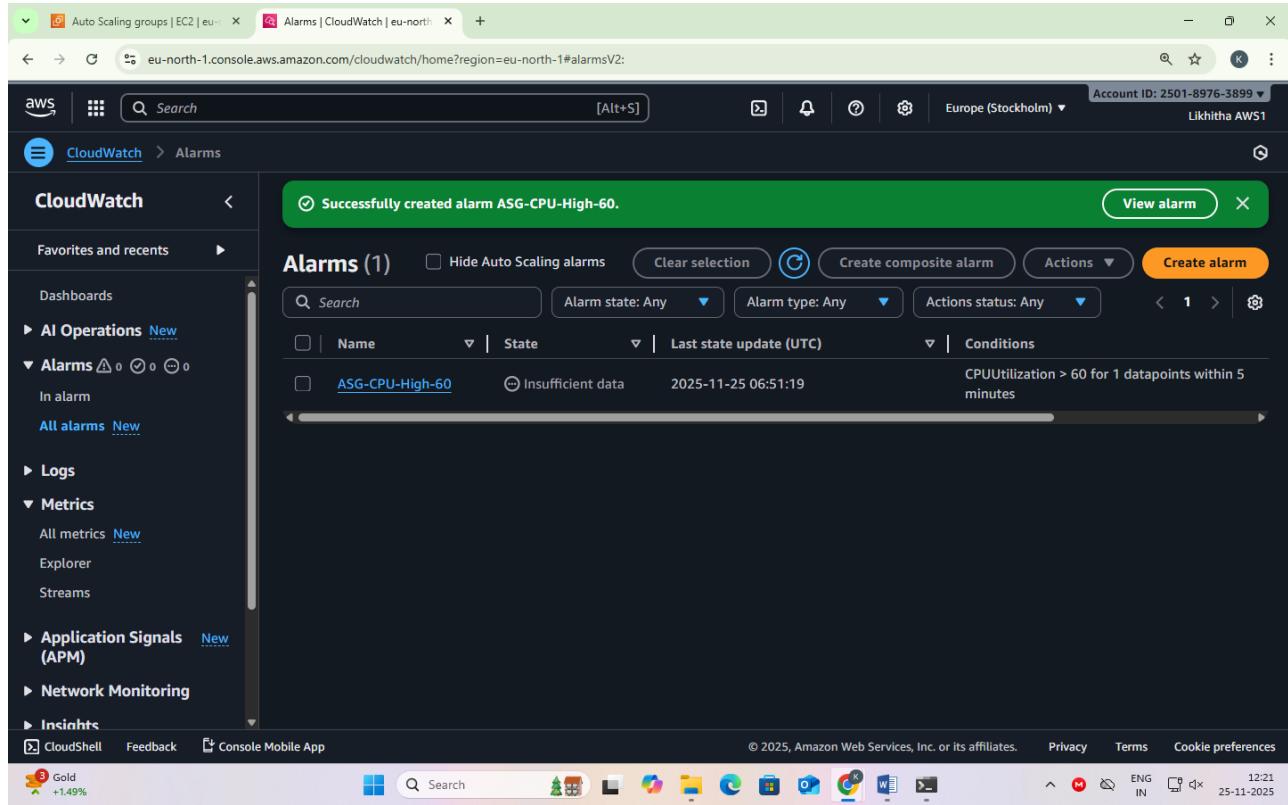
Name and create

Alarm name: ASG-CPU-High-60

Description (optional): CPUUtilization > 60% for 1 minute (ASG-WebServer)

Click Next, then Create alarm.

You now have one metric alarm with no actions.



## PART 2 – Attach High-CPU alarm to ASG as Scale-OUT Policy

Open ASG automatic scaling

Go to EC2 → left menu Auto Scaling groups.

Click ASG-WebServer.

Go to the Automatic scaling tab.

Create Scale-OUT policy

Click Create dynamic scaling policy.

Policy type: Simple scaling

Scaling policy name:

CloudWatch alarm:

Click the dropdown and select ASG-CPU-High-60.

Take the action:

Action: Add

Value: 1

Unit: capacity units

And then wait: 300 seconds (default is fine).

Click Create.

Now your ASG knows:  
When alarm ASG-CPU-High-60 goes to ALARM → Add 1 instance.

The screenshot shows the AWS CloudWatch Auto Scaling group details page for the 'ASG-WebServer' group. The left sidebar lists various AWS services. The main content area shows the 'Automatic scaling' tab selected. A callout box highlights the 'Dynamic scaling policies' section, which contains one policy named 'ScaleOut-CPU60'. The policy details are as follows:

- Policy type:** Simple scaling
- Enabled or disabled:** Enabled
- Execute policy when:** ASG-CPU-High-60
- breaches the alarm threshold:** CPUUtilization > 60 for 1 consecutive periods of 300 seconds for the metric dimensions: AutoScalingGroupName = ASG-WebServer
- Take the action:** Add 1 capacity units

### PART 3 – Create Low-CPU Alarm (for Scale-IN)

Create another alarm in CloudWatch

Go again to CloudWatch → Alarms → Create alarm.

Select metric exactly like before:

EC2 → By Auto Scaling Group → ASG-WebServer → CPUUtilization → Select metric.

Set condition (CPU < 20)

Statistic: Average

Period: 1 minute

Conditions:

Threshold type: Static

“Whenever CPUUtilization is...” → Lower

“than...” → 20

Datapoints to alarm: 1 out of 1 (default).

Click Next.

Skip actions again

Do not add SNS / Auto Scaling / EC2 actions.

Click Next → if popup appears, Proceed without actions.

Name and create

Alarm name: ASG-CPU-Low-20

Description (optional): CPUUtilization < 20% for 1 minute (ASG-WebServer)

Next → Create alarm.

#### **PART 4 – Attach Low-CPU alarm to ASG as Scale-IN Policy**

Open ASG automatic scaling again

Back to EC2 → Auto Scaling groups → ASG-WebServer.

Go to Automatic scaling tab.

Create Scale-IN policy

Click Create dynamic scaling policy.

Policy type: Simple scaling

Scaling policy name: scalein-cpu20

CloudWatch alarm:

Choose ASG-CPU-Low-20 from the dropdown.

Take the action:

Action: Remove

Value: 1

Unit: capacity units

And then wait: 300 seconds.

Click Create.

#### **CHECK the resources created in STEP 8-**

In EC2 → Auto Scaling groups → ASG-WebServer → Automatic scaling

- You should see:
- ScaleOut-CPU60 – Simple scaling – Add 1 capacity units – Alarm: ASG-CPU-High-60
- scalein-cpu20 – Simple scaling – Remove 1 capacity units – Alarm: ASG-CPU-Low-20

In CloudWatch → Alarms

- ASG-CPU-High-60
- ASG-CPU-Low-20
- Both will show Actions: No actions – this is OK because ASG is using them.

## **STEP 9 — Perform Stress Testing for Auto Scaling Validation**

To artificially increase CPU load on EC2 instances and observe automatic scaling behavior triggered by CloudWatch alarms.

### **Part A — Connect to the ASG-launched EC2 Instance**

Go to AWS Console → EC2 → Instances  
Select the instance launched by ASG  
(NOT the original base instance you created manually)  
Click Connect  
Choose EC2 Instance Connect → Connect

### **Part B — Install Stress Testing Tool**

For Amazon Linux 2023:  
`sudo dnf install stress-ng -y`  
Confirm installation:  
`stress-ng --version`

### **Part C — Apply CPU Load (Trigger Scale Out)**

Run stress tool for 2 minutes with 4 CPU workers:  
`stress-ng --cpu 4 --timeout 120`  
✓ This will spike CPU usage above **60%**  
✓ Your Scale-Out Policy should activate  
✓ CloudWatch alarm → ALARM state → ASG launches 1 more instance

### **Part D — Monitor Auto Scaling Activity**

Check scaling progress in:

| Service                              | What to Monitor                               |
|--------------------------------------|---|
| EC2 → Auto Scaling Groups → Activity | Shows launching of new instance               |
| EC2 → Instances                      | New instance appears with different host-name |
| CloudWatch → Alarms                  | High CPU alarm turns ALARM                    |

Scaling may take **2–4 minutes**  
Refresh page to see updates

### **Part E — Test Load Balancing**

Copy ALB DNS Name → paste in browser → refresh multiple times:  
You should see:

Hello from Instance 1 — ip-xx-xx-xx-xx

Hello from Instance 2 — ip-aa-aa-aa-aa

✓ Confirms ALB is distributing traffic across multiple instances

## Part F — Scale In (CPU comes down)

After 2 minutes, stress test stops automatically → CPU drops below 20%

✓ Low CPU alarm triggers

✓ ASG removes the extra instance

✓ Only 1 instance remains (desired capacity)

Scaling-in may take 3–5 minutes

### Extra points

This exercise imitates a production-grade architecture used in companies.

#### What is a Target Group?

A Target Group is a collection of EC2 instances that receive traffic from the Application Load Balancer (ALB).

It defines:

- Which instances to send traffic to
- On which port (example: 80)
- Health check rules (path /, interval, threshold)

In simple words:

Target Group = list of servers behind the load balancer.

#### Why do we need two steps: Create Launch Template and Create Auto Scaling Group?

Because both have different roles:

Launch Template = WHAT to launch

It contains the configuration of one EC2 instance, such as:

- AMI
- Instance type
- Security Group
- Key pair
- Storage
- User data

This is just a blueprint.

Auto Scaling Group = WHEN & HOW MANY to launch

The ASG uses the Launch Template to automatically:

- Launch new instances
- Remove instances
- Maintain desired capacity

- Scale based on CPU/memory demand

### **Where to find http://<alb-dns-name>?**

You can find the ALB DNS Name in the AWS Console:

Path: EC2 → Load Balancers → Select your ALB → Description tab

There you will see: DNS name: mywebapp-alb-12345678.ap-south-1.elb.amazonaws.com

Use it in your browser as: http://mywebapp-alb-12345678.ap-south-1.elb.amazonaws.com

This is the public URL of your Load Balancer.

### **What is ALB DNS?**

ALB DNS is the publicly accessible DNS name automatically created by AWS for your Application Load Balancer.

Example: myapp-alb-123456789.ap-south-1.elb.amazonaws.com

When a user types this URL:

- Traffic goes to the ALB
- ALB forwards to Target Group
- Target Group sends request to one of the EC2 instances

ALB DNS = The website URL of your Load Balancer.

DNS means Domain Name System.

DNS is a system that converts domain names to IP addresses.

Example: www.amazon.com → 52.95.120.1

A DNS Server is only one part of this system.

DNS = System that translates names to IPs.

DNS Server = Machine that performs the translation.

### **Why Do we NEED CloudWatch alarms for ASG scaling exercise?**

We are using “Simple Scaling”

- Based on CPU > 60 and CPU < 20
- Which requires manually created CloudWatch alarms

Without the alarms, ASG has NO trigger, so it cannot scale.

### **Delete these resources in this exact order**

To avoid dependency errors:

#### **1. Delete Auto Scaling Group (ASG)**

- EC2 → Auto Scaling Groups
- Select ASG-WebServer
- Actions → Delete

This will automatically terminate any instances managed by ASG.

## **2. Delete Launch Template**

- EC2 → Launch Templates
- Select your template
- Actions → Delete template

## **3. Delete Application Load Balancer**

- EC2 → Load Balancers
- Select your ALB
- Actions → Delete

## **4. Delete Target Group**

- EC2 → Load Balancing → Target Groups
- Select your target group
- Actions → Delete

## **\*\*5. Delete the Custom AMI**

- EC2 → AMIs
- Select your AMI → **Deregister**
- Check Snapshots → delete the related snapshot also.

## **6. Delete any Manual EC2 Instances and other resources**

If you still have any instances running:

- EC2 → Instances → Select → **Terminate**

### **Resource clean-up NOTE:**

- **ALB** charges per hour
- **ASG instances** can launch unexpectedly
- **Custom AMI snapshot** costs storage
- **Running EC2** will cost hourly

**DATE: 21-11-25**

## **Exercise-15:**

### **Hosting a WordPress Content Management Website on Amazon Lightsail.(Only Demo)**

**Amazon LightSail** is a beginner-friendly cloud platform in AWS that provides:

- Virtual servers (simplified EC2)
- Fixed monthly pricing (predictable cost)
- Simple UI and easy setup
- One-click application launch (WordPress, LAMP, Node.js, etc.)
- Built-in networking (Static IP, DNS, Firewall)
- Automatic SSH terminal
- Snapshots (backups)

It is ideal for:

- Personal websites
- Student projects
- Small web apps
- WordPress blogs
- Quick prototyping

### **Why Lightsail Instead of EC2?**

Lightsail = “EC2 with all hard things simplified.”

| Feature | EC2                    | Lightsail                 |
|---------|------------------------|---------------------------|
| Pricing | Pay per hour           | Fixed monthly             |
| Setup   | Complex (VPC, SG, AMI) | Simple                    |
| SSH     | Need .pem or PuTTY     | Built-in SSH              |
| DNS     | Basic                  | Easy DNS panel            |
| Apps    | Install manually       | One-click WordPress, LAMP |

### **3. What is WordPress in Lightsail?**

WordPress is a full-stack content management system (CMS) pre-installed on Lightsail.

It includes:

Frontend (UI)

- HTML, CSS, JS
- Themes
- Templates
- Page layouts

Backend

- PHP (server-side logic)

- WordPress core engine

Database

- MySQL/MariaDB

Stores pages, media, lessons, posts, settings.

Server

- Apache web server

Hosting

- Lightsail VM + static IP + firewall

Thus, WordPress is a full-stack application (LAMP stack).

#### **4. Lightsail Free-Tier Eligibility**

- 3 months free
- Only for the 512 MB RAM plan
- Suitable for WordPress, but slightly slow
- After 3 months → ~₹350–₹400/month
- recommended: 1 GB RAM plan.

### **PART A — Create Server on Lightsail**

Step 1 — Open Lightsail Console

AWS Console → Search “Lightsail” → Open it.

Lightsail dashboard appears.

Step 2 — Create Instance

Click Create Instance.

Step 3 — Select Platform:

Choose Linux/Unix. (WordPress works best on Linux)

Step 4 — Select Blueprint → App + OS → WordPress

[Lightsail will install automatically:

- Apache
- PHP
- MySQL/MariaDB
- WordPress application

No manual setup required.]

Step 5 — Choose Instance Plan

512 MB RAM (free-tier for 3 months but slower)

Step 6 — Name the Instance

Give a name (example MyCMS)

Click Create Instance.

Step 7 — Wait Until Status = Running

Takes 1–2 minutes.

## PART B — Access and Configure WordPress

### Step 8 — Connect Through Browser-Based SSH

Click **Connect** → **Connect using SSH**

(Lightsail gives built-in SSH. No need for .pem file.)

### Step 9 — Retrieve WordPress Admin Password

In SSH terminal, run:

```
cat bitnami_application_password
```

This outputs a long randomly generated password.

Copy it — you need it for login. [example: INBV03eMcWX:]

### Step 10 — Open Your WordPress Website

Copy the **Public IP** from the instance details.

Visit: <http://<your-lightsail-ip>>

You will see a WordPress homepage.

### Step 11 — Login to WordPress Admin

Visit: <http://<your-lightsail-ip>/wp-admin>

Login:

- Username: **user**
- Password: *paste the password from SSH*

You now have full access to your site's backend.

## PART C — Build ‘CMS Website’ [Experiential learning]

### Step 12 — Install a Clean Education Theme

Go to:

Dashboard → Appearance → Themes → Add New

Recommended theme: Astra (best)

Activate the theme. [install and activate]

### Step 13 — Create Main Website Pages

Dashboard → Pages → Add New

Create:

1. Home
2. Courses & Subjects
3. Lesson Notes
4. Assignments
5. Study Material (PDFs/PPTs)
6. Contact

### Step 14 — Create Lesson-Wise Pages

### Step 15 — Upload PDFs, PPTs, Notes

Go to:

Dashboard → Media → Add New

Upload:

- PDFs
- DOCX files
- PPTs
- Images
- Notes

Then insert into pages using “Add Media”.

#### Step 16 — Create Menus

Dashboard → Appearance → Menus

Add:

Home | Courses | Notes | Assignments | Contact

Add submenus under Courses and Notes.

### PART D — Secure & Finalize

#### Step 18 — Attach Static IP

Lightsail → Networking → Create Static IP → Attach to instance.

Reason:

Without this, your IP changes → site breaks.

#### Step 19 — Take Snapshot

Lightsail → Snapshots → Create snapshot.

Acts as a backup of your entire site.

**DATE: 21-11-25**

## **Exercise–16: Building a Basic Python Flask Web Application — Fundamentals of Web Requests & Form Handling (Pre-Requisite for AWS RDS Connectivity Lab)**

[Prerequisite for AWS RDS exercise. This particular exercise is a local Python Flask exercise. No AWS service is used here.]

To design and implement a simple Web Application using Python Flask that:

- Displays an HTML form to accept Name and Password.
- Reads the form data on the server side when the form is submitted.
- Displays the submitted values on a new result page.

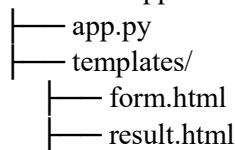
After completing this exercise, you will be able to:

- Understand how an HTML form sends data to the server.
- Explain how Flask receives form data using the request object.
- Read request parameters (request.form) in Flask.
- Understand basic frontend-backend communication.
- Handle a simple HTTP POST request in a Flask application.

### **Folder Structure**

Create the following structure manually:

FlaskFormApp/



### **Requirements**

- Python installed (3.x)
- Flask library
- Any code editor (VS Code / PyCharm / Notepad etc.)
- Web browser (Chrome / Edge / Firefox)

### **STEP 1: Install Flask**

Open **Command Prompt / Terminal** and run:  
pip install flask

### **STEP 2: Create Project Folder**

Create a folder, for example:  
C:\Users\MCA\FlaskFormApp

### **STEP 3: Create templates Folder**

Inside C:\Users\MCA\FlaskFormApp, create a sub-folder named templates  
FlaskFormApp/templates/  
All HTML files will be placed in this templates folder.

### **STEP 4: Create HTML Form — templates/form.html**

Create a new file form.html inside templates folder and type the following code:

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML Form</title>
</head>
<body>
    <h2>Enter Your Details</h2>
    <form action="/submit" method="post">
        Name: <input type="text" name="uname"><br><br>
        Password: <input type="password" name="pwd"><br><br>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

#### **Note:**

- action="/submit" – sends the form data to the /submit route.
- method="post" – form data is sent using HTTP POST method.
- name="uname" and name="pwd" – these names are used in Flask to read values.

### **STEP 5: Create Result Page — templates/result.html**

Create another file result.html inside templates folder:

```
<!DOCTYPE html>
<html>
<head>
    <title>Result</title>
</head>
<body>
    <h2>Form Submission Result</h2>
    <p><strong>Name:</strong> {{ name }}</p>
    <p><strong>Password:</strong> {{ password }}</p>
</body>
</html>
```

#### **Note:**

- {{ name }} and {{ password }} are **placeholders** (Jinja2 template syntax) that Flask will fill with the values sent from app.py.

## STEP 6: Create Flask Backend — app.py

In the FlaskFormApp folder, create app.py and type:

```
from flask import Flask, request, render_template

app = Flask(__name__)

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

@app.route('/submit', methods=['POST'])
def submit():
    name = request.form['uname']
    password = request.form['pwd']
    return render_template('result.html', name=name, password=password)

if __name__ == "__main__":
    app.run(debug=True)
```

### Explanation:

- @app.route('/') → Home URL, shows the form using form.html.
- @app.route('/submit', methods=['POST']) → Handles form submission.
- request.form['uname'] → Reads the value of input with name="uname".
- render\_template('result.html', name=name, password=password) → Sends values to result page.

## STEP 7: Run the Application

- Open **Command Prompt / Terminal** inside the FlaskFormApp folder.
- Run:
- python app.py
- You will see something like:
- \* Running on http://127.0.0.1:5000
- Open a browser and type the URL: <http://127.0.0.1:5000>
- To stop the Flask server, go to the terminal where it is running and press **Ctrl + C**.

## 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.xxxxxxx.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.

### **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:

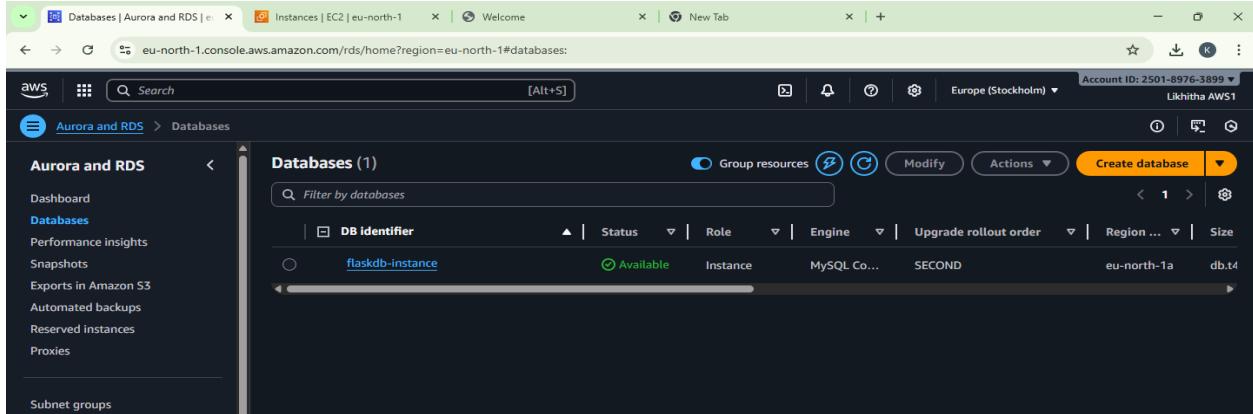
nano db\_config.py → paste code for db\_config.py  
nano app.py → paste code for app.py  
nano templates/register.html → paste register page  
nano templates/login.html → paste login page  
nano templates/welcome.html → paste welcome page

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

## 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 selected. Key details shown include:

- Endpoint:** flaskdb-instance.cze4a8sai1wq.eu-north-1.rds.amazonaws.com
- Port:** 3306
- Networking:** Availability Zone: eu-north-1a, VPC: vpc-064b87dde6ec95cda, Subnet group: rds-ec2-db-subnet-group-1, Subnets: subnet-0ffa6398a5a58ffd9
- Security:** VPC security groups: rds-ec2-1 (sg-0f10a79937b8305cd) (Active), Publicly accessible: No, Certificate authority: rds-ca-rsa2048-g1, Certificate authority date: May 25, 2021, 07:20:07Z - Oct 26, 2024, 07:20:07Z

## 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

```

[ec2-user@ip-172-31-31-82 templates]$ nano error.html
[ec2-user@ip-172-31-31-82 templates]$ sudo dnf install mariadb105 -y
Last metadata expiration check: 0:19:59 ago on Wed Dec 17 06:41:22 2025.
Dependencies resolved.
=====
Package           Architecture Version      Repository  Size
=====
Installing:
mariadb105      x86_64       3:10.5.29-1.amzn2023.0.1   amazonlinux 1.5 M
Installing dependencies:
mariadb-connector-c      x86_64       3.3.10-1.amzn2023.0.1   amazonlinux 211 k
mariadb-connector-c-config  noarch     3.3.10-1.amzn2023.0.1   amazonlinux 9.9 k
mariadb105-common      x86_64       3:10.5.29-1.amzn2023.0.1   amazonlinux 28 k
perl-Sys-Hostname    x86_64       1.23-477.amzn2023.0.7    amazonlinux 16 k
Transaction Summary
=====
Install 5 Packages

Total download size: 1.8 M
Installed size: 19 M
Downloading Packages:
(1/5): mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64.rpm          4.8 MB/s | 211 kB  00:00
(2/5): mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch.rpm      218 kB/s | 9.9 kB  00:00
(3/5): mariadb105-10.5.29-1.amzn2023.0.1.x86_64.rpm                  24 MB/s | 1.5 MB  00:00
(4/5): mariadb105-common-10.5.29-1.amzn2023.0.1.x86_64.rpm            1.2 MB/s | 28 kB  00:00
(5/5): perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64.rpm             681 kB/s | 16 kB  00:00
Total                                         17 MB/s | 1.8 MB  00:00
Running transaction check

```

- 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;

## 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;

## Expected Output

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

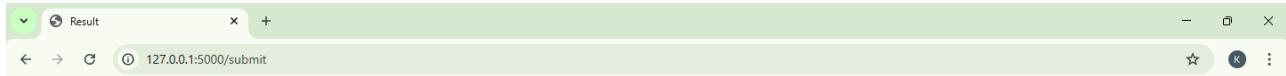
“Welcome, <username>! You have successfully logged in.”

- For invalid credentials:

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

The screenshot shows a web browser window with the following details:

- Title Bar:** HTML Form
- Address Bar:** 127.0.0.1:5000
- Content Area:**
  - Form Title:** Enter Your Details
  - Fields:** Name: [input field], Password: [input field]
  - Buttons:** [Submit] button



### 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)**.

## **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

### **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 AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like 'Dashboard', 'Tables', 'Explore items', 'PartQL editor', etc. The main area is titled 'Scan or query items' with a 'Scan' button selected. It shows a table named 'MCA\_StudentLabInternals' with three rows of data. The table structure includes columns for 'USN (String)', 'CourseCode (String)', 'IA1Marks', and 'Name'. The data rows are:

|                          | USN (String) | CourseCode (String) | IA1Marks | Name    |
|--------------------------|--------------|---------------------|----------|---------|
| <input type="checkbox"/> | 1MS24MCA002  | CCL301              | 14       | Chithra |
| <input type="checkbox"/> | 1MS24MCA001  | CCL301              | 18       | Arun    |
| <input type="checkbox"/> | 1MS24MCA001  | DBS301              | 20       | Arun    |

### 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**.

The screenshot shows the AWS DynamoDB console with the following details:

- Left Sidebar:** Shows the navigation menu under "DynamoDB" with "Explore items" selected.
- Top Bar:** Shows the URL as "eu-north-1.console.aws.amazon.com/dynamodbv2/home?region=eu-north-1#item-explorer?filter1Comparator=EQUAL&filter1Name=USN&filter1Type=S&filter1Value=1MS24MC...".
- Table Selection:** "Select a table or index" dropdown set to "Table - MCA\_StudentLabInternals".
- Attribute Selection:** "Select attribute projection" dropdown set to "All attributes".
- Filters:** "Filters - optional" section with two conditions:
  - Attribute name: USN, Condition: Equal to, Value: 1MS24MCA001
  - Attribute name: CourseCode, Condition: Equal to, Value: CCL301
- Run Button:** A prominent orange "Run" button.
- Success Message:** "Completed - Items returned: 1 - Items scanned: 3 - Efficiency: 33.33% - RCU consumed: 2".
- Table View:** "Table: MCA\_StudentLabInternals - Items returned (1)".
 

|                          | USN (String) | CourseCode (String) | IATMarks | Name |
|--------------------------|--------------|---------------------|----------|------|
| <input type="checkbox"/> | 1MS24MCA001  | CCL301              | 18       | Arun |
- Bottom Footer:** Includes links for "CloudShell", "Feedback", "Console Mobile App", and copyright information: "© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences".

## 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.

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore Items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under the 'Explore Items' section, 'Explore Items' is selected. The main area shows a table named 'MCA\_StudentLabInternals'. A search bar at the top has 'Search' and '[Alt+S]' placeholder text. To the right of the search bar are account information (Account ID: 2501-8976-3899, Region: Europe (Stockholm), User: Likhitha AWS1) and a refresh button. Below the search bar, there are dropdown menus for 'Select a table or index' (set to 'Table - MCA\_StudentLabInternals') and 'Select attribute projection' (set to 'All attributes'). A 'Partition key' section shows 'Attribute' USN and 'Value' 1MS24MCA001. A 'Sort key - optional' section shows 'Attribute' CourseCode and 'Value' Equal to. There's also a 'Filters - optional' section with a 'Run' button and a 'Reset' button. At the bottom of this section, a green status bar says 'Completed - Items returned: 2 - Items scanned: 2 - Efficiency: 100% - RCU's consumed: 0.5'. Below this, a table titled 'Table: MCA\_StudentLabInternals - Items returned (2)' is displayed. It shows two rows of data:

|                          | USN (String) | CourseCode (String) | IA1Marks | Name |
|--------------------------|--------------|---------------------|----------|------|
| <input type="checkbox"/> | 1MS24MCA001  | CCL301              | 18       | Arun |
| <input type="checkbox"/> | 1MS24MCA001  | DBS301              | 20       | Arun |

At the bottom of the table view, there are buttons for 'Actions' and 'Create item'. The footer of the page includes links for CloudShell, Feedback, and Console Mobile App, along with copyright information (© 2025, Amazon Web Services, Inc. or its affiliates.) and links for Privacy, Terms, and Cookie preferences.

### 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.

### 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 20
- Click Save changes

Verify updated value appears.

The screenshot shows the AWS DynamoDB console. The left sidebar is titled 'DynamoDB' and includes options like Dashboard, Tables, Explore items, PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below this is a section for 'DAX' with options for Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled 'Items | Amazon DynamoDB' and 'DynamoDB Scan Results'. It shows a search bar and navigation buttons. A table titled 'MCA\_StudentLabInternal' is displayed with 5 rows. The table has columns: USN (String), CourseCode (String), IA1Marks, and Name. The data is as follows:

|   | USN (String) | CourseCode (String) | IA1Marks | Name    |
|---|--------------|---------------------|----------|---------|
| 1 | 1MS24MCA002  | CCL301              | 20       | Chithra |
| 2 | 1MS24MCA001  | CCL301              | 18       | Arun    |
| 3 | 1MS24MCA001  | DBS301              | 20       | Arun    |
| 4 |              |                     |          |         |

Below the table, a message says 'Completed - Items returned: 3 - Items scanned: 3 - Efficiency: 100% - RCU consumed: 2'. The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with copyright information and privacy terms.

## 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 with the 'Explore items' page for the 'MCA\_StudentLabInternals' table. The table has two items:

| USN         | CourseCode | IA1Marks | Name    |
|-------------|------------|----------|---------|
| 1MS24MCA002 | CCL301     | 20       | Chithra |
| 1MS24MCA001 | CCL301     | 18       | Arun    |

## 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.

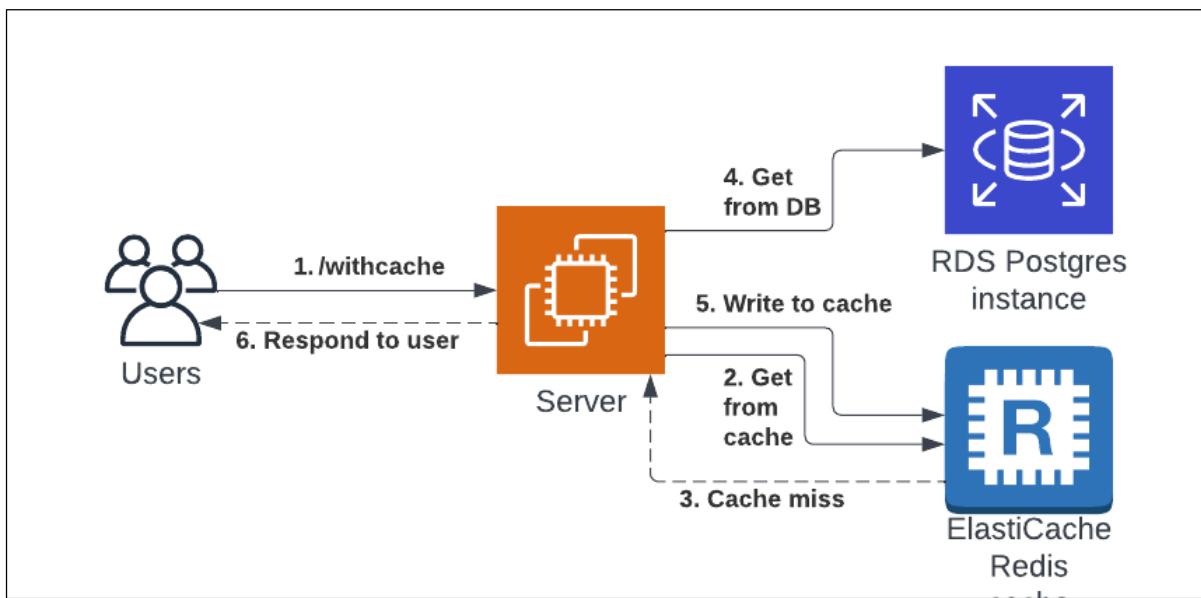
The screenshot shows the AWS DynamoDB console with the 'Tables' page. A success message at the top states: "The request to delete the 'MCA\_StudentLabInternals' table has been submitted successfully." Below this, the table list shows 0 tables.

## 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 + valkey-cli ready

Phase-2 → Redis cache created

Phase-3 → Connect EC2 → Redis 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

- Go to AWS Console → EC2
- 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 EC2

##### Inbound Rules

| Type | Port | Source                      |
|------|------|-----------------------------|
| SSH  | 22   | My IP (or Anywhere for lab) |

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.

The screenshot shows the AWS EC2 Instances page. The left sidebar is collapsed. The main area displays two instances:

| Name              | Instance ID         | Instance state | Type     | Status check      | Alarm status  | Availability Zone |
|-------------------|---------------------|----------------|----------|-------------------|---------------|-------------------|
| assign-01         | i-064b11ded047da83f | Stopped        | t3.micro | -                 | View alarms + | eu-north-1a       |
| RedisClient-AL... | i-0f61991ad6d4cf959 | Running        | t3.micro | 3/3 checks passed | View alarms + | eu-north-1b       |

The 'RedisClient-AL...' instance is selected. The details panel at the bottom shows:

- Details** tab is active.
- Instance summary**:
  - Instance ID: i-0f61991ad6d4cf959
  - IPv6 address: -
  - Public IPv4 address: 13.60.231.153 | open address
  - Instance state: Running
  - Private IP DNS name (IPv4 only): ec2-13-60-231-153.eu-north-1.compute.amazonaws.com | open address

## 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 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.

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

### **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 **Primary Endpoint**
  - Example: lab-redis.xxxxxxx.cache.amazonaws.com

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

The screenshot shows the AWS ElastiCache Management Console interface. The top navigation bar includes tabs for 'ElastiCache Management Console' and 'DynamoDB Scan Results'. Below the navigation is a search bar and account information: 'Account ID: 2501-8976-3899' and 'Likhitha AWS1'. The main content area displays the 'lab-redis' cluster details. On the left, a sidebar lists resources like 'Valkey caches', 'Memcached caches', and 'Redis OSS caches'. The central panel shows the 'Cluster details' for 'lab-redis', listing various configuration parameters such as engine version (7.1.0), node type (cache.t4g.micro), and status (Available). The ARN of the cluster is also visible.

| Cluster name           | Description   | Node type                                    | Status  |
|------------------------|---|--|---|
| lab-redis              | Easy created demo cluster on 2025-12-31T06:05:18.765Z                     | cache.t4g.micro                              | Available   |
| Engine                 | Redis   | Global datastore                             | Global datastore role   |
| Update status          | Up to date  | Shards                                       | Number of nodes   |
| Data tiering           | Disabled  | Multi-AZ                                     | Encryption in transit   |
| Encryption at rest     | Enabled   | Parameter group<br>default.redis7.cluster.on | Transit encryption mode<br>Required   |
| Configuration endpoint | <a href="#">clustercfg.lab-redis.83ghjh.eur1.cache.amazonaws.com:6379</a> | Primary endpoint                             | ARN<br><a href="#">arn:aws:elasticache:eu-north-1:250189763899:replicationgroup:lab-redis</a> |
| Data migration         | No active migrations  | Reader endpoint                              |   |

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.

## **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 Primary Endpoint from Phase-2.

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

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

### **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  $\leq 60$
- 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

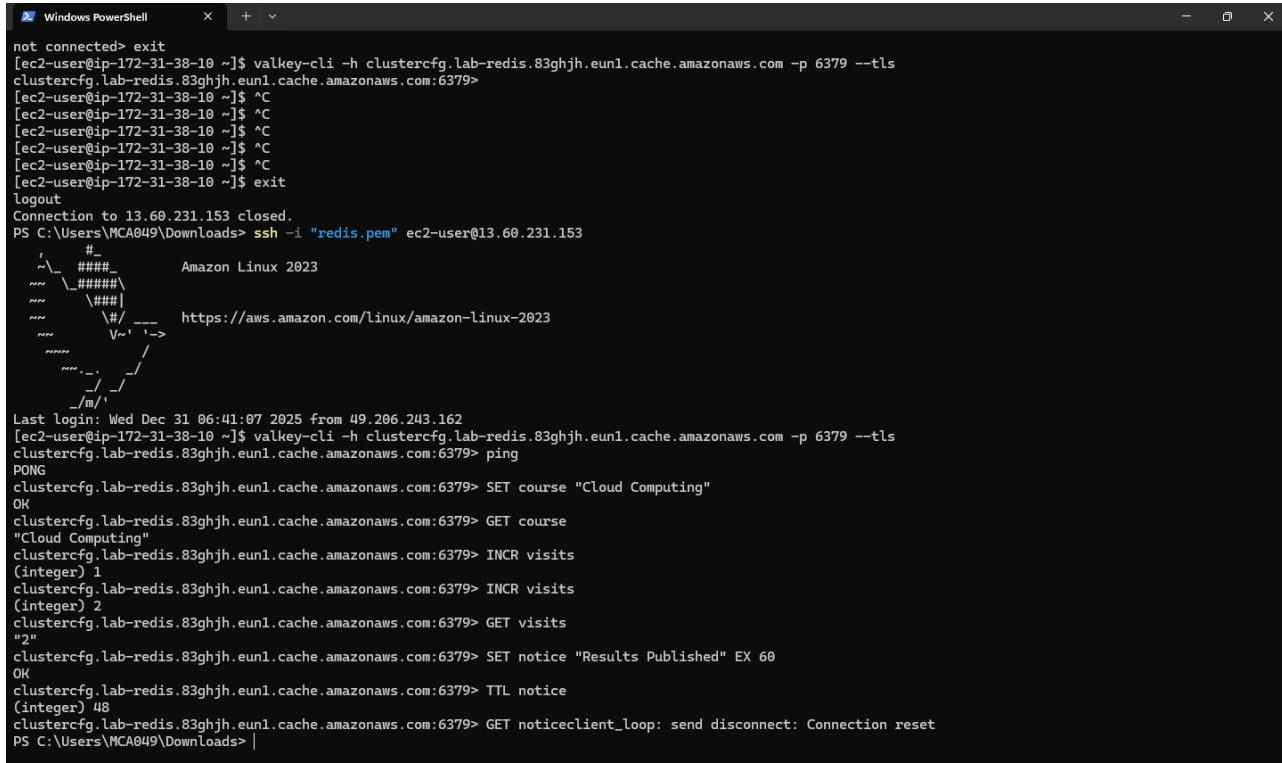
GET course

##### **Expected Output**

(nil)

#### **Step 5: Exit Redis Client**

EXIT



```

not connected> exit
[ec2-user@ip-172-31-38-10 ~]$ valkey-cli -h clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com -p 6379 --tls
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379>
[ec2-user@ip-172-31-38-10 ~]$ ^C
[ec2-user@ip-172-31-38-10 ~]$ exit
logout
Connection to 13.60.231.153 closed.
PS C:\Users\MC049\Downloads> ssh -i "redis.pem" ec2-user@13.60.231.153
      _#
     /###_
    /#####\
   /#####
  /#/
 V`--> https://aws.amazon.com/linux/amazon-linux-2023
  /_
 /_/
 /_/
 /_m/
Last login: Wed Dec 31 06:41:07 2025 from 49.206.243.162
[ec2-user@ip-172-31-38-10 ~]$ valkey-cli -h clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com -p 6379 --tls
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> ping
PONG
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> SET course "Cloud Computing"
OK
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> GET course
"Cloud Computing"
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> INCR visits
(integer) 1
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> INCR visits
(integer) 2
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> GET visits
"2"
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> SET notice "Results Published" EX 60
OK
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> TTL notice
(integer) 48
clustercfg.lab-redis.83ghjh.eun1.cache.amazonaws.com:6379> GET noticeclient_loop: send disconnect: Connection reset
PS C:\Users\MC049\Downloads> |

```

### 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                     |

| Problem      | Reason      | Fix               |
|--------------|-------------|-------------------|
| (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.

**DATE: 05-12-25**

**Exercise–20:** Deploy a simple Flask application on AWS Elastic Beanstalk and verify it using the public URL.

**Phase A — Create the Flask App**

**Step A1: On your system, create a folder**

Create a folder: eb-flask-lab

**Step A2: Create application.py**

Create a file named application.py:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello from Flask on Elastic Beanstalk!"

@app.route("/health")
def health():
    return "OK"

if __name__ == "__main__":
    app.run()
```

**Step A3: Create requirements.txt**

Create a file named **requirements.txt** and add:

```
Flask==3.0.3
gunicorn==22.0.0
```

**Step A4: Create Procfile (MOST IMPORTANT)**

Create a file named **exactly**:

Procfile (no extension)

Contents:

```
web: gunicorn application:app
```

[Windows Notepad method:

File name: Procfile, Save as type: All Files, Encoding: UTF-8]

### **Step A5: Verify folder contents**

Your folder must contain exactly these 3 files:

eb-flask-lab

```
└── application.py  
└── requirements.txt
```

└── Procfile (Type should show as “File”, NOT Procfile.txt)

### **Phase B — Create the ZIP correctly (root-level ZIP)**

#### **Step B1: Select the 3 files**

Select:

- application.py
- requirements.txt
- Procfile

#### **Step B2: Create ZIP**

Right click → Compress to → ZIP file

Rename the ZIP as:

eb-flask-lab.zip

#### **Step B3: Confirm ZIP contents (one-time check)**

Open the ZIP and confirm it shows (at top level):

application.py

requirements.txt

Procfile

If you see a folder inside the ZIP, that is wrong.

### **Phase C — Deploy on Elastic Beanstalk (AWS Console)**

#### **Step C1: Open Elastic Beanstalk**

AWS Console → Search Elastic Beanstalk → Open

Ensure region is Asia Pacific (Mumbai) (ap-south-1)

#### **Step C2: Create Environment**

Click Create environment

Environment tier - Select: Web server environment

#### **Application information**

- Application name: EB-Flask-Lab (or any name)

#### **Environment information**

- Environment name: EB-Flask-Lab-env (or any name)
- Domain: leave blank (auto)

#### **Platform**

- Platform: Python
- Platform branch: latest Python on Amazon Linux
- Platform version: recommended

### **Step C3: Upload your code (IMPORTANT SETTINGS)**

In Application code:

1. Select: Upload your code
2. Version label: type v1
3. Source code origin: Local file  
(Do NOT choose “Public S3 URL”)
4. Click Choose file and select:
5. eb-flask-lab.zip

Presets

Select: Single instance (free tier eligible)

Click Next

### **Phase D — Configure Service Access (IAM Roles)**

On Configure service access page:

#### **Step D1: Service role**

If dropdown shows “No options”:

- Click Create role (opens IAM)
- Use case: Elastic Beanstalk
  
- Keep default permissions (AWS auto selects)
- Create role name:
- aws-elasticbeanstalk-service-role

Return to EB tab → click Refresh → select this role.

#### **Step D2: EC2 instance profile**

If dropdown shows “No options”:

- Click Create role
- Use case: EC2
- Attach policy: AWSElasticBeanstalkWebTier
- Role name: aws-elasticbeanstalk-ec2-role

Return to EB tab → refresh → select this role.

#### **Step D3: EC2 Key pair**

Leave blank (optional)

Click Next

Leave remaining pages as default → Create environment

### **Part E — Wait for Deployment**

#### **Step E1: Monitor**

Open Events tab.

Final success condition:

- Green message: Environment successfully launched
- Health: OK

- Domain URL appears

The screenshot shows the AWS Elastic Beanstalk console with the environment 'EB-Flask-Lab-env-1' selected. The left sidebar shows the application 'EB-Flask-Lab' and the environment 'EB-Flask-Lab-env-1'. The main area displays the 'Environment overview' with a green 'OK - 2 health issues' status. It shows the domain 'EB-Flask-Lab-env-1.eba-xftppzux.ap-south-1.elasticbeanstalk.com'. The 'Platform' section indicates Python 3.14 running on 64bit Amazon Linux 2023/4.9.0. The 'Events' tab is selected, showing a single event from January 2, 2026, at 10:13:51 UTC+5:30, stating 'Successfully launched environment: EB-Flask-Lab-env-1'.

## Phase F — Test the Application

### Step F1: Open the domain URL

Click the Domain link shown in EB.

Expected output: Hello from Flask on Elastic Beanstalk!

### Step F2: Test /health

In the browser address bar, append /health

Example: <http://<your-domain>.elasticbeanstalk.com/health>

Expected output: Hello from Flask on Elastic Beanstalk!

The screenshot shows a web browser window with the URL 'eb-flask-lab-env-1.eba-xftppzux.ap-south-1.elasticbeanstalk.com'. The page displays the text 'Hello from Flask on Elastic Beanstalk!'

## Phase G — Cleanup (must do after lab)

### Step G1: Terminate Environment

Elastic Beanstalk → Environment

### Actions → Terminate environment

(Optional) After termination: Delete application if required.

## **Common Mistakes (quick checklist)**

Procfile must be Procfile (no .txt)

ZIP must contain files at root level (not folder)

Upload must be **Local file** (not Public S3 URL)

IAM roles must be created/selected if “No options”

### **1. What is Elastic Beanstalk?**

Elastic Beanstalk is a Platform as a Service (PaaS) provided by AWS that allows developers to deploy and manage applications without manually handling the underlying infrastructure such as EC2, load balancers, or auto scaling.

The user only uploads the application code, and Elastic Beanstalk automatically:

- Creates required AWS resources
- Deploys the application
- Handles scaling, monitoring, and health checks

### **2. What does “Single instance environment” mean?**

A Single instance environment means the application runs on one EC2 instance only, without a load balancer or auto scaling.

It is mainly used for:

- Learning and lab exercises
- Development and testing
- Low-traffic applications

It is cost-effective and simple, but not fault-tolerant.

### **3. Why do we need requirements.txt?**

The requirements.txt file lists all Python libraries and their versions required for the application.

Elastic Beanstalk uses this file to:

- Automatically install dependencies using pip
- Ensure the application runs consistently across environments

Without requirements.txt, the application may fail due to missing modules.

### **4. What is Procfile used for?**

A Procfile tells Elastic Beanstalk how to start the application.

It specifies:

- The process type (e.g., web)
- The command to run the application (e.g., Gunicorn for Flask)

Example:

web: gunicorn application:app

Without a Procfile, Elastic Beanstalk may not know which command to execute, leading to deployment errors.

### **5. What happens if you zip the parent folder instead of the files?**

If the parent folder is zipped, Elastic Beanstalk cannot locate key files like:

- application.py

- requirements.txt
- Procfile

As a result:

- Deployment fails
- Application shows errors such as “*Application version not found*” or *502 Bad Gateway*

Elastic Beanstalk expects all required files at the root level of the ZIP file.

**EXERCISE 21:** Deploy a Flask App on AWS **Elastic Beanstalk** and Perform CRUD on **DynamoDB** (Using AWS SDK/boto3)

A) LOCALLY — Prepare Folder + Code (Before AWS)

1) Create folder

Create a folder named: flask-ddb-lab

2) Inside it, create 4 files

Your folder must look like:

flask-ddb-lab/  
application.py  
requirements.txt  
Procfile  
runtime.txt

3) Paste code into application.py

```
from flask import Flask, request, jsonify
import boto3
import os

app = Flask(__name__)

REGION = os.getenv("AWS_REGION", "ap-south-1")
TABLE_NAME = os.getenv("TABLE_NAME", "Students")

dynamodb = boto3.resource("dynamodb", region_name=REGION)
table = dynamodb.Table(TABLE_NAME)

@app.route("/")
def home():
    return "Flask + DynamoDB on Elastic Beanstalk is working!"

@app.route("/health")
def health():
    return jsonify(status="ok")

# CREATE
@app.route("/student", methods=["POST"])
def create_student():
    data = request.get_json()
    table.put_item(Item={
        "StudentID": data["StudentID"],
        "Name": data["Name"],
        "Dept": data.get("Dept", "")})
```

```

        })
    return jsonify(message="Student created"), 201

# READ
@app.route("/student/<student_id>", methods=["GET"])
def get_student(student_id):
    resp = table.get_item(Key={"StudentID": student_id})
    item = resp.get("Item")
    if not item:
        return jsonify(error="Student not found"), 404
    return jsonify(item)

# UPDATE
@app.route("/student/<student_id>", methods=["PUT"])
def update_student(student_id):
    data = request.get_json()
    resp = table.update_item(
        Key={"StudentID": student_id},
        UpdateExpression="SET #n = :n, Dept = :d",
        ExpressionAttributeNames={"#n": "Name"},
        ExpressionAttributeValues={":n": data["Name"], ":d": data.get("Dept", "")},
        ReturnValues="UPDATED_NEW"
    )
    return jsonify(message="Student updated", updated=resp["Attributes"])

# DELETE
@app.route("/student/<student_id>", methods=["DELETE"])
def delete_student(student_id):
    table.delete_item(Key={"StudentID": student_id})
    return jsonify(message="Student deleted")

```

4) requirements.txt

```

Flask==3.0.0
boto3==1.34.0
gunicorn==21.2.0

```

5) Procfile

```

web: gunicorn application:app

```

6) runtime.txt

```

python-3.11

```

7) Zip it

Select these 4 files together → right click → Send to → Compressed (zipped) folder.

Rename the zip: flask-ddb-lab.zip

When you open the zip, you must directly see:

- application.py
- requirements.txt
- Procfile
- runtime.txt

(No extra folder inside.)

## B) AWS — DynamoDB Table

Create table

AWS Console → DynamoDB → Tables → Create table

- Table name: Students
  - Partition key: StudentID (String)
- Create → wait till ACTIVE

## C) AWS — IAM Role (for EB EC2 to access DynamoDB)

Create role

AWS Console → IAM → Roles → Create role

- Trusted entity: AWS service
  - Use case: EC2
  - Attach policy: AmazonDynamoDBFullAccess
  - Role name: EB-EC2-DynamoDB-Role
- Create role

[once role is created, for next time it will be automatically selected. No need to create again]

## D) AWS — Elastic Beanstalk Environment (Sample app first)

Create EB app/environment

AWS Console → Elastic Beanstalk → Create application

- Environment tier: Web server environment
- Application name: flask-ddb-lab
- Platform: Python
- Application code: Sample application

- Presets: Single instance (free tier eligible)

Configure service access

On Configure service access page:

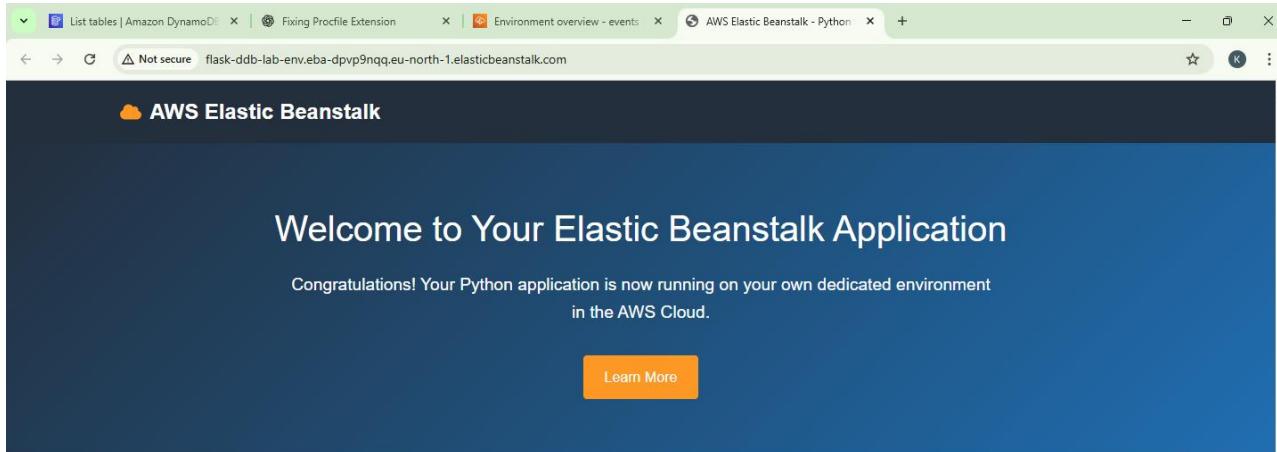
- Service role: leave default
  - EC2 instance profile: select **EB-EC2-DynamoDB-Role**
- Next → Next → Review → Create environment

Wait until you can open the EB domain and see sample page.

The screenshot shows the AWS Elastic Beanstalk Environment overview page for the environment 'Flask-ddb-lab-env'. The left sidebar shows the navigation path: Elastic Beanstalk > Environments > Flask-ddb-lab-env. The main content area is divided into two main sections: 'Environment overview' and 'Platform'. In the 'Environment overview' section, it shows 'Health' (Pending), 'Domain' (Flask-ddb-lab-env.eba-dpvp9nq.eu-north-1.elasticbeanstalk.com), and 'Application name' (flask-ddb-lab). In the 'Platform' section, it shows 'Platform' (Python 3.14 running on 64bit Amazon Linux 2023/4.9.0), 'Running version' (none), and 'Platform state' (Supported). Below these sections is a 'Events' tab, which is currently selected. It displays a table of events with three entries:

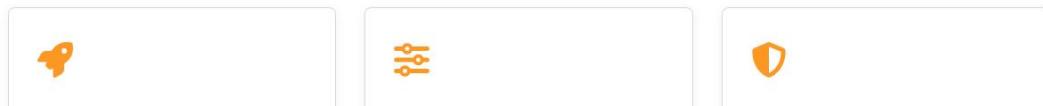
| Time                                | Type | Details   |
|-------------------------------------|------|---|
| January 2, 2026 10:39:54 (UTC+5:30) | INFO | Successfully launched environment: Flask-ddb-lab-env    |
| January 2, 2026 10:38:48 (UTC+5:30) | INFO | Instance deployment completed successfully.             |
| January 2, 2026 10:38:44 (UTC+5:30) | INFO | Instance deployment successfully generated a 'Profile'. |

Click on Domain: [Flask-ddb-lab-env.eba-jtqiug36.ap-south-1.elasticbeanstalk.com](https://Flask-ddb-lab-env.eba-jtqiug36.ap-south-1.elasticbeanstalk.com)



## Benefits of AWS Elastic Beanstalk

Discover why thousands of developers rely on AWS Elastic Beanstalk to deploy and manage their applications.



### E) AWS — Deploy Your ZIP

Upload and deploy

EB Environment → Upload and deploy

- Upload: flask-ddb-lab.zip
  - Version label: v1
- Click Deploy

After deploy, opening the EB domain should show:  
“Flask + DynamoDB on Elastic Beanstalk is working!”



## F) Add Environment Variables in Elastic Beanstalk (MANDATORY)

Step 1

Go to: AWS Console → Elastic Beanstalk → Environments → flask-ddb-lab-env

Step 2

Click: Configuration

Step 3

Under Software, click: Edit

Step 4

Scroll to Environment properties

Add these two entries:

| Name       | Value      |
|------------|------------|
| TABLE_NAME | Student    |
| AWS_REGION | ap-south-1 |

Step 5

Click: Apply

Step 6

Wait for Environment update to complete

(Status may show Updating / Pending — wait until it finishes)

## G) Verify Application After Env Update

Open the EB Environment URL in browser:

<http://<your-eb-domain>/>

Expected output: Flask + DynamoDB on Elastic Beanstalk is working!

## H) Test Health Endpoint (GET)

Step 1

Open AWS CloudShell (or Command Prompt / PowerShell)

Step 2

Run: curl http://flask-ddb-lab-env.eba-rziry7cp.us-east-2.elasticbeanstalk.com/health

Expected output: {"status":"ok"}

#### AWS CloudShell - Steps

1. In AWS Console (top right)
2. Search CloudShell in the search bar
3. Wait until terminal opens (you'll see \$ prompt)
4. Paste the following command (replace domain):

#### I) CREATE Item in DynamoDB (POST)

##### Step 1

Run:

```
curl -X POST http://Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com /Student \
-H "Content-Type: application/json" \
-d '{"StudentID":"101","Name":"Anita","Dept":"MCA"}'
```

Expected output:

```
{"message":"Student created"}
```

#### J) READ Item from DynamoDB (GET)

Run:

```
curl http://<your-eb-domain>/student/101
```

Expected output (JSON with student data).

#### K) UPDATE Item (PUT)

Run:

```
curl -X PUT http://<your-eb-domain>/student/101 \
-H "Content-Type: application/json" \
-d '{"Name":"Anita S","Dept":"MCA-III"}'
```

Expected output:

```
{"message":"Student updated"}
```

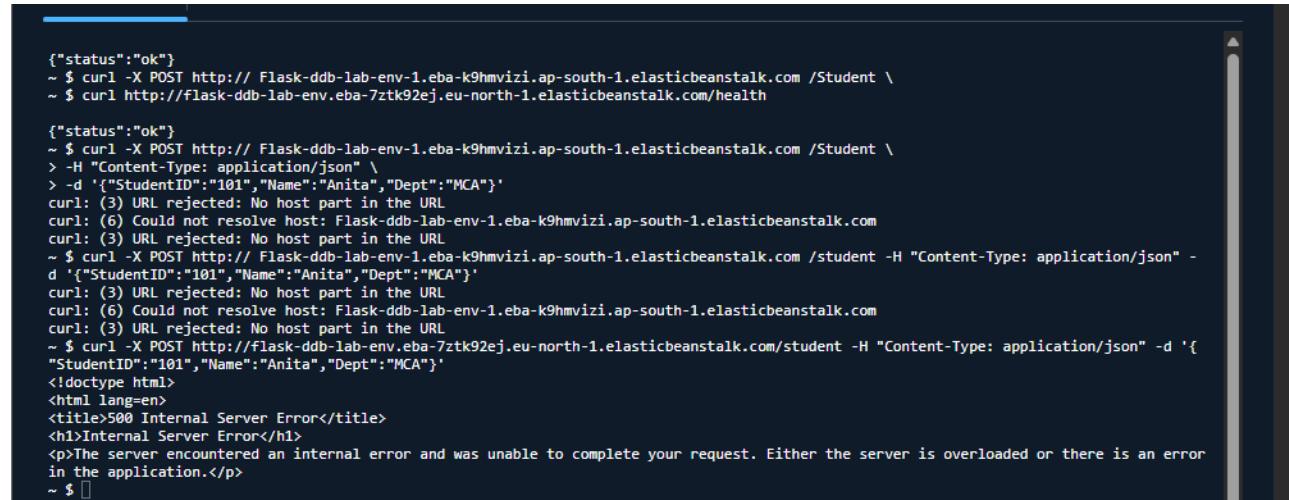
#### L) DELETE Item (DELETE)

Run:

```
curl -X DELETE http://<your-eb-domain>/student/101
```

Expected output:

```
{"message":"Student deleted"}
```



```
{"status":"ok"}  
~ $ curl -X POST http:// Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com /Student \  
~ $ curl http://flask-ddb-lab-env.eba-7ztk92ej.eu-north-1.elasticbeanstalk.com/health  
{"status":"ok"}  
~ $ curl -X POST http:// Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com /Student \  
> -H "Content-Type: application/json" \  
> -d '{"StudentID": "101", "Name": "Anita", "Dept": "MCA"}'  
curl: (3) URL rejected: No host part in the URL  
curl: (6) Could not resolve host: Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com  
curl: (3) URL rejected: No host part in the URL  
~ $ curl -X POST http:// Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com /student -H "Content-Type: application/json" -  
d '{"StudentID": "101", "Name": "Anita", "Dept": "MCA"}'  
curl: (3) URL rejected: No host part in the URL  
curl: (6) Could not resolve host: Flask-ddb-lab-env-1.eba-k9hmviz.ap-south-1.elasticbeanstalk.com  
curl: (3) URL rejected: No host part in the URL  
~ $ curl -X POST http://flask-ddb-lab-env.eba-7ztk92ej.eu-north-1.elasticbeanstalk.com/student -H "Content-Type: application/json" -d '{  
"StudentID": "101", "Name": "Anita", "Dept": "MCA"}'  
<!DOCTYPE html>  
<html lang=en>  
<title>500 Internal Server Error</title>  
<h1>Internal Server Error</h1>  
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.</p>  
~ $
```

#### M) Verify in DynamoDB Console

Go to:

AWS Console → DynamoDB → Tables → Students → Explore table items

Confirm record creation / update / deletion.

## **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: "2-09-09"

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:

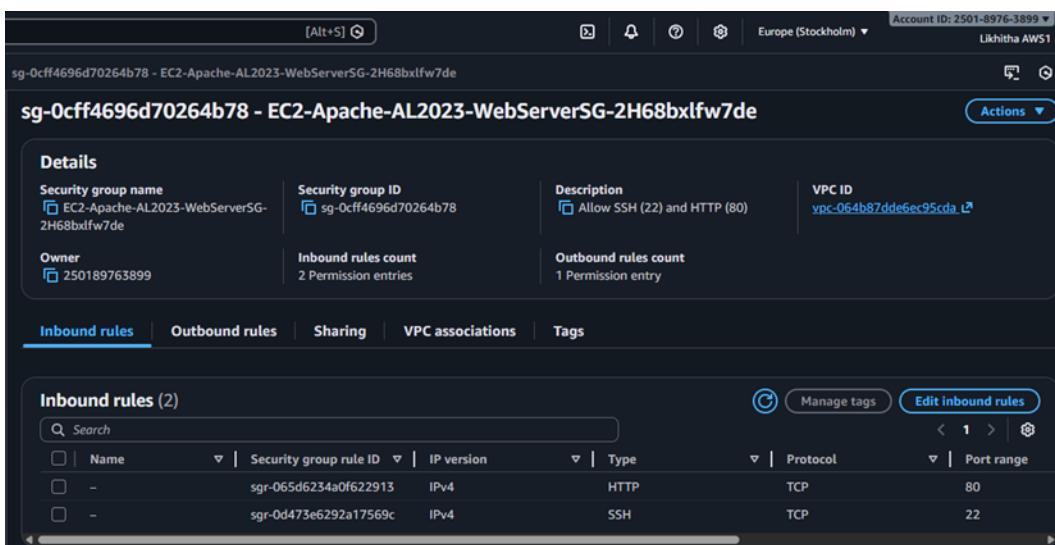
**Apache Installed via CloudFormation UserData (Amazon Linux 2023)!**



---

## Part H — Verify in EC2 (Optional)

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



## 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.

### **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

The screenshot shows the AWS CloudFormation console interface. On the left, there's a sidebar with 'Stacks (2)' and a list of stacks: 'EC2-Pull-S3-Website' (Status: CREATE\_COMPLETE) and 'S3-Website-Bucket' (Status: CREATE\_COMPLETE). The main area is titled 'S3-Website-Bucket'. It has tabs for 'Stack info', 'Events', 'Resources', 'Outputs' (which is selected), 'Parameters', 'Template', 'Change sets', and 'Git sync'. Under the 'Outputs' tab, it says '(1)'. There is one output entry: 'Key' is 'BucketName', 'Value' is 's3-website-bucket-websitebucket-jcullfsrzrh85', and 'Description' and 'Export name' are both '-'.

---

## 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

| Name       | Type | Last modified                          | Size    | Storage class |
|------------|------|--|---------|---------------|
| index.html | html | January 13, 2026, 11:44:53 (UTC+05:30) | 221.0 B | Standard      |

## 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**

**Stacks (2)**

**EC2-Pull-S3-Website**

**Stack info**

**Overview**

| Stack ID  | Description   |
|---|---|
| arn:aws:cloudformation:eu-north-1:250189763899:stack/EC2-Pull-S3-Website/27d6a570-f04a-11f0-8711-0e77f4a00893 | Launch EC2 (Amazon Linux 2023), install Apache, and pull index.html from S3 |
| Status  | Detailed status   |
| CREATE_COMPLETE   | -   |
| Status reason   | Root stack  |
| -   | -   |
| Parent stack  | Created time  |
| -   | 2026-01-15 12:06:13 UTC+0530  |
|   | Updated time  |
|   | -   |
| Deleted time  | Drift status  |
| -   | NOT_CHECKED   |
| Last drift check time   | Termination protection  |
| -   | Deactivated   |

## PART 4 — Verify Output

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

✓ Expected page:

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

EC2 + S3 Demo

Not secure ec2-56-228-24-132.eu-north-1.compute.amazonaws.com

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

## 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?

## 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**

The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Functions' (selected), 'Actions', and 'Create function'. On the left, there's a sidebar with 'Lambda > Functions'. The main area displays a table titled 'Functions (1/1)'. The table has columns: Function name, Description, Package type, Runtime, Type, and Last modified. One row is visible, showing 'StudentGradeLogger' with 'Zip' as the package type, 'Python 3.11' as the runtime, 'Standard' as the type, and '5 minutes ago' as the last modified time. A search bar at the top of the table says 'Search by attributes or search by keyword'. To the right of the table, there's a 'Tutorials' section with a heading 'Create a simple web app'. It says 'In this tutorial you will learn how to:' and lists two bullet points: 'Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage' and 'Invoke your function through its function URL'. There are 'Learn more' and 'Start tutorial' buttons at the bottom of this section.

### 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 console for the 'StudentGradeLogger' function. The 'Test' tab is selected. A green box highlights the status message: 'Executing function: succeeded (logs ↗)'. Below it, a 'Details' link is visible. The 'Test event' section contains a 'Test event' button, a 'Delete' button, a 'CloudWatch Logs Live Tail' button, a 'Save' button, and a 'Test' button. The 'Invocation type' section shows 'Synchronous' selected. The 'Event name' field contains 'ValidInput'. On the right side, there's a sidebar titled 'Create a simple web app' with a 'Start tutorial' button.

## Step 7: Test invalid input (Missing Marks)

Create another test event:

- Event name: MissingMarks

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

Run Test

Expected:

- statusCode: 400
- message: Missing 'Marks'

The screenshot shows the AWS Lambda console with the 'StudentGradeLogger' function selected. The 'Test' tab is active, displaying a green success message: 'The test event "ValidInput2" was successfully saved.' Below this, a red box highlights an error message: 'Executing function: failed (logs) [?] Details'. A 'Diagnose with Amazon Q' button is also visible. On the left, the 'Test event' section shows an event named 'ValidInput2' with options to 'Delete', 'CloudWatch Logs Live Tail', 'Save', and 'Test'. The 'Edit saved event' button is highlighted. The 'Invocation type' is set to 'Synchronous'. The 'Event name' dropdown also contains 'ValidInput2'. The right side of the screen features a 'Tutorials' sidebar with a 'Create a simple web app' section, which includes a 'Start tutorial' button.

## Step 8: Test invalid marks range

Event name: InvalidMarks

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

Expected:

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

## 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):
  - o 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

Amazon SNS > Subscriptions > Create subscription To exit full screen, press and hold Esc

### Create subscription

**Details**

**Topic ARN**  
arn:aws:sns:us-east-1:893904836080:NotificationTopic

**Protocol**  
The type of endpoint to subscribe  
Email

**Endpoint**  
An email address that can receive notifications from Amazon SNS.  
likhithacs.02@gmail.com

ⓘ After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)  
This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)  
Send undeliverable messages to a dead-letter queue.

[Cancel](#) [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

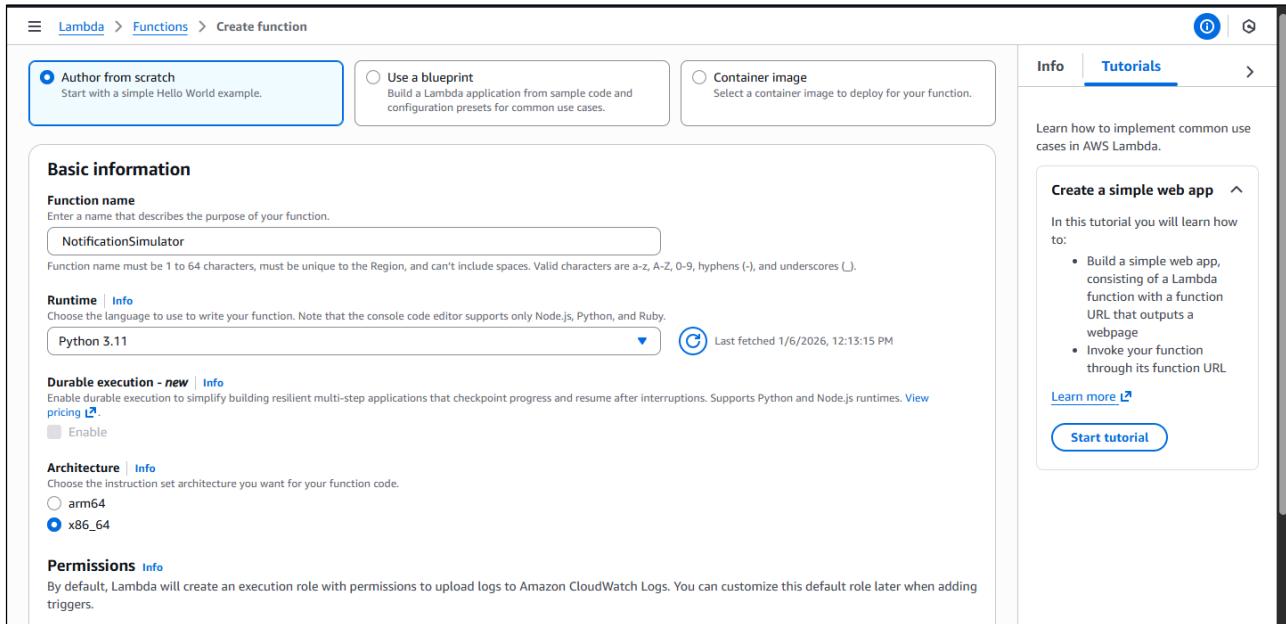
#### Step 6: Open Lambda

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)

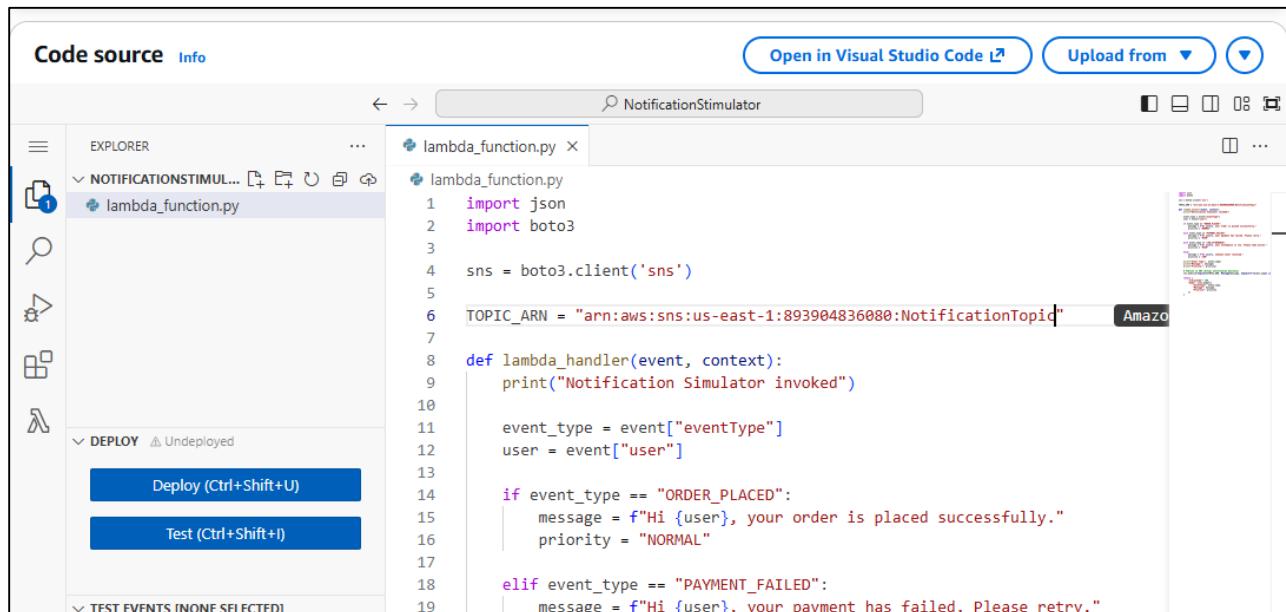
# Publish to SNS (Actual notification delivery)
sns.publish(TopicArn=TOPIC_ARN, Message=message, Subject=f"{event_type} [{priority}]")

return {
    "statusCode": 200,
    "body": json.dumps({
        "EventType": event_type,
        "Message": message,
        "Priority": priority
    })
}

```

## Click Deploy

Wait for “Successfully updated function...”



The screenshot shows the AWS Lambda Code source interface. The left sidebar has an 'EXPLORER' section with a folder named 'NOTIFICATIONSTIMUL...' containing 'lambda\_function.py'. Below it is a 'DEPLOY' section with 'Undeployed' status, 'Deploy (Ctrl+Shift+U)', and 'Test (Ctrl+Shift+I)' buttons. At the bottom is a 'TEST EVENTS [NONE SELECTED]' section. The main area shows the code for 'lambda\_function.py':

```

import json
import boto3

sns = boto3.client('sns')

TOPIC_ARN = "arn:aws:sns:us-east-1:893904836080:NotificationTopicId"

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."

```

## 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"  
}
```

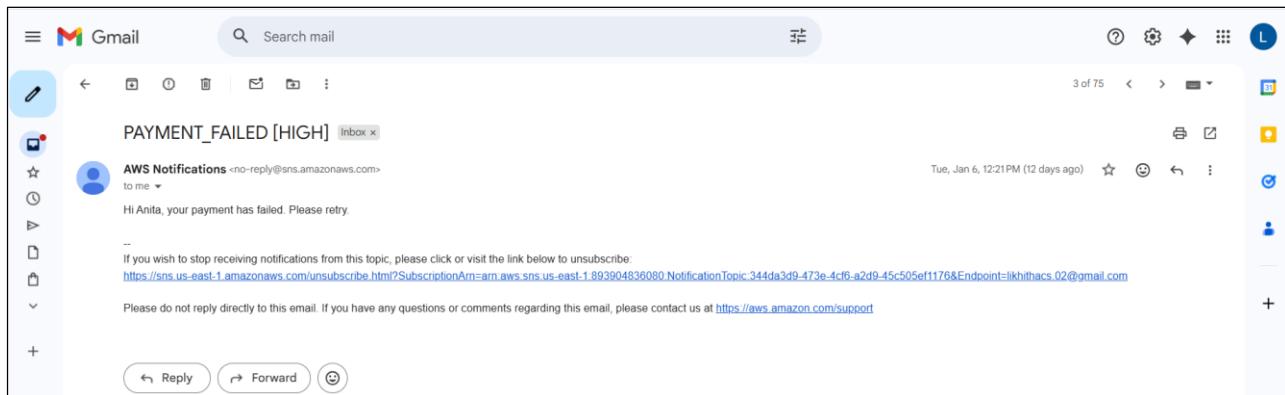
Click Save

#### 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**

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

| Timestamp   | Message  |
|---|--|
| No older events at this moment. <a href="#">Retry</a>                                     |  |
| ▶ 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. <a href="#">Auto retry paused.</a> <a href="#">Resume</a> |  |

## 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**

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

Now your CSV is in S3.

The screenshot shows the AWS S3 'Upload: status' page. At the top, it displays the destination as 's3://athena-lab-lcs-050'. Below this, under the 'Summary' section, there are two boxes: 'Succeeded' (containing '1 file, 215.0 B (100.00%)') and 'Failed' (containing '0 files, 0 B (0%)'). The 'Files and folders' tab is selected, showing a table with one row: 'students.csv' (text/csv, 215.0 B, Succeeded). A 'Find by name' search bar is also present.

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

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

This is required so Athena can store query outputs.

The screenshot shows the 'Query in Amazon SageMaker Unified Studio' interface. It features a sidebar with 'Data' settings (Data source: AwsDataCatalog, Catalog: None, Database: Choose a database) and a main area for 'Query 1'. A message at the top states: 'Before you run your first query, you need to set up a query result location in Amazon S3.' Another message below says: 'Athena now supports typeahead code suggestions to speed up SQL query development. Typeahead suggestions are turned on by default. You can change this setting in query editor preferences.' The bottom of the screen shows a standard Windows taskbar with various icons.

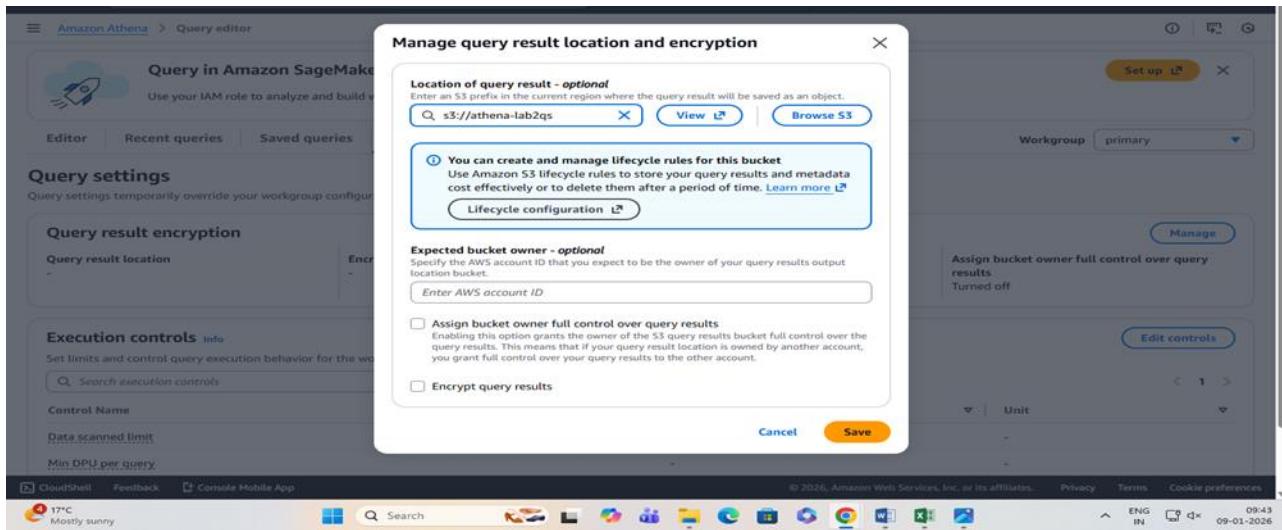
## Part D — Create a Database in Athena

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. In the top-left pane, there's a sidebar with 'Tables and views' and a table named 'students'. The main area contains a SQL query: 'SELECT \* FROM students;'. Below the query, the 'Run' button is highlighted. To the right, the 'Query results' tab is active, showing the output of the query. The results table has columns: #, studentid, name, dept, marks, and result. The data is as follows:

| # | 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 single table 'students'. The main area shows the SQL query: 'SELECT StudentID, Name, Dept, Marks FROM students WHERE Result = 'Fail';'. The 'Query results' tab is active, showing the output of the query. The results table has columns: #, StudentID, Name, Dept, and Marks. The data is as follows:

| # | StudentID | Name | Dept | Marks |
|---|-----------|------|------|-------|
| 1 | 104       | John | MCA  | 35    |
| 2 | 106       | Arun | MCA  | 49    |

### 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.

**DATE: 31-12-25**

### Exercise-27: Smart Sensor Monitoring System using AWS IoT Core

Cloud-Based IoT Data Ingestion and Processing using AWS - To understand how IoT device data is sent to the cloud, processed automatically, and logged using AWS managed services.

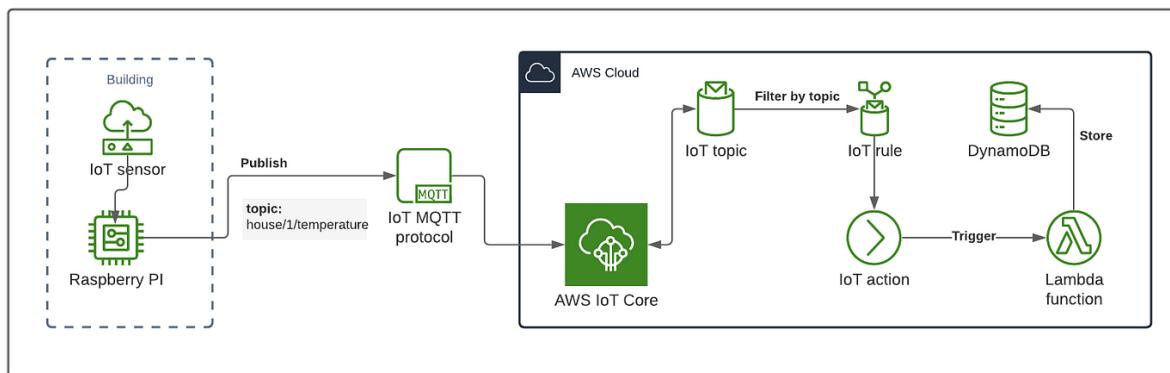
In this lab, a simulated IoT device sends sensor data (such as temperature and humidity) in JSON format to the AWS cloud.

The data is received by AWS IoT Core, which acts as the central message broker for IoT devices.

Whenever new sensor data arrives:

- AWS IoT Core detects the event
- The data is processed automatically
- The processed output is stored or logged in the cloud for monitoring and verification

This exercise demonstrates how real-time device data can be handled without managing servers, highlighting the principles of IoT, cloud computing, and event-driven architecture.



#### Concepts Covered

- Internet of Things (IoT)
- Device-to-Cloud communication
- Event-driven processing
- JSON data format
- Serverless cloud services

- Real-time data handling

## **Input**

Simulated sensor data (example: temperature and humidity values)

## **Output**

- IoT message successfully received by AWS
- Automatic processing triggered
- Cloud logs showing received and processed data

## **Learning Outcome**

After completing this lab, students will be able to:

- Explain how IoT devices communicate with the cloud
- Understand the role of AWS IoT Core in IoT solutions
- Describe event-based data processing in cloud environments
- Relate IoT concepts to real-world smart systems (smart homes, healthcare, industry)

### **STEP 1: Open IoT Service**

- Core dashboard

### **STEP 2: Create an IoT Thing (Device)**

1. Go to Manage → Things
2. Click Create things
3. Choose Create single thing
4. Give a name (example: TempSensor01)
5. Skip advanced settings
6. Create the thing

This represents a sensor device (even though no real hardware is used).

### **STEP 3: Create Device Certificate**

1. Choose Auto-generate certificate
2. Activate the certificate
3. Download:
  - o Device certificate
  - o Private key
  - o Root CA
4. Attach the certificate to the Thing

**Certificate = identity of the device**

### **STEP 4: Create and Attach IoT Policy**

1. Go to Secure → Policies
2. Create a new policy
3. Allow:
  - o Connect
  - o Publish
  - o Subscribe
  - o Receive
4. Use \* (for lab simplicity)
5. Attach the policy to the certificate

**Policy = permission for device to talk to AWS**

### **STEP 5: Test Device Messages (No Hardware)**

Go to Test → MQTT test client

Subscribe to a topic

Example: sensor/temperature

Publish a message:

```
{  
  "deviceId": "TempSensor01",  
  "temperature": 32,
```

```
"humidity": 65
```

```
}
```

Verify message appears instantly

IoT data successfully reached the cloud

## STEP 6: Create a Rule (Automation)

1. Go to Act → Rules
2. Create a rule
3. Rule query example:
4. SELECT \* FROM 'sensor/temperature'
5. Choose action:
  - Send to CloudWatch Logs  
*(or Lambda / DynamoDB if required)*

Rule = “when data arrives, do something”

## STEP 7: Verify Output

1. Open CloudWatch → Logs
2. Check log group created by IoT Rule
3. Confirm sensor data entries

Automatic processing confirmed

Sensor data is sent to AWS IoT Core, where rules automatically process and store the data without using any server.

ap-northeast-1.console.aws.amazon.com/Iot/home?region=ap-northeast-1#host

AWS IoT [Alt+5]

MQTT test client

Subscribe to a topic Publish to a topic

Topic Filter Info  
The topic filter describes the topicID to which you want to subscribe. The topic filter can include MQTT wildcard characters.

sensor/temperature

► Additional configuration

Subscribe

Subscriptions

**sensor/temperature**

Message payload

```
{<empty>} {> sensor/temperature} {> "humidity": 65}
```

Pause Clear Export Edit

► Additional configuration

Publish

▼ sensor/temperature

```
{<empty>} {> sensor/temperature} {> {<empty>}} {> "deviceID": "TempSensor01", "temperature": 32, "humidity": 65}
```

January 20, 2026, 12:28:15 (UTC+0530)

► Properties

Test

► Device Advisor

**MQTT test client**

Device Location

Query connectivity status

Manage

► All devices

► Greengrass devices

► LPWAN devices

Software packages

► Remote actions

► Message routing

Retained messages

► Security

Logs Updated

Device software

Billing groups

Settings

Feedback Console Mobile App

© 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookies preferences

The screenshot shows the AWS IoT MQTT test client interface. On the left, there's a sidebar with navigation links like 'AWS IoT', 'Monitor', 'Connect', 'Test', 'Manage', and 'Logs'. Under 'Test', 'MQTT test client' is selected. The main area has tabs for 'Subscribe to a topic' and 'Publish to a topic'. A 'Topic Filter' input field contains 'sensor/temperature'. Below it, a 'Message payload' section shows a JSON object with a single entry: 'humidity': 65. There are buttons for 'Subscribe' (orange), 'Pause', 'Clear', 'Export', and 'Edit'. A 'Publish' button is also present. In the 'Published' section, a message is shown with a timestamp: 'January 20, 2026, 12:28:15 (UTC+0530)'. The message content is identical to the one in the payload. Below the message, there's a 'Properties' section. At the bottom of the page, there are links for 'Feedback' and 'Console Mobile App', along with standard copyright and privacy information.

**DATE: 02-01-26**

**Exercise-28:** Accelerate an S3 Static Website Using Amazon CloudFront (CDN)

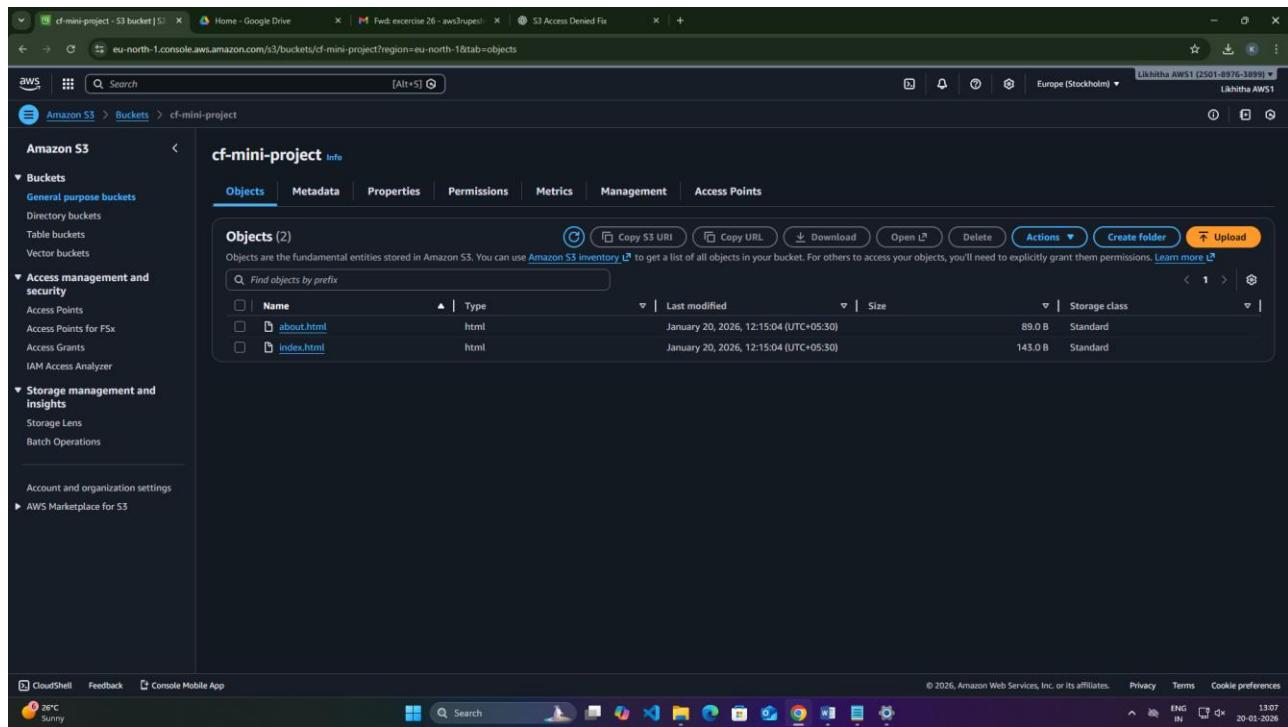
**Pre-requisite**

- You already have an **S3 bucket** with at least:
  - index.html
  - (optional) error.html

**Part A - Ensure S3 is ready**

- Open **S3 → your bucket**
- Go to **Properties → Static website hosting**
- Enable it, set:
  - Index document: index.html
  - Error document: error.html (optional)

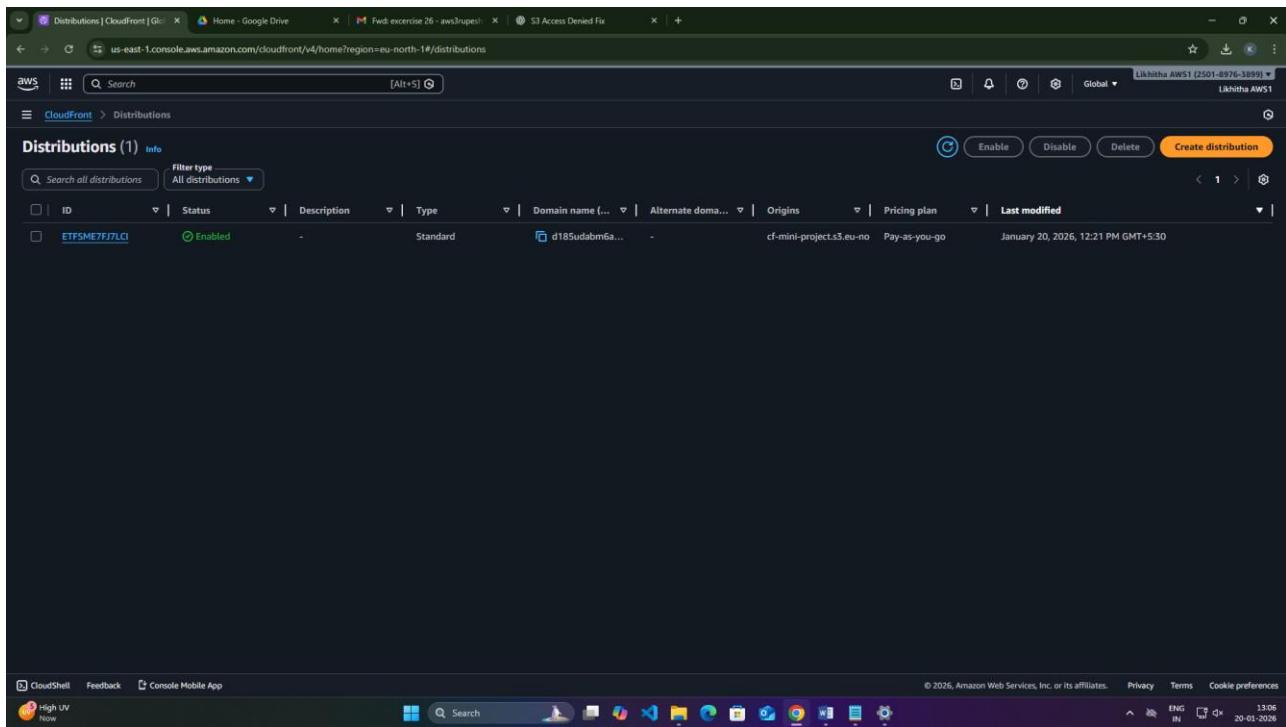
**Note down the S3 Website endpoint shown there.**



## Part B - Create CloudFront Distribution

- Go to **CloudFront** → **Create distribution**
- **Origin domain**
  - Select your **S3 bucket** (not the website endpoint)
- **Origin access**
  - Choose **Origin access control (OAC)** (recommended)
  - Click **Create control setting** if asked
- **Viewer protocol policy**
  - Choose **Redirect HTTP to HTTPS**
- **Default root object**
  - Set: index.html
- Click **Create distribution**

Copy the **Distribution domain name** (looks like dxxxxx.cloudfront.net)



### Part C - Allow CloudFront to read the bucket (important)

After creating distribution, CloudFront usually shows a message like “**S3 bucket policy needs update**”.

- Click **Copy policy**
- Go to **S3 → Bucket → Permissions → Bucket policy**
- Paste and **Save**

This step ensures **public users cannot directly access S3**, but CloudFront can.

### Part D - Test

- Wait until Distribution status becomes **Enabled** (and “Deployed”)
- Open in browser:
  - <https://dxxxxx.cloudfront.net>

Expected:

- Your index.html loads via CloudFront.

### Viva

- **S3** stores the website files

- **CloudFront** caches them at edge locations (faster)
- **OAC** ensures S3 is not public; CloudFront accesses it securely

**DATE: 02-01-26**

## **Exercise-29 Mini Project –**

### **Global Static Website Delivery using S3 + CloudFront with Cache Update Demo**

Host a static website in S3, deliver it through CloudFront (CDN), then do a content update and observe caching behavior (old page still showing). Finally, force the latest version using CloudFront Invalidations.

#### **A) Prerequisites**

- AWS Account
- An S3 bucket
- 2 small HTML files ready:
  - index.html
  - about.html

#### **index.html (Version 1)**

```
<html>
  <body>
    <h1>My CloudFront Mini Project</h1>
    <h2>Version: V1</h2>
    <p>Loaded via CloudFront CDN</p>
  </body>
</html>
```

#### **about.html**

```
<html>
  <body>
    <h1>About Page</h1>
    <h2>Version: V1</h2>
  </body>
</html>
```

## B) Step-by-step Execution

### Step 1: Create S3 Bucket

Go to S3 → Create bucket

Bucket name: cf-mini-project-<yourname>

Region: (keep default)

Block Public Access: Keep ON

Click Create bucket

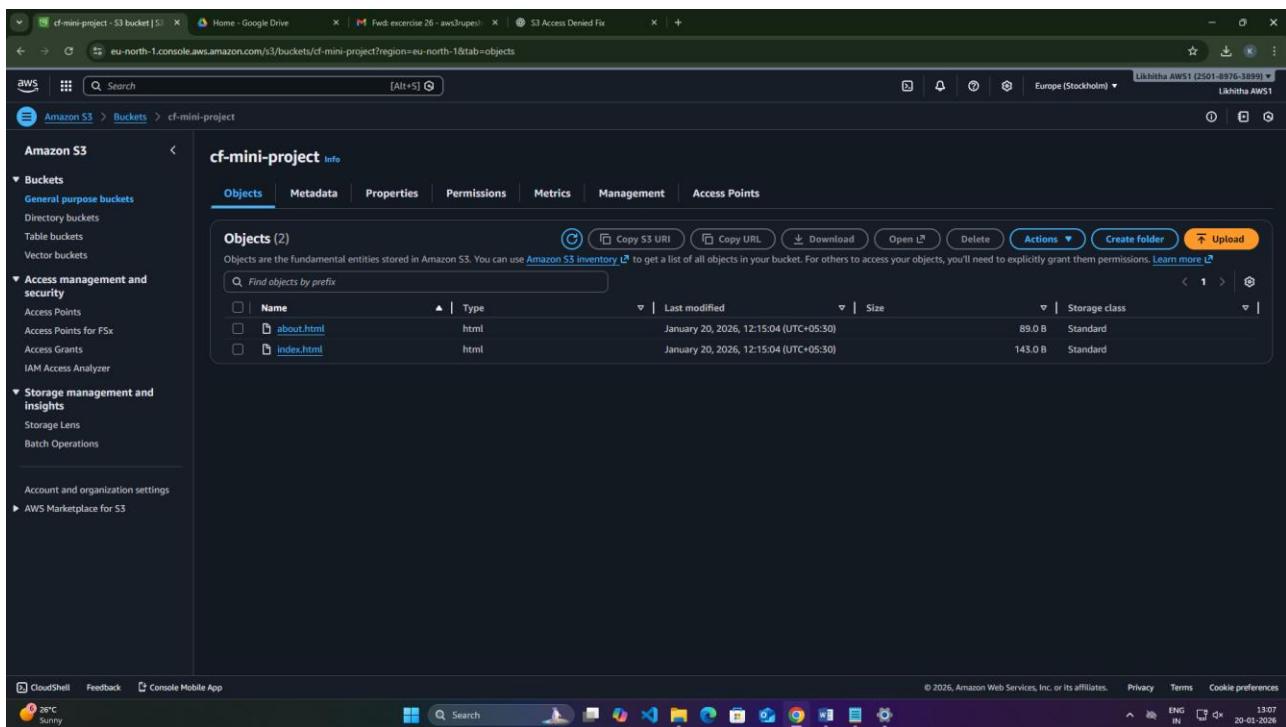
### Step 2: Upload Website Files

Open your bucket → Upload

Upload:

- index.html
- about.html

Click Upload



The screenshot shows the AWS S3 console interface. On the left, there's a navigation sidebar with sections like 'General purpose buckets', 'Access management and security', and 'Storage management and insights'. The main area is titled 'cf-mini-project' and shows a table of objects. The table has columns for Name, Type, Last modified, Size, and Storage class. There are two entries: 'about.html' and 'index.html', both of which are HTML files. The 'about.html' file was last modified on January 20, 2026, at 12:15:04 UTC+05:30, and is 89.0 B in size, stored in the Standard storage class. The 'index.html' file was last modified on January 20, 2026, at 12:15:04 UTC+05:30, and is 143.0 B in size, also stored in the Standard storage class.

| Name       | Type | Last modified                          | Size    | Storage class |
|------------|------|--|---------|---------------|
| about.html | html | January 20, 2026, 12:15:04 (UTC+05:30) | 89.0 B  | Standard      |
| index.html | html | January 20, 2026, 12:15:04 (UTC+05:30) | 143.0 B | Standard      |

### **Step 3: Create CloudFront Distribution (with secure access)**

Go to CloudFront → Create distribution

Origin domain

- Select your S3 bucket (from dropdown)

Origin access

- Choose Origin access control (OAC)
- Click Create control setting (if asked) → Create

Viewer protocol policy

- Select Redirect HTTP to HTTPS

Default root object

- Enter: index.html

Click Create distribution

### **Step 4: Add Bucket Policy for OAC (important)**

After distribution creation, CloudFront page will show a message like:

- “S3 bucket policy needs to be updated”

Click Copy policy

Go to S3 → your bucket → Permissions → Bucket policy

Paste policy → Save changes

**Now: S3 is NOT public, but CloudFront can fetch objects.**

The screenshot shows the AWS CloudFront console with a single distribution listed. The distribution details are as follows:

| ID            | Status  | Description | Type     | Domain name     | Alternate domain | Origins                          | Pricing plan  | Last modified                       |
|---------------|---------|-------------|----------|-----------------|------------------|----------------------------------|---------------|-------------------------------------|
| ETFSME7FJ7LCI | Enabled | -           | Standard | d185udabin6a... | -                | cf-mini-project.s3.eu-nor... (1) | Pay-as-you-go | January 20, 2026, 12:21 PM GMT+5:30 |

## Step 5: Test the Website Using CloudFront

Go back to CloudFront

Open your distribution

Copy the Distribution domain name, like:

- dxxxxx.cloudfront.net

Open in browser:

- https://dxxxxx.cloudfront.net

Also test:

- https://dxxxxx.cloudfront.net/about.html

Screenshot evidence:

- CloudFront distribution details page (domain + status)
- Browser output showing **Version V1**

## C) Cache Demo (Version Update + See Old Content)

### Step 6: Update Website Content in S3 (V2)

Edit your index.html and change:

From:  
<h2>Version: V1</h2>  
To:  
<h2>Version: V2</h2>

Now re-upload updated index.html:

S3 → bucket → Upload → add updated index.html

Upload

### **Step 7: Observe CloudFront Caching**

Immediately refresh:

- <https://dxxxxx.cloudfront.net>

Many times you may still see **V1** (cached), even though S3 has V2.

This is the caching behavior demonstration.

Tip for students:

- Do a hard refresh: Ctrl + Shift + R
- Still might show V1 because cache is on CloudFront edge.

Take screenshot if it still shows V1.

### **D) Force Latest Content Using Invalidation**

#### **Step 8: Create CloudFront Invalidations**

CloudFront → your distribution

Go to Invalidations tab

Click Create invalidation

In “Object paths”, enter:

- /index.html  
(or use /\* to clear everything)

Click **Create invalidation**

Wait until status becomes **Completed**.

### **Step 9: Verify Latest Version**

Refresh the CloudFront URL again: <https://dxxxxx.cloudfront.net>

Now it should show **Version: V2**

Screenshot evidence:

- Invalidation status “Completed”
- Browser showing V2

### **Cleanup (to avoid charges)**

1. CloudFront: Disable distribution → wait → Delete distribution
2. S3: Empty bucket → Delete bucket

### **Mini-Project Output Checklist:**

S3 bucket file list (index.html, about.html)

CloudFront distribution domain name

Browser output **V1**

S3 updated file showing **V2**

Browser still showing **V1** (optional if observed)

CloudFront invalidation completed

Browser output **V2**

### **Viva Questions**

1. Why use CloudFront instead of accessing S3 directly?
2. What does “cache” mean?
3. Why was invalidation needed?
4. What is OAC and why is it useful?

**DATE: 07-01-26**

## **Exercise–30 Mini Project – Deploying a Containerized App on AWS using Amazon EKS**

- Create an EKS cluster on AWS
- Add worker nodes (managed node group)
- Connect using kubectl
- Deploy Nginx and expose it

### **Region**

Use **Mumbai (ap-south-1)** (or keep consistent with your account).

### **Prerequisites**

- Logged in with root email + password
- **AWS CloudShell**

### **Method: CloudShell + eksctl**

#### **Step A1: Open CloudShell**

AWS Console (Mumbai region) → click **CloudShell** icon (terminal opens in browser)

#### **Step A2: Check tools**

Run:

```
aws --version
```

```
kubectl version --client
```

```
eksctl version
```

If eksctl is not found, install it ([CloudShell often has it](#)).

#### **Step A3: Create EKS cluster + node group (single command)**

Run:

```
eksctl create cluster \
--name eks-lab-cluster \
--region ap-south-1 \
--nodes 2 \
--node-type t3.medium \
--managed
```

### If eksctl asks / fails due to IAM (rare, but in case)

Run: aws sts get-caller-identity

If it shows:

- Your root account ID
- ARN ending with :root

Then IAM is 100% fine.

The screenshot shows the 'Create EKS cluster' step in the AWS CloudFormation console. The configuration includes:

- Name:** eks-lab-cluster
- Kubernetes version:** 1.34
- Cluster IAM role:** AmazonEKSAutoClusterRole
- Node IAM role:** AmazonEKSAutoNodeRole
- VPC:** vpc-0e981149ea5a3e352 | Default
- Subnets:** subnet-08ec1fd6ec3b97a (ap-south-1b), subnet-0f7e0674c140581e4 (ap-south-1c), and subnet-002a199bdd5c38bea

This automatically:

- Creates a **new VPC** with correct subnets/routes
- Creates the **EKS cluster**
- Creates the **Managed Node Group (worker nodes)**
- Configures kubeconfig for you

Wait until it completes (it will print “cluster created”)

## Step A4: Verify nodes

kubectl get nodes

Expected: 2 nodes in Ready state.

## Step A5: Deploy Nginx

kubectl create deployment webapp --image=nginx

kubectl get pods

```

$ kubectl get nodes
No resources found in default namespace.

$ kubectl create deployment webapp --image=nginx
deployment.apps/webapp created

$ kubectl expose deployment webapp --type=LoadBalancer --port=80
service/webapp exposed

$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
webapp-c89577dd5-2c2s5  0/1   Pending   0          9s
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
webapp-c89577dd5-2c2s5  0/1   Pending   0          13s
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
webapp-c89577dd5-2c2s5  0/1   Pending   0          17s
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
webapp-c89577dd5-2c2s5  0/1   Pending   0          28s
$ kubectl delete deployment webapp
deployment.apps "webapp" deleted from default namespace

$ kubectl create deployment webapp --image=nginx
deployment.apps/webapp created

$ kubectl delete deployment webapp
deployment.apps "webapp" deleted from default namespace

$ kubectl delete service webapp
service "webapp" deleted from default namespace

$ kubectl get pods
No resources found in default namespace.

$ kubectl get services
NAME           CLUSTER-IP  EXTERNAL-IP  PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>    443/TCP   41m
$ kubectl create deployment webapp --image=nginx
deployment.apps/webapp created

$ kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
webapp   0/1     1           0           10s
$ kubectl expose deployment webapp --type=LoadBalancer --port=80
service/webapp exposed

$ kubectl get svc -w

```

## Step A6: Expose using LoadBalancer

```
kubectl expose deployment webapp --type=LoadBalancer --port=80
```

```
kubectl get svc
```

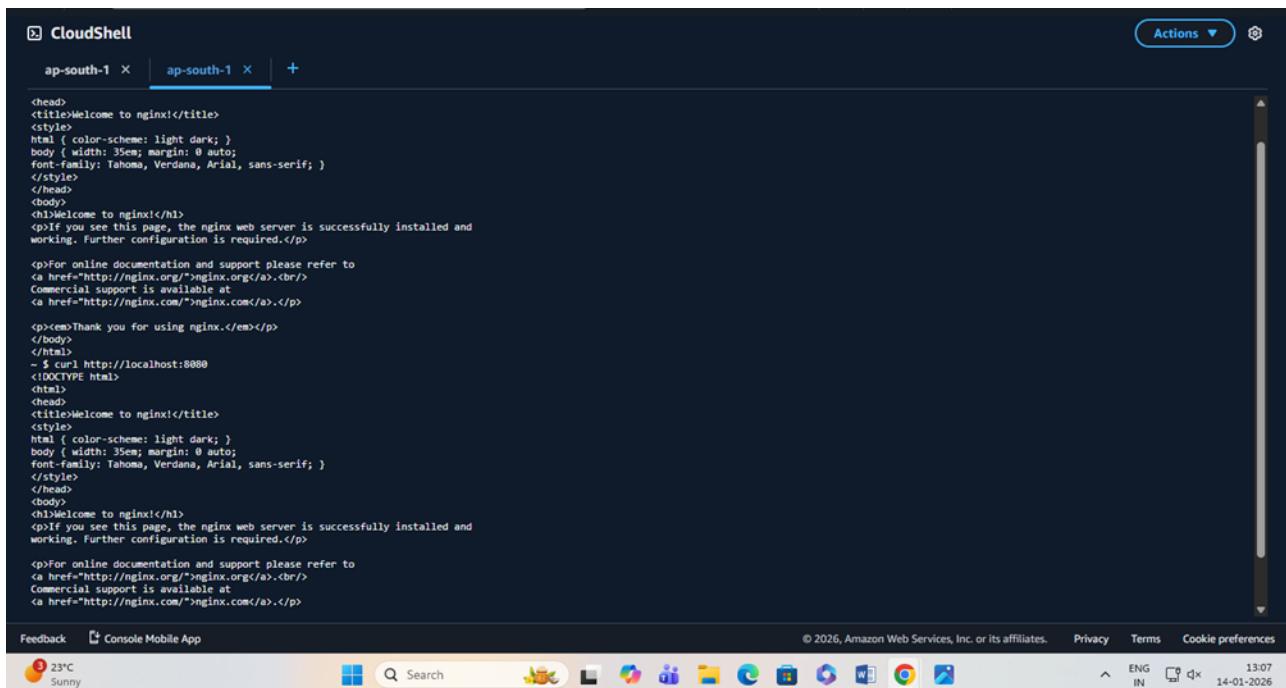
Wait until EXTERNAL-IP becomes a value (not <pending>).

## Step A7: Open in browser

Copy the EXTERNAL-IP and open:

<http://<EXTERNAL-IP>>

Expected: Nginx welcome page



The screenshot shows the AWS CloudShell interface. The terminal window displays the output of a curl command to the Nginx welcome page at port 8080. The output is as follows:

```
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com">nginx.com</a>.</p>
<em>Thank you for using nginx.</em></p>
</body>
</html>
- $ curl http://localhost:8080
<DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com">nginx.com</a>.</p>
```

The CloudShell interface includes tabs for 'ap-south-1' and 'Actions'. At the bottom, there are links for 'Feedback', 'Console Mobile App', and various AWS services like Lambda, S3, and CloudWatch. The status bar shows the date '14-01-2026' and time '13:07'.

