RAMAIAH
Institute of Technology

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

LABORATORY MANUAL

III Semester

Batch:2024-26

Name: Chethan V

USN: 1MS24MC022

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

Table of Lab Programs

Sl. No.	Programs/Exercise/Topic
1.	**AWS Account Setup** and Configuration. AWS Console Overview. Enable **MFA** for root account. 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) **Enable Versioning**. **Cross-Region Replication rule.**
5.	Overview of EC2. To Launch a Windows EC2 Instance and Connect via **RDP Client**.
6.	Launch a Linux EC2 Instance and Connect using SSH through **PowerShell** on Windows machine. [pem file] Launch a Linux EC2 Instance and Connect using SSH through **PuTTY** on Windows machine. [ppk file] Launch a Linux EC2 Instance and Connect using SSH through **Linux terminal** on Linux machine. [pem file] Launch a Linux EC2 Instance and Connect through **EC2 connect** option on the console.
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.
8.	**Custom AMI** - Create a Custom AMI from a Working EC2 Instance. - Launch an EC2 Instance using a Custom AMI. - Delete the custom AMI [Deregister and delete snapshot]
9.	**Mini-Project:** Hosting a Multi-Page Website using EC2 and S3 Submission: 11-11-2025, Marks: 5
10.	Creation and Configuration of a Custom AWS **Virtual Private Cloud** (VPC). [Including Public and Private Subnets, Internet Gateway, NAT Gateway, Route Tables]
11.	Launching and Configuring EC2 Instances for **Web Server in Public Subnet** and **Database Server in Private Subnet** Using NAT Gateway for Outbound Access.
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.
13.	**Stress Testing** a Linux EC2 Instance (CPU Load Test)

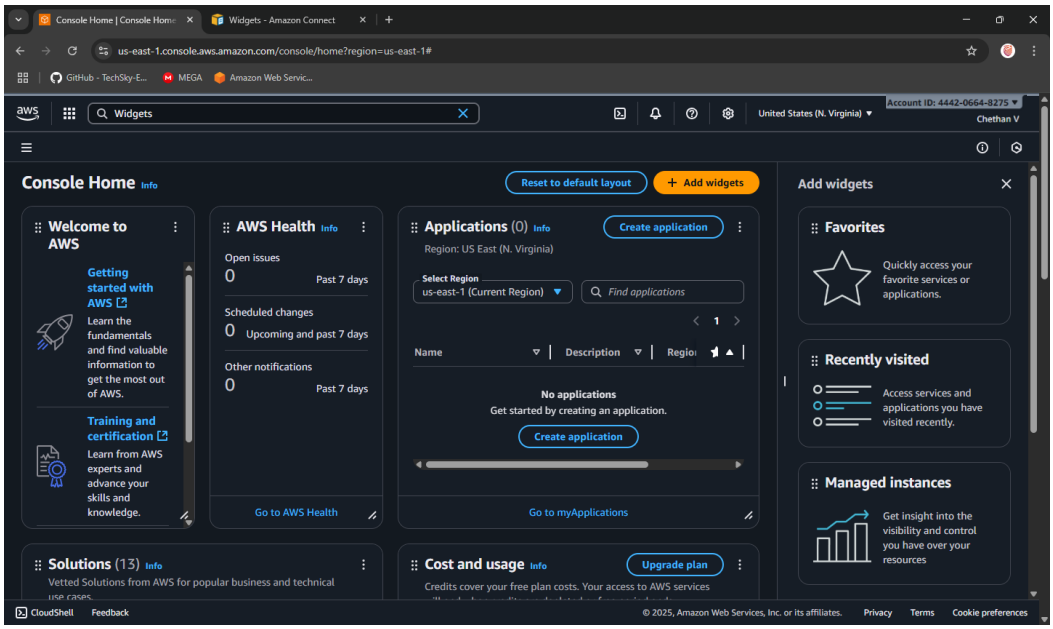
14.	**Mini Project:** Deploying a Load-Balanced Web Application using **Application Load Balancer** (ALB), **Auto Scaling Group** (ASG), Custom AMI, and Target Group on AWS with CloudWatch Alarms & Stress Testing for Auto Scaling Validation. Submission: 22-11-2025, Marks: 5
15.	Hosting a WordPress Content Management Website on Amazon **Lightsail**. (only demo)
16.	Building a Basic Python Flask Web Application — Fundamentals of Web Requests & Form Handling (**Pre-Requisite for AWS RDS Connectivity Lab**)
17.	Flask Web App with Registration & Login Using AWS **RDS** (MySQL) (Full Deployment on AWS EC2 + AWS RDS)
18.	Creating and Operating a NoSQL Key-Value Database using Amazon **DynamoDB**
19.	**ElastiCache** (Redis) as an In-Memory Cache
20.	Deploy a simple Flask application on AWS **Elastic Beanstalk** and verify it using the public URL.
21.	Deploy a Flask App on AWS **Elastic Beanstalk** and Perform CRUD on **DynamoDB** (Using AWS SDK/boto3)
22.	**CloudFormation** – Launch EC2 (Amazon Linux 2023) + Install Apache using UserData
23.	Static Content Pulled from S3 using **CloudFormation** [EC2 + S3]
24.	AWS **Lambda:** Input Processing, Business Logic Execution, and CloudWatch Logging.
25.	Event-Driven Notification System using **AWS Lambda** and Amazon **SNS**
26.	Analyze data in a CSV File stored in S3 Using **Amazon Athena** (No Server)
27.	Smart Sensor Monitoring System using **AWS IoT Core**. Cloud-Based IoT Data Ingestion and Processing using AWS
28.	Accelerate an S3 Static Website Using **Amazon CloudFront** (CDN)
	Mini Project – Global Static Website Delivery using S3 + CloudFront with Cache Update
	Mini Project – Deploying a Containerized App on AWS using **Amazon EKS**

Date: 8-10-2025

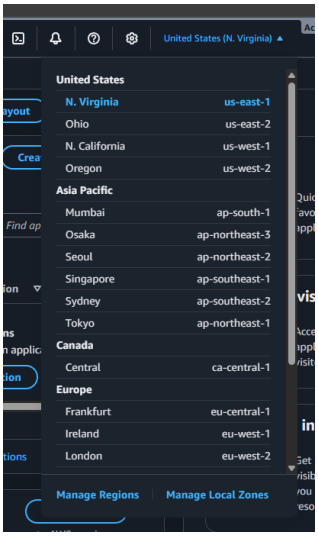
Exercise: AWS Account Setup and Configuration. AWS Console Overview. Enable MFA. Create AWS budget alert.

AWS Console overview / AWS Home page/ AWS Dash board

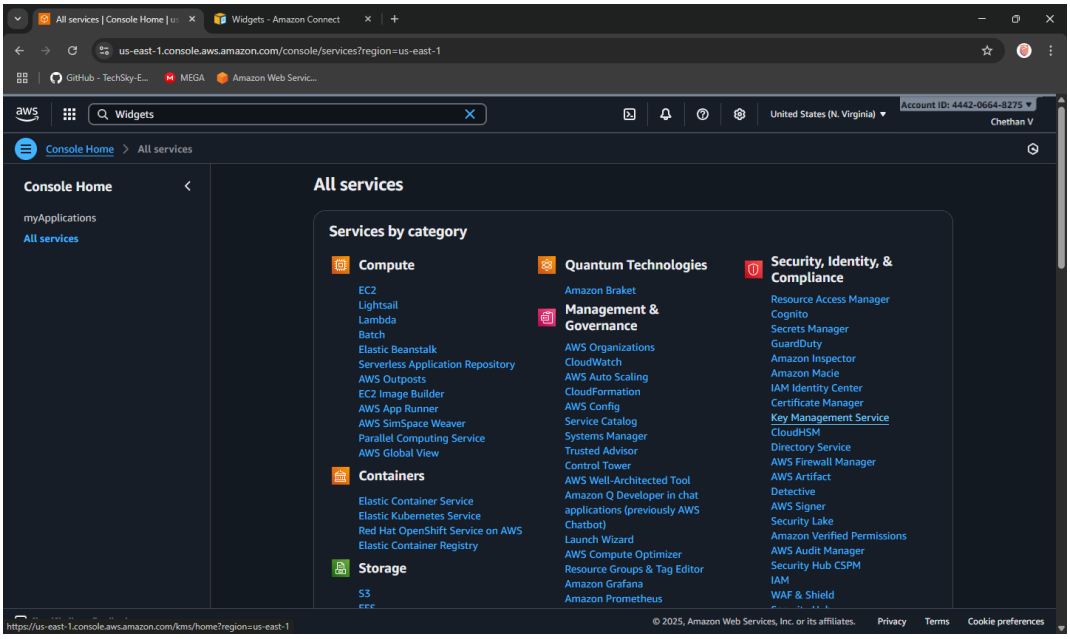
Widgets – default view/ Add or remove widgets - small panels on the dashboard showing metrics or shortcuts; users can add or remove them as needed.



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

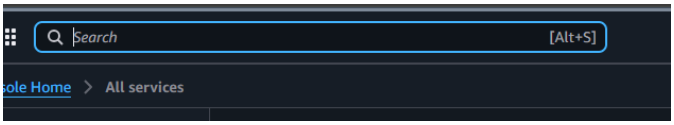


Services – A categorized list of all AWS offerings such as Compute, Storage, Database, etc.



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.

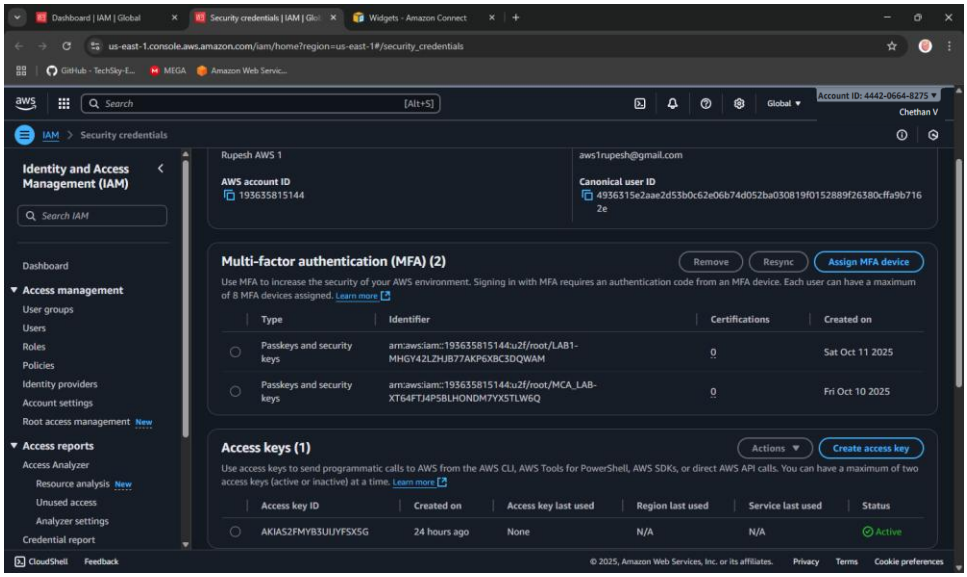
1. Select **your phone** (or “Use another device” if it prompts).

- 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

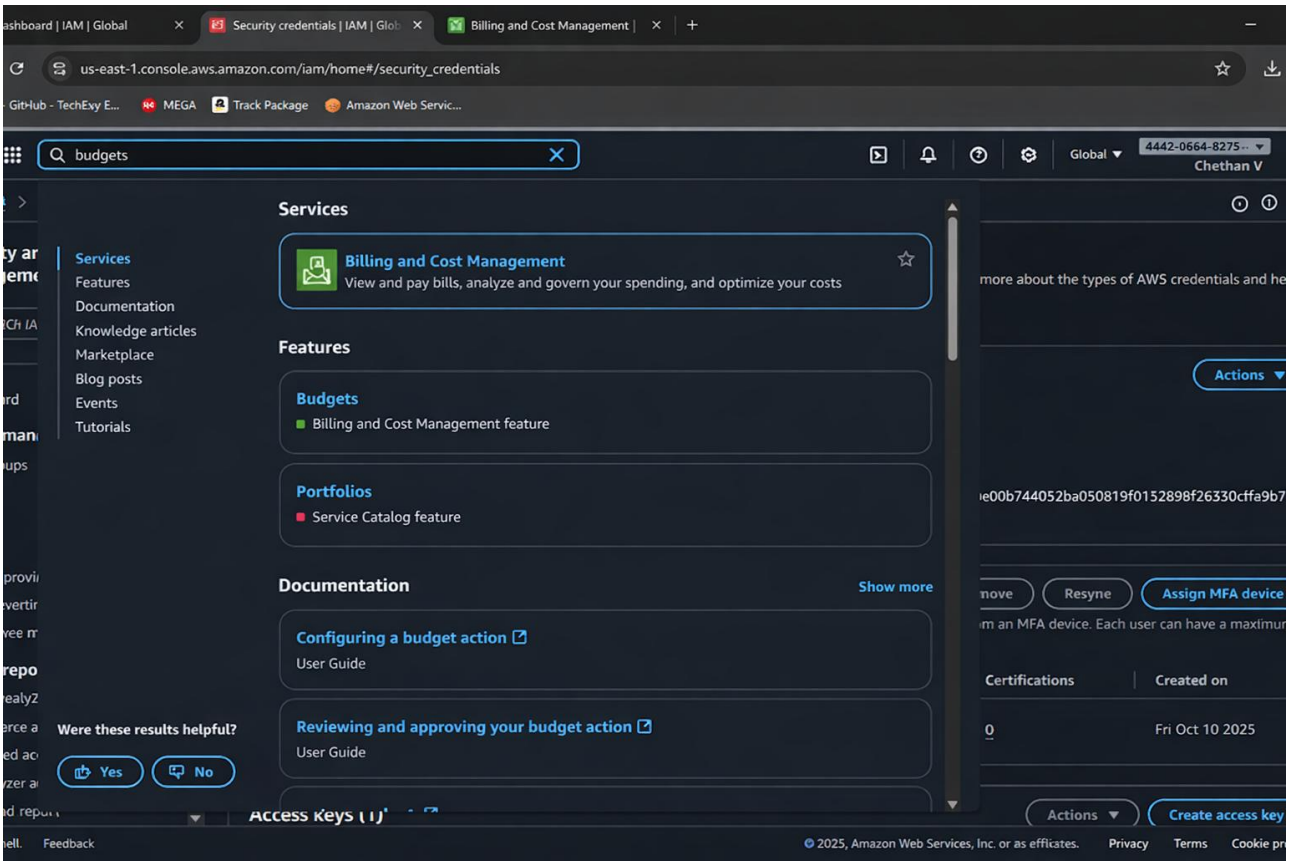
Step 1:

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

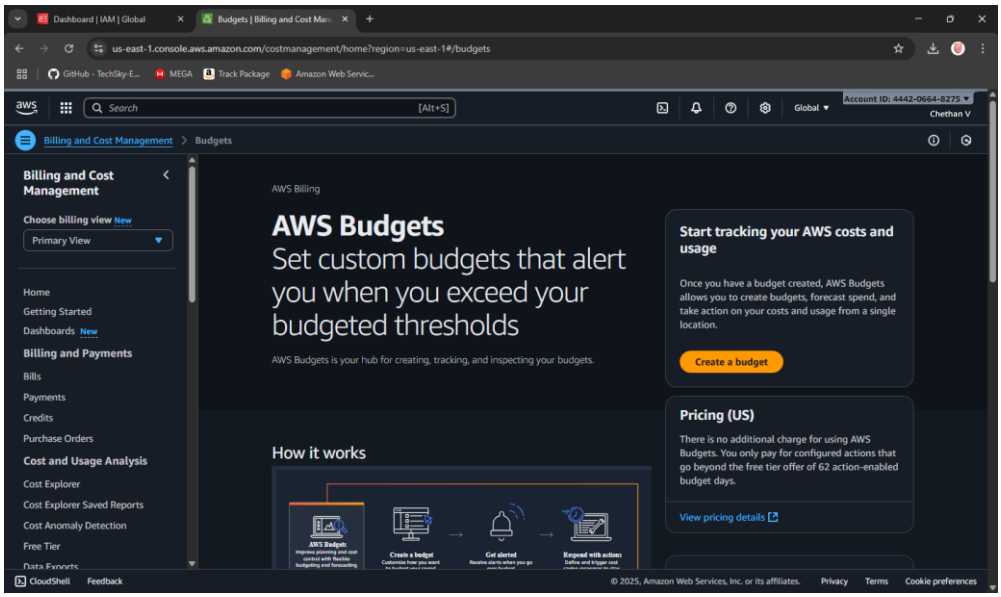
Click on Create Budget button



Under Budget setup select ‘Customize’ option

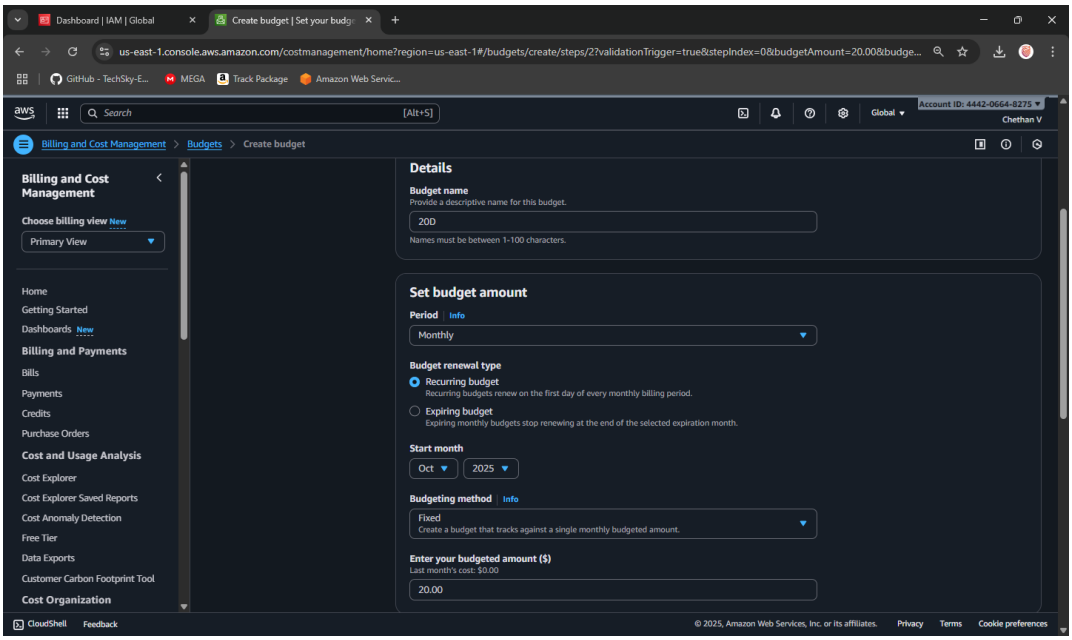
Under Budget types select ‘Cost budget’ option

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.

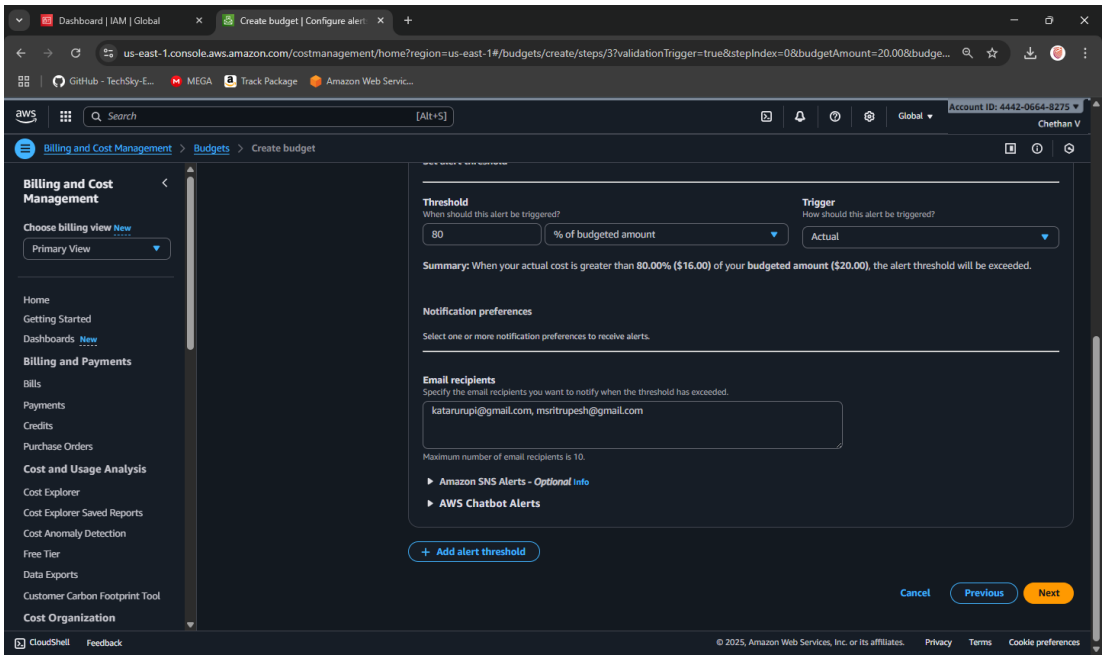
Click on Next button.



Step 4: Configure alerts

Click on Add an alert threshold

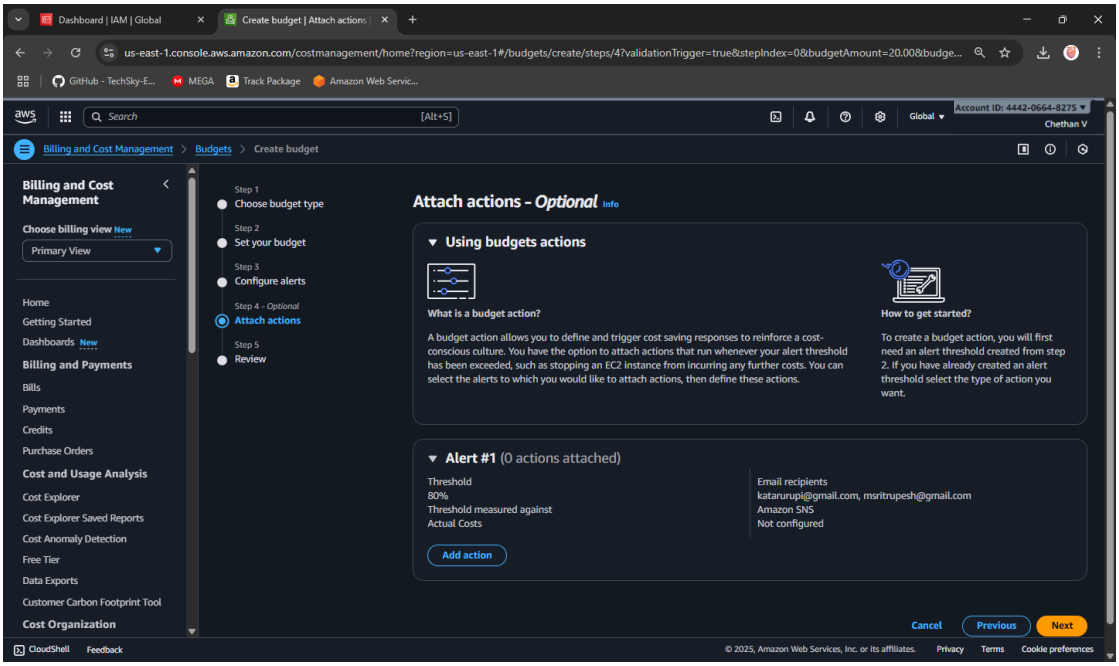
Under Set alert threshold



Set Threshold: 80 and Trigger: Actual

Email recipients: enter your email id.

Click on Next button.



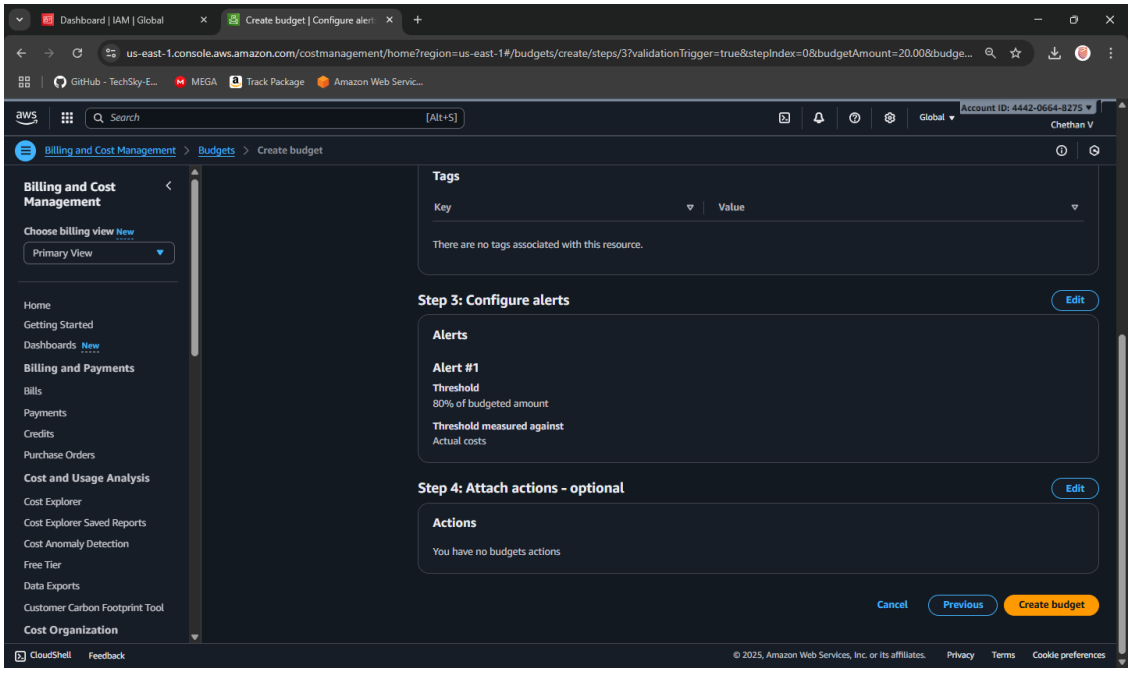
Step 5: Under Attach actions – leave it on default

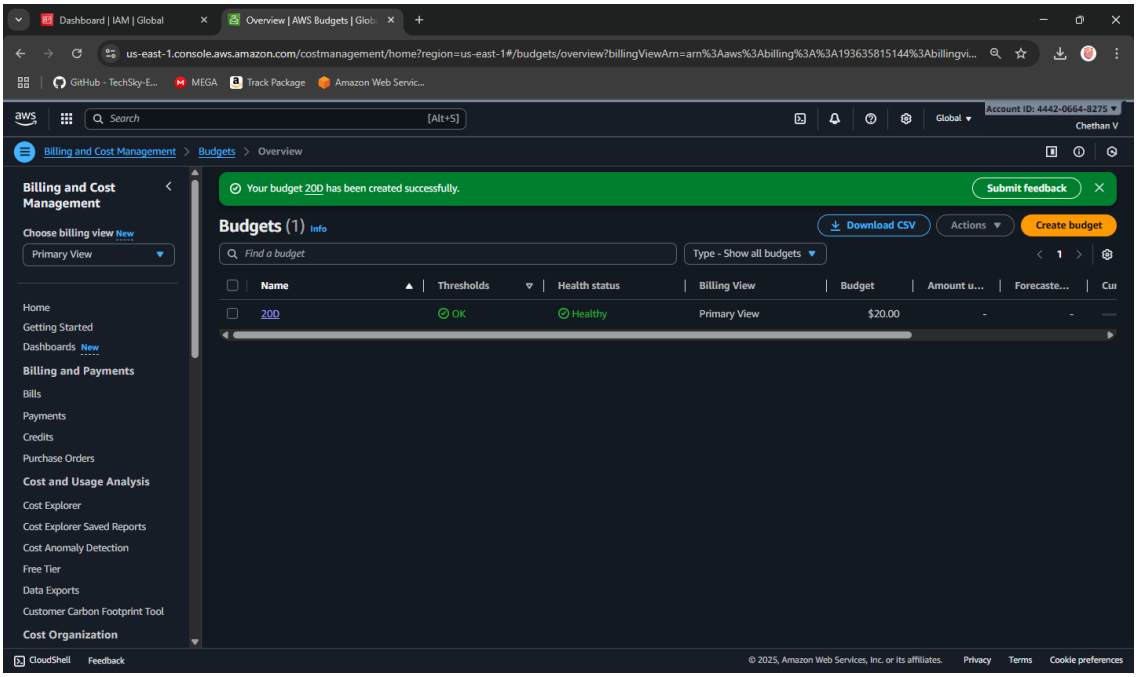
Click on Next button.

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.





Date: 9-10-2025

Exercise: 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

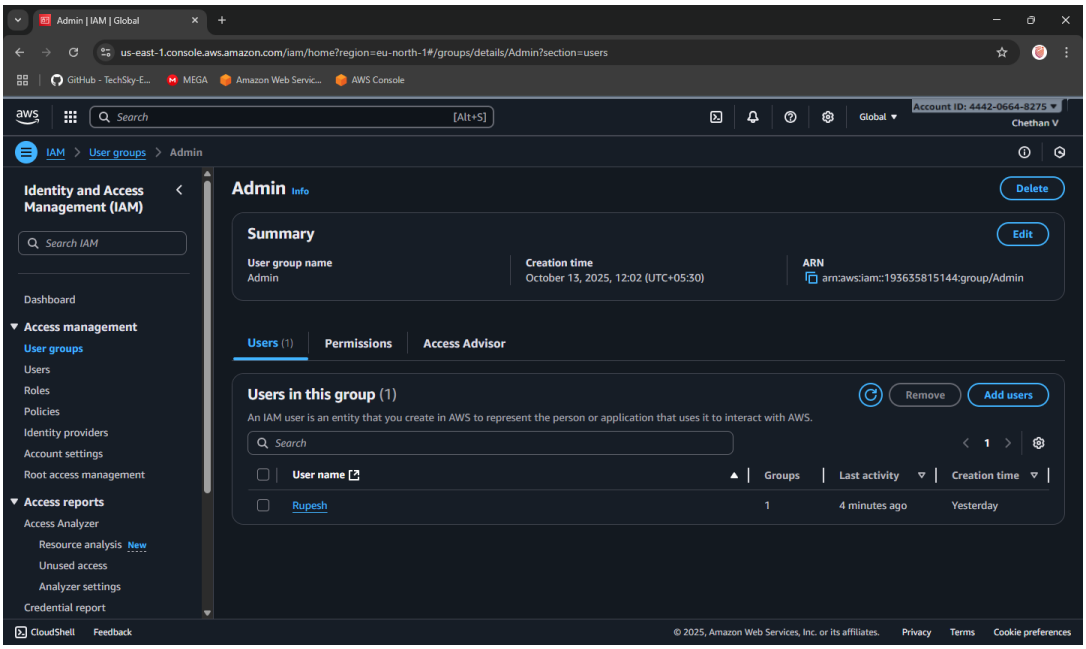
- You can attach IAM policies to define what members of the group can do.
  Select the following:
  - AdministratorAccess → Full access to all services.
  - IAMUserChangePassword

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.



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:

1. Add user to group – Assign predefined permission groups.
2. Copy permissions from an existing user.
3. 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.

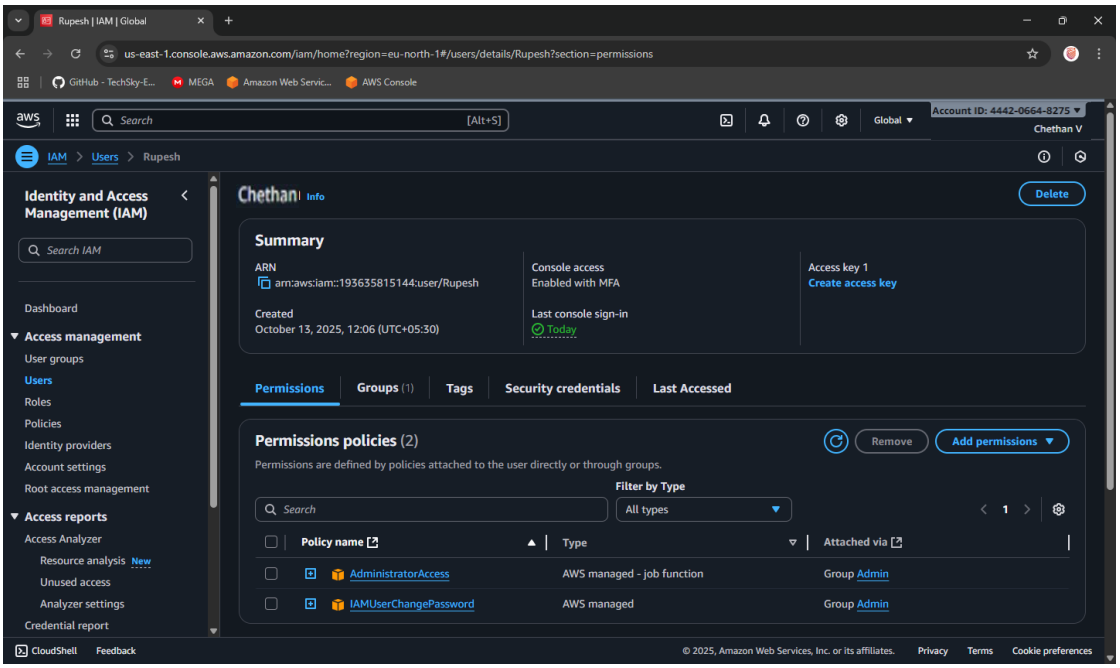
Step 7: Save Credentials

- 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.



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

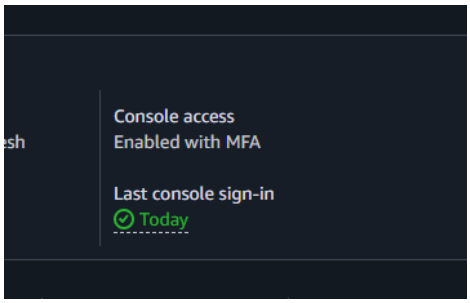
- If you select Virtual MFA device:
    1. Open your Google Authenticator or Authy app on your phone.
    2. Scan the QR code shown on the AWS screen.
    3. The app starts generating 6-digit codes.

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.



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:

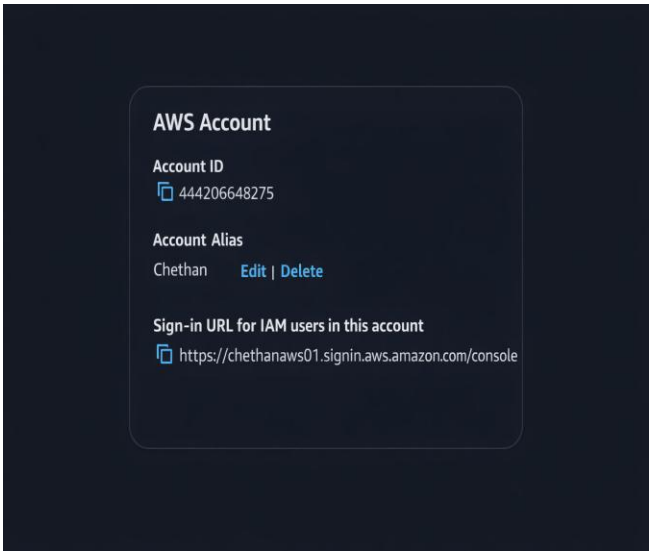
- Under the “AWS Account” section, find “Account Alias”.
- Click on “Create” (or “Edit” if one already exists).

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.



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:

- Data (the actual file),
- Metadata (information about the file),
- 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: 99.999999999% (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.

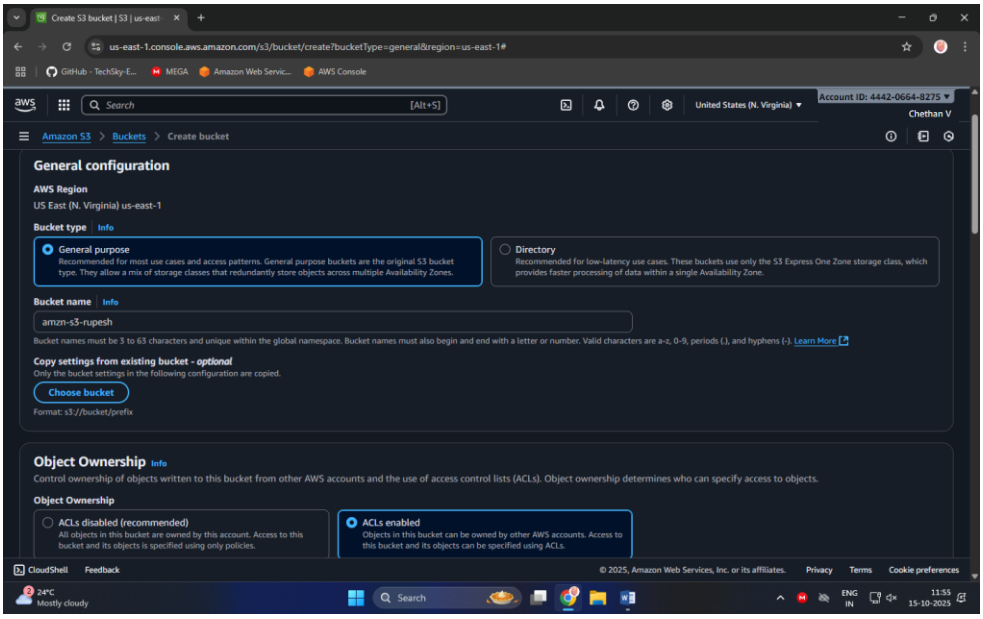
Steps to Create an S3 Bucket and Upload an Image

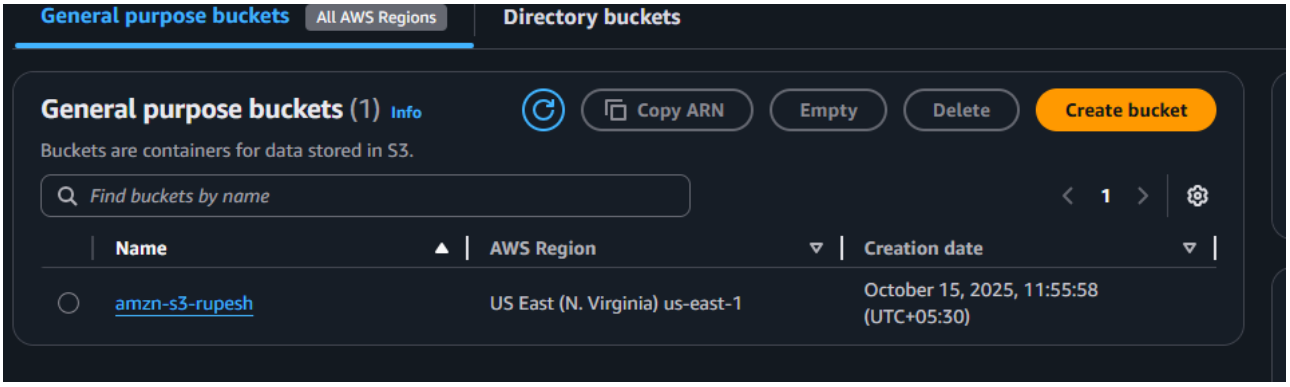
Step 1: Open the S3 Service

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

Step 2: Create a New Bucket

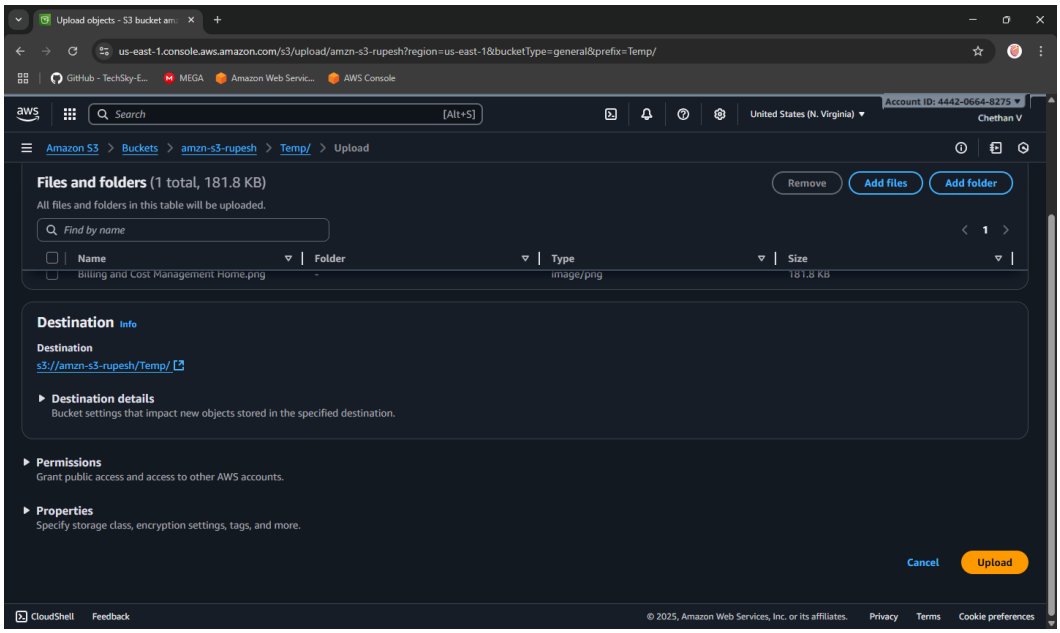
- Click “Create bucket”.
- Bucket type: General purpose
- Bucket name: Enter a name (Bucket names must be globally unique and lowercase.)
- AWS Region: Choose the region nearest to you.
- Scroll down and uncheck:
  “Block all public access” → Uncheck this (for viewing file publicly).
  Confirm the warning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

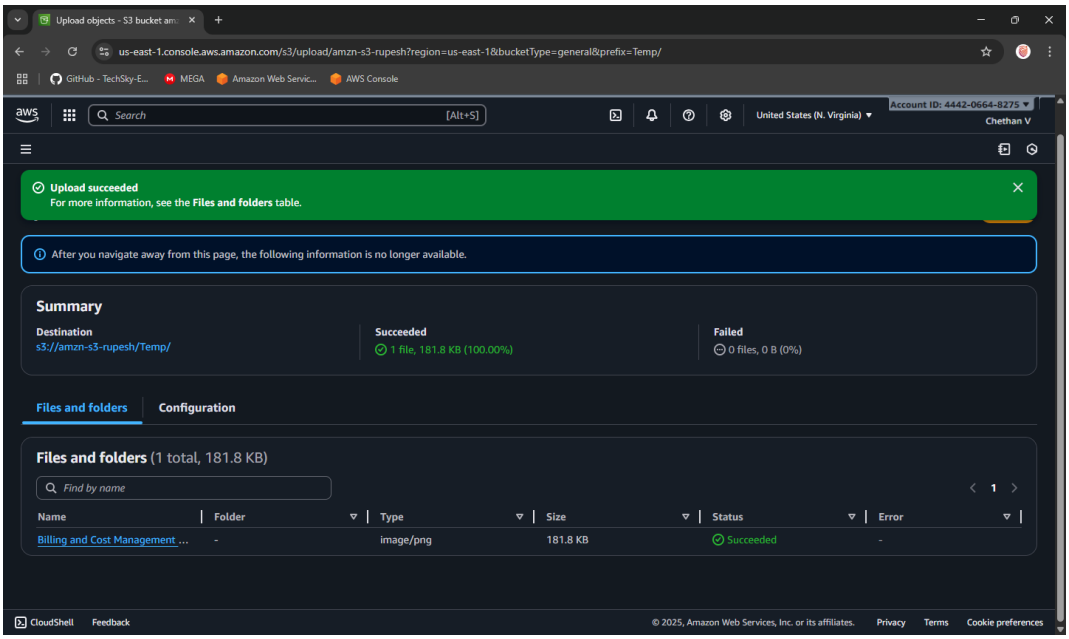




Step 3: Upload an Image File

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.



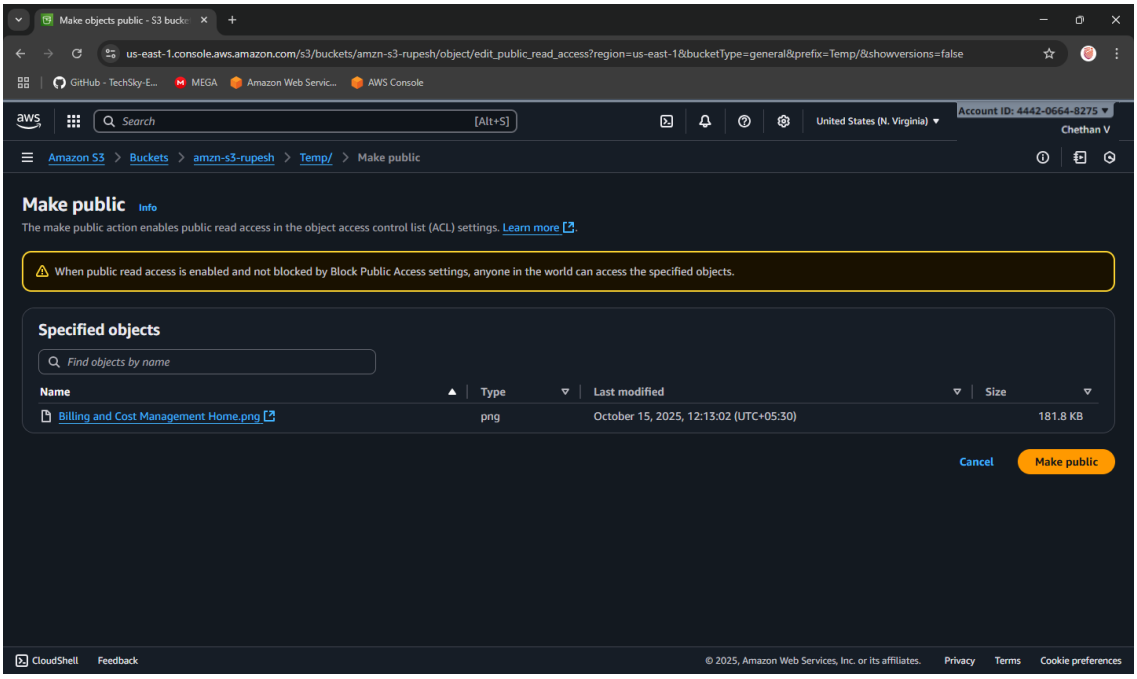


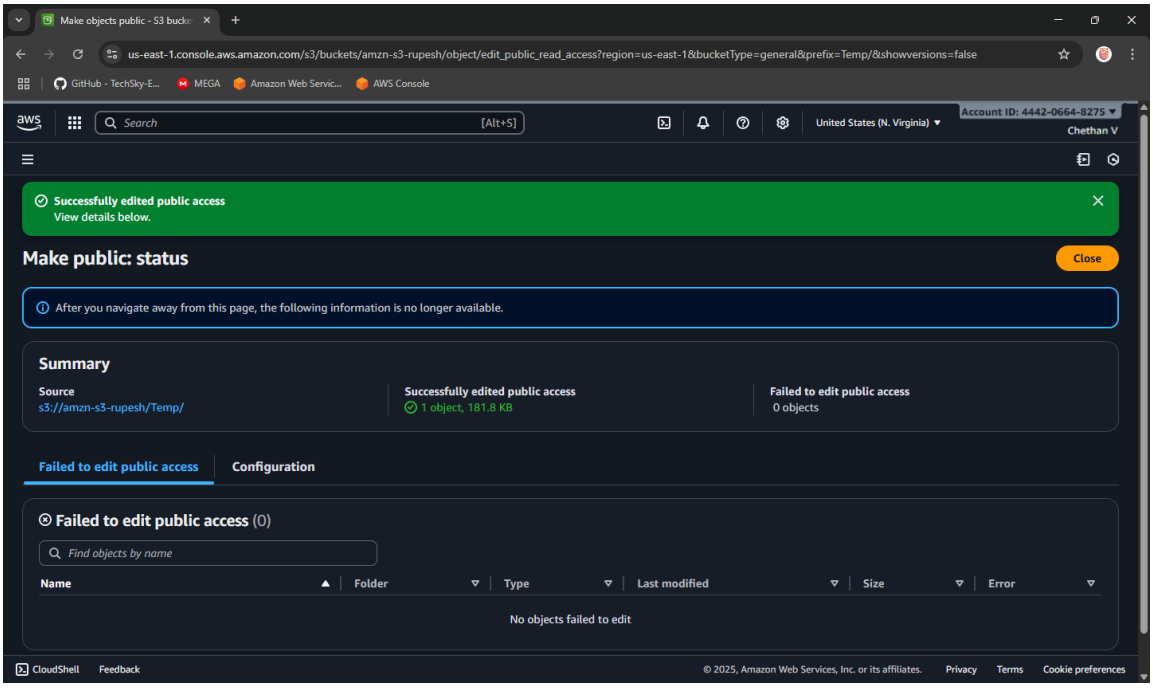
Step 4: Make the Object Public

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.





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.

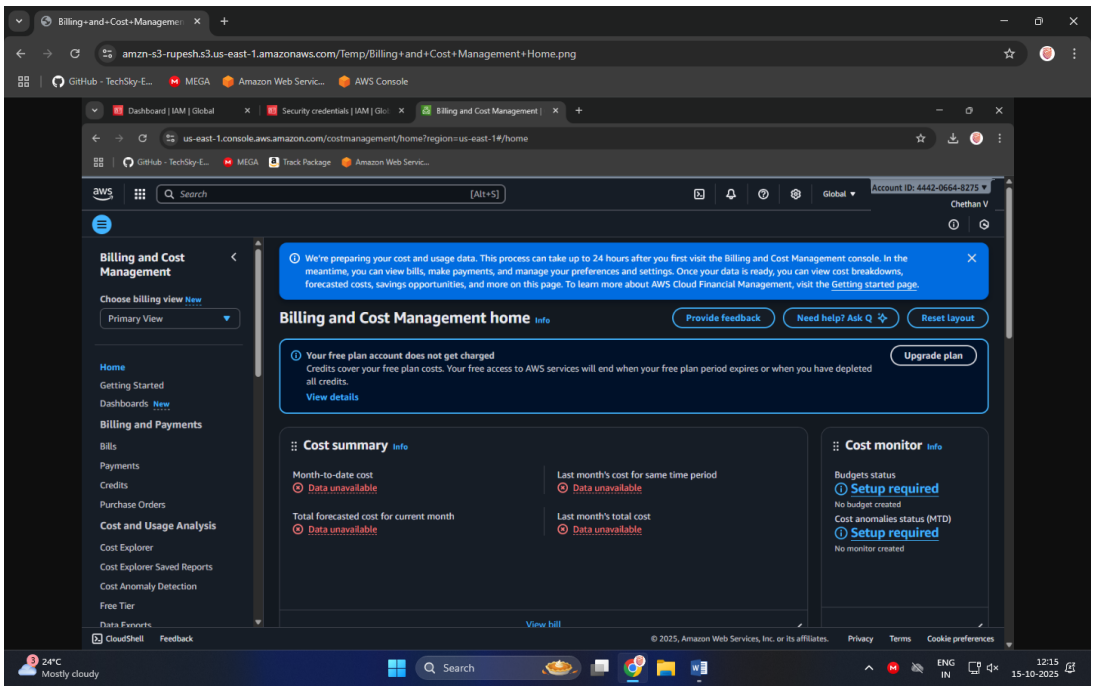
The image is displayed directly from the S3 bucket — meaning AWS S3 is serving that object over HTTP.

Entity tag (Etag)

56303f1ccd09c02bfe41d02f3c14303b

Object URL

https://amzn-s3-rupesh.s3.us-east-1.amazonaws.com/Temp/Billing+and+Cost+Management+Home.png



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].

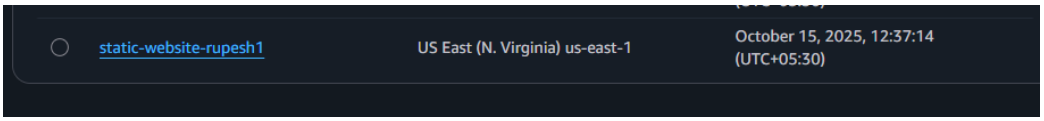
When you enable "Static Website Hosting," your S3 bucket acts like a web server, and AWS provides a public website URL to access it.

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).
- Uncheck "Block all public access."
- Click Create bucket.



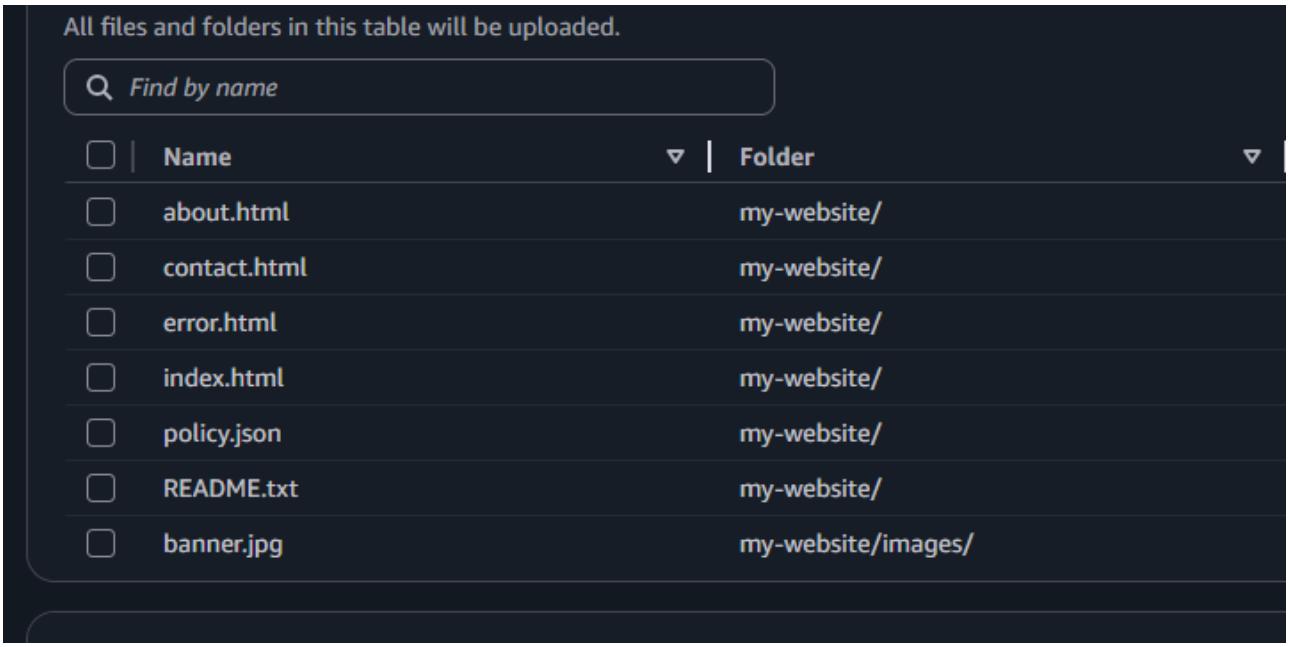
Step 2: Prepare Website Files

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

my-website/

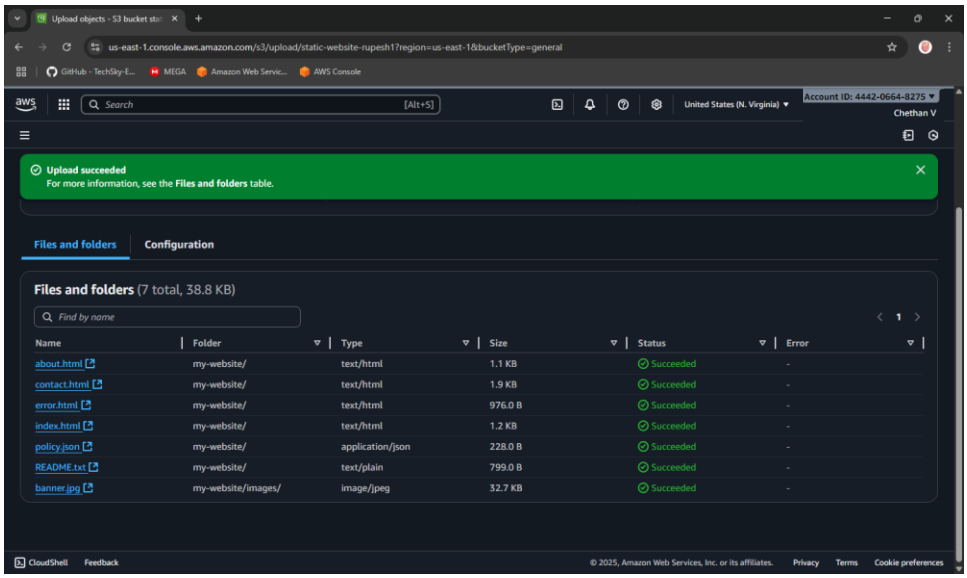
```
|
├── index.html
├── about.html
├── contact.html
├── error.html
└── images/
    └── banner.jpg
```

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:

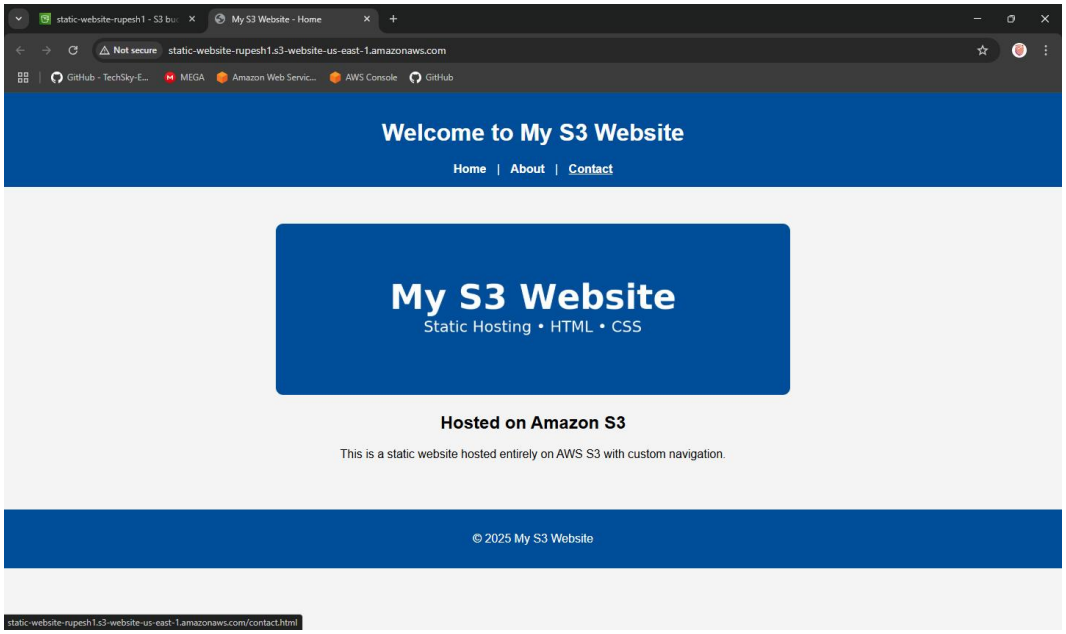
- Go to the Permissions tab → Bucket Policy → Edit.
- Paste the following policy (replace bucket name):

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

- Save the changes.

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



When Will the Error Page Be Shown?

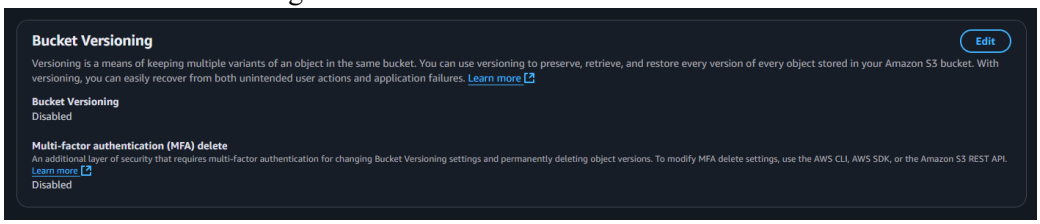
If a user enters a wrong URL or tries to access a file that doesn't exist (e.g., /abc.html), Amazon S3 automatically displays the file you set as the **Error document** (error.html).

Amazon S3 Versioning

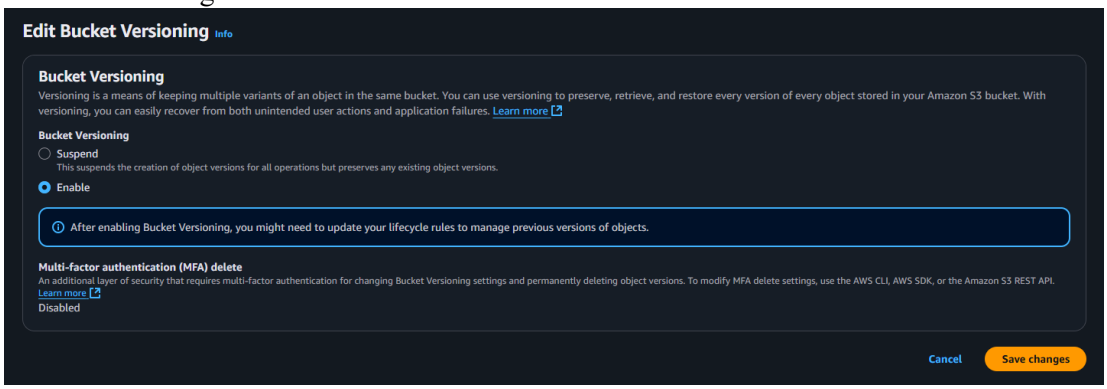
Versioning allows you to keep multiple versions of an object in a bucket.
If a file is accidentally deleted or overwritten, you can recover the previous version.
Each version gets a unique version ID.

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.)

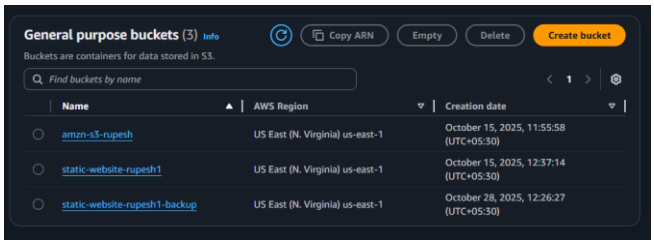
Cross-Region Replication (CRR)

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

Step 1: Enable Versioning on both:
Source bucket
Destination bucket

Step 2: Create Destination Bucket (Example: my-static-web-backup)
Choose a different region (e.g., us-east-1)



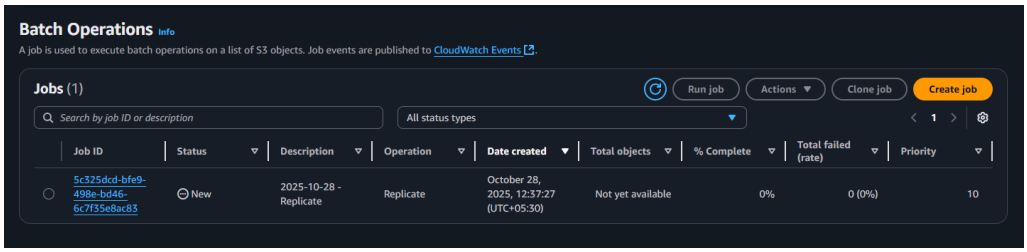
Step 3: Set Bucket Policies & IAM Role
Give replication permission:
Go to S3 → Source bucket → Management → Replication rules → Create rule

Step 4: Create Replication Rule
Choose Entire bucket or Prefix-based replication
Destination: Select the destination bucket

[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
Enable the rule

Step 5: Save
Any new objects uploaded to the source bucket will automatically replicate to the destination region.



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

Reminder 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.

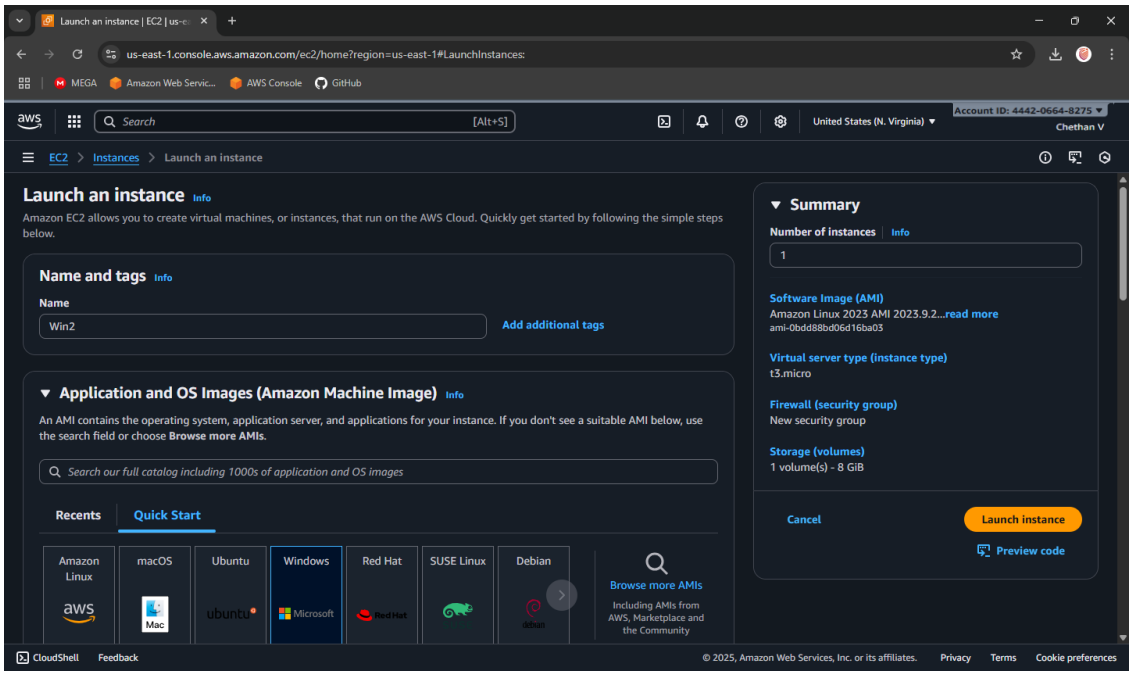
Steps to Launch a Windows EC2 Instance and Connect via RDP Client

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) →

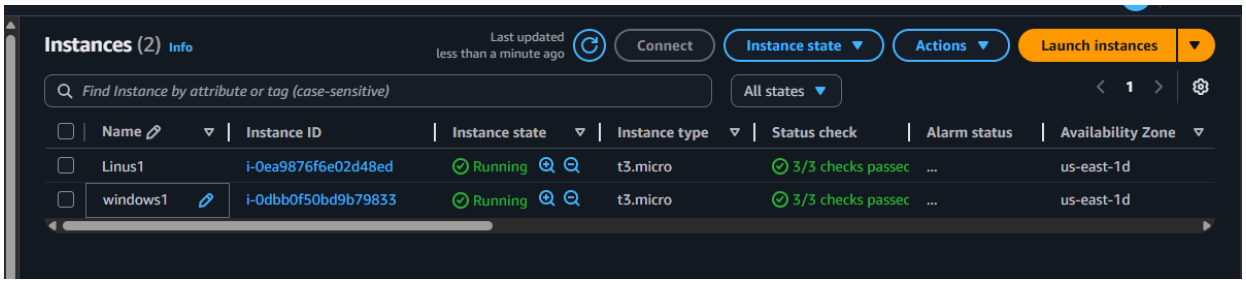
- Select Create security group.
- Allow RDP (port 3389) access from My IP (for better security) or anywhere (0.0.0.0/0).

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.

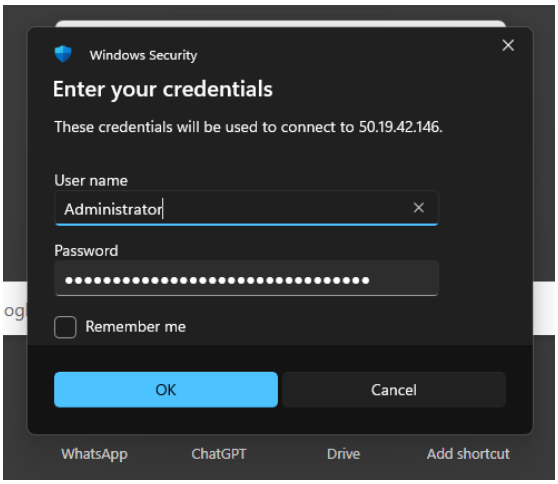
Step 8: Get the Administrator Password

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:

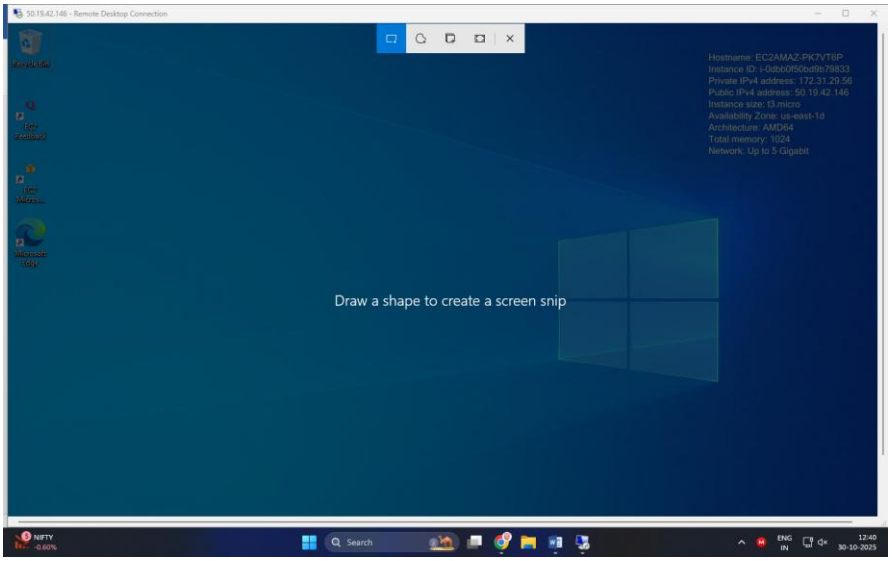
1. Open Remote Desktop Connection (from Start Menu).
2. Enter your instance's Public IPv4 address.
3. Click Connect → Enter:
    - Username: Administrator
    - Password: *(the decrypted password from AWS)*
4. Click OK → accept the certificate → the remote Windows desktop opens!

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.



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.

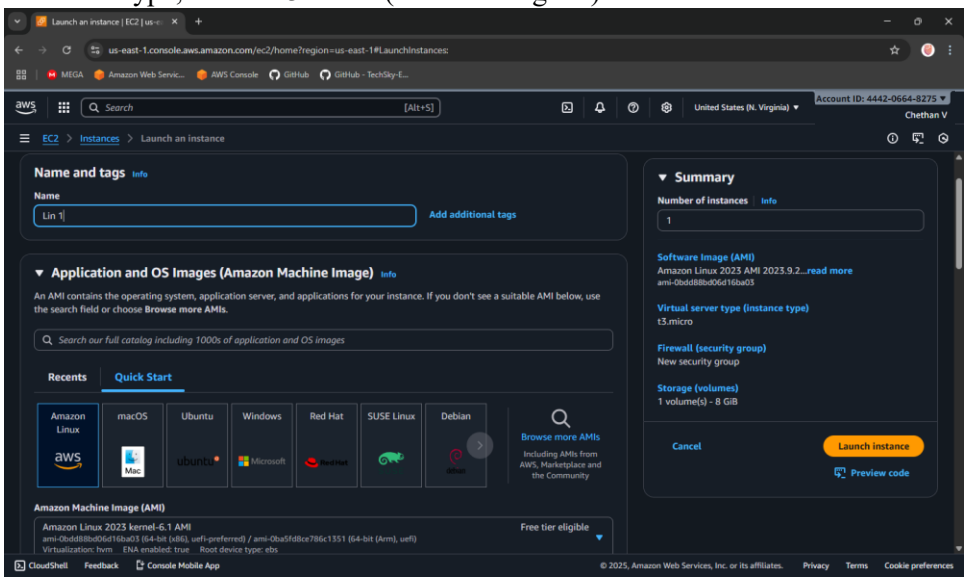
Scenario	Key File Format	Explanation
Using PuTTY on Windows	.ppk	The .ppk file is specific to the GUI based PuTTY application, a popular SSH client for Windows system.
Using PowerShell on Windows, Linux terminal on Linux	.pem	The .pem file is the default AWS key format used by the OpenSSH client available 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:

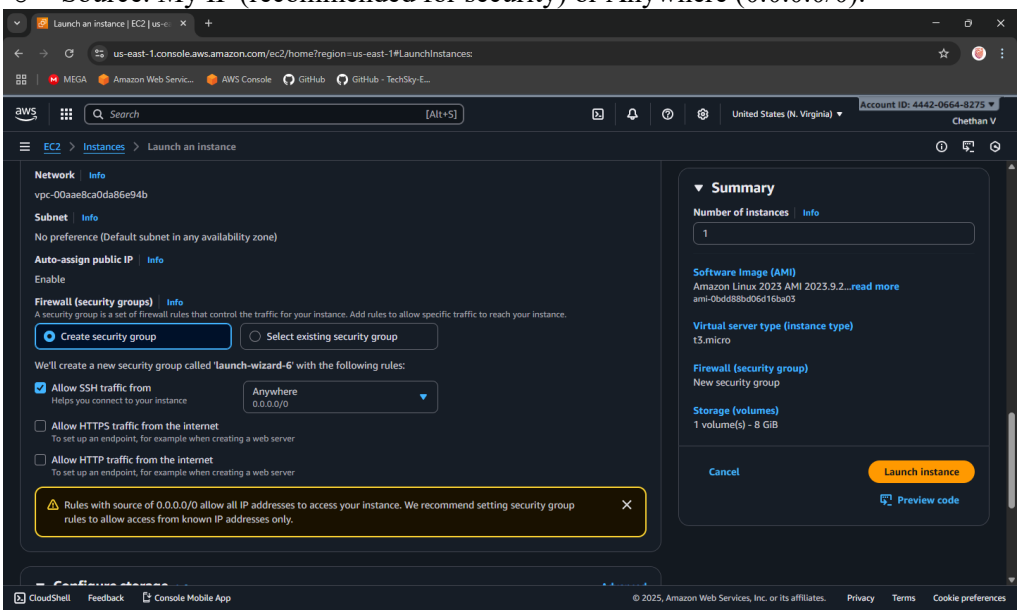
- Key pair type: RSA
- Private key file format: .pem (for SSH via PowerShell/Linux)

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).

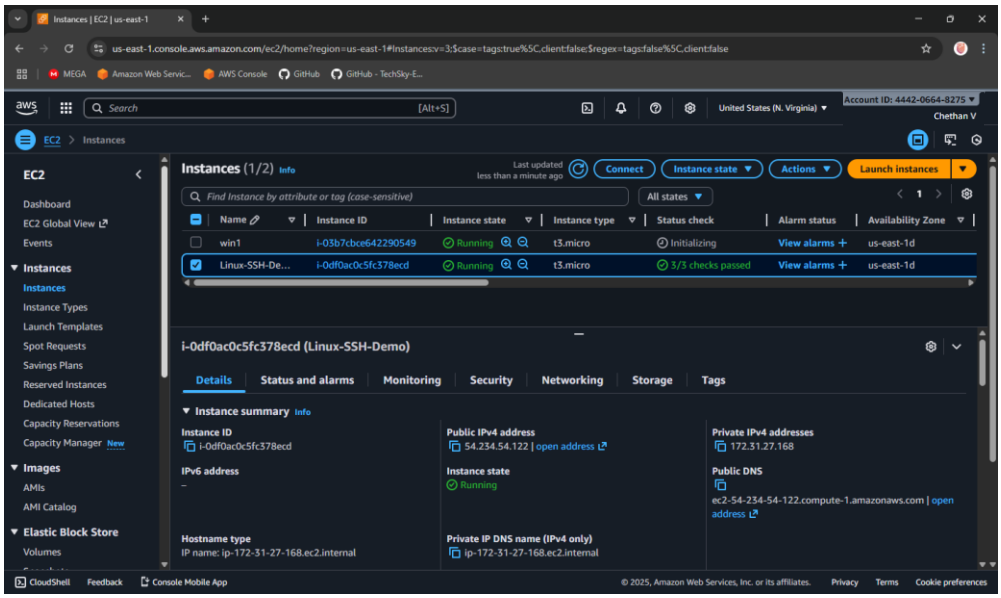


Step 6: Launch the Instance

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

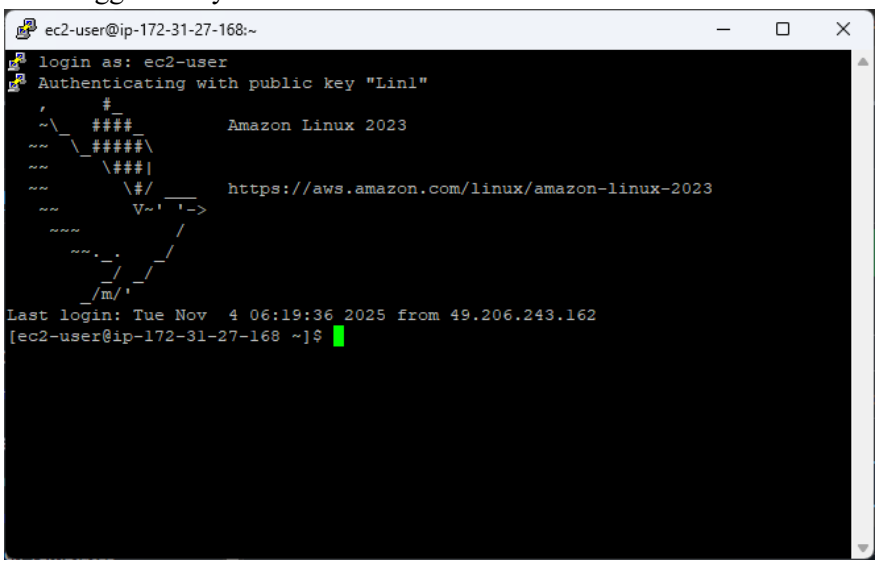
Select your instance → In the summary section →
Copy the Public IPv4 address — you'll use this to connect.



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>
  ssh -i "01.pem" ec2-user@< 54.234.54.122>
- When prompted, type **yes** to continue connecting.
- You'll now be logged into your EC2 Linux terminal.



(B) On Linux Terminal (Ubuntu / macOS)

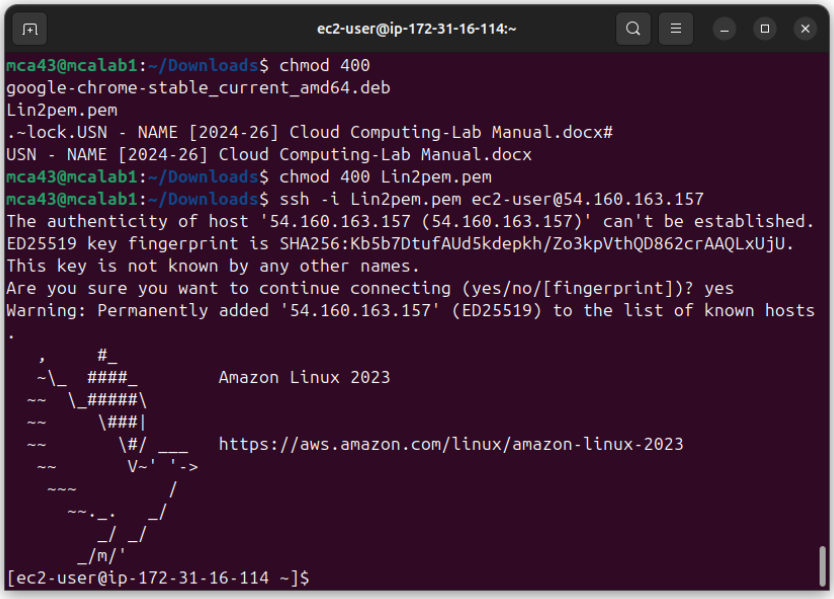
- Open **Terminal**.
- Navigate to the directory where your .pem file is stored.
- Set proper permission for the key file:
  chmod 400 keyfile.pem
- Connect to the instance:
  ssh -i keyfile.pem ec2-user@<Public-IP-address>
- Type **yes** when prompted.
- You will be connected to your EC2 instance remotely.

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

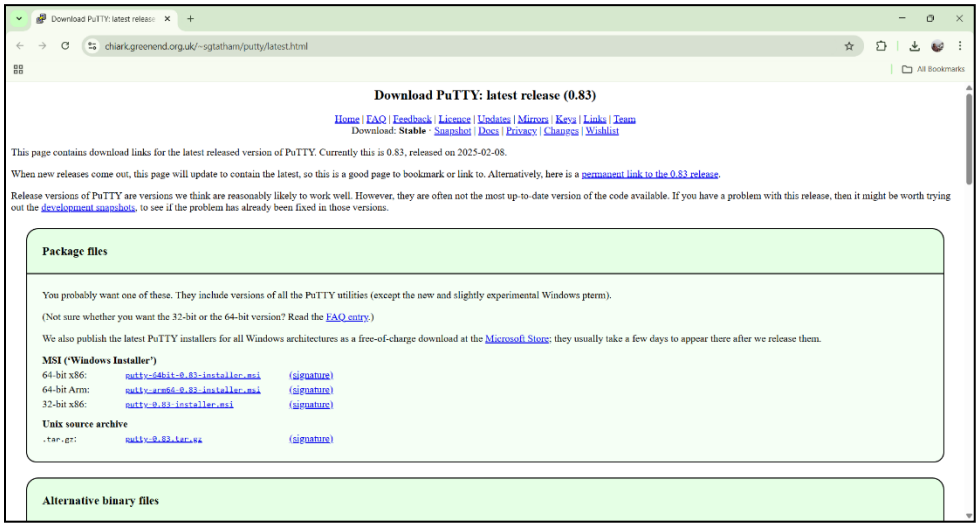
Go to the EC2 console.
Select your instance → Instance State → Stop Instance (to avoid charges).



Steps to Install PuTTY on Windows

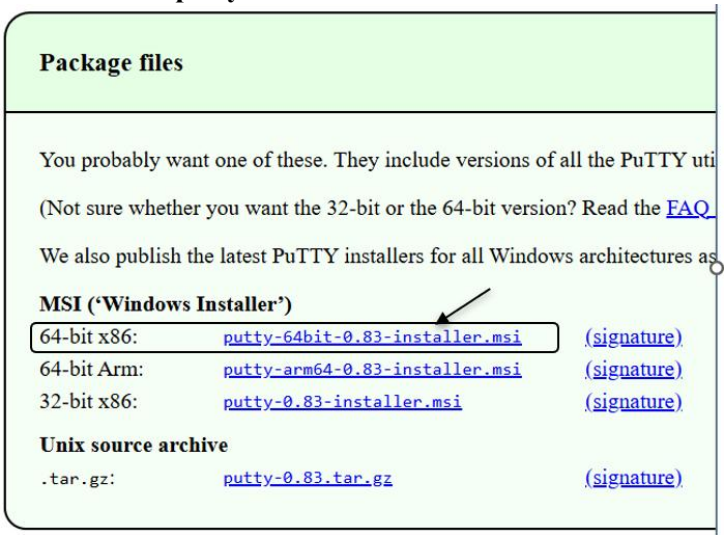
PuTTY is a client program for the SSH, Telnet and Rlogin network protocols. These protocols allow you to interact with a remote server as if you were sitting right in front of it.
It is primarily used in the Windows platform.
In an era where cloud computing and remote servers are the norm, being able 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.

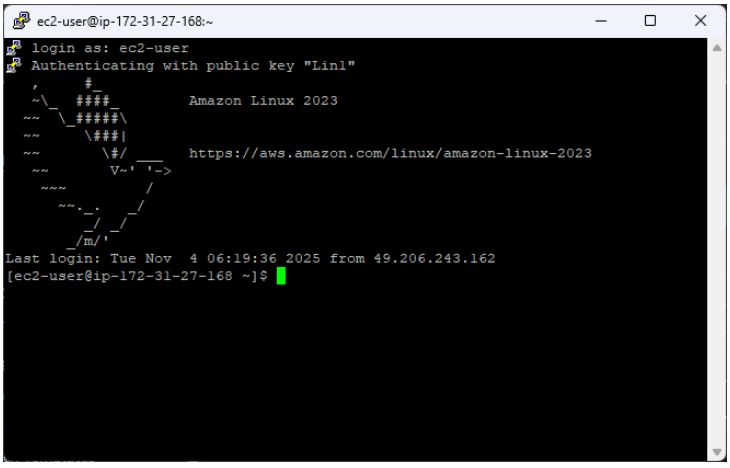
Run this command in your local terminal before connecting:

chmod 400 keypair.pem

chmod 400 MyApacheServer.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

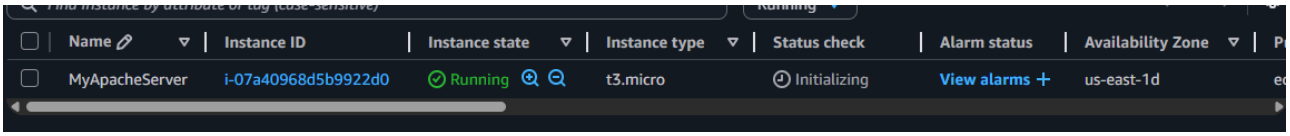
Click → Allow SSH traffic

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

Step 7: Launch Instance

Review → Click **Launch Instance**.

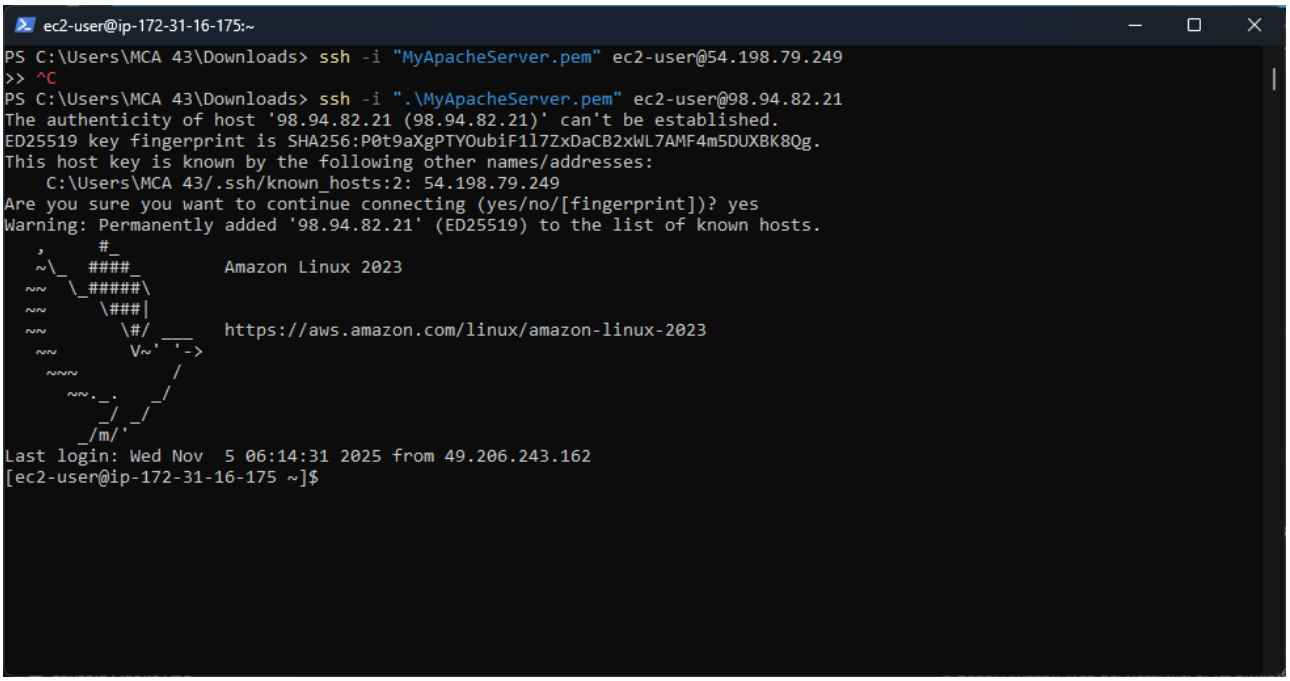
Wait for the instance to start.



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 " MyApacheServer.pem" ec2-user@ 54.198.79.249
- 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

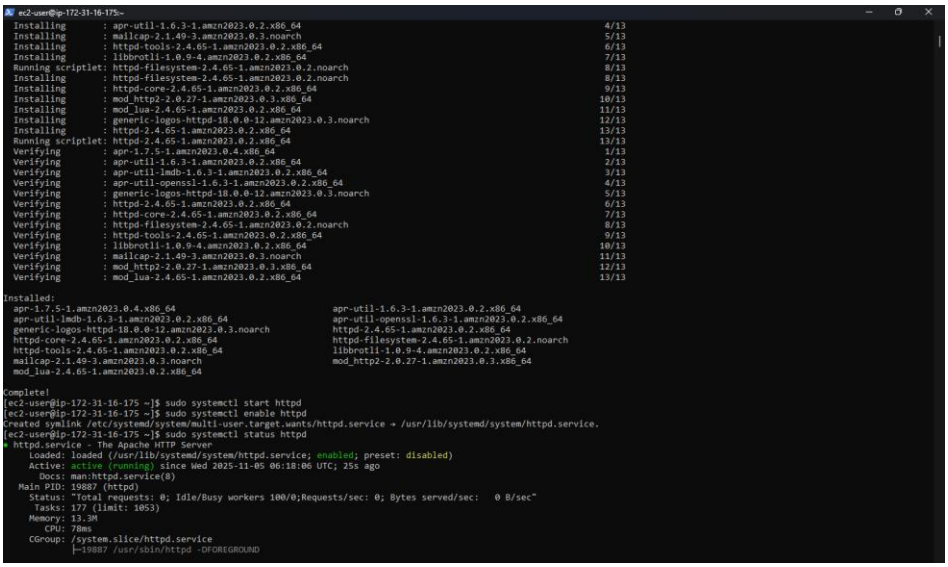
Step 1: Update the Packages
sudo yum update -y

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.]

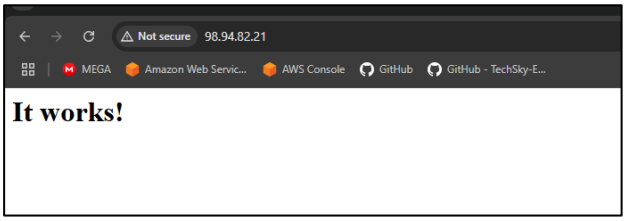


Step 7: Test Apache Server

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]

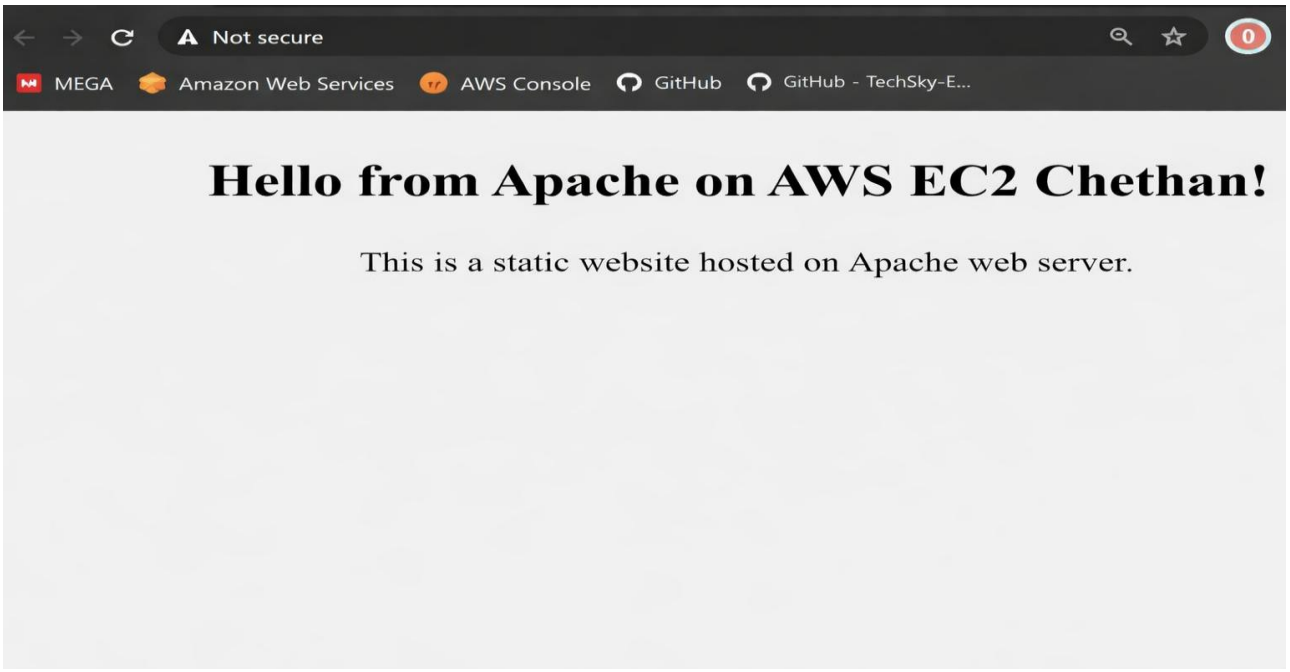
Step 4: Restart Apache
sudo systemctl restart httpd

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

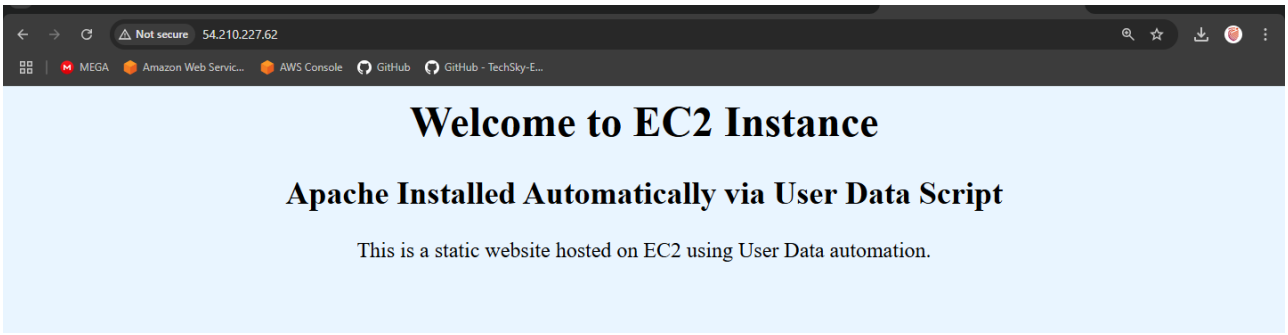
Wait for the status → Running

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!



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

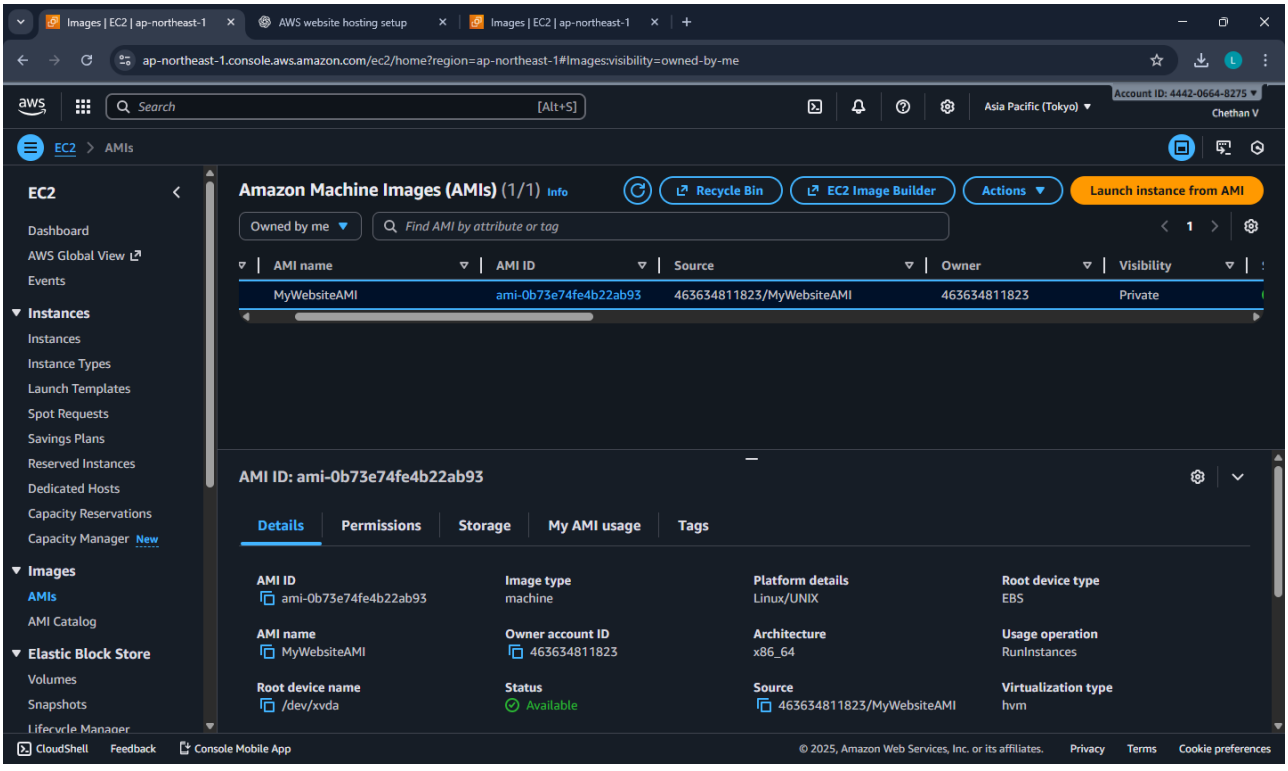
Click **Create image**.

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.



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.

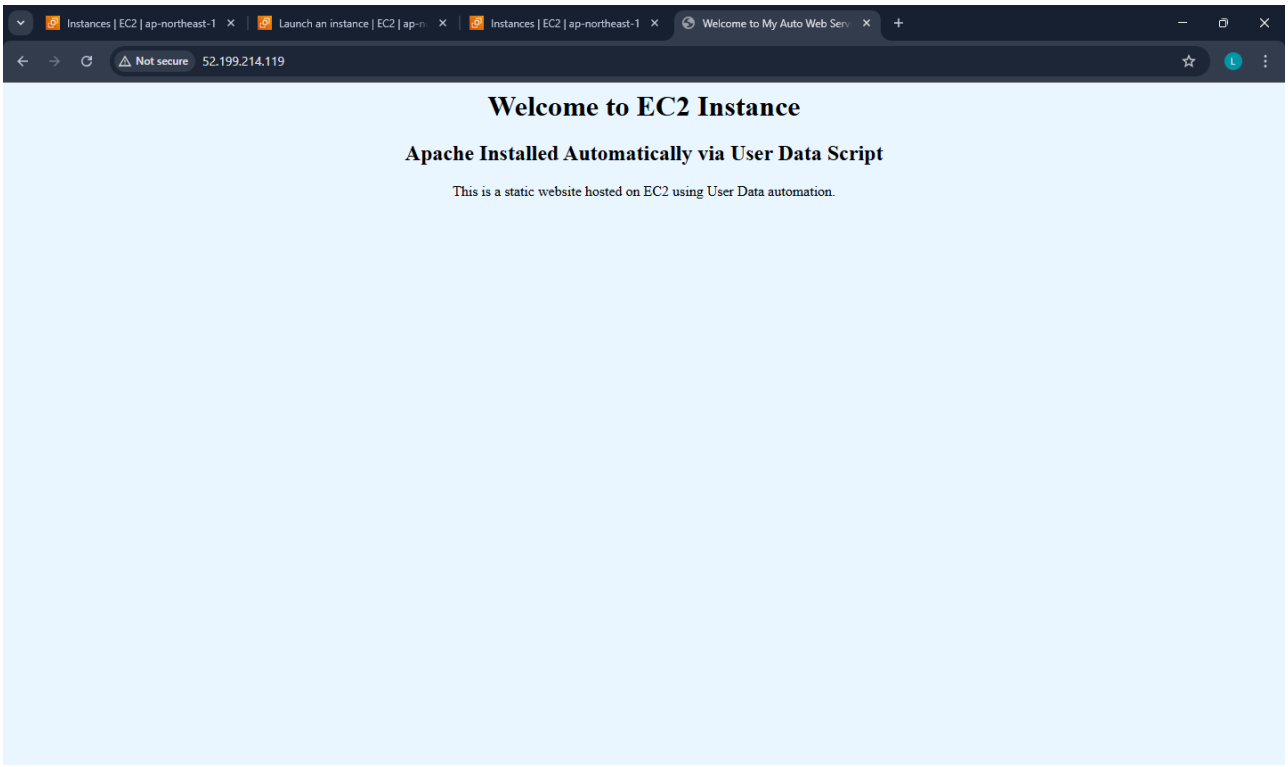
Step 7: Access the Website

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.

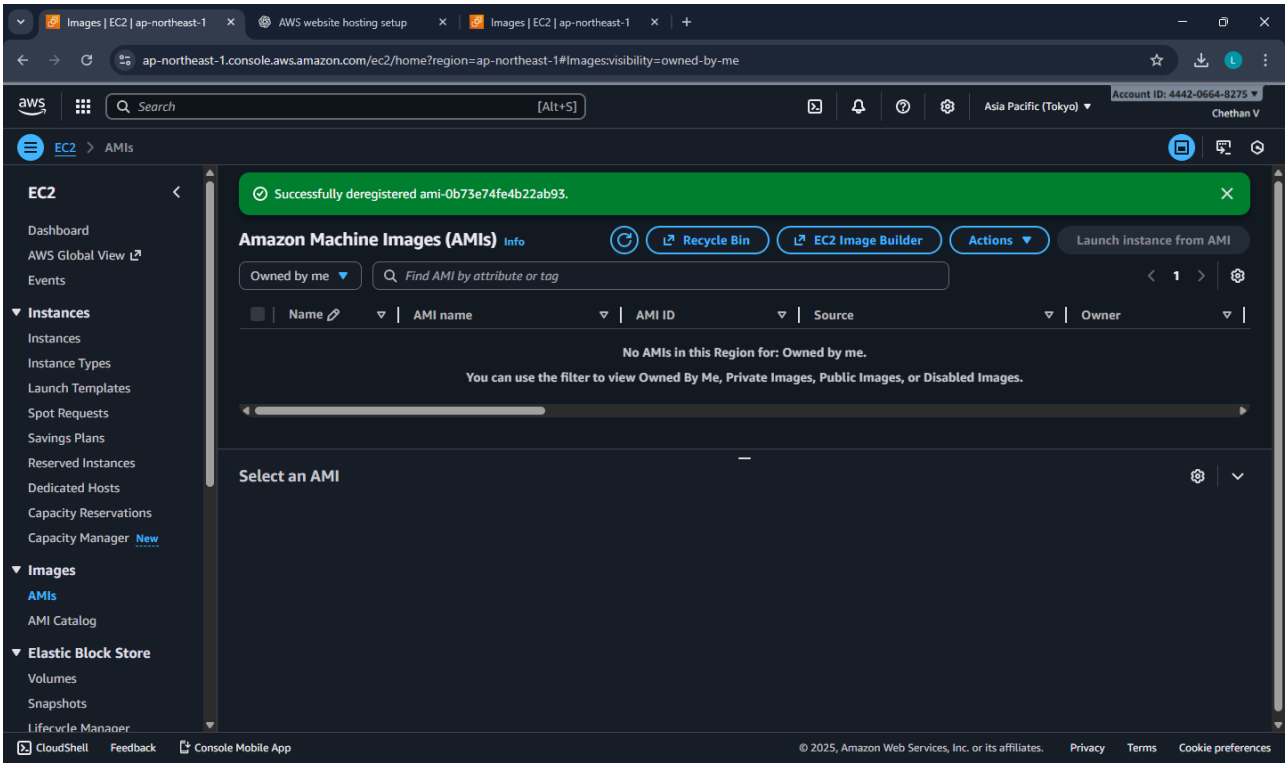
Step 3 – Deregister the AMI

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).



Step 4 – Delete the Associated Snapshot

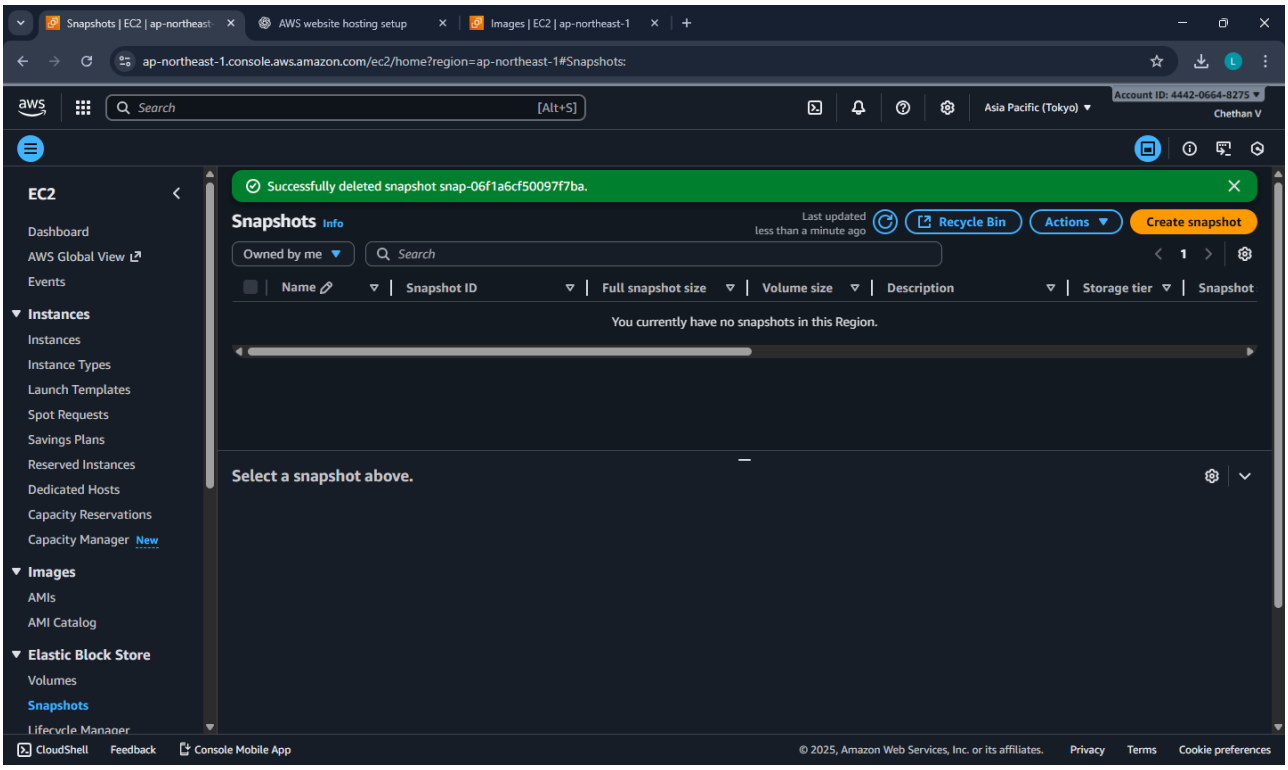
To fully free up storage space (and avoid charges):

In the left navigation pane, click Elastic Block Store → Snapshots.

Find the snapshot linked to your deleted AMI.

You can check the Description column; it usually mentions the AMI ID.

Select the snapshot → Actions → Delete snapshot → Confirm Delete.



- The AMI is now deregistered (unavailable for future instance launches).
- The snapshot is also deleted, freeing up EBS storage and costs.

Date: 11-11-2025

Exercise 9:

MINI-PROJECT

Hosting a Multi-Page Website using EC2 and S3

To design and deploy a multi-page website hosted on an EC2 Linux instance (Apache) that fetches static assets (CSS, JS, images) from an Amazon S3 bucket.

This project demonstrates a two-tier cloud hosting model:

This mini-project demonstrates hybrid hosting: EC2 acts as the web server while S3 delivers static content. Together, they form a scalable, cost-effective architecture similar to real-world cloud applications.

Layer	Service	Role
Web Server	EC2 (Apache on Linux)	Hosts HTML pages (index.html, about.html, etc.)
Object Storage	S3	Stores and delivers static files (style.css, banner.jpg, favicon.ico)

Integration Flow: The EC2 web pages use public S3 object URLs to load all static assets.

OVERALL ORDER

1. Create S3 bucket
2. Upload static assets to S3 (CSS + images)
3. Make them public & copy URLs
4. Create EC2, install Apache
5. Create HTML files on EC2 and paste S3 URLs
6. Test in browser

Follow this order Because HTML needs the S3 URLs — so S3 must be ready first.

STEP 1: Create S3 bucket

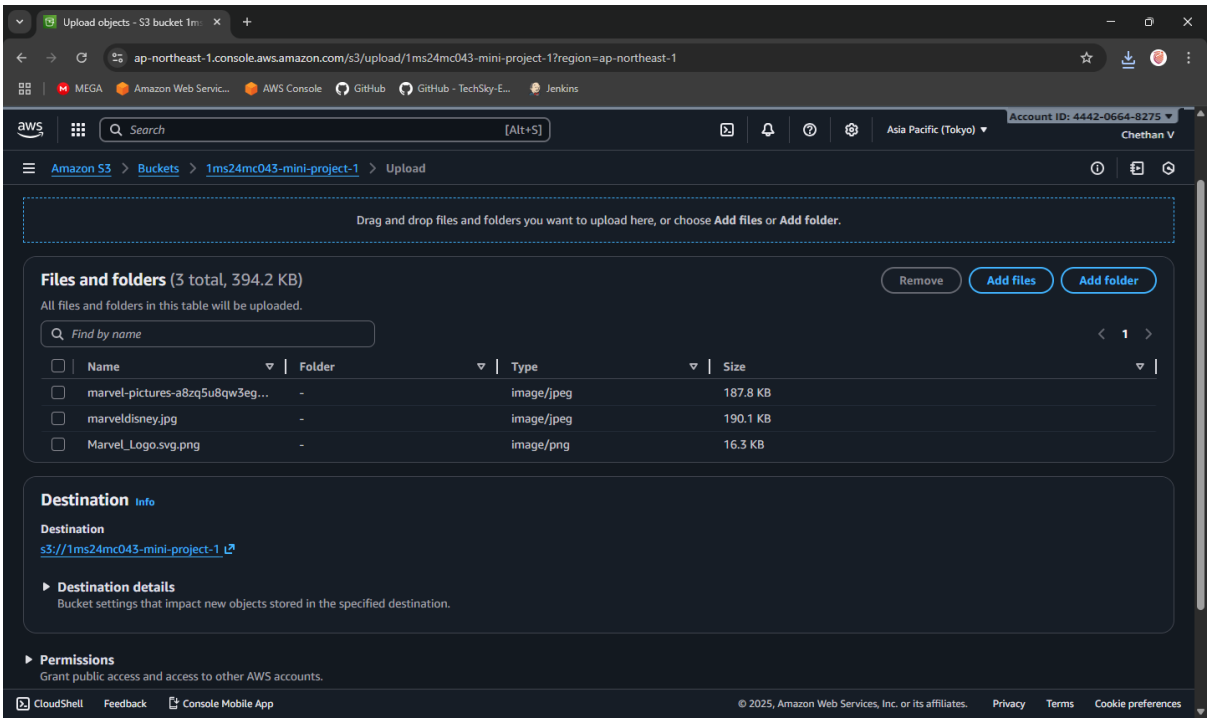
Open **S3 → Create bucket**

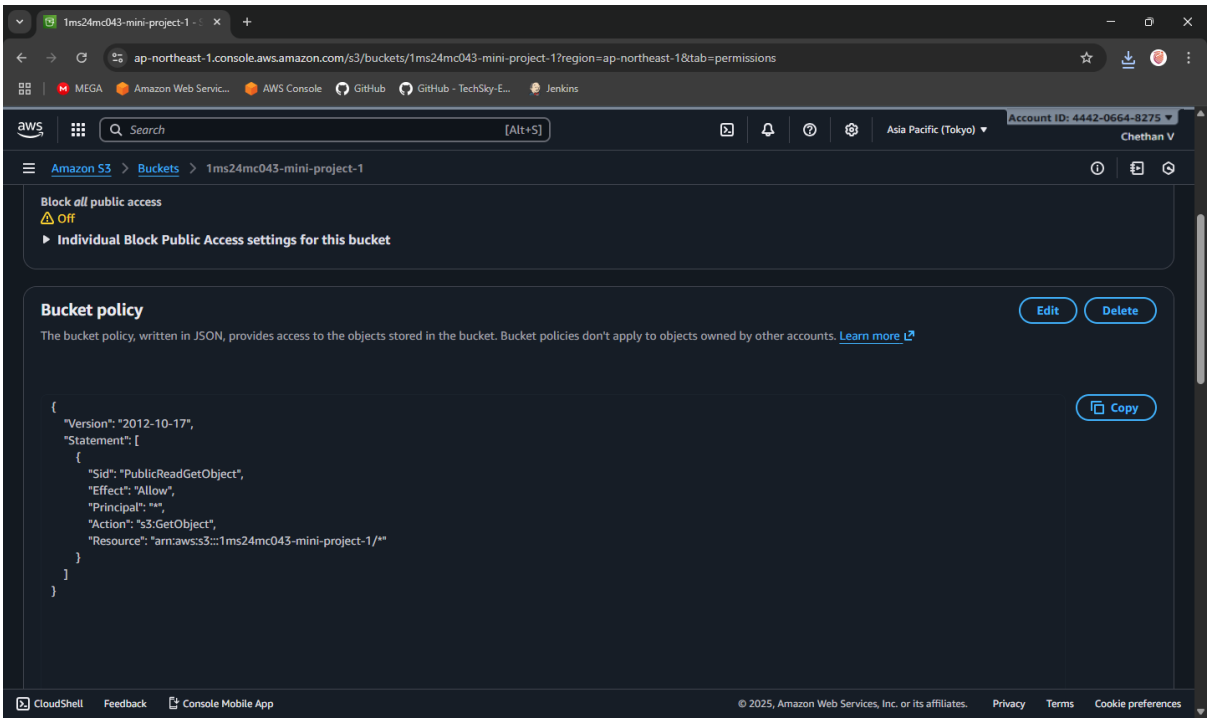
Name: my-miniweb-demo-bucket (or any unique name)

Region: same as EC2

Uncheck "Block all public access" (for lab)

Create.



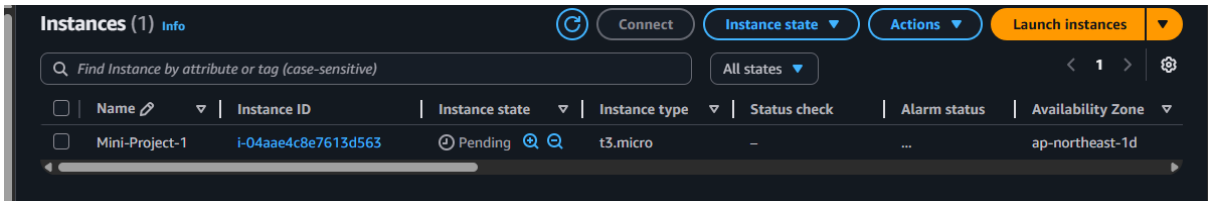


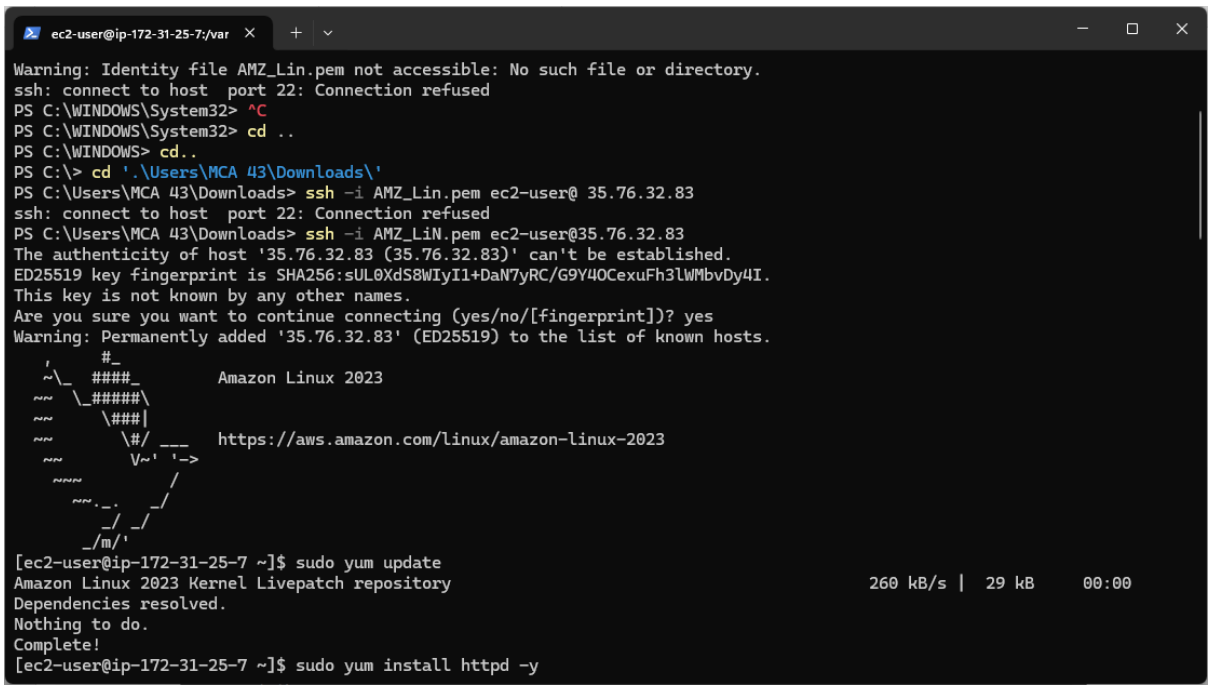
STEP 2: Create STATIC FILES (on your laptop first)

You need **three** static objects to upload to S3:

1. style.css
2. banner.jpg

3. about.jpg





Create a CSS file and save it as – style.css:

```
/* style.css */
body {
  font-family: Arial, sans-serif;
  background-color: #f8f8f8;
  margin: 0;
  padding: 0;
}

header {
  background: #007bff;
  color: white;
  padding: 15px;
```

```
  text-align: center;
}

.container {
  padding: 20px;
  background: white;
  max-width: 900px;
  margin: 20px auto;
  box-shadow: 0 0 5px rgba(0,0,0,0.1);
}

a {
  color: #007bff;
  text-decoration: none;
}

nav a {
  margin-right: 15px;
}

img {
  max-width: 100%;
}
```

1. Open **Notepad** (or Notepad++, VS Code, anything).
2. Paste the CSS.
3. Click **File → Save As…**
4. In the **File name** box, type: style.css
5. In **Save as type**, select **All Files (*.*)**
6. Save it.
7. Then upload style.css to your S3 bucket.

After you upload it to S3, copy the **Object URL** and use it in your HTML like this:
<link rel="stylesheet" href="https://your-bucket-name.s3.amazonaws.com/style.css">

Images:

- Save any image as **banner.jpg** (for homepage)
- Save another image as **about.jpg**

STEP 3: Upload to S3

In your bucket:

Click **Upload** → add:

- ✓ style.css

- ✓ banner.jpg
- ✓ about.jpg

After upload → select each object → **Make public**.

For each file, note the **Object URL**. It will look like:

- https://my-miniweb-demo-bucket.s3.amazonaws.com/style.css
- https://my-miniweb-demo-bucket.s3.amazonaws.com/banner.jpg
- https://my-miniweb-demo-bucket.s3.amazonaws.com/about.jpg

These links will be used in HTML.

Add bucket policy:

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

(Replace bucket name.)

STEP 4: Create EC2 and install Apache

EC2 → Launch instance → Amazon Linux 2 → t3.micro

Security group: allow **HTTP (80)** and **SSH (22)**

Connect with SSH:

ssh -i "yourkey.pem" ec2-user@<EC2-Public-IP>

Install Apache:

sudo yum update -y

sudo yum install httpd -y

sudo systemctl start httpd

sudo systemctl enable httpd

Test: open http://<EC2-Public-IP> → Apache page appears.

STEP 5: Create HTML files on EC2

Go to Apache folder:

cd /var/www/html

We will create **3 pages**:

1. index.html (home)
2. about.html
3. contact.html

While writing HTML, **replace** the S3 links with your actual bucket links.

5.1 index.html

sudo nano index.html [Paste this (change bucket name in 2 places)]:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Mini Web Demo</title>
    <!-- CSS coming from S3 -->
    <link rel="stylesheet" href="https://my-miniweb-demo-bucket.s3.amazonaws.com/style.css">
  </head>
  <body>
    <header>
      <h1>Welcome to My Website</h1>
      <nav>
        <a href="index.html">Home</a>
        <a href="about.html">About</a>
        <a href="contact.html">Contact</a>
      </nav>
    </header>

    <div class="container">
      <h2>Home Page</h2>
      <p>This website is hosted on an EC2 instance, but the images and CSS are coming from
S3.</p>

      <!-- Image coming from S3 -->
      <img src="https://my-miniweb-demo-bucket.s3.amazonaws.com/banner.jpg" alt="Banner
Image">
    </div>
  </body>
</html>
```

Save → Ctrl+O, Enter → Ctrl+X.

5.2 about.html

sudo nano about.html [Paste this (change bucket name in 2 places)]:

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Us</title>
    <link rel="stylesheet" href="https://my-miniweb-demo-bucket.s3.amazonaws.com/style.css">
  </head>
  <body>
    <header>
      <h1>About This Project</h1>
      <nav>
        <a href="index.html">Home</a>
        <a href="about.html">About</a>
        <a href="contact.html">Contact</a>
      </nav>
    </header>

    <div class="container">
      <h2>What are we doing here?</h2>
      <p>We are demonstrating integration between EC2 (for HTML) and S3 (for static assets).</p>
      <p>HTML files are stored in /var/www/html on EC2.</p>
      <p>CSS and images are stored in S3 and accessed using public object URLs.</p>

      <!-- Optional image from S3 -->
      <img src="https://my-miniweb-demo-bucket.s3.amazonaws.com/about.jpg" alt="About
Image">
    </div>
  </body>
</html>
```

5.3 contact.html

sudo nano contact.html [Paste this (change bucket name in 1 place)]:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact</title>
    <link rel="stylesheet" href="https://my-miniweb-demo-bucket.s3.amazonaws.com/style.css">
  </head>
  <body>
```

```
  <header>
    <h1>Contact Us</h1>
    <nav>
      <a href="index.html">Home</a>
      <a href="about.html">About</a>
      <a href="contact.html">Contact</a>
    </nav>
  </header>

  <div class="container">
    <h2>Get in touch</h2>
    <p>Email: demo@example.com</p>
    <p>This page is also hosted on EC2, but it is using the same CSS from S3 for a consistent
look.</p>
  </div>
 </body>
</html>
```

STEP 6: Test

1. Open: http://<EC2-Public-IP>/ → home page
2. Click **About** → /about.html
3. Click **Contact** → /contact.html
4. Right-click → Inspect → Network → you will see CSS and images loading from S3.

QUICK STORAGE SUMMARY

Stored in S3 (public):

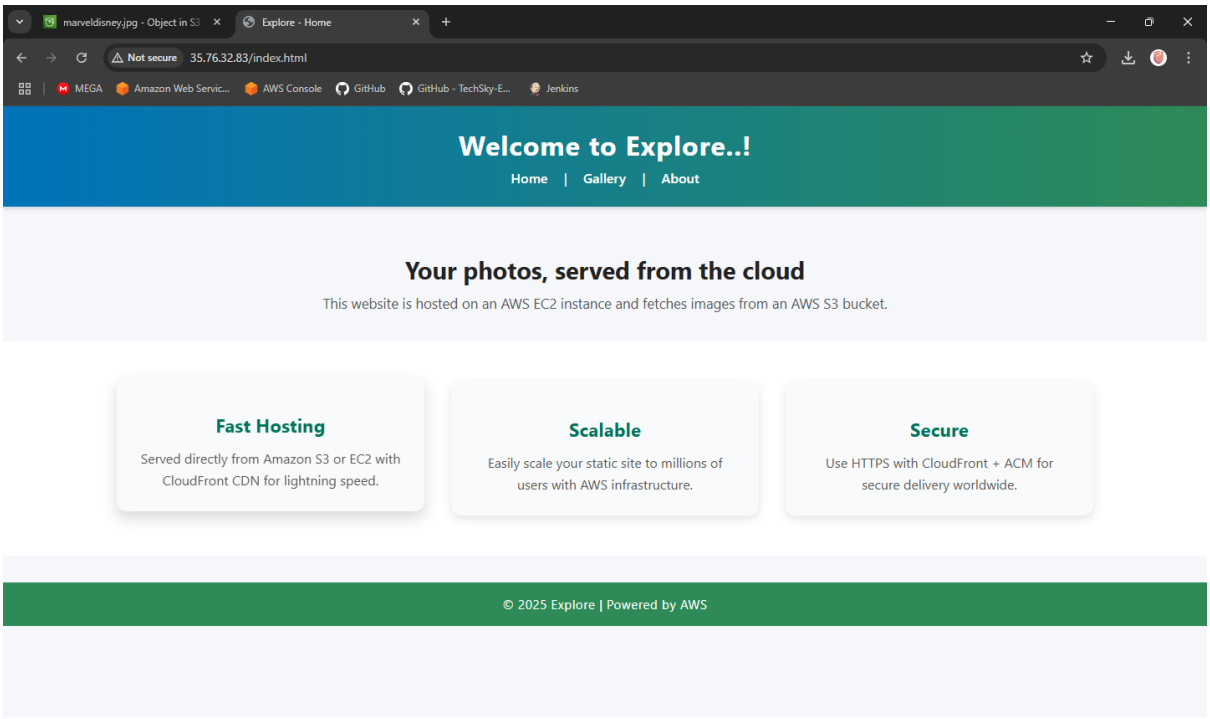
- style.css
- banner.jpg
- about.jpg (optional)

Stored in EC2 (/var/www/html):

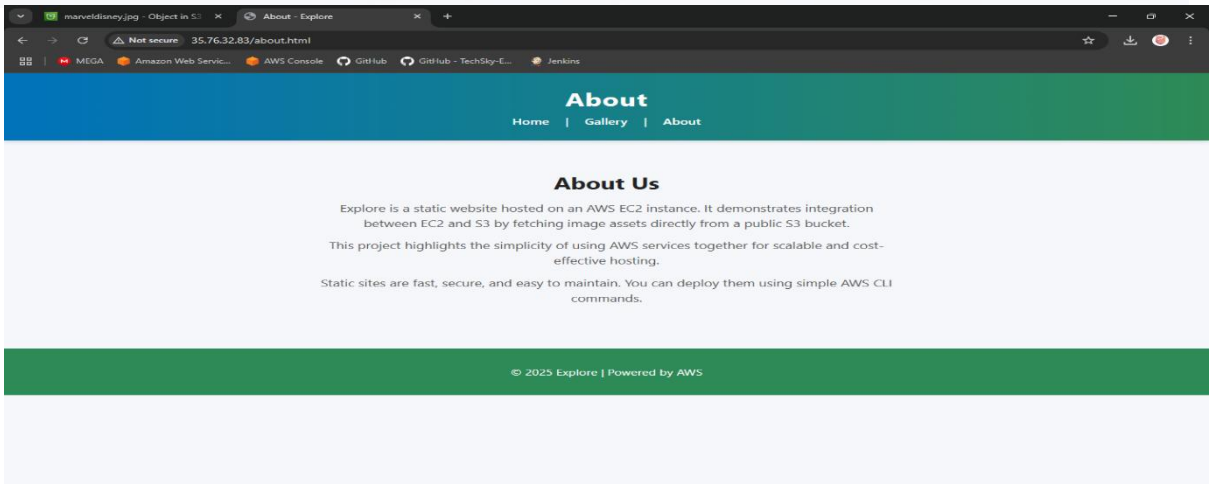
- index.html
- about.html
- contact.html

Linked using S3 Object URL inside the HTML:

- <link rel="stylesheet" href="https://your-bucket.s3.amazonaws.com/style.css">
- <img src="https://your-bucket.s3.amazonaws.com/banner.jpg">


Explore - Home
Not secure
35.76.32.83/index.html
MEGA
Amazon Web Servic...
AWS Console
GitHub
GitHub - TechSky-E...
Jenkins
Welcome to Explore..!
Home | Gallery | About
Your photos, served from the cloud
This website is hosted on an AWS EC2 instance and fetches images from an AWS S3 bucket.
Fast Hosting
Served directly from Amazon S3 or EC2 with CloudFront CDN for lightning speed.
Scalable
Easily scale your static site to millions of users with AWS infrastructure.
Secure
Use HTTPS with CloudFront + ACM for secure delivery worldwide.
© 2025 Explore | Powered by AWS


Not Secure
35.75.73.23/gallery.html
Gallery
Home | Gallery | About
Images Fetched from S3
Images served from AWS S3 | © 2025 Explore


About - Explore
Not secure
35.76.32.83/about.html
About
Home | Gallery | About
About Us
Explore is a static website hosted on an AWS EC2 instance. It demonstrates integration between EC2 and S3 by fetching image assets directly from a public S3 bucket.
This project highlights the simplicity of using AWS services together for scalable and cost-effective hosting.
Static sites are fast, secure, and easy to maintain. You can deploy them using simple AWS CLI commands.
© 2025 Explore | Powered by AWS

Sample Viva Questions

No	Question	Expected Understanding
1	How is S3 integrated with EC2?	EC2 HTML uses direct S3 object URLs for images/CSS.
2	Why store static assets in S3?	S3 provides cost-efficient, durable storage and faster access.
3	What happens if the S3 bucket is private?	Browser shows AccessDenied for assets; need GetObject permission.
4	What is a Favicon?	A small icon displayed on the browser tab.
5	How to enable Versioning in S3?	Select bucket → Properties → Enable Versioning.
6	After stopping and starting EC2, will files persist?	Yes, stored on EBS volume.
7	What is the benefit of this two-tier design?	Demonstrates decoupled web and asset delivery layers.

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 (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 (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)

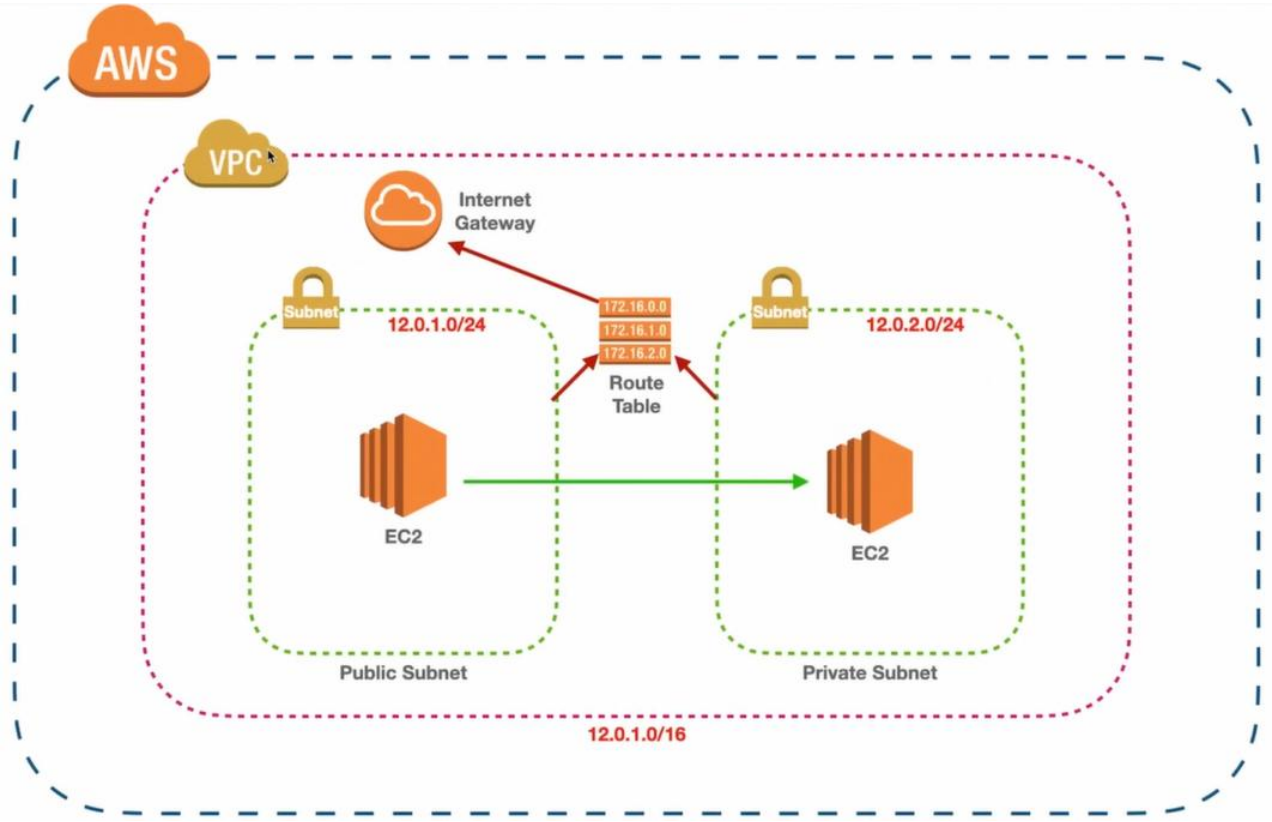
- A **static, public IPv4 address** that can be attached to an EC2 or NAT Gateway.
- Remains constant even if the instance is stopped or restarted.
- Useful for **NAT Gateways**, load balancers, or fixed server access.

Example: Two-Tier Architecture

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)

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

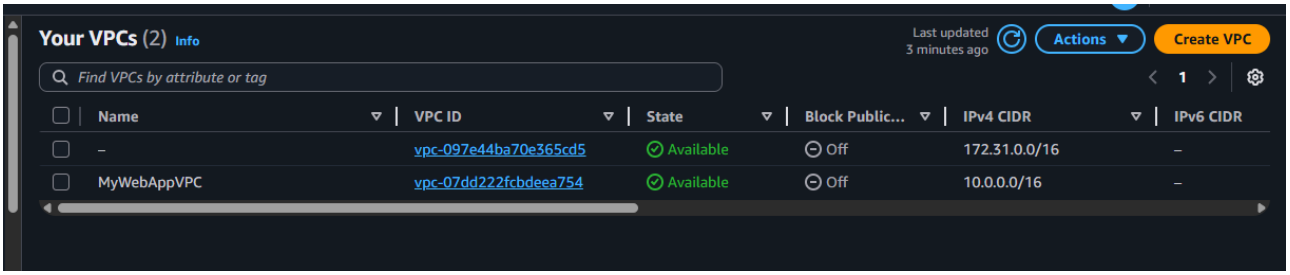
To design and configure a **Virtual Private Cloud (VPC)** in AWS with:

- 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.

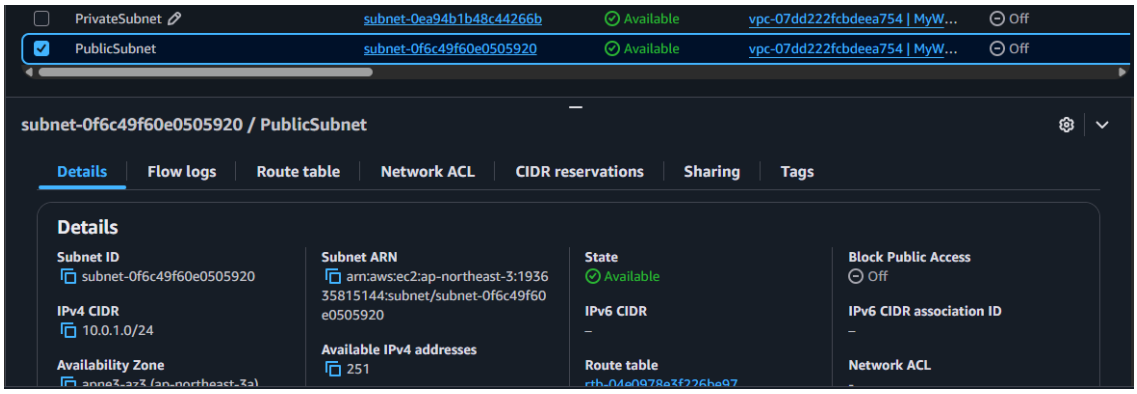


Step 2 – Create Subnets

We'll create two subnets inside the VPC.

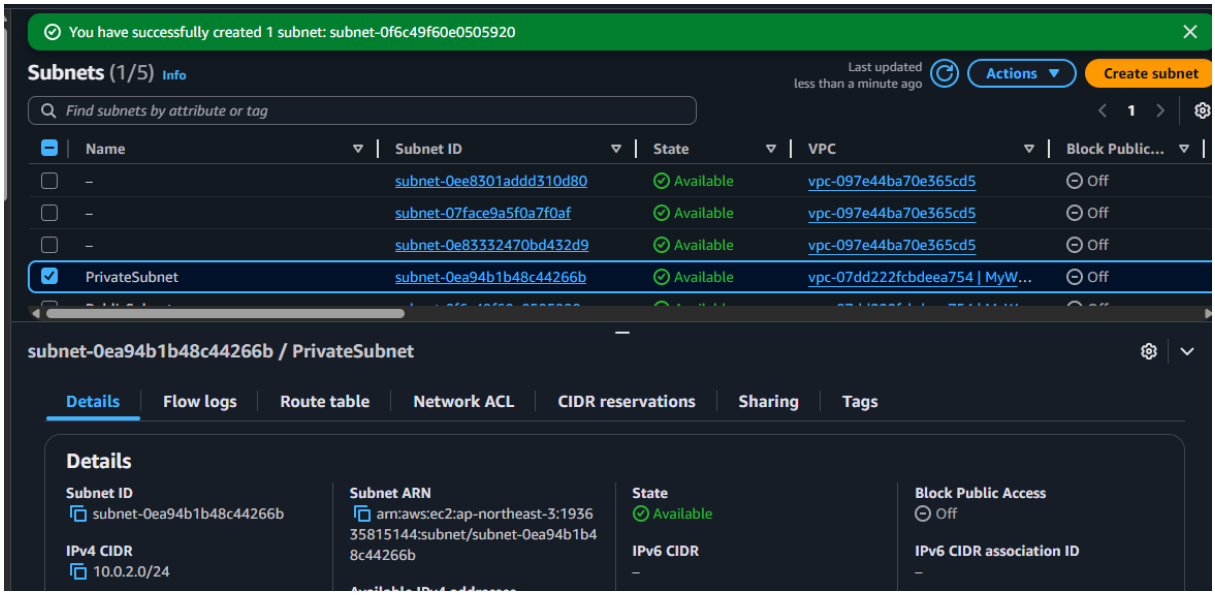
(a) Public Subnet

- 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.



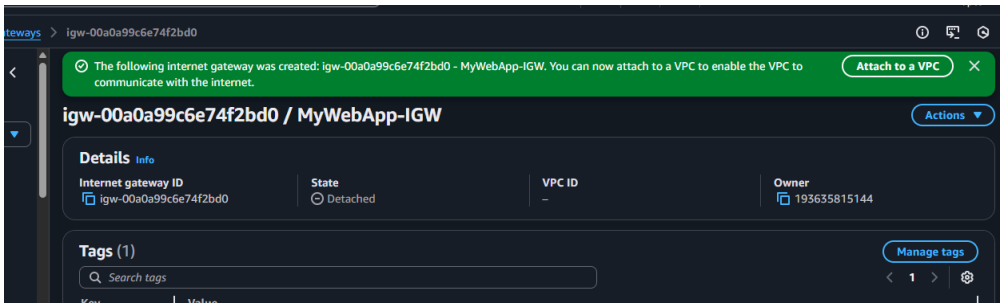
(b) Private Subnet

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



Step 3 – Create and Attach an Internet Gateway

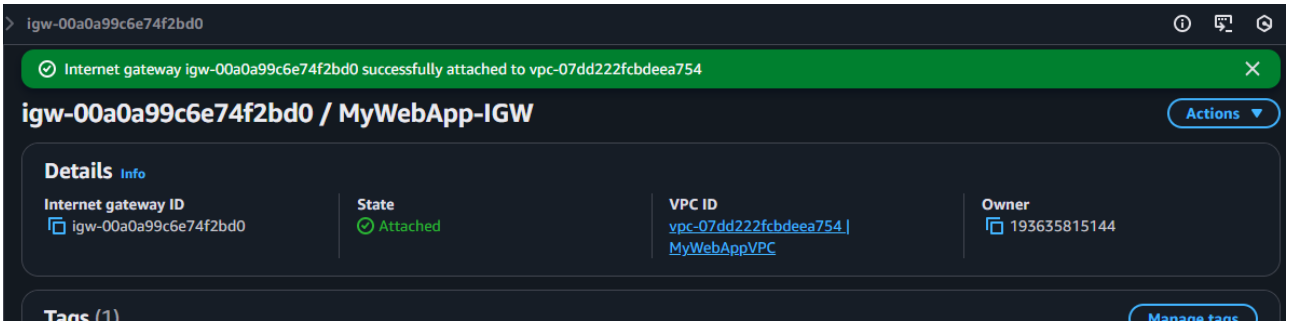
- In the left navigation pane, scroll down and click on “Internet Gateways”.
- Internet Gateways → Create Internet Gateway.
    - Name: MyWebApp-IGW
- Click Create Internet Gateway



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.

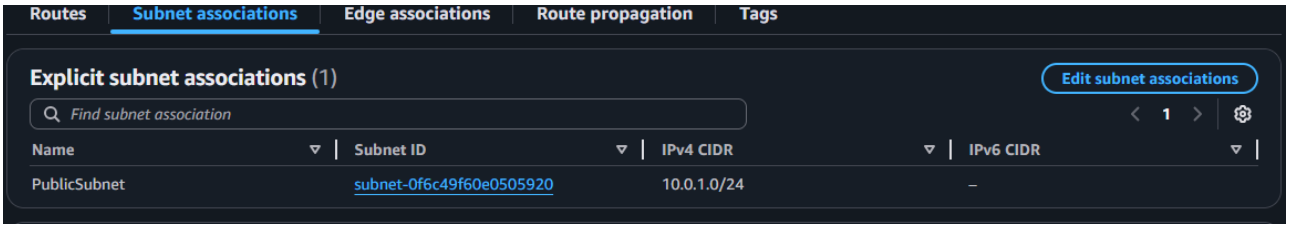


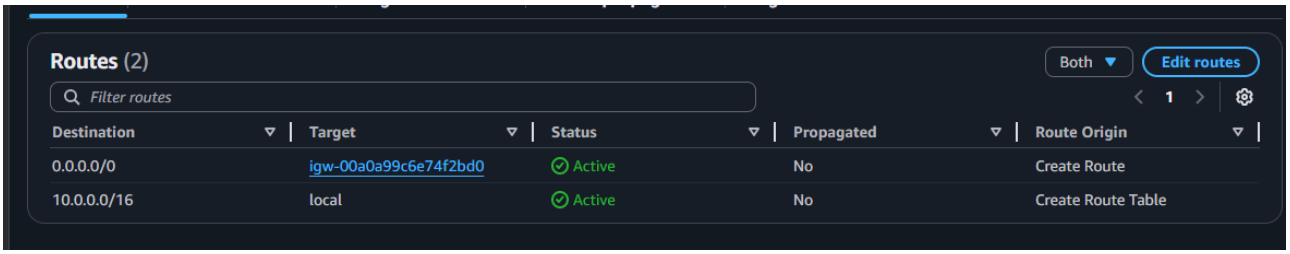
Step 4 – Configure Route Table for IGW

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.





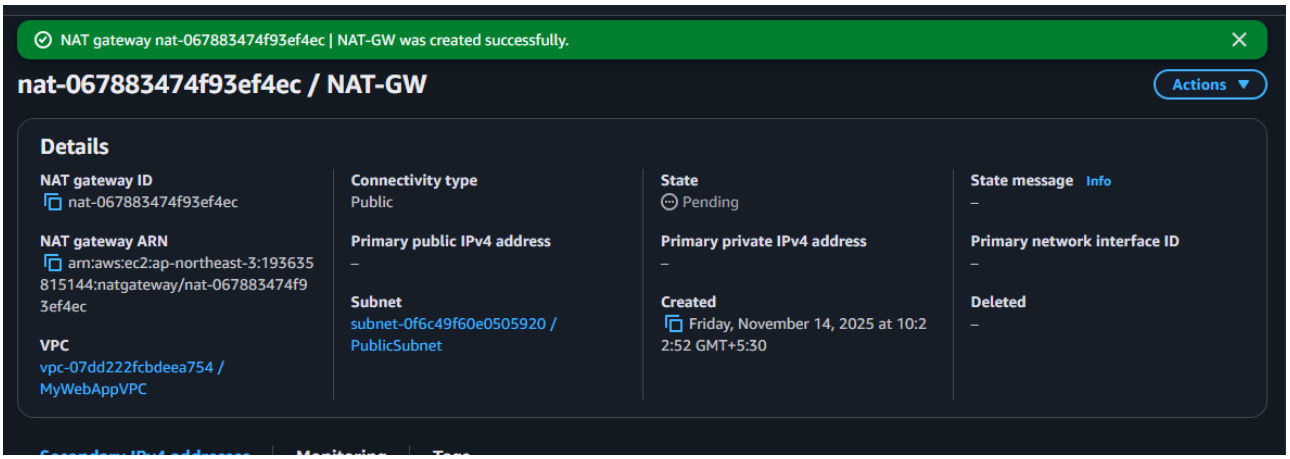
Step 5 – Create a NAT Gateway

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.

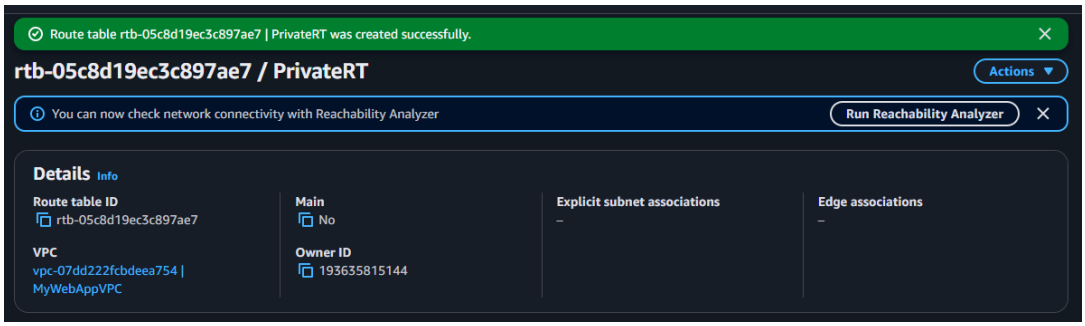
Note: NAT Gateway is a **paid resource**.



Step 6 – Configure Route Table for IGW

Private Route Table

- Create another Route Table → Name PrivateRT.

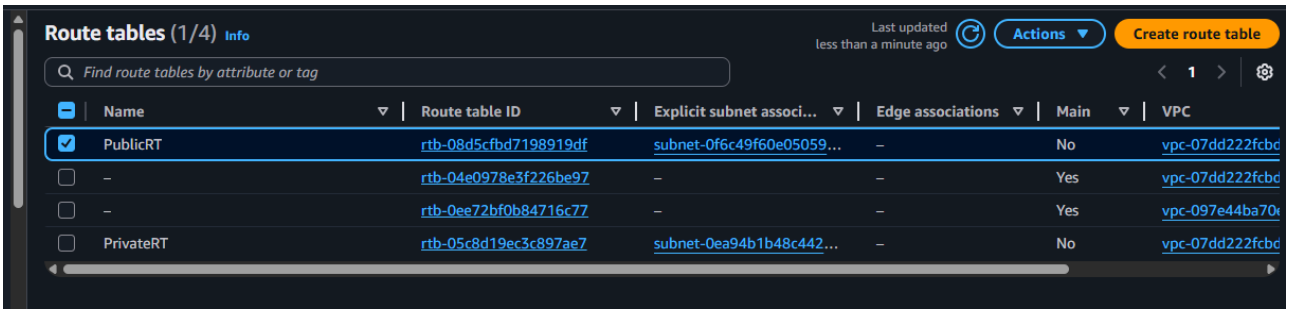


- Under Routes → Edit routes → Add route
    - Destination: 0.0.0.0/0
    - Target: NAT Gateway (MyWebApp-NATGW)
- Under Subnet Associations → Select PrivateSubnet → Save.

PrivateSubnet traffic goes through the NAT Gateway.

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 → t2.micro / 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.

Step 3 — Connect to Public Windows Instance

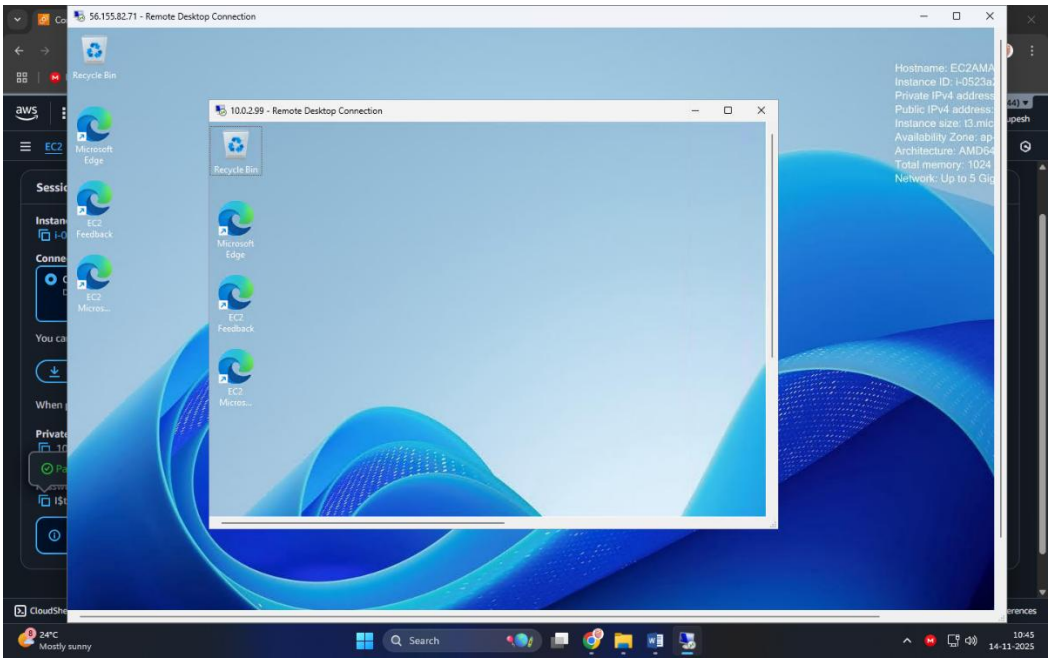
1. Select WebInstance → Click Connect.
2. Choose RDP Client tab.

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.

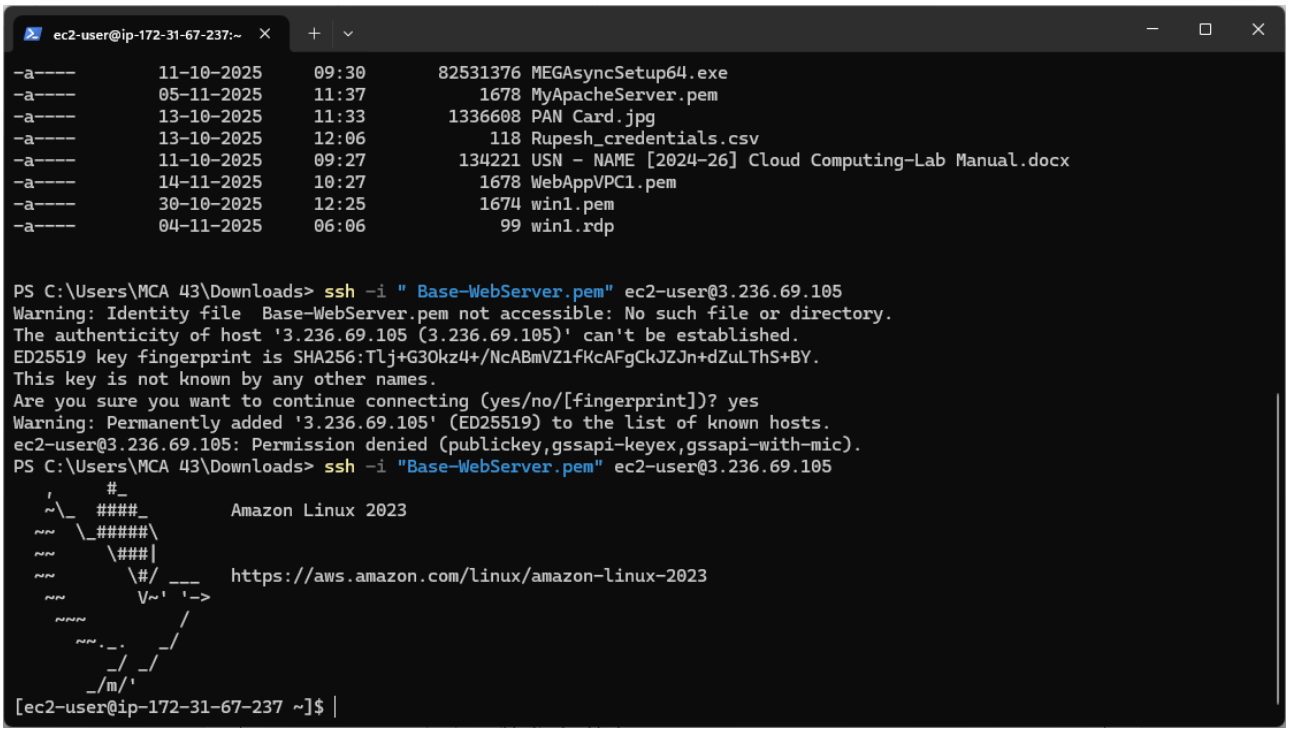
VPC + Subnets

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
- ssh -i "Base-WebServer.pem" ec2-user@ 44.213.68.154



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

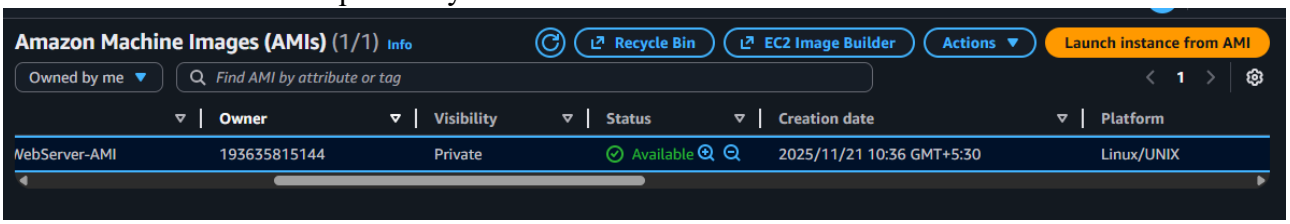
- Move to Web Root Folder - cd /var/www/html
- Create a webpage with hostname:
  - echo "<h1>Hello from Instance 1 — $(hostname)</h1>" | sudo tee /var/www/html/index.html

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

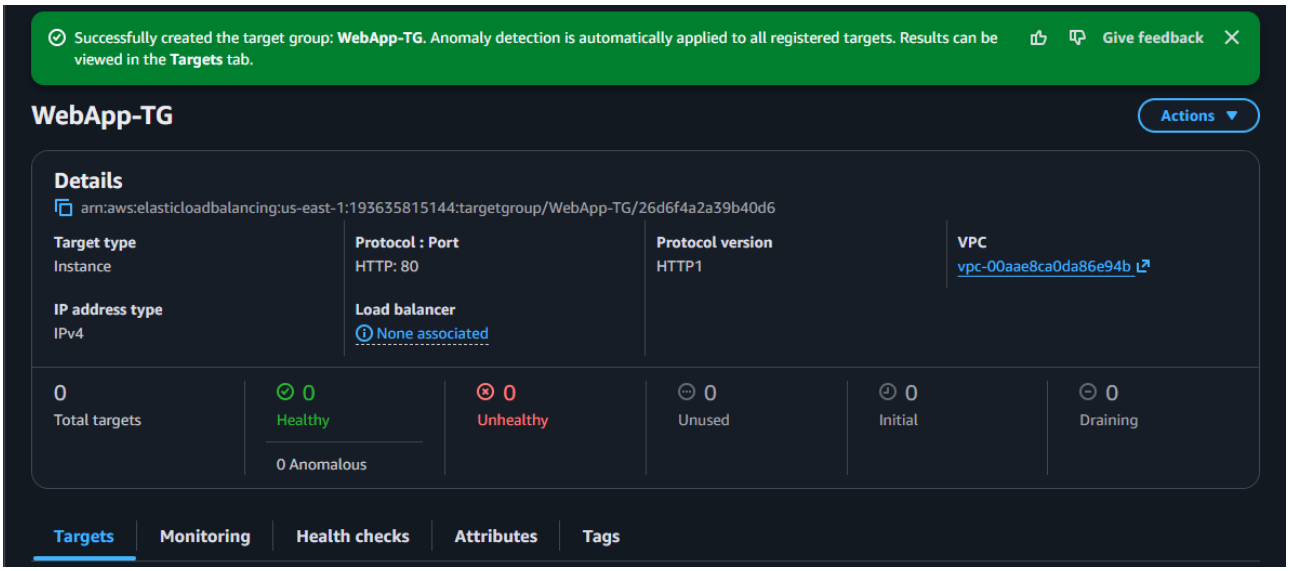
This AMI now contains Apache + your index.html.



STEP 4 — Create Target Group

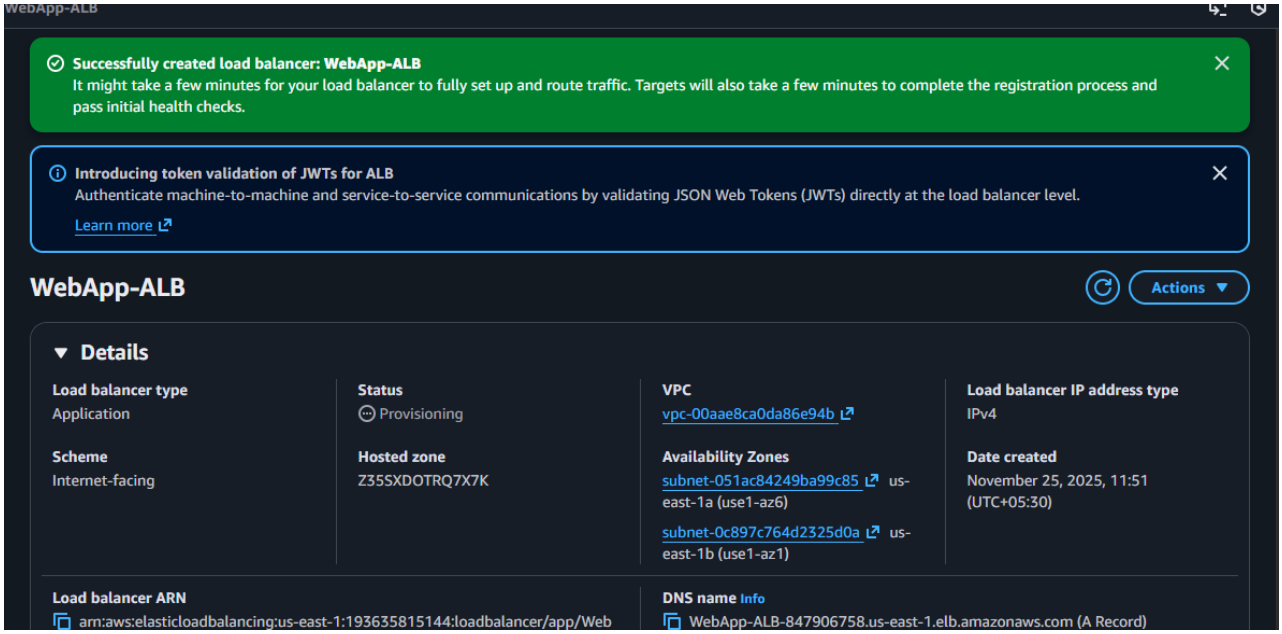
- EC2 → Target Groups → Create Target Group
- Target type: Instances
- Name: WebApp-TG
- Protocol: HTTP
- Port: 80
- Health Check Path: /
- Create (do not register instances manually)

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



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

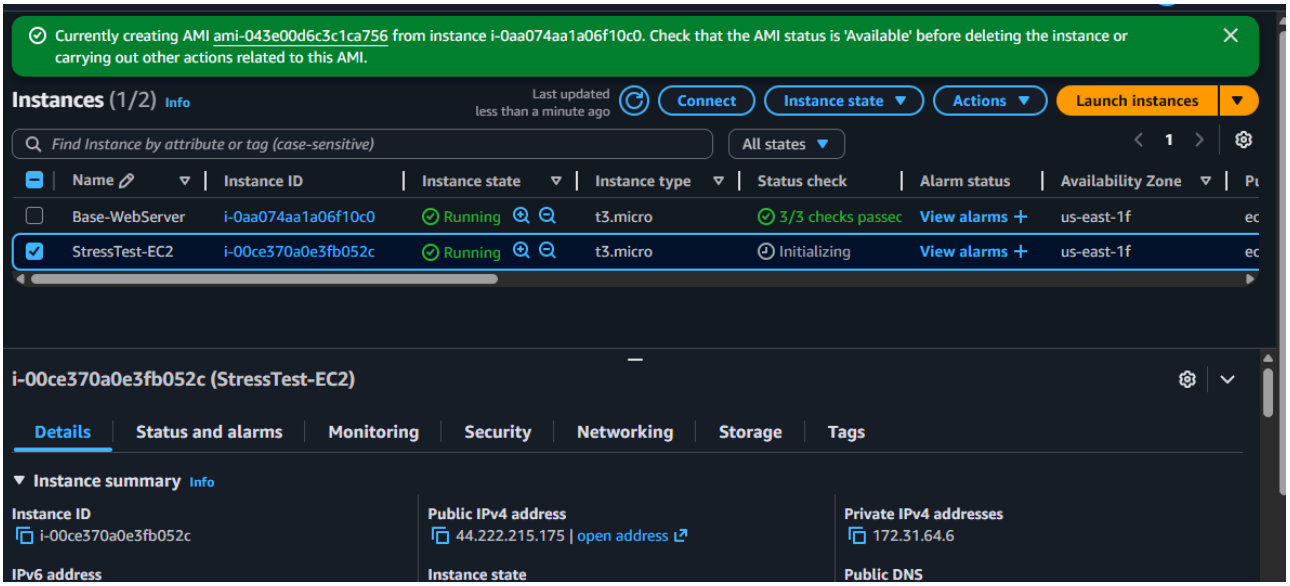
DATE: 20-11-25

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 "AMZ-LIN.pem" ec2-user@ 44.221.74.199

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

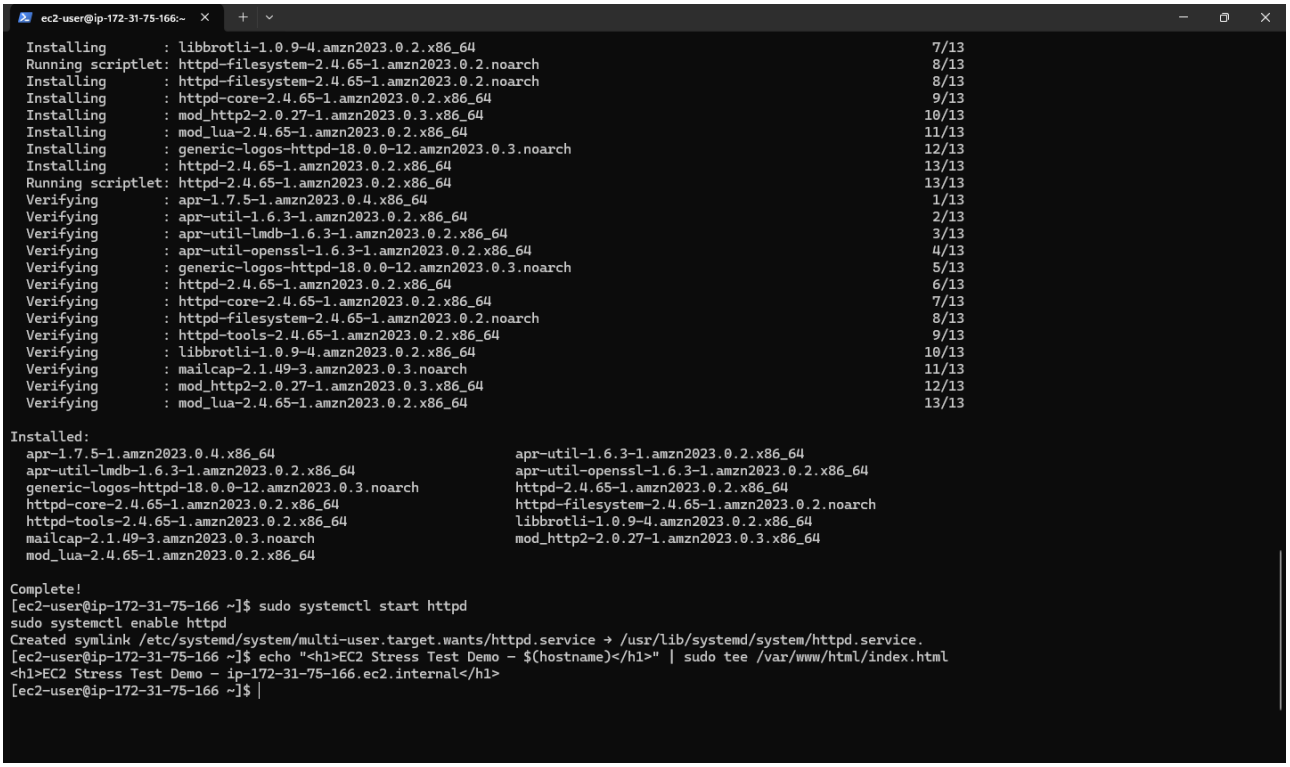
sudo yum install httpd -y

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 ~
```

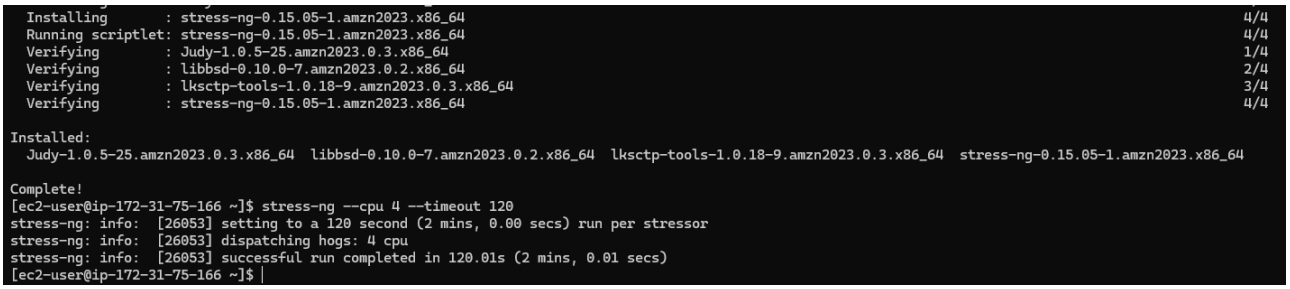
Run CPU stress for 2 minutes using 4 CPU workers:

```
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

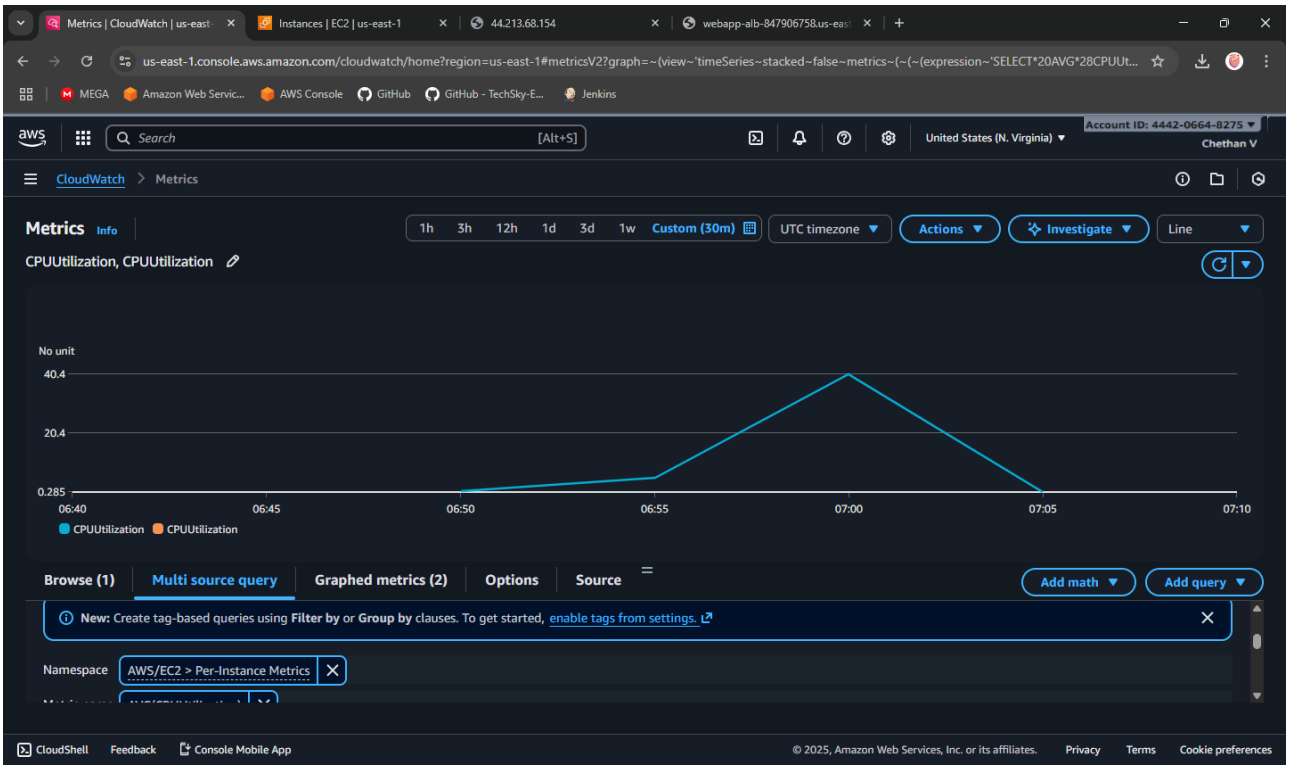
Go to CloudWatch → Metrics → EC2 → Per-Instance Metrics

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.

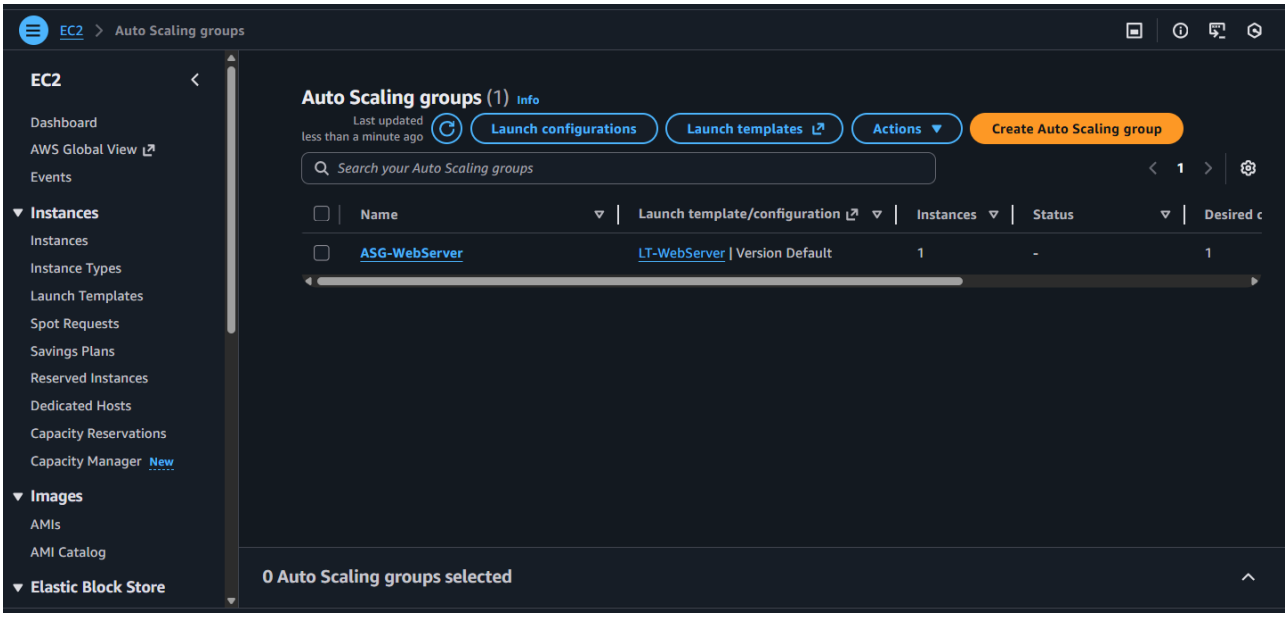
Verify ASG Instance Creation

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-xxxxxxxxxxxx (Running)



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.

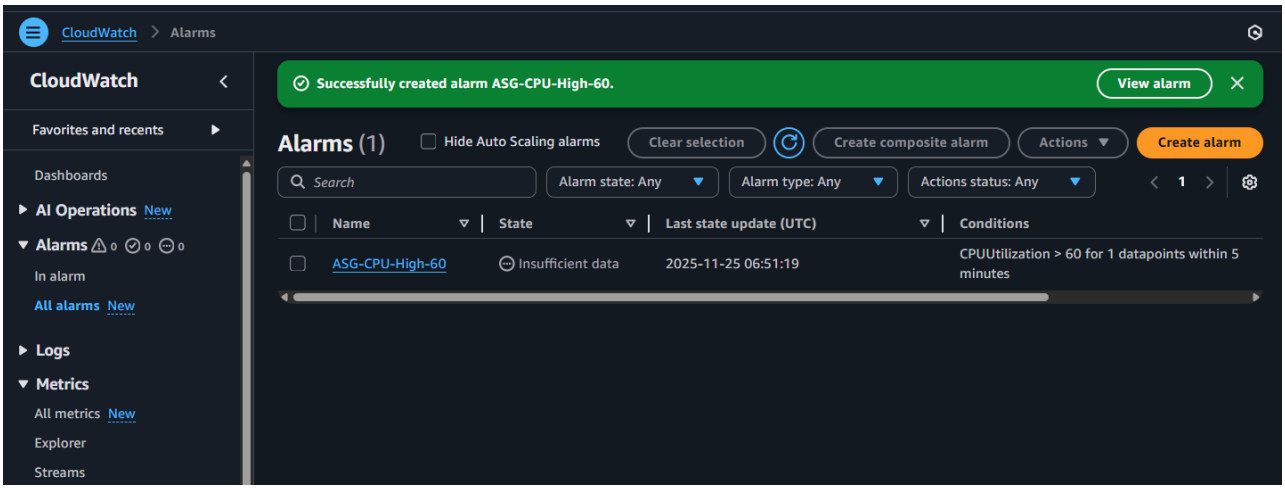
Name and create

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: ScaleOut-CPU60

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.

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.

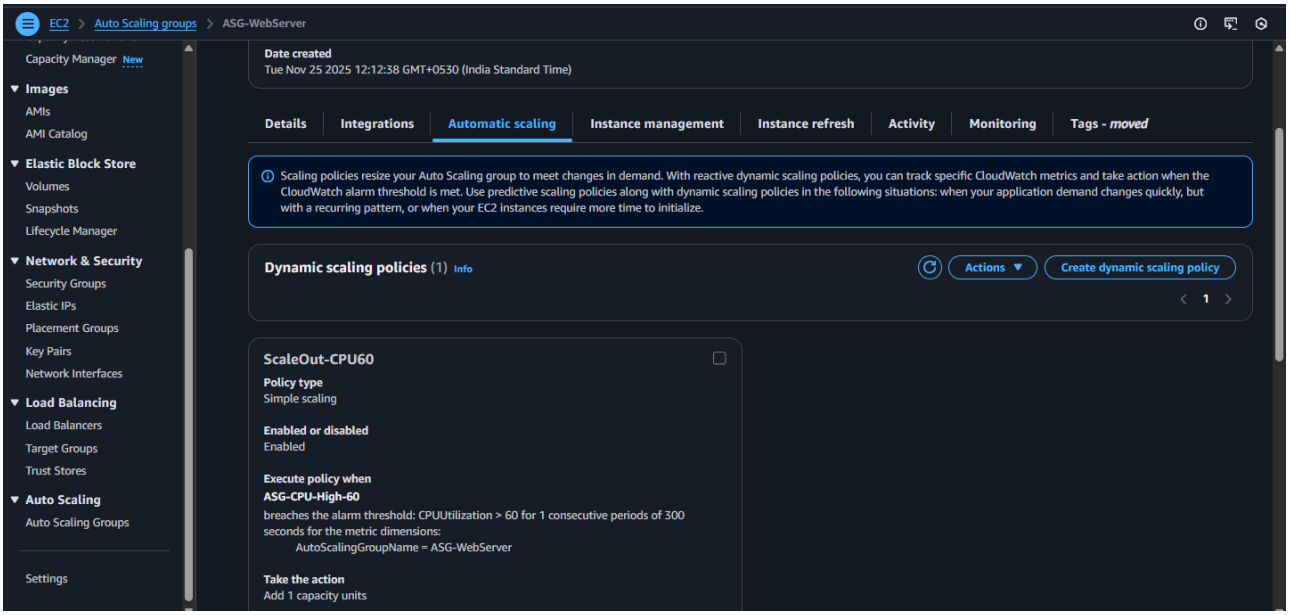
Click Next → if popup appears, Proceed without actions.

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 hostname
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.

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/
├── app.py
├── templates/
    ├── form.html
    ├── result.html
```

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:

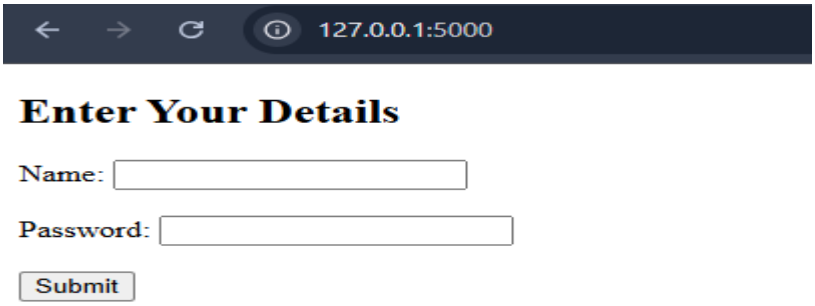
```
from flask import Flask, request, render_template

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.xxxxxx.ap-south-1.rds.amazonaws.com
   'user': 'admin',                    # master username created in RDS
   'password': 'YourPasswordHere',          # master password created in RDS
   'database': 'flaskdb'                  # database name created in MySQL
}
```

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

6.2 app.py

This is the **main Flask application**.

app.py

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

app = Flask(__name__)

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

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

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

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

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

    cur.close()
    conn.close()

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

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

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

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

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

    cur.close()
    conn.close()

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

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

7. HTML Templates (Frontend)

All HTML files go inside the templates/ folder.

7.1 templates/register.html

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

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

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

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

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

7.2 templates/login.html

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

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

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

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

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

7.3 templates/welcome.html

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

7.4 templates/success.html

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

7.5 templates/error.html

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

Deployment Steps –

PHASE 1 – Launch EC2 Linux Instance

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

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

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

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

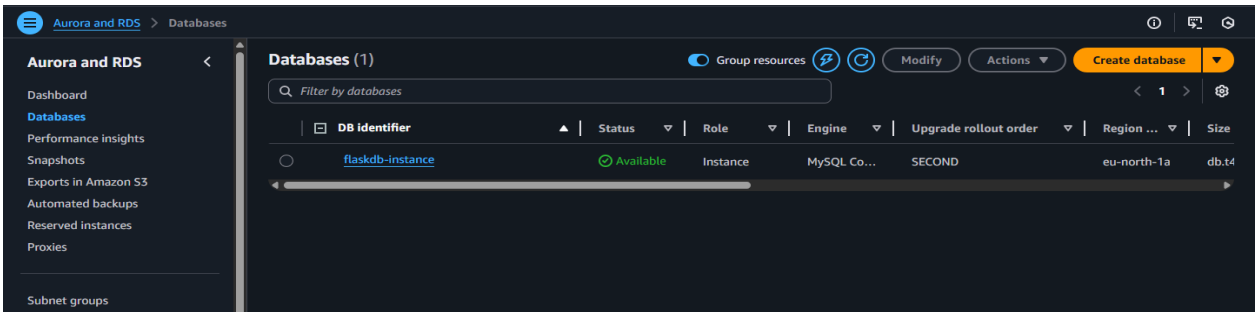
- Create project folder:
  mkdir FlaskLoginApp
  cd FlaskLoginApp
- Create templates folder:
  mkdir templates
- Create the Python and HTML files using nano:
  nano db_config.py → paste code for db_config.py
  nano app.py → paste code for app.py

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".



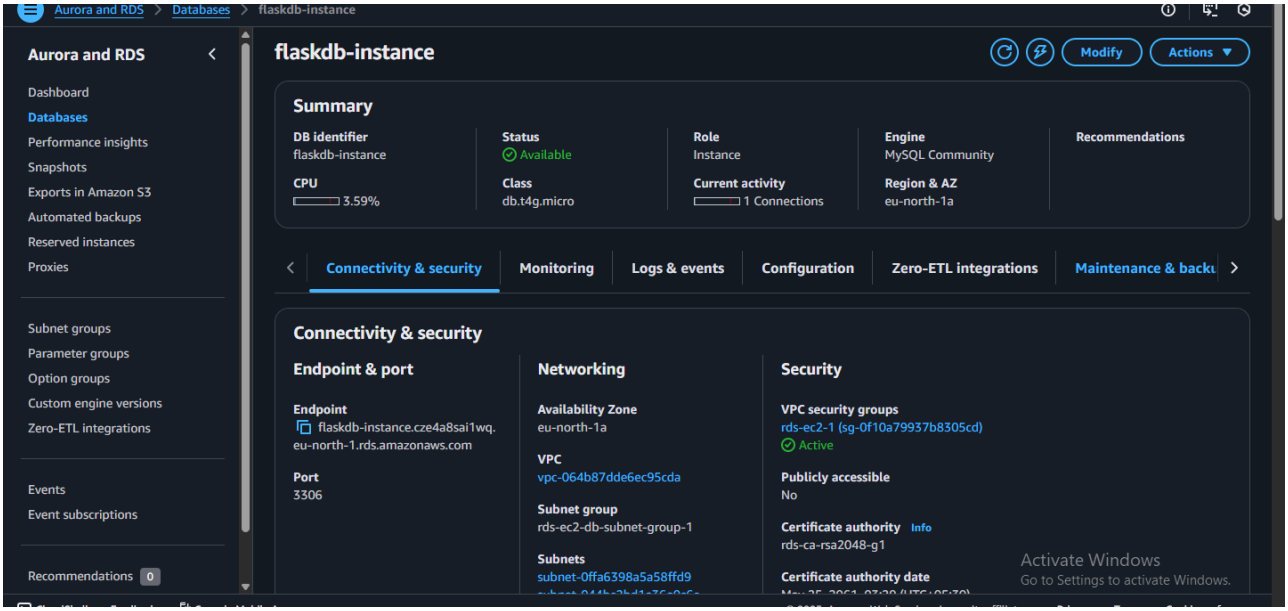
PHASE 5 – Verify RDS Security Group (Auto-configured)

- 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.



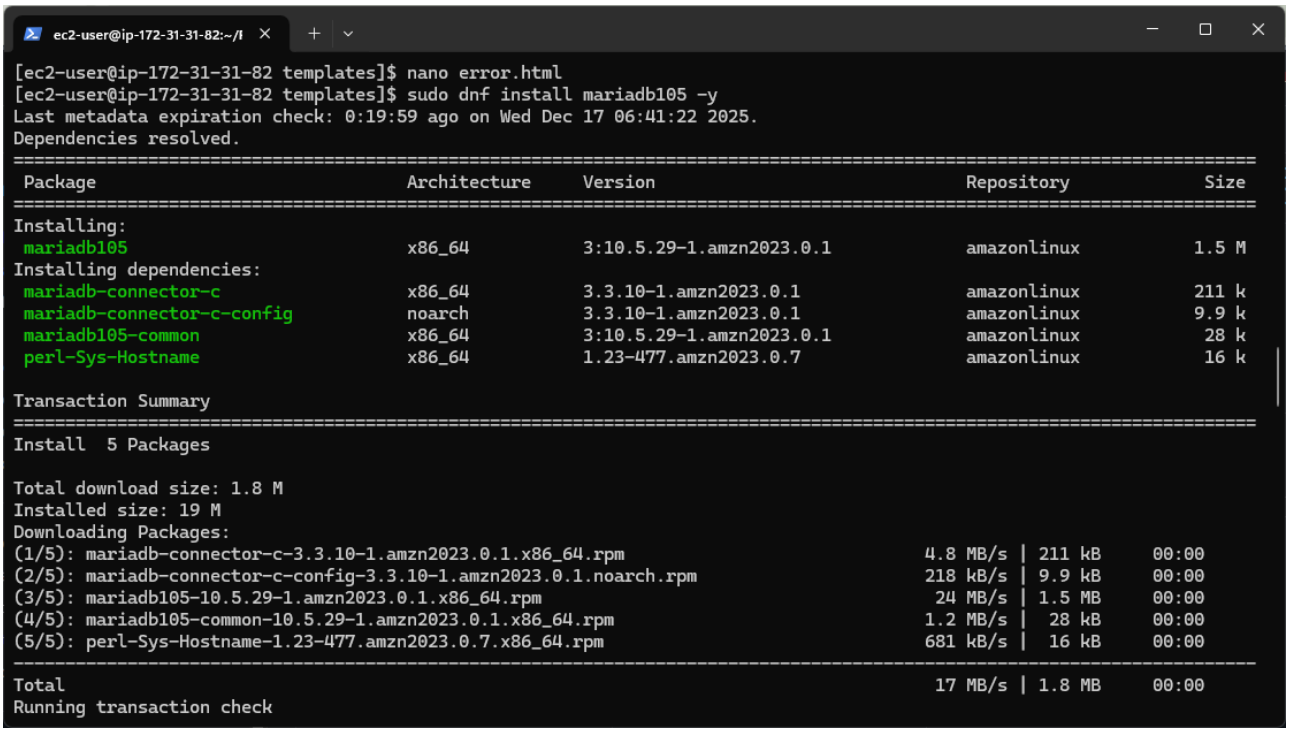
PHASE 6 – Create Database and Table in RDS from EC2

- From the EC2 terminal, install MySQL client (MariaDB client):
  sudo dnf install mariadb105 -y
  # if this fails, try: sudo dnf install mariadb -y
- Connect to the RDS MySQL instance from EC2:
  mysql -h <RDS-endpoint> -u admin -p
  Example: mysql -h flaskdb-instance.xxxxxx.ap-south-1.rds.amazonaws.com -u admin -p
- Inside the MySQL prompt, run following commands:

  CREATE DATABASE flaskdb;

  USE flaskdb;

  CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  password VARCHAR(50) NOT NULL
  );
- Exit MySQL: exit;



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;

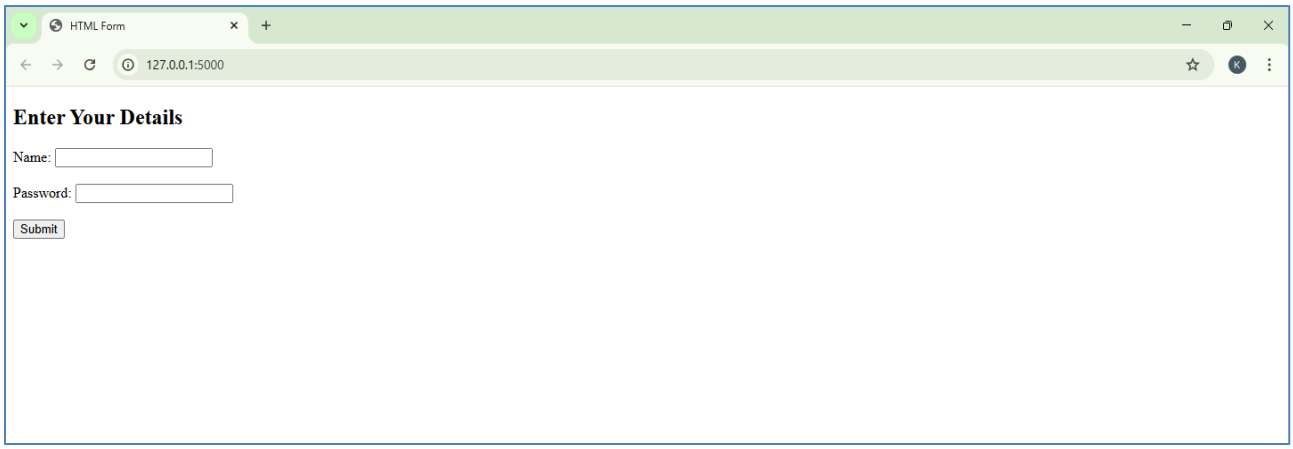
Expected Output

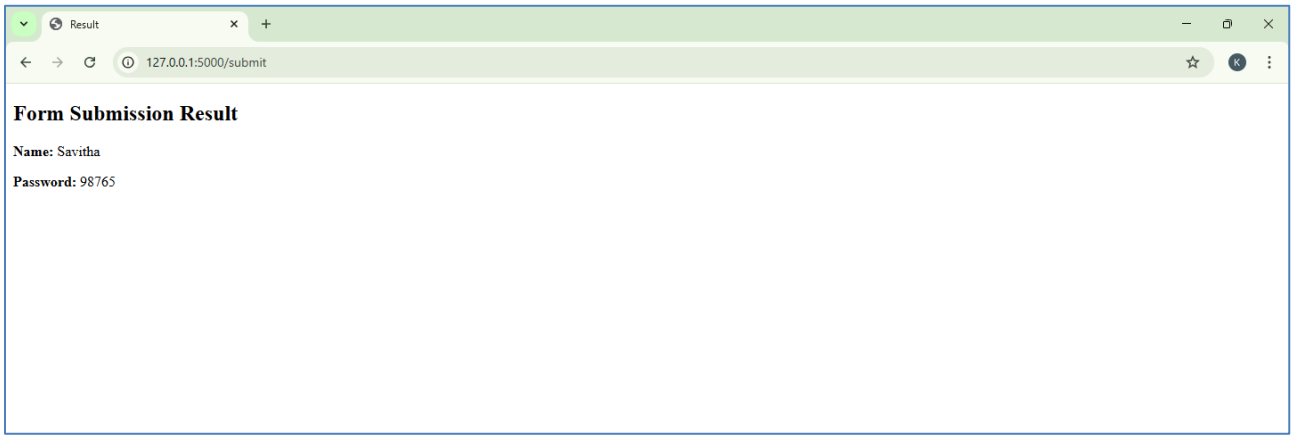
1. Registration Page
    - User enters Name and Password.
    - On submit, data is inserted into users table in RDS MySQL.
2. Login Page
    - User enters same Name and Password.
    - Flask executes SELECT query on users table to check match.
3. Welcome Page
    - For valid credentials:

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

- For invalid credentials:

"Invalid Username or Password" message with link to try again.





Result

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

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

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

Sample Viva Questions

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

   AWS RDS databases are private by default and do not allow any incoming connections unless explicitly permitted by security group rules.
   Since the Flask application runs on EC2, it must be allowed to connect to the RDS MySQL database.
   Therefore, the EC2 security group must be added to the RDS inbound rules to allow:
   - Port: 3306 (MySQL)
   - Source: EC2 Security Group

   This ensures that only the EC2 instance running the application can access the database.
6. What is the purpose of db_config.py?
7. What is **SQL Injection**? How does using parameterized queries (%s) help prevent it?

   SQL Injection:
   A technique where malicious SQL code is entered as user input.
   In this application, it is prevented using parameterized queries (%s).
8. What is the difference between **Registration workflow** and **Login workflow** in this app?
9. Why do we use host='0.0.0.0' in app.run() on EC2?

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

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

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

   This allows:

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

What happens if we don't use 0.0.0.0?

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

In this exercise

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

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

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

Expected Learning Outcomes

After completing this exercise, students will be able to:

- Configure and connect a **Flask application** to an **AWS RDS MySQL** database.
- Store **form data** into a MySQL table using **INSERT** queries.
- Retrieve and validate data using **SELECT** queries from Flask.
- Understand **Registration vs Login** logic.
- Explain how **EC2 acts as the backend application server**.
- Understand **Security Group rules** for EC2 ↔ RDS communication.
- Perform a **basic full-stack deployment** on AWS using Flask, HTML, and MySQL (RDS).

from db_config import db_config

Meaning:

You are importing the dictionary named db_config from the file db_config.py.

This file contains your AWS RDS connection details:

python

Copy code

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

So when you write:

python

Copy code

from db_config import db_config

You are simply bringing this dictionary into app.py.

✔ Why is this needed?

Because your app.py code uses:

python

Copy code

mysql.connector.connect(**db_config)

This needs access to:

host

user

password

database

All of those are stored in the db_config dictionary.

So we import it to use it inside the main Flask application.

✔ Why separate files?

1 Clean code

app.py contains only application logic.

db_config.py contains only database settings.

2 Easy updates

If the RDS endpoint or password changes → only update db_config.py, not the whole app.

3 Security & maintainability

Credentials a

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:

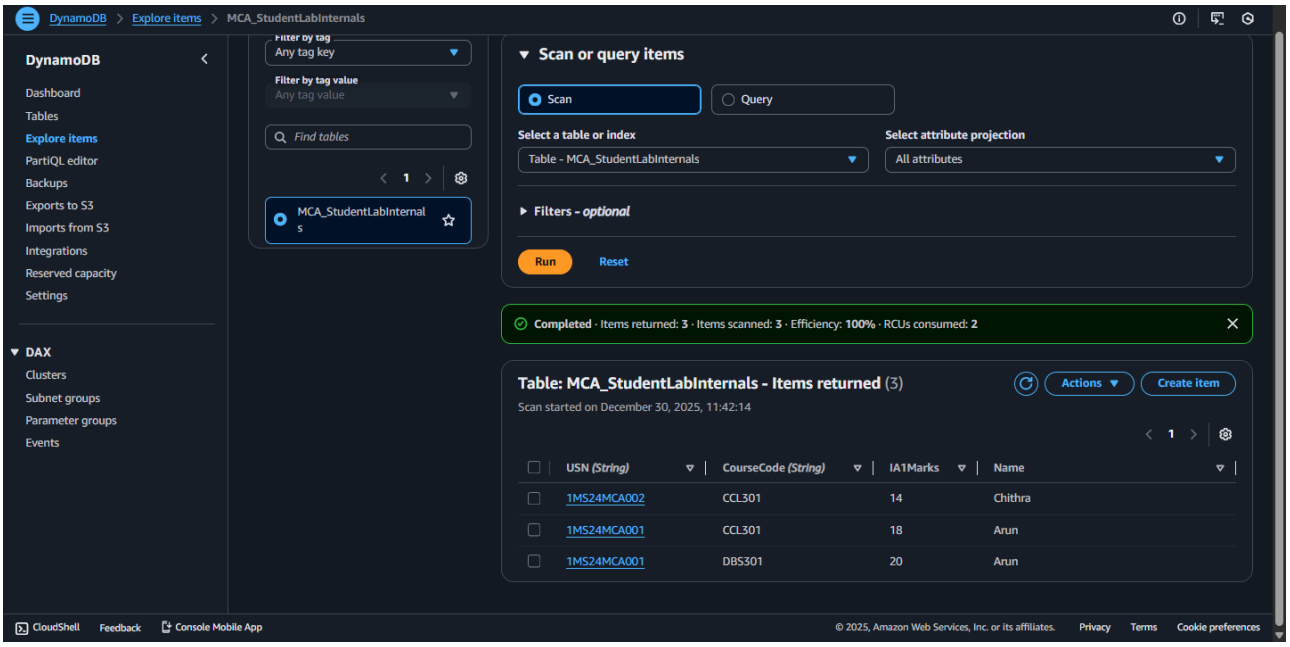
- USN = 1MS24MCA001
- CourseCode = DBS301
- Name = Arun
- Semester = 3
- Attendance = 88
- IA1Marks = 20

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.

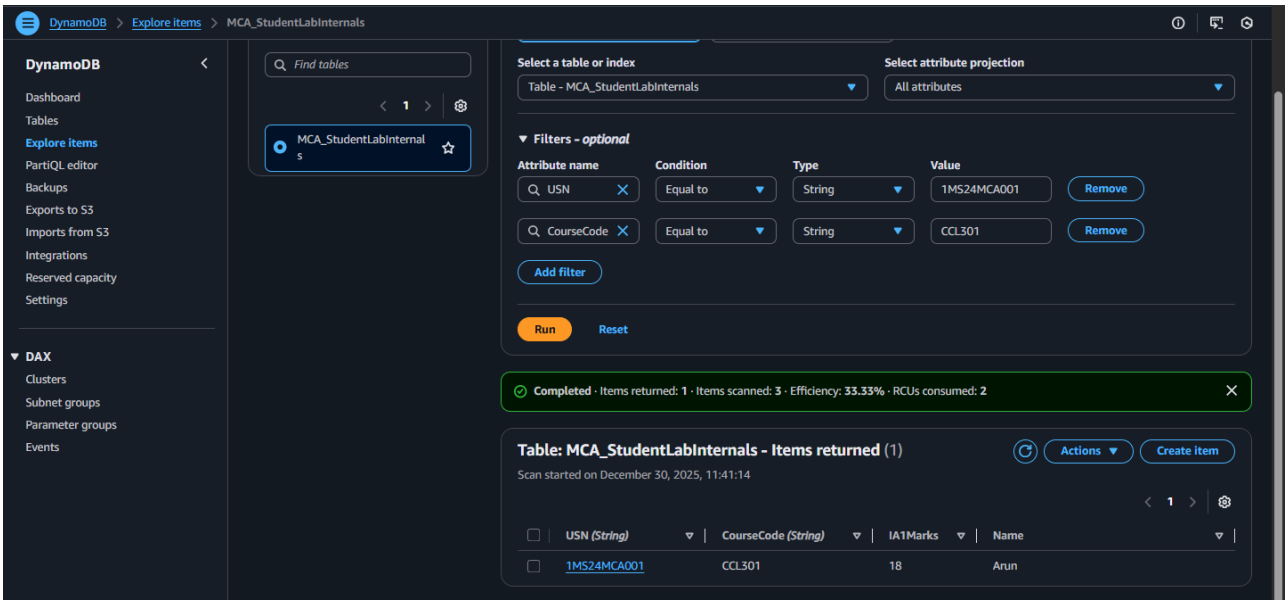


Phase 3: Read Data (Get / Query / Scan)

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

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

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



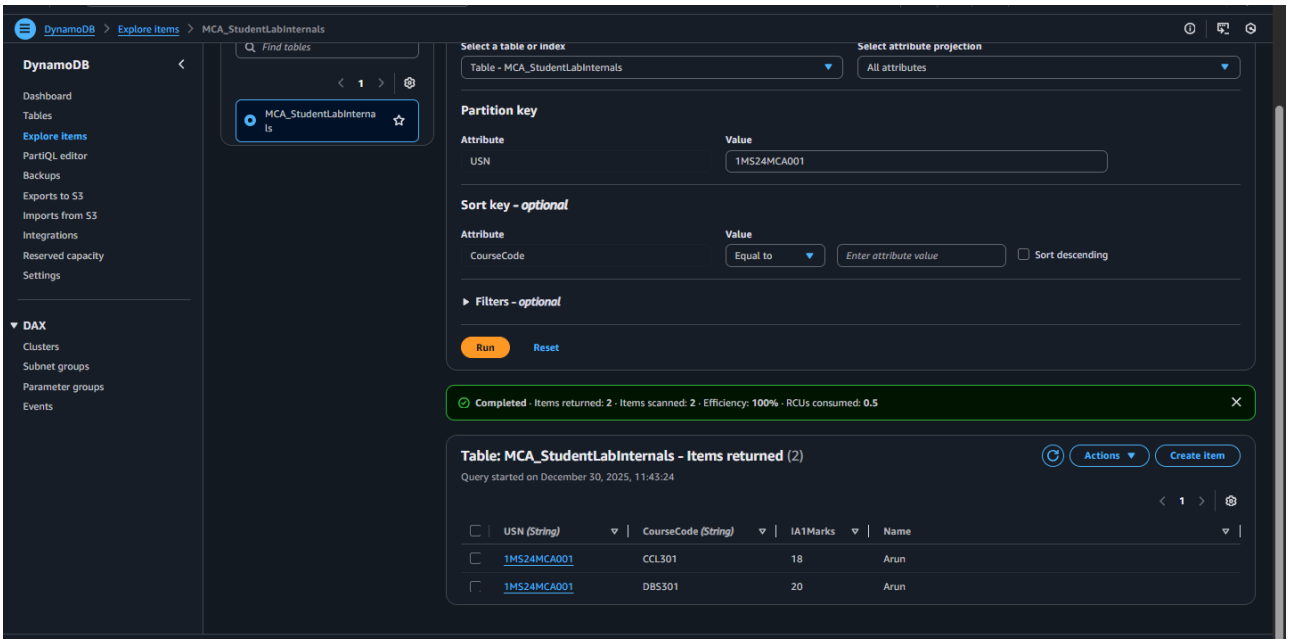
Step 10: Query – fetch all courses for one student

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

Expected output:

- Items for Arun in both CCL301 and DBS301

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



Step 11: Scan (Not for large tables)

- Click Scan
- Run scan without filters

Expected output:

- All items (Arun + Chitra)

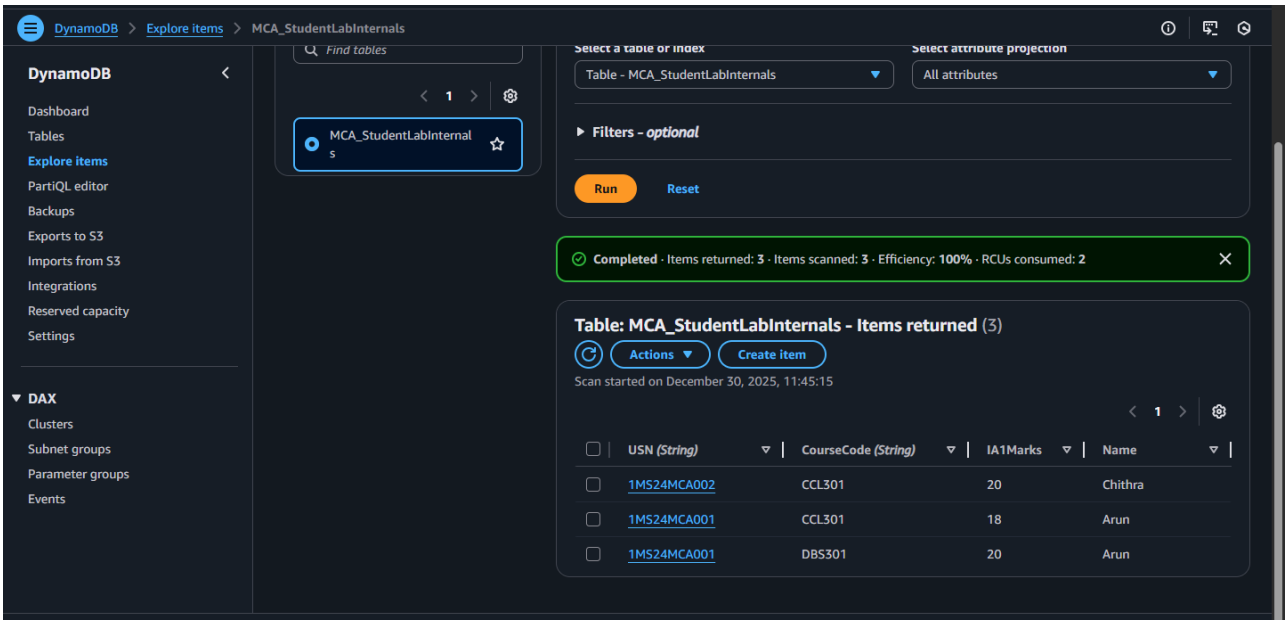
Key concept:
Scan reads the entire table → slow/costly for big tables.

Phase 4: Update Data

Step 12: Update IA1 marks of Chitra for CCL301

- Open item where:
  USN = 1MS24MCA002
  CourseCode = CCL301
- Click Edit
- Change: IA1Marks from 14 to 16
- Click Save changes

Verify updated value appears.

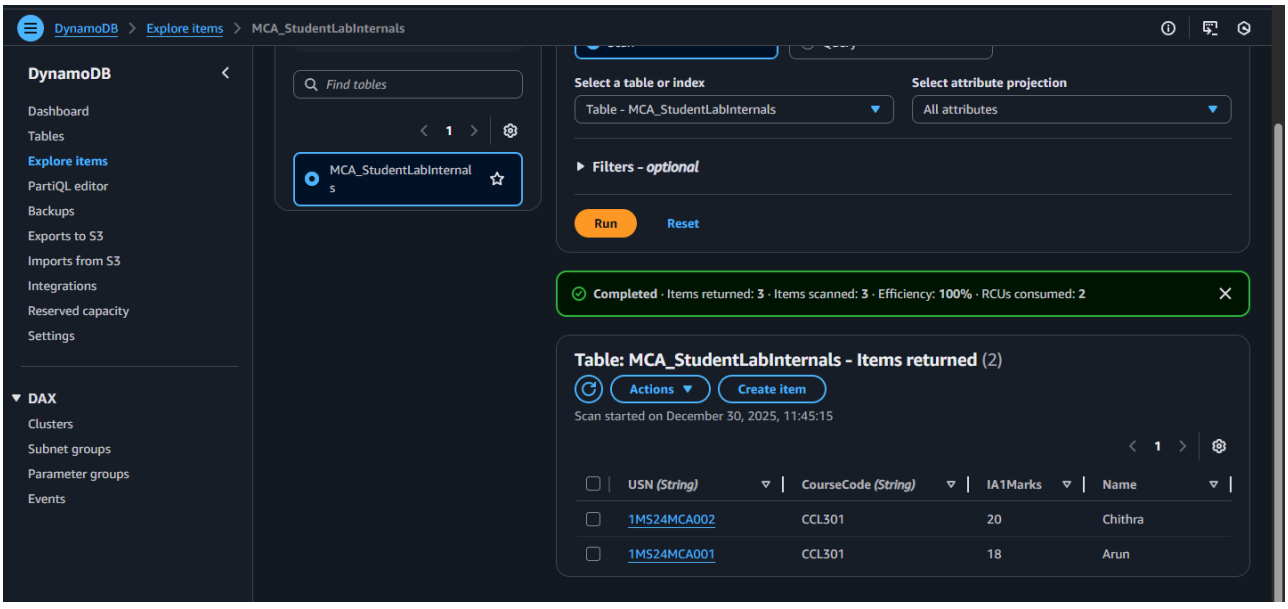


Phase 5: Delete Data

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

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

Now total items should reduce by 1.

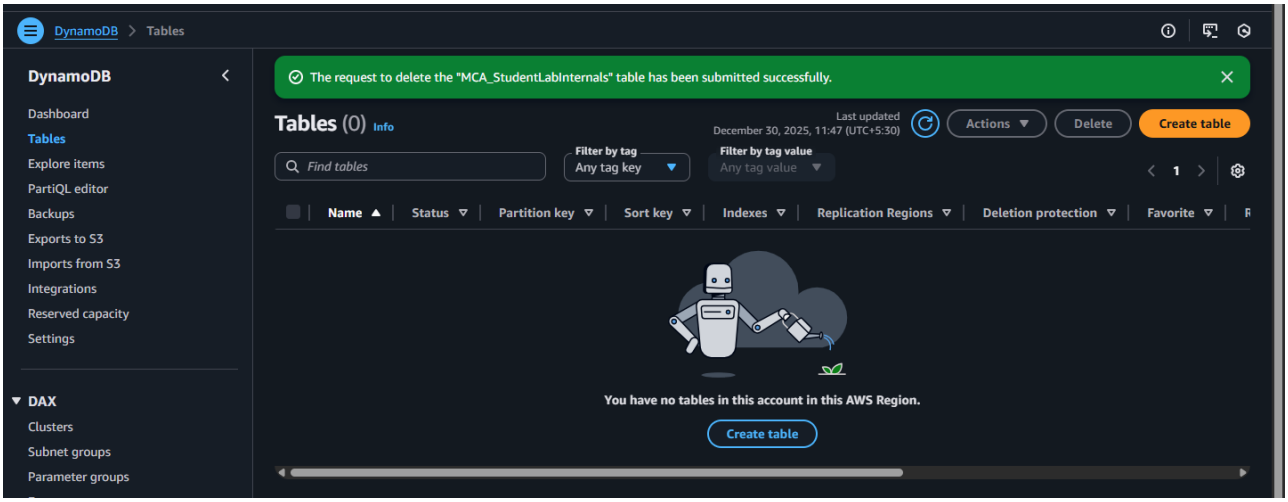


Phase 6: Cleanup (Must Do to avoid charges)

Step 15: Delete the Table

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

Ensure table disappears from list.

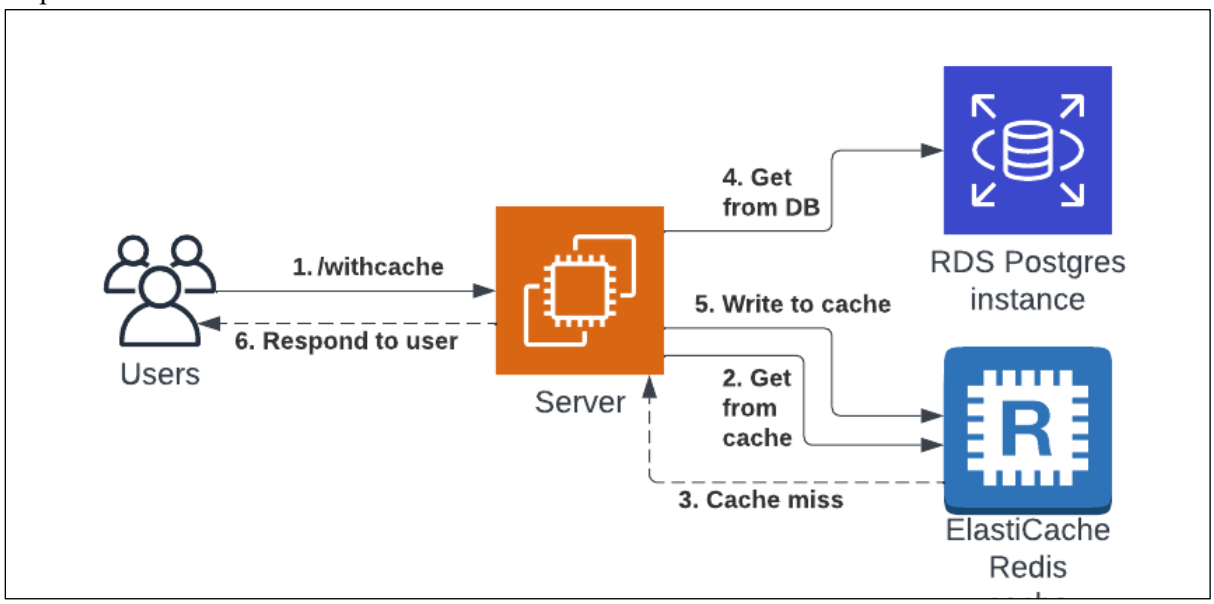


DATE: 03-12-25

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

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

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



This exercise provides **exposure** to an industry-used in-memory data store.
Redis improves performance by serving frequently accessed data from memory and is typically used **in front of databases** in real-world applications.

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

Phase-1 → EC2 creation + valkey-cli installation
Phase-2 → Redis cache creation
Phase-3 → Connect EC2 → Redis engine and run commands

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

Purpose of Phase-1
To create an EC2 instance using **Amazon Linux 2023** that will act as a **client machine** to connect to Amazon ElastiCache (Redis OSS) using **valkey-cli**.

Step 1: Select AWS Region

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

Step 2: Launch EC2 Instance

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

Step 3: Configure Instance Basics

- Name: RedisClient-AL2023
- AMI: Select Amazon Linux 2023 AMI
- Instance Type: Select t3.micro (Free Tier eligible)
- Key Pair: Select an existing key pair OR create a new one (RSA, .pem)
- Network & Security
  - Network
    - VPC: Default VPC
    - Subnet: No preference
    - Auto-assign public IP: Enabled
- Security Group (VERY IMPORTANT)
  Create a new security group:
  - Security Group Name: SG-RedisClient
  - Description: Security group for Redis client EC2
    Inbound Rules

Type	Port	Source
SSH	22	Anywhere 0.0.0.0/0

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

Step 5: Storage

- Keep default

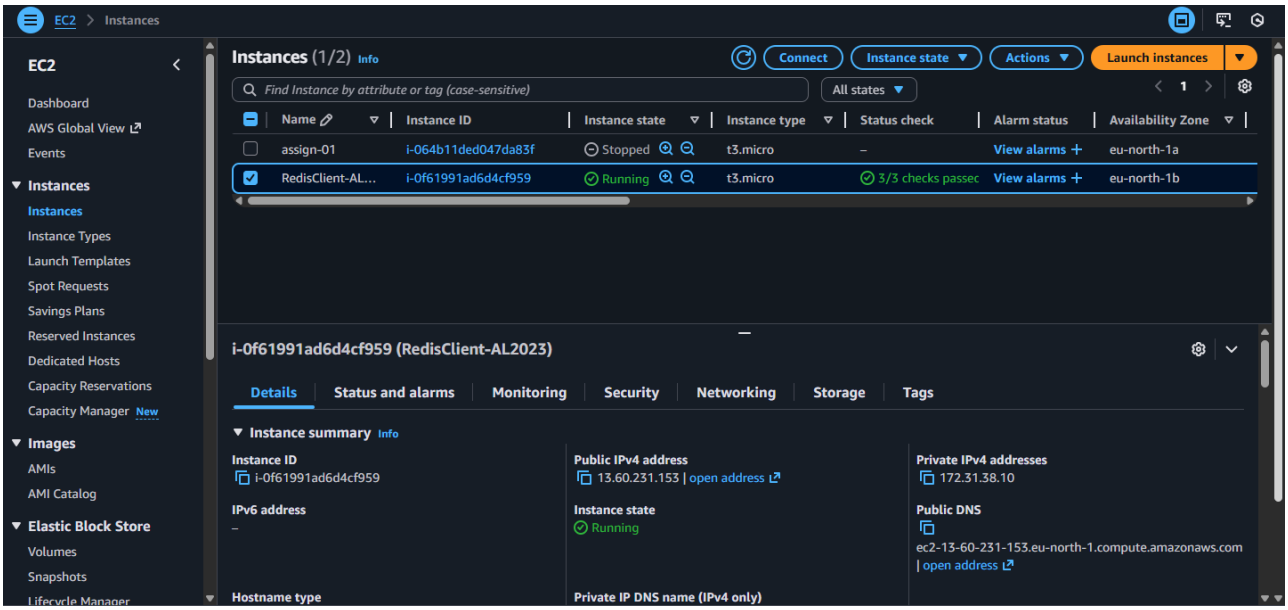
Step 6: Launch Instance

- Click Launch instance
- Wait until Instance State = Running

Step 7: Connect to EC2

1. EC2 → Instances → select RedisClient-AL2023
2. Click Connect
3. Choose EC2 Instance Connect
4. Click Connect

You are now inside the EC2 terminal.



Step 8: Install Valkey Client

Run the following commands:

sudo dnf update -y

sudo dnf install -y valkey

Verify installation:

valkey-cli --version

Expected Output (example)

valkey-cli 8.x.x

This confirms the EC2 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

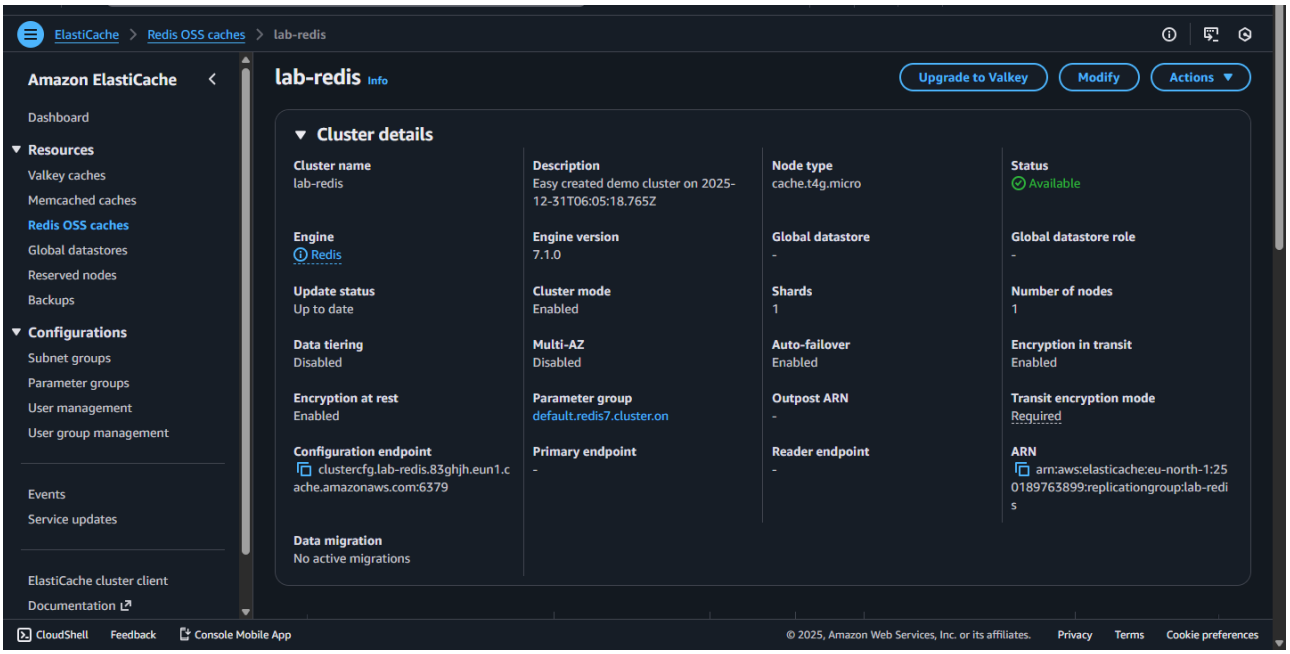
1. Review all settings
2. Click Create
3. Wait until Status = Available

This may take a few minutes.

Step 11: Note the Redis Endpoint

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

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



Note:

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

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

Purpose of Phase-3
To connect the EC2 client (Amazon Linux 2023) to the ElastiCache Redis OSS cluster using valkey-cli and demonstrate basic in-memory cache operations.

Pre-checks - Before connecting, confirm:

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

Step 1: Connect to EC2

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

You are now inside the EC2 terminal.

Step 2: Verify Valkey Client
Run: valkey-cli --version
Expected Output (example)
valkey-cli 8.x.x
This confirms the EC2 is ready to act as a Redis-compatible client.

Step 3: Connect to ElastiCache Redis
Use the Configuration Endpoint from Phase-2.
valkey-cli -h < Configuration_ENDPOINT> -p 6379

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

Expected Result
You should see a prompt like:
lab-redis.xxxxxx.cache.amazonaws.com:6379>
This means the connection is successful.
Step 4: Execute Redis / Valkey Commands
These commands simulate what an application does internally.

Test Connectivity
PING
Expected Output
PONG

Store and Retrieve Data (SET / GET)

SET course "Cloud Computing"
GET course
Expected Output
"Cloud Computing"
Demonstrates **key–value storage in memory**.

Counter Example (INCR)
INCR visits
INCR visits
GET visits
Expected Output
"2"
NOTE: Common real-world use - page views, hit counters.

4.4 Set Data with Expiry (TTL)
SET notice "Results Published" EX 60
TTL notice
GET notice
Expected Output

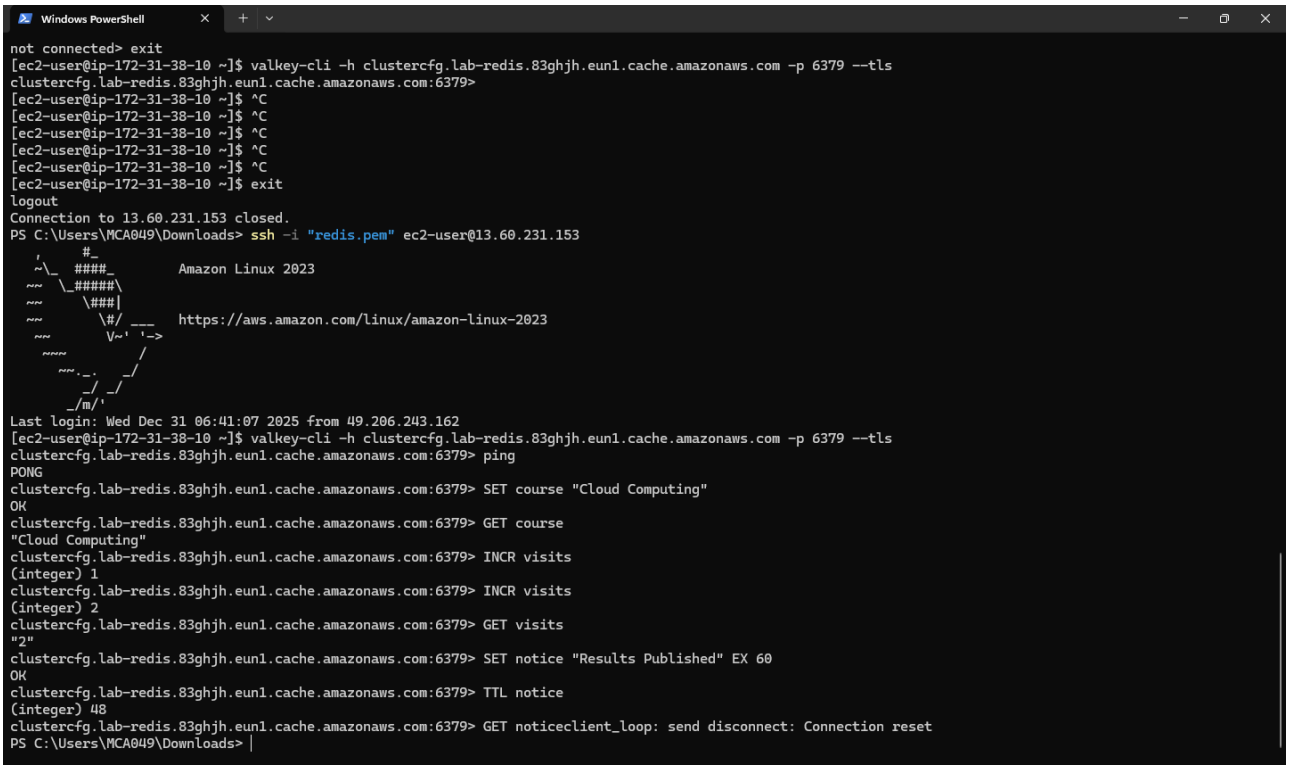
- TTL shows a value **≤ 60**
- GET returns:

"Results Published"
Shows **temporary cache data**.

4.5 Verify Automatic Expiry
Wait ~60 seconds, then run:
GET notice
Expected Output
(nil)
Confirms data is **automatically removed from memory**.

4.6 Delete Data Manually
DEL course
GET course
Expected Output
(nil)

Step 5: Exit Redis Client
EXIT



Note:

"The application first checks Redis.
If data is present, it is returned immediately from memory.
If not present, the application fetches data from the database and stores it in Redis with a TTL.
Redis automatically removes the data after expiry."

Phase-3 Outcome

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

Common Errors & Quick Fix

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

Clean-Up (Mandatory)

Step 1: Delete Redis Cluster

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

Step 2: Terminate EC2 Instance

- EC2 → Instances → Terminate RedisClient-EC2

Step 3: Optional

- Delete unused security groups

Redis Commands Explanation

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

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”:

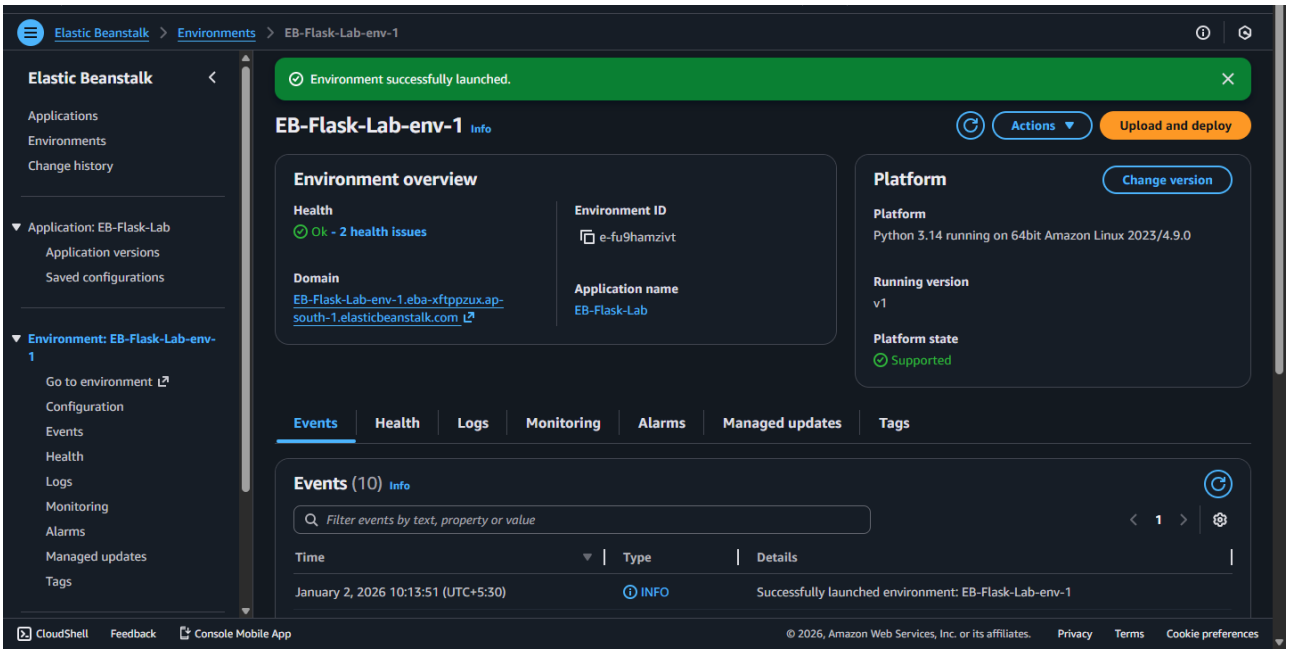
- Click Create role
- Use case: EC2
- Attach policy: AWSElasticBeanstalkWebTier
- Role name: aws-elasticbeanstalk-ec2-role

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



Phase F — Test the Application

Step F1: Open the domain URL

Click the Domain link shown in EB.

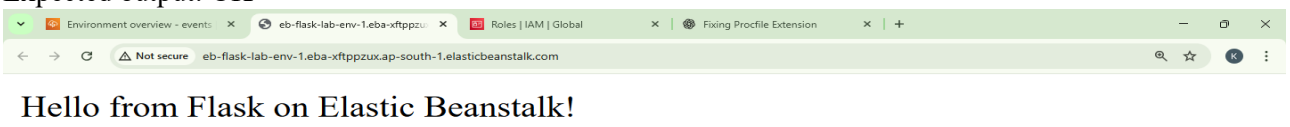
Expected output: Hello from Flask on Elastic Beanstalk!

Step F2: Test /health

In the browser address bar, append /health

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

Expected output: OK



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.

DATE: 10-12-25

Exercise–21: Deploy a Flask App on AWS Elastic Beanstalk and Perform CRUD on DynamoDB (Using AWS SDK/boto3)

Deploy a simple Python Flask web app on AWS Elastic Beanstalk (EB) that creates, reads, updates, and deletes items in DynamoDB using AWS APIs via boto3.

A) LOCALLY — Prepare Folder + Code (Before AWS)

Create folder

Create a folder named: flask-ddb-lab

Inside it, create 4 files

Your folder must look like:

flask-ddb-lab/
- application.py
- requirements.txt
- Procfile
- runtime.txt

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")
```

requirements.txt

```
Flask==3.0.0
boto3==1.34.0
gunicorn==21.2.0
```

Procfile

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

runtime.txt

```
python-3.11
```

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

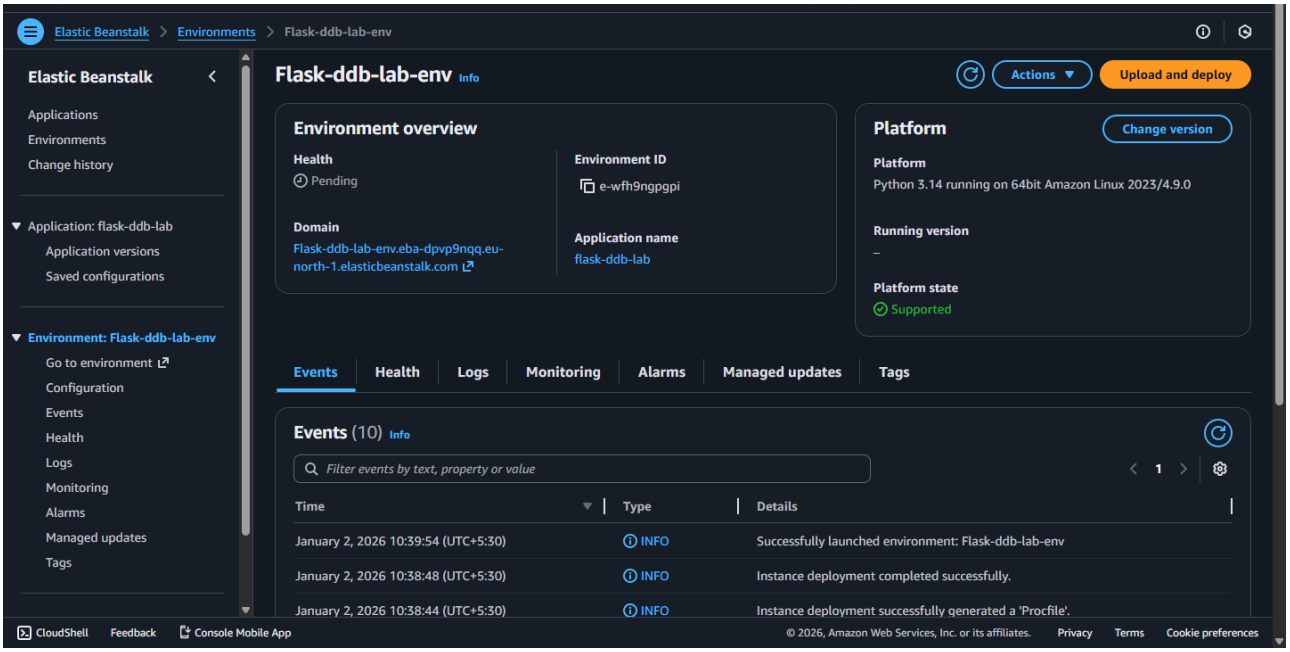
- Platform: **Python**
- Application code: **Sample application**
- Presets: **Single instance (free tier eligible)**

Configure service access

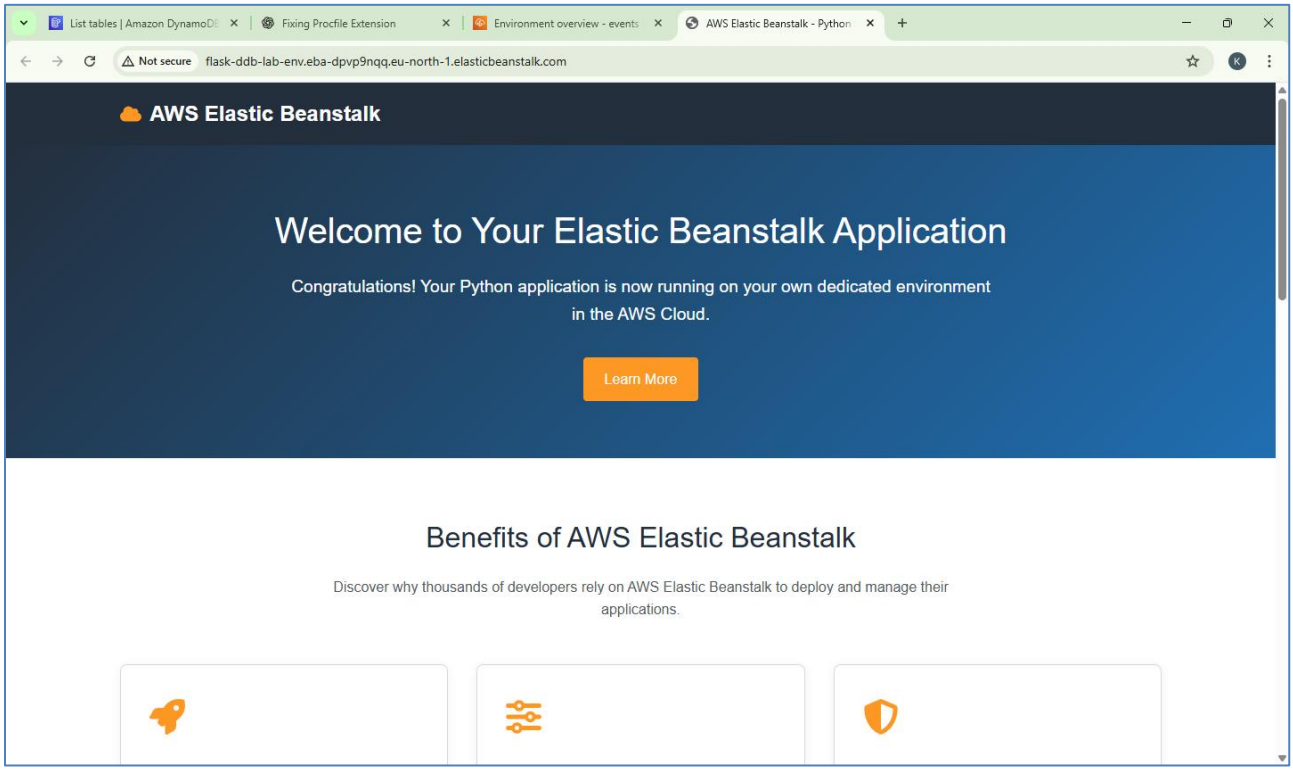
On **Configure service access** page:

- 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.



Click on Domain: Flask-ddb-lab-env.eba-jtqiug36.ap-south-1.elasticbeanstalk.com



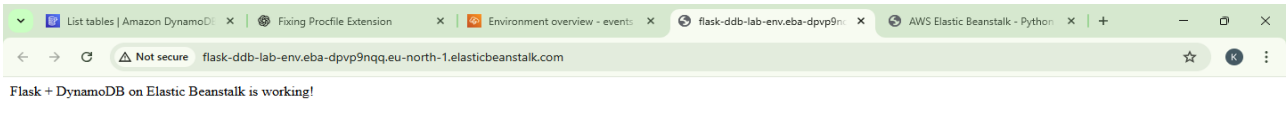
E) AWS — Deploy Your ZIP

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)

Go to: AWS Console → Elastic Beanstalk → Environments → flask-ddb-lab-env

Click: Configuration

Under **Software**, click: Edit

Scroll to **Environment properties**

Add these two entries:

Name	Value
TABLE_NAME	Student
AWS_REGION	ap-south-1

Click: Apply

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)

Open **AWS CloudShell** (or Command Prompt / PowerShell)

Run: curl http://<your-eb-domain>/health

Expected output: {"status":"ok"}

I) AWS CloudShell - Steps

In AWS Console (top right)

Search **CloudShell** in the search bar

Wait until terminal opens (you'll see $ prompt)

Paste the following command (replace domain):

1) CREATE Item in DynamoDB (POST)

Run:

curl -X POST http:// Flask-ddb-lab-env-1.eba-k9hmvizi.ap-south-1.elasticbeanstalk.com /Student \

-H "Content-Type: application/json" \

-d '{"StudentID":"101","Name":"Anita","Dept":"MCA"}'

Expected output:

{"message":"Student created"}

2) READ Item from DynamoDB (GET)

Run:

curl http://<your-eb-domain>/student/101

Expected output (JSON with student data).

3) UPDATE Item (PUT)

Run:

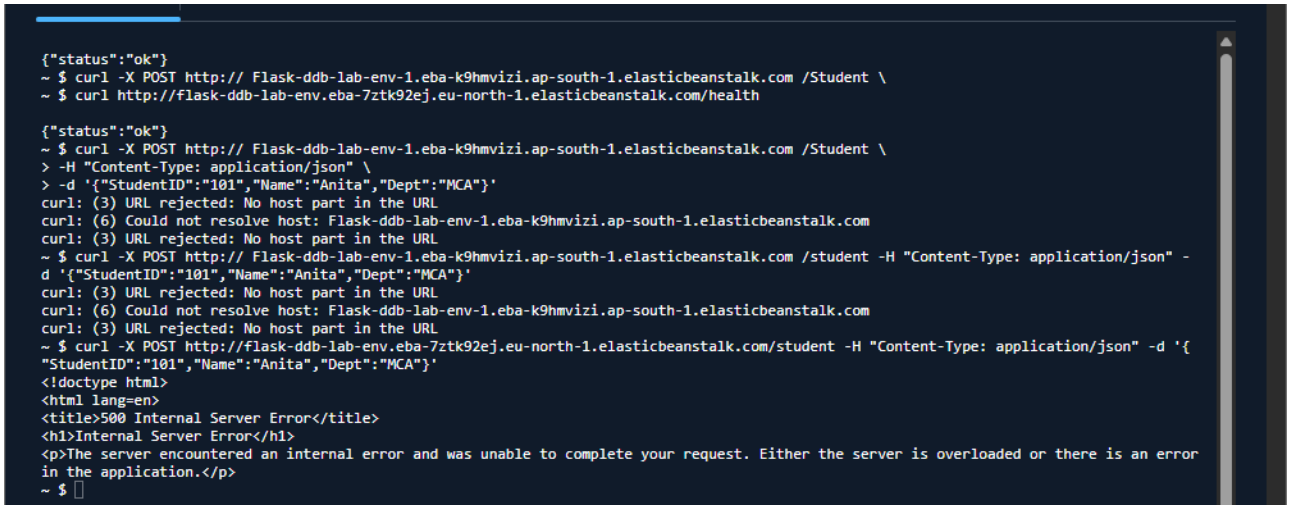
curl -X PUT http://<your-eb-domain>/student/101 \

-H "Content-Type: application/json" \

-d '{"Name":"Anita S","Dept":"MCA-III"}'

Expected output:

{"message":"Student updated"}



4) DELETE Item (DELETE)

Run:

curl -X DELETE http://<your-eb-domain>/student/101

Expected output:

{"message":"Student deleted"}

5) Verify in DynamoDB Console

Go to:

AWS Console → DynamoDB → Tables → Students → Explore table items

Confirm record creation / update / deletion.

DATE: 12-12-25

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.

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

Click Create key pair
A file will download like: pemkeypair.pem
Note: CloudFormation uses key pair NAME (example: pemkeypair), not the file name extension.

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

Create YAML file on your computer

- Open Notepad
- Paste the full YAML template given below
- Save as: ec2-apache-al2023.yaml
    1. Save type: All files
    2. 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 later on.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Launch EC2 (Amazon Linux 2023) and install Apache

Parameters:
```

```
  KeyName:
    Type: AWS::EC2::KeyPair::KeyName
    Description: Select an existing EC2 Key Pair

  AmazonLinux2023AMI:
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/al2023-ami-kernel-default-x86_64

Resources:
  WebServerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow SSH and HTTP
      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
      ImageId: !Ref AmazonLinux2023AMI
      SecurityGroupIds:
        - !Ref WebServerSG
      UserData:
        Fn::Base64: |
          #!/bin/bash
          dnf update -y
          dnf install -y httpd
          systemctl enable httpd
          systemctl start httpd
          echo "<h1>Apache Installed via CloudFormation (Amazon Linux 2023)</h1>" >
/var/www/html/index.html

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

Part C — Create CloudFormation Stack (Console Steps)

Step C1: Open CloudFormation

- AWS Console → Search CloudFormation
- Click Stacks
- Click Create stack → With new resources (standard)

Step C2: Prepare template (select correct options)

- Under Prepare template:
  Select Choose an existing template
- Under Template source:
  Select Upload a template file
- Click Choose file → select ec2-apache-al2023.yaml
- 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)

- Leave everything as default
- Click **Next**

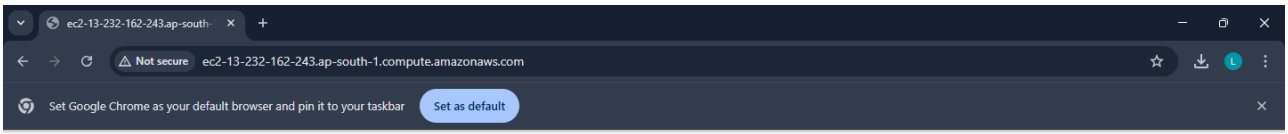
Part F — Review and Create

- Scroll down
- Click **Create stack**

Part G — Monitor Stack Creation

- Wait for Stack status to become: CREATE_COMPLETE
- Open the stack → click Outputs tab
- Copy WebsiteURL
- Paste URL in browser

- Expected Output in browser:
  Apache Installed via CloudFormation UserData (Amazon Linux 2023)

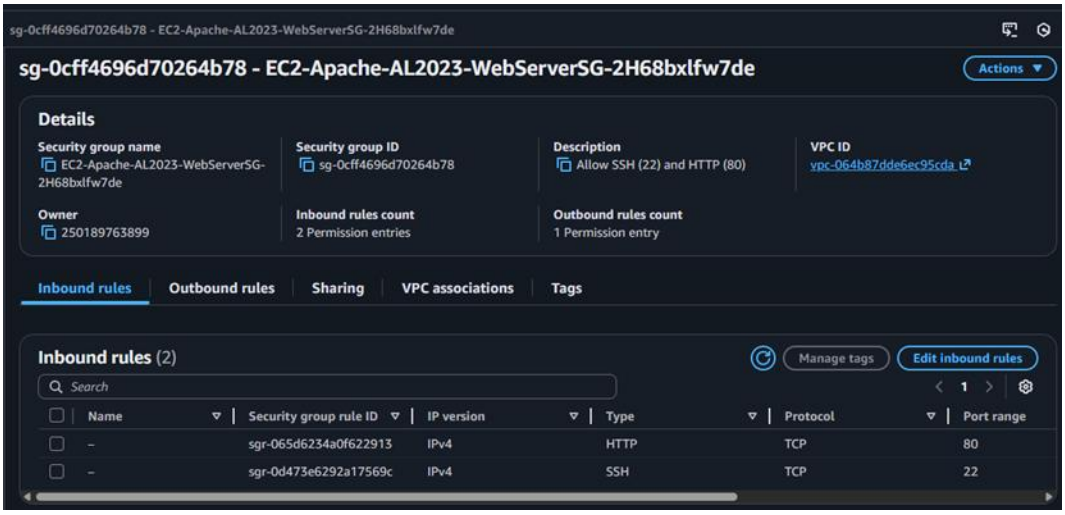


Apache Installed via CloudFormation UserData (Amazon Linux 2023)!

Part H — Verify in EC2

Go to **EC2 → Instances**
Instance should be running
Security group should allow:

- SSH 22
- HTTP 80



Troubleshooting (Common Errors)

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)

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

Clean-up

To avoid charges:

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

This deletes EC2 + Security Group automatically.

DATE: 17-12-25

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

Create an **S3 bucket** using CloudFormation.

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

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

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

Pre-requisites

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

One-time: Create Key Pair

EC2 → Key Pairs → Create key pair

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

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

Step 1.1: Create S3 template file

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

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Create an S3 bucket to store website content (index.html)
Resources:
  WebsiteBucket:
    Type: AWS::S3::Bucket

Outputs:
  BucketName:
    Description: S3 Bucket Name (use this to upload index.html)
    Value: !Ref WebsiteBucket
```

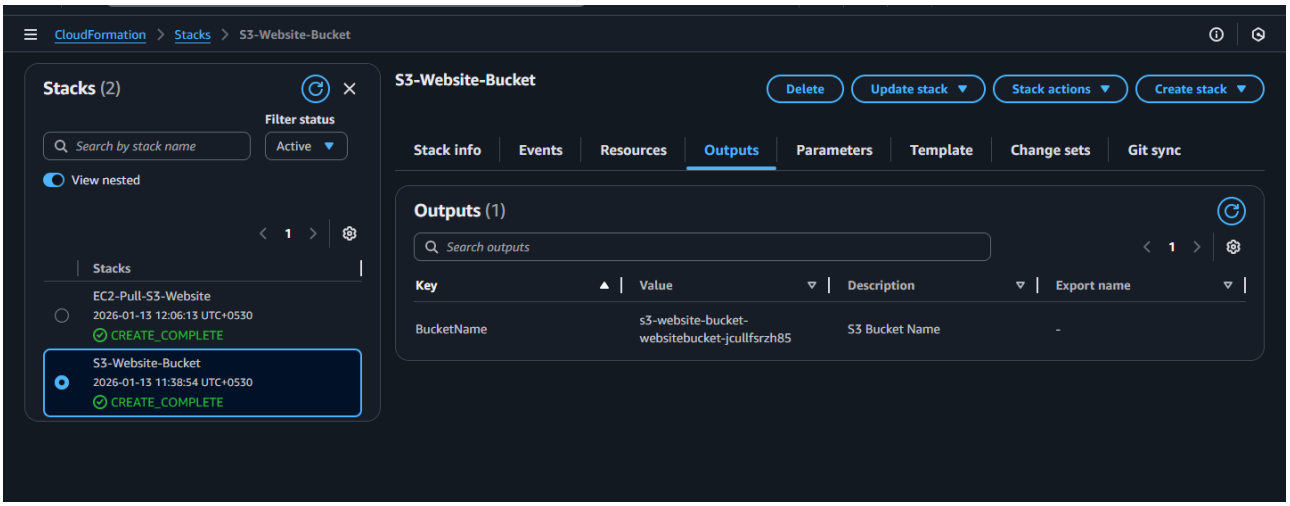
Step 1.2: Create stack

CloudFormation → Stacks → Create stack → With new resources

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

Step 1.3: Copy bucket name

Open stack → **Outputs** → copy **BucketName**



PART 2 — Upload Static Content to S3 (Manual)

Step 2.1: Create index.html on your PC

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

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

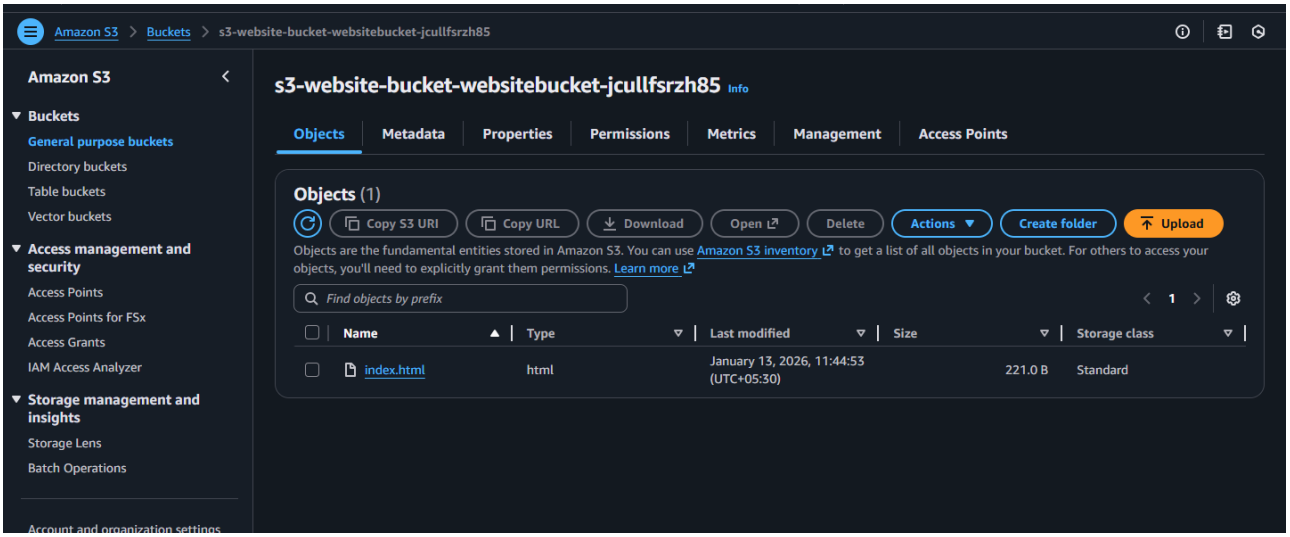
</html>

Step 2.2: Upload to S3 bucket

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

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

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



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

This stack will:

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

Step 3.1: Create EC2 template file

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

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

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

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

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

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

Resources:
  WebServerRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: S3ReadOnlyForWebsiteBucket
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - s3:GetObject
                  - s3:ListBucket
                Resource:
                  - !Sub "arn:aws:s3:::${BucketName}"
                  - !Sub "arn:aws:s3:::${BucketName}/*"

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

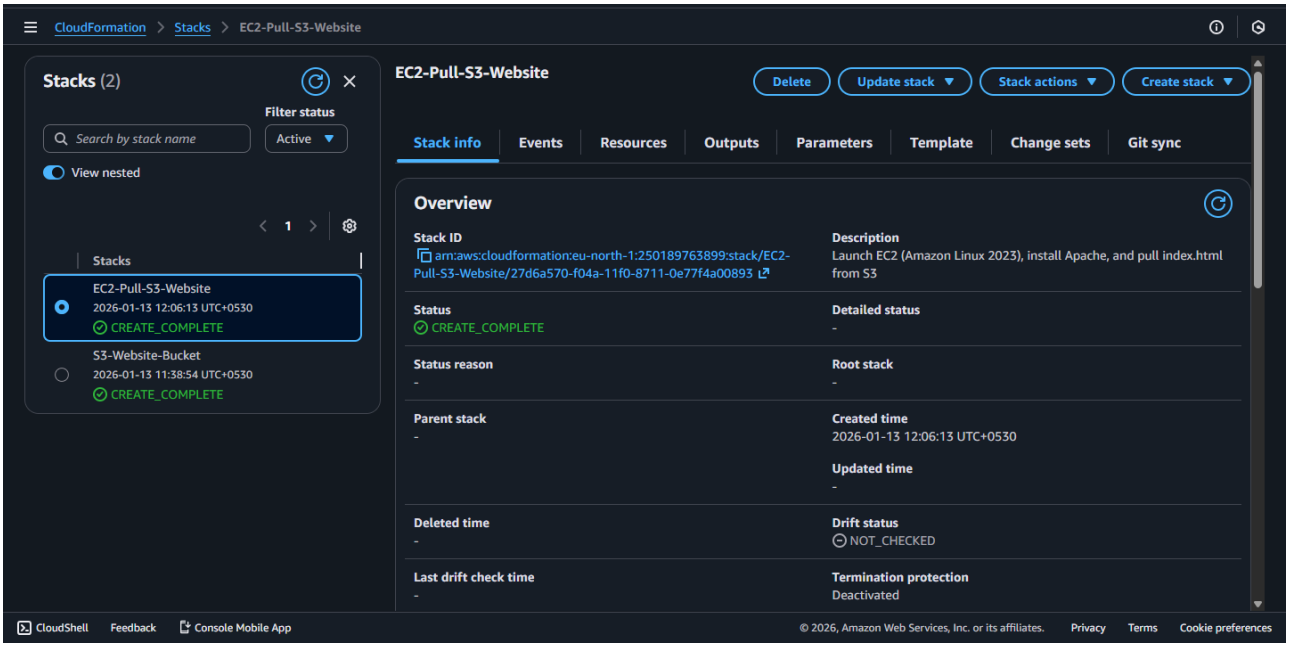
```
  WebServerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow SSH (22) and HTTP (80)
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
  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

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

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

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

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

          systemctl restart httpd

Outputs:
  WebsiteURL:
    Description: Open this URL in browser
    Value: !Sub "http://${WebServerInstance.PublicDnsName}"
```

Step 3.2: Create stack

CloudFormation → Stacks → Create stack

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

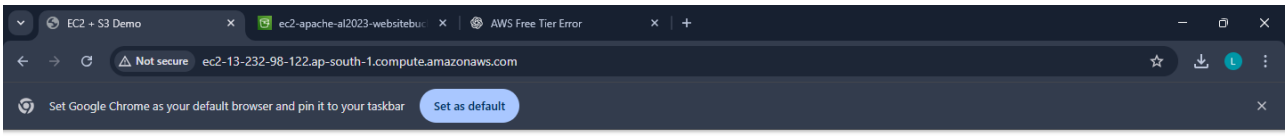
Create stack → wait for **CREATE_COMPLETE**



PART 4 — Verify Output

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

Expected page:
"Hello! This page was pulled from S3 to EC2 automatically…"



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

Deployed using CloudFormation + UserData

Troubleshooting (comm

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 questions

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

DATE: 19-12-25

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

Create one Lambda function that:

- prints a welcome message
- reads JSON input event
- performs business logic (grade calculation)
- writes logs → view them in CloudWatch Logs

Create the Lambda Function

Step 1: Open Lambda

AWS Console → Search Lambda → Open AWS Lambda

Step 2: Create function

Click Create function

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

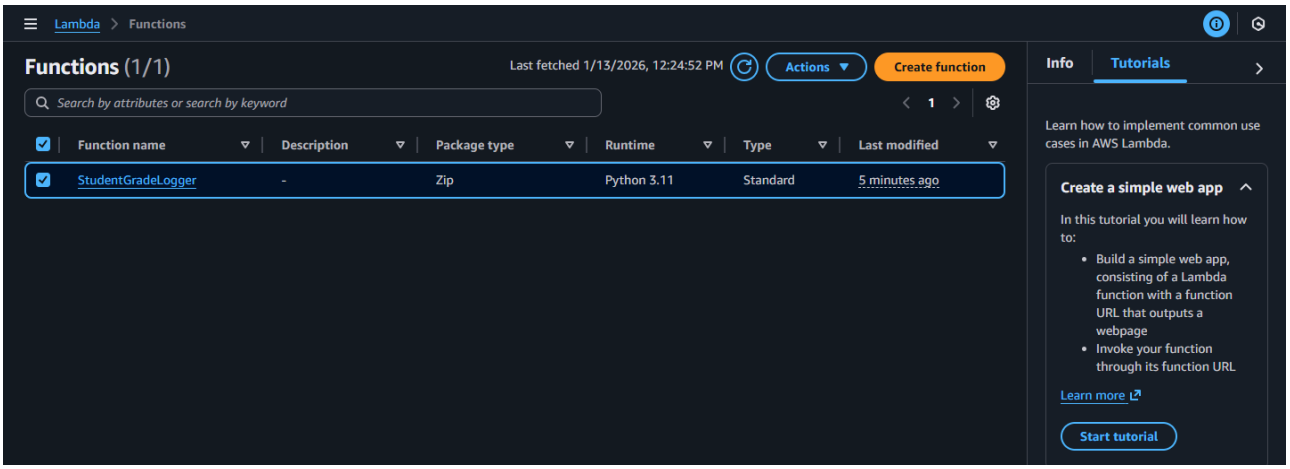
Step 3: Permissions (Execution Role)

Under "Permissions"

- Choose: Create a new role with basic Lambda permissions

This automatically gives permission to write logs to CloudWatch.

Click **Create function**



Add Code (Business Logic + Logs)

Step 4: Paste this code

In **Code** tab → lambda_function.py → paste and **Deploy**

```
import json
```

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

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

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

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

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

Click **Deploy**.

Test with Input Events (JSON)

Step 5: Create a test event (Valid case)

Go to Test (top right) → Configure test event

- Event name: ValidInput
- Paste this JSON:

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

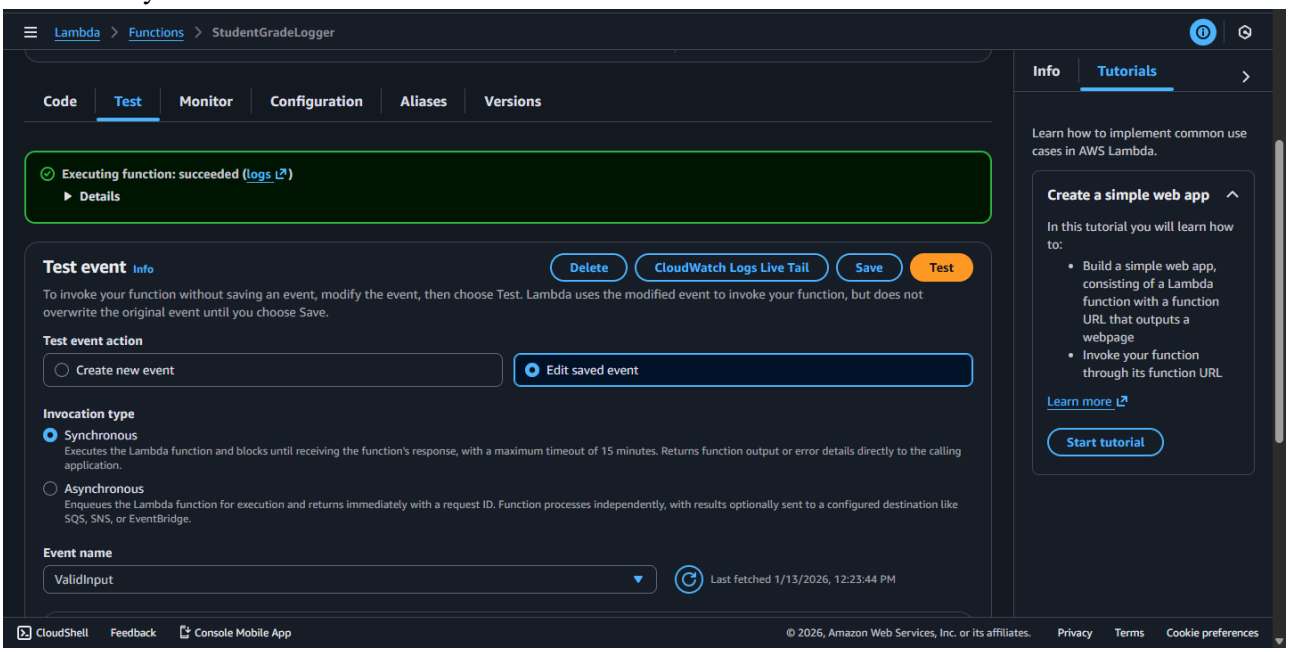
Click **Save**

Step 6: Run test

Click **Test**

Expected response (sample):

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



Step 7: Test invalid input (Missing Marks)

Create another test event:

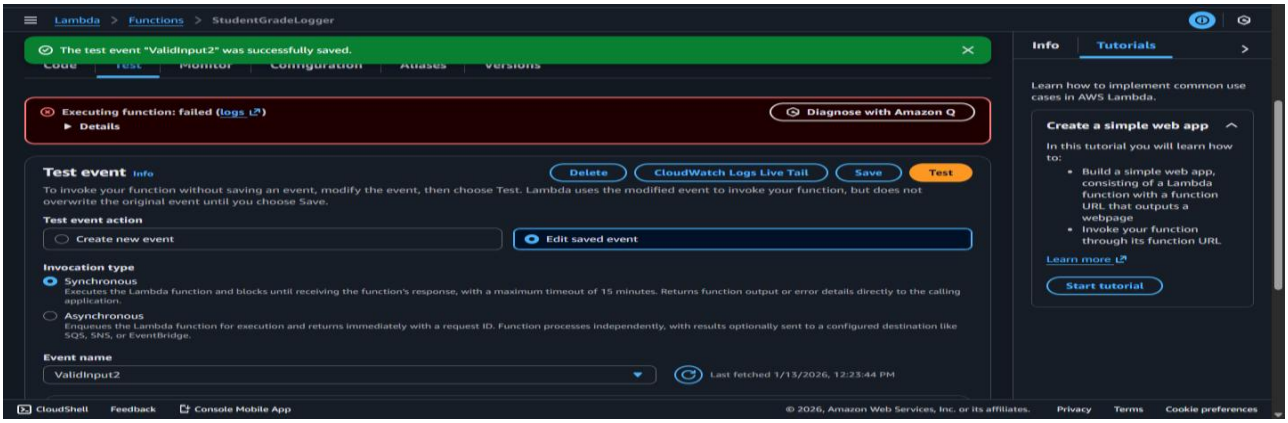
- Event name: MissingMarks

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

Run **Test**

Expected:

- statusCode: 400



- message: Missing 'Marks'

Step 8: Test invalid marks range

Event name: InvalidMarks

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

Expected:

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

View Logs in CloudWatch

Step 9: Open logs from Lambda directly

On the Lambda function page:

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

You will see:

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

Cleanup (Avoid charges, keep account clean)

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

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

DATE: 24-12-25

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

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

Create SNS Topic + Email Subscription

Step 1: Open SNS

AWS Console → Search SNS → Open Simple Notification Service

Step 2: Create a Topic

SNS → Topics → Create topic

- Type: Standard
- Name: NotificationTopic

  Click Create topic

Step 3: Copy Topic ARN

Open the created topic → copy Topic ARN

(You will paste it into Lambda code later)

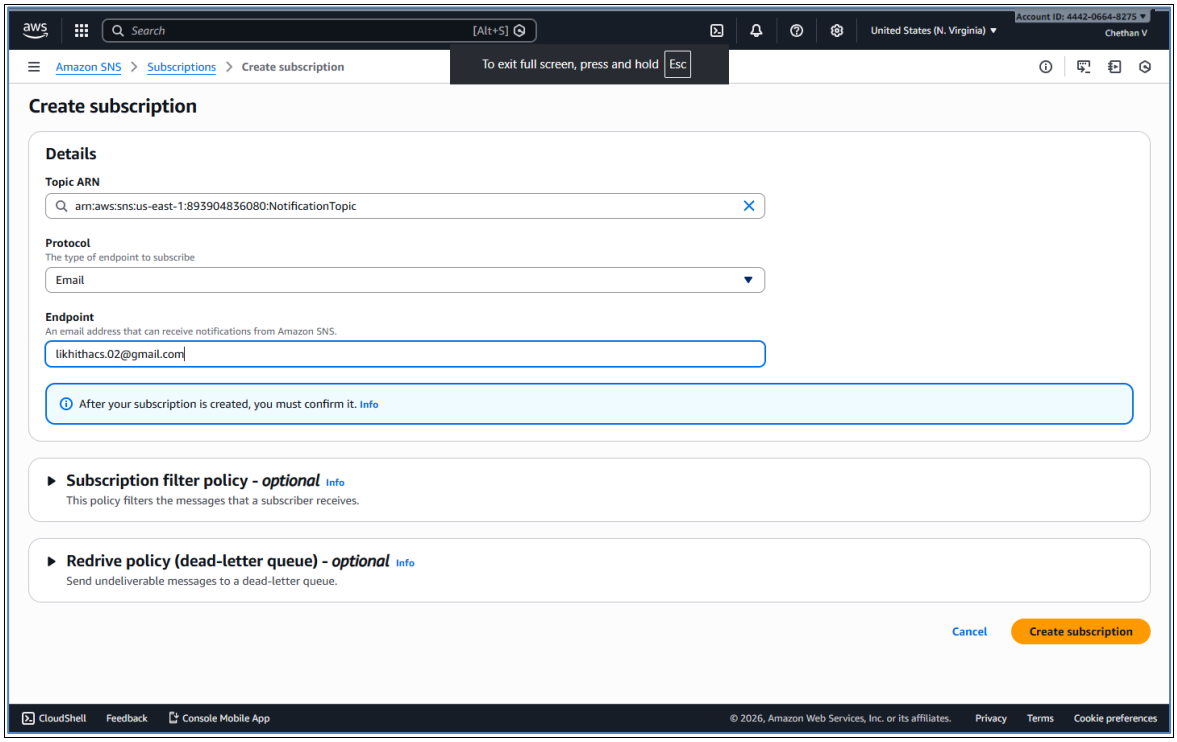
Step 4: Create an Email Subscription

Inside the same topic:

Click Create subscription

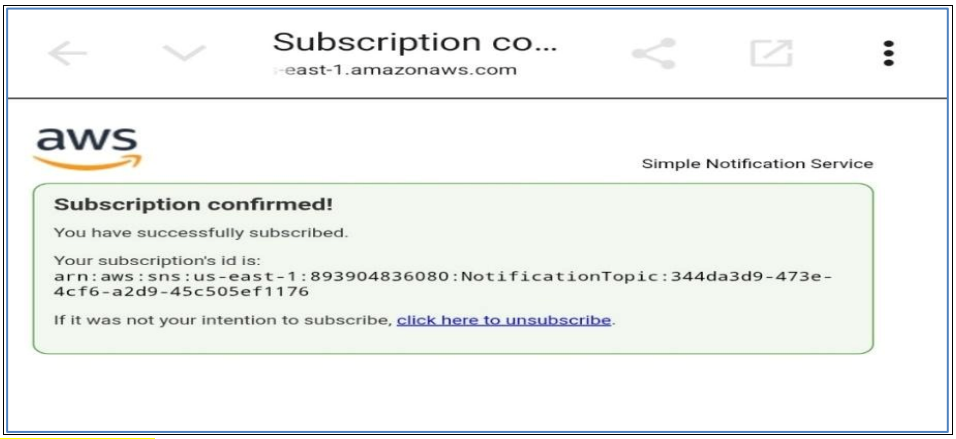
- Protocol: Email
- Endpoint: (your email id)

  Click Create subscription



Step 5: Confirm Subscription

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



Create Lambda Function

Step 6: Open Lambda

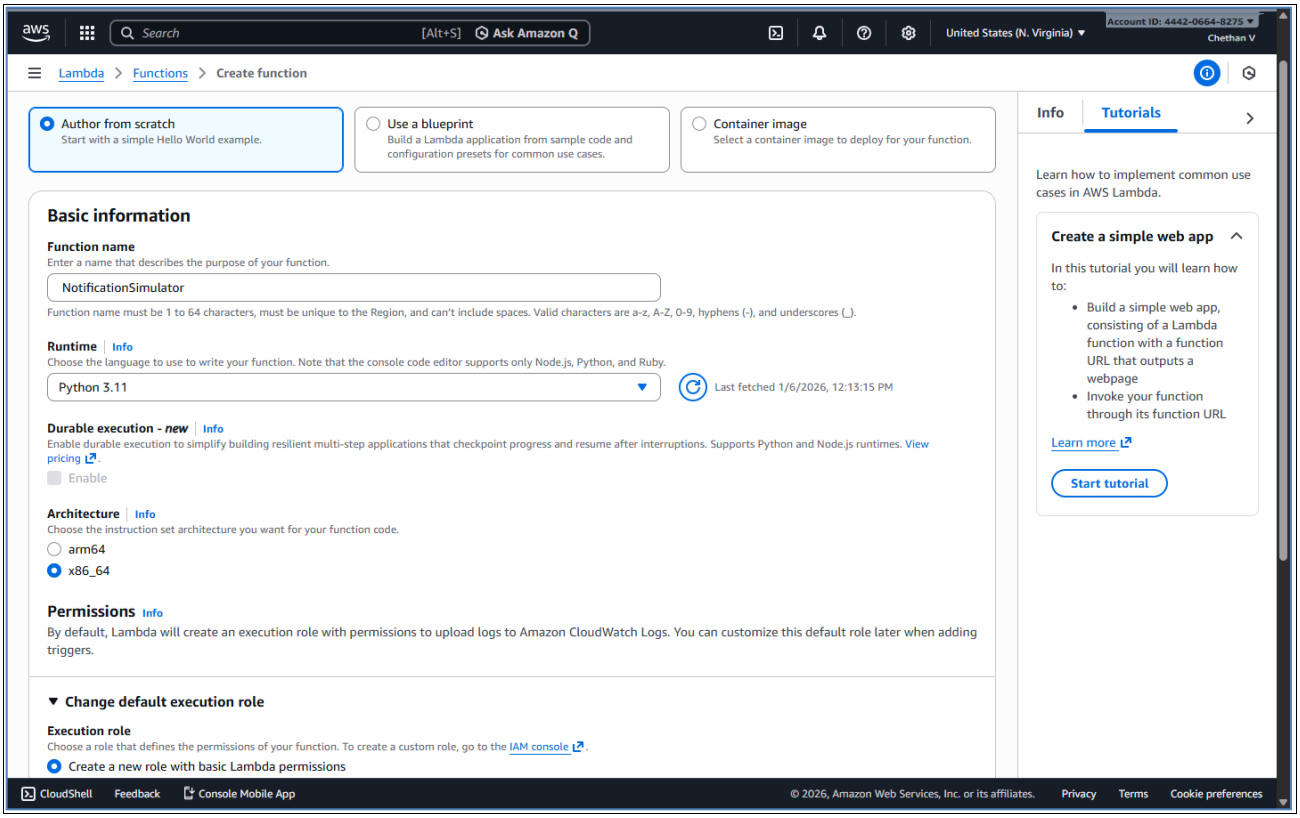
AWS Console → Search Lambda → Open AWS Lambda

Step 7: Create function

Click Create function

- Select: Author from scratch
- Function name: NotificationSimulator
- Runtime: Python 3.11

- Permissions: Create a new role with basic Lambda permissions
  Click Create function



Add Lambda Code (with SNS Publish)

Step 8: Paste code in lambda_function.py

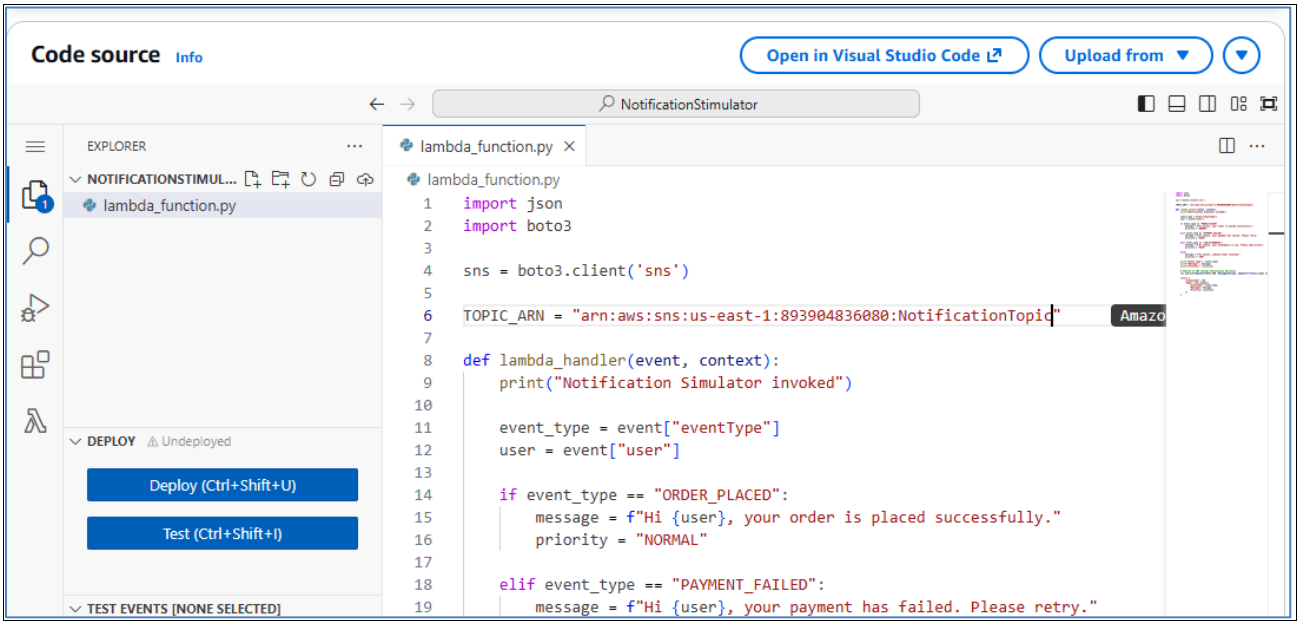
Lambda → **Code** tab → open lambda_function.py → remove existing code → paste:

Replace <SNS_TOPIC_ARN> with your copied Topic ARN.

```
import json
import boto3

sns = boto3.client('sns')

TOPIC_ARN = "<SNS_TOPIC_ARN>"

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

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

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

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

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

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

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

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

    return {
        "statusCode": 200,
        "body": json.dumps({
            "EventType": event_type,
            "Message": message,
            "Priority": priority
        })
    }
```

Click **Deploy**
Wait for “Successfully updated function…”



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"
}
```

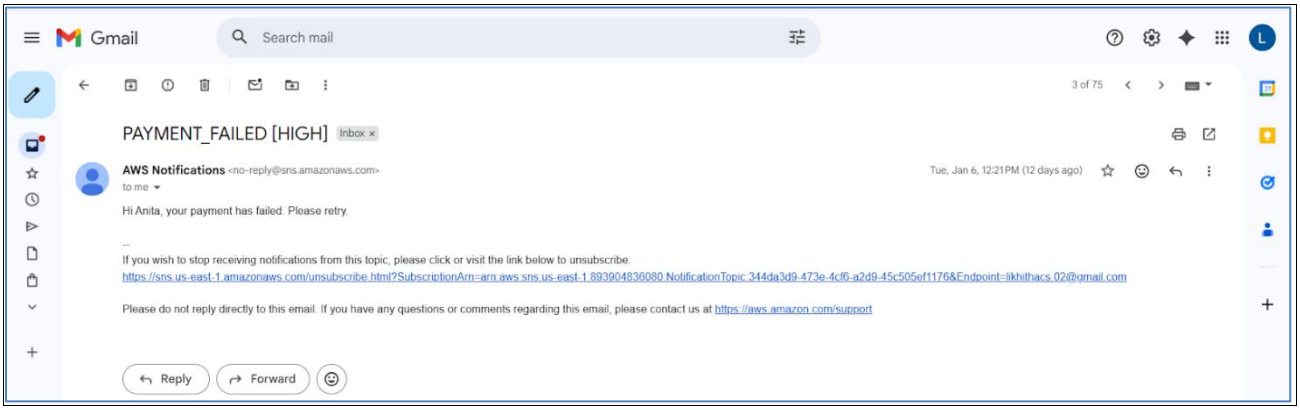
Click **Save**

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.”

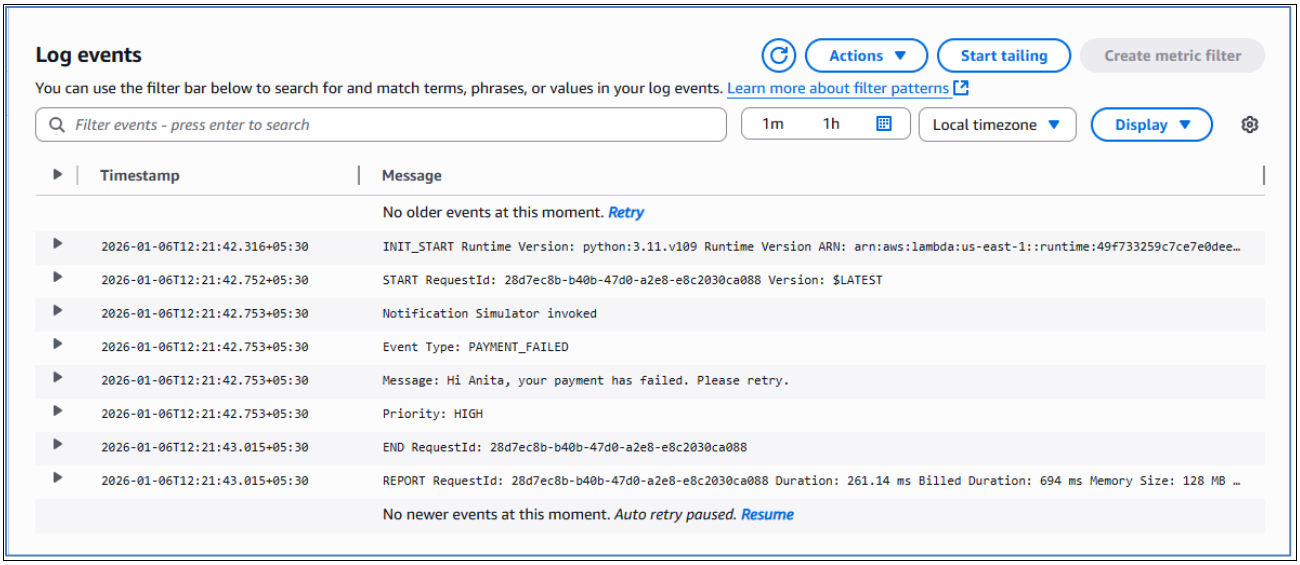


Verify CloudWatch Logs

Step 14: View logs

Lambda → Monitor → View CloudWatch logs
Open latest log stream.
Verify these prints exist:

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



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)

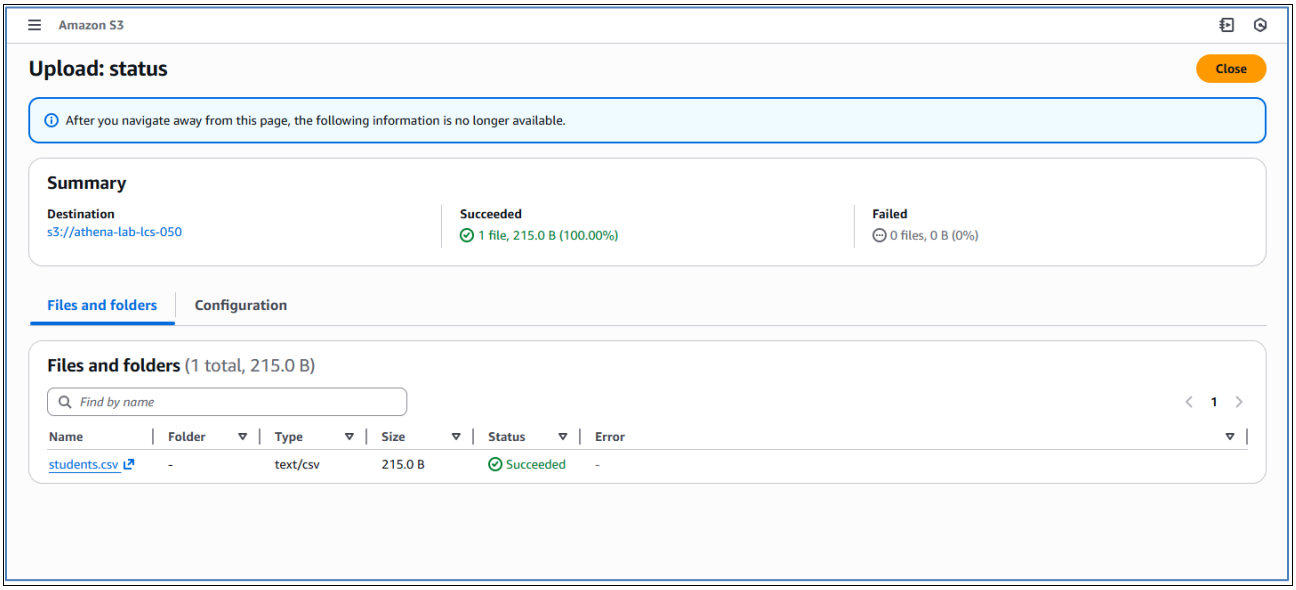
1. Open **Notepad**
2. Paste this data and save as: **students.csv**

```
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**

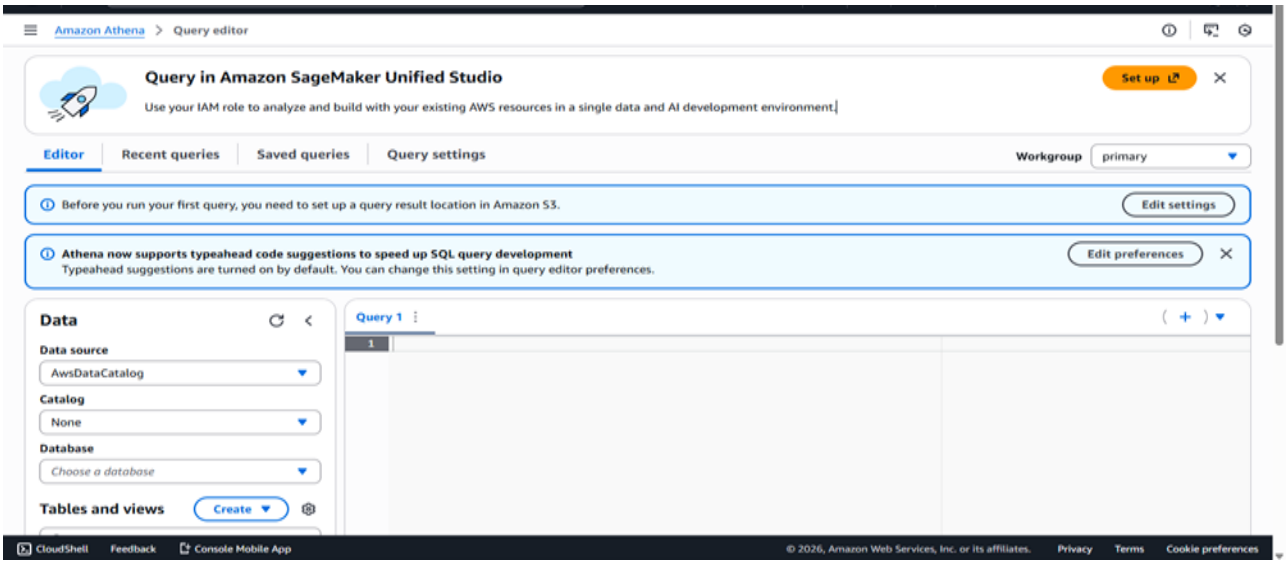
Now your CSV is in S3.



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.



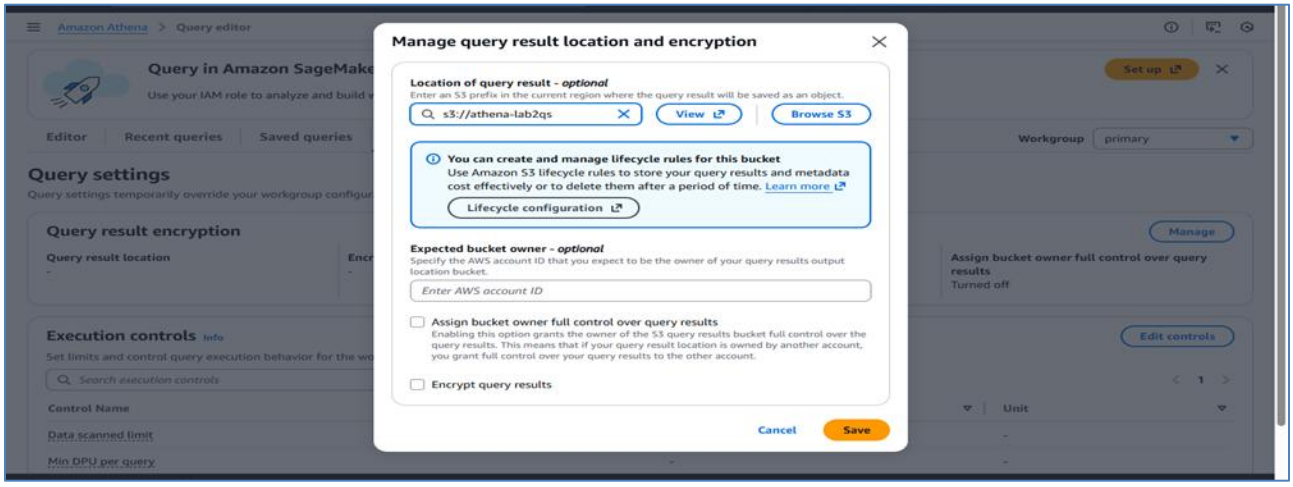
Part D — Create a Database in Athena

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

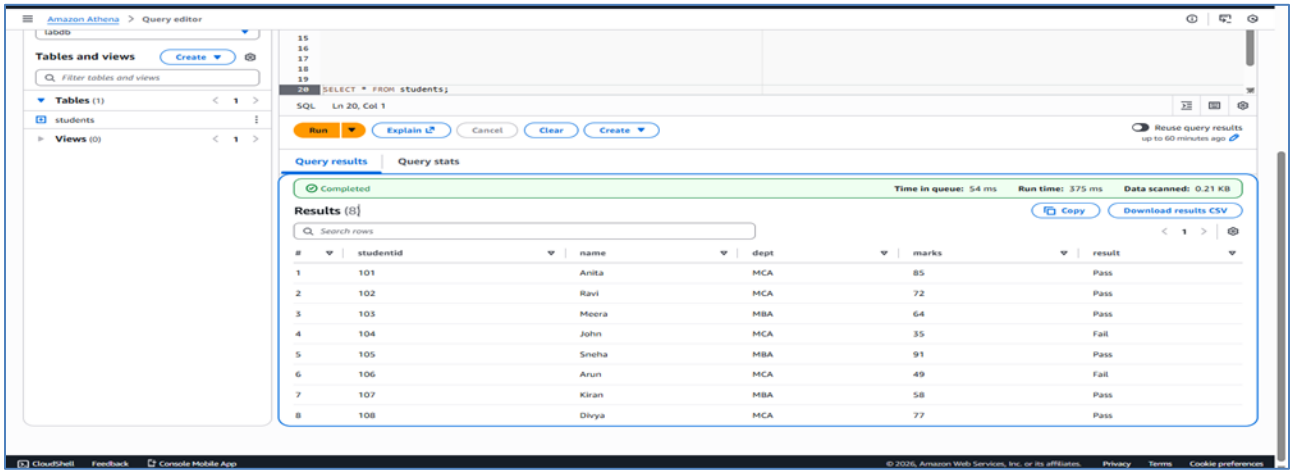
SELECT * FROM students;

Expected: 8 records.

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)

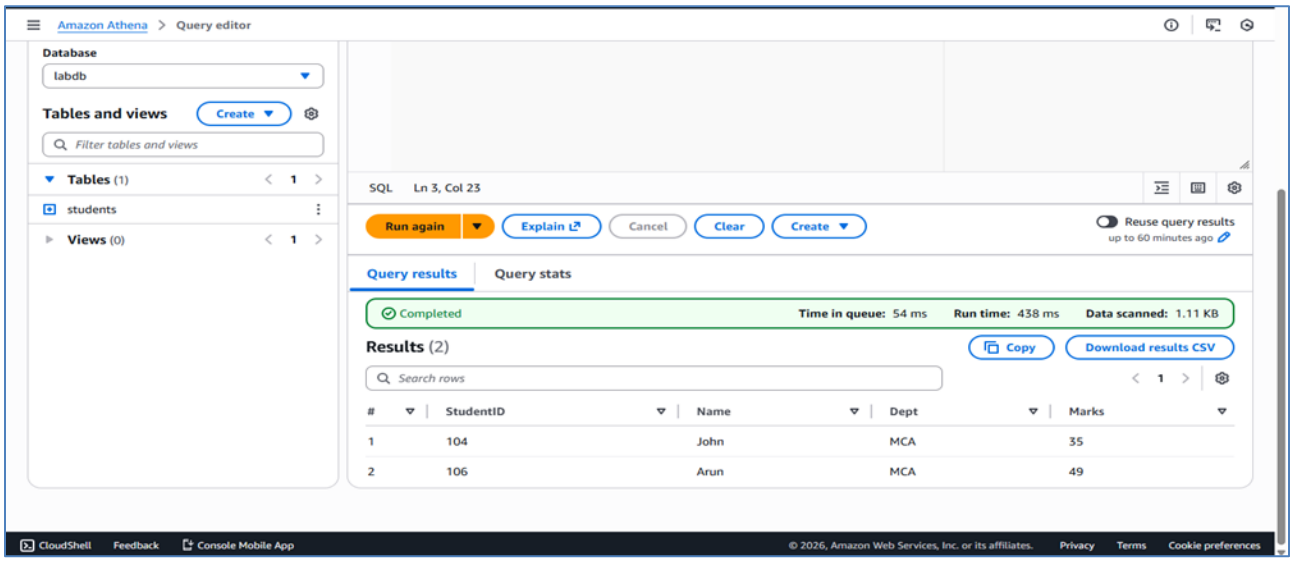
7) List failed students

SELECT StudentID, Name, Dept, Marks

FROM students

WHERE Result = 'Fail';

Expected: John, Arun



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:
    - students.csv
    - 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

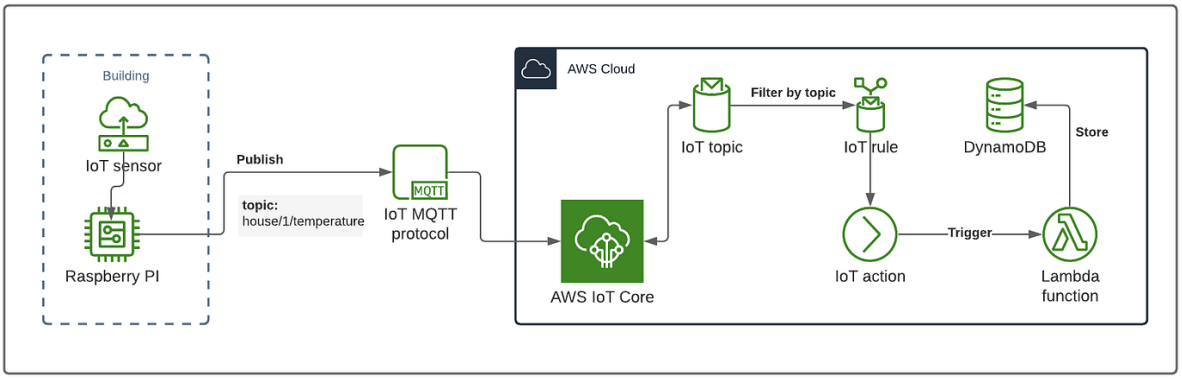
Cloud-Based IoT Data Ingestion and Processing using AWS - To understand how IoT device data is sent to the cloud, processed automatically, and logged using AWS managed services.

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:
    - Device certificate
    - Private key
    - 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:
    - Connect
    - Publish
    - Subscribe
    - 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)

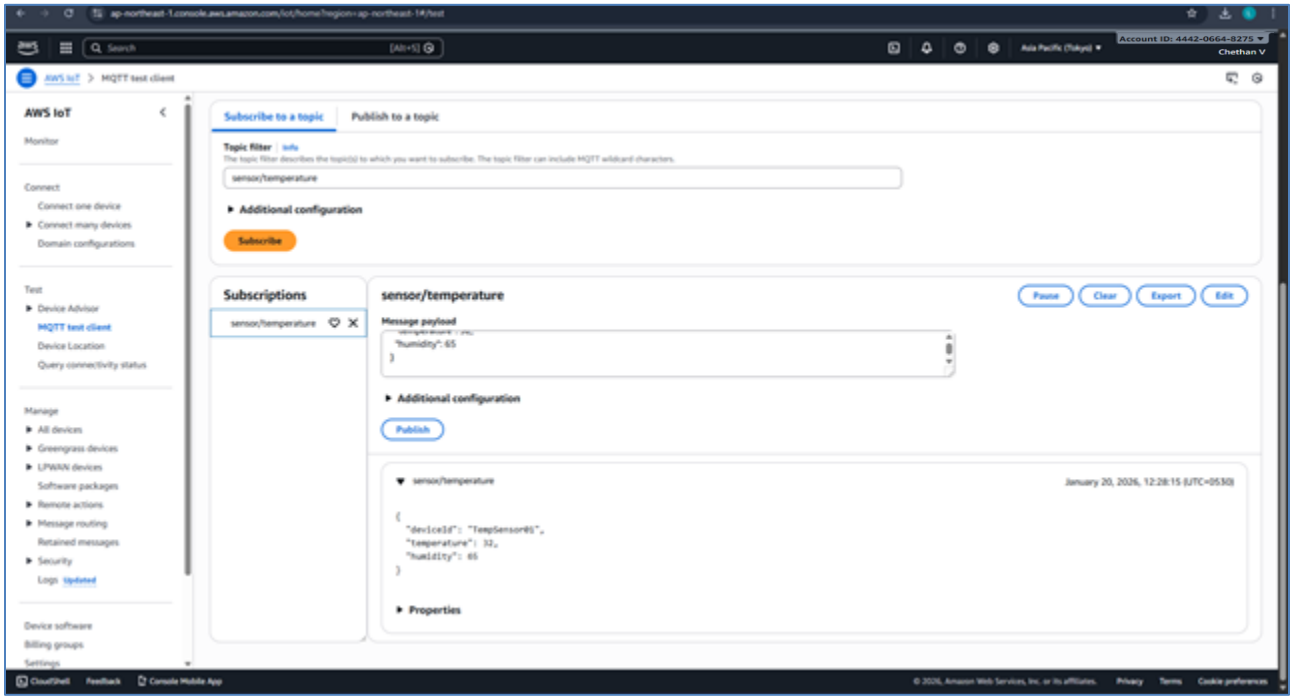
Rule = "when data arrives, do something"

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.


AWS IoT
MQTT test client
Subscribe to a topic
Publish to a topic
Topic filter
sensor/temperature
Additional configuration
Subscribe
Subscriptions
sensor/temperature
Pause
Clear
Export
Edit
Message payload
"humidity": 65
Additional configuration
Publish
sensor/temperature
January 20, 2026, 12:28:15 (UTC+0530)
{
"deviceId": "TempSensor01",
"temperature": 32,
"humidity": 65
}
Properties

DATE: 02-01-26

Exercise–28: Accelerate an S3 Static Website Using Amazon CloudFront (CDN)

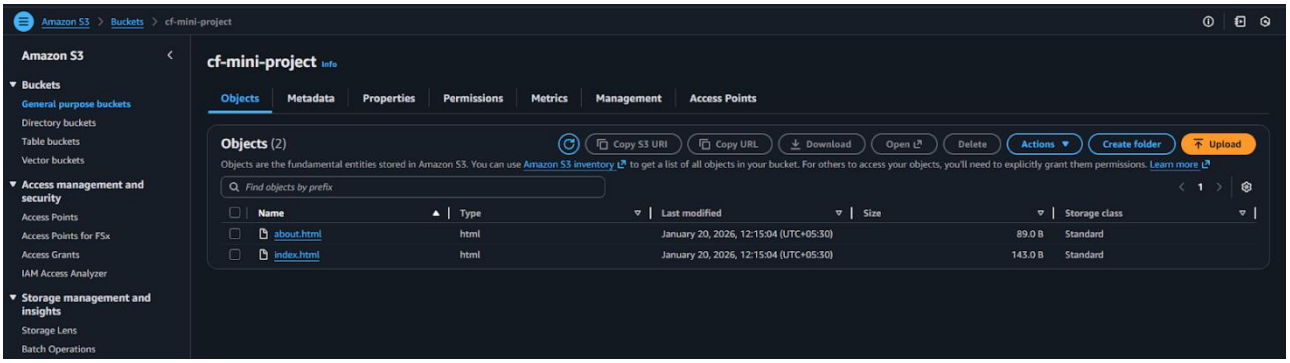
Pre-requisite

- 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 Invalidation.

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

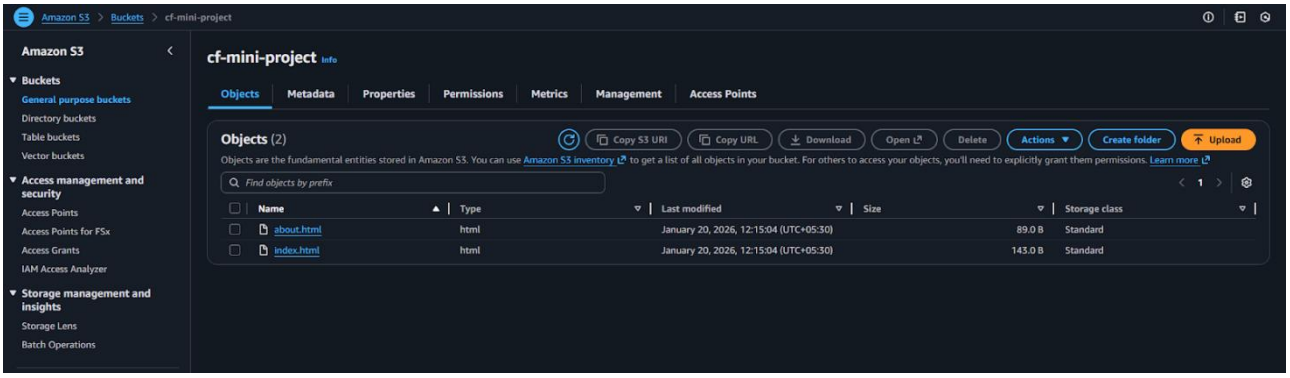
Step 2: Upload Website Files

Open your bucket → Upload

Upload:

- index.html
- about.html

Click Upload



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:

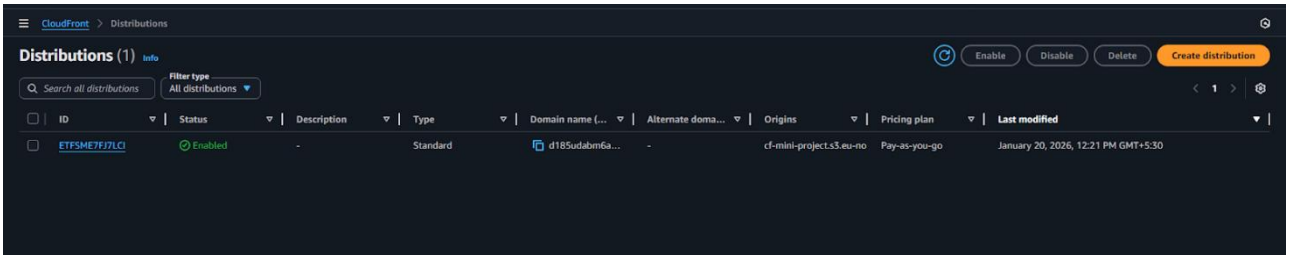
- "S3 bucket policy needs to be updated"

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.



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 Invalidation

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
```

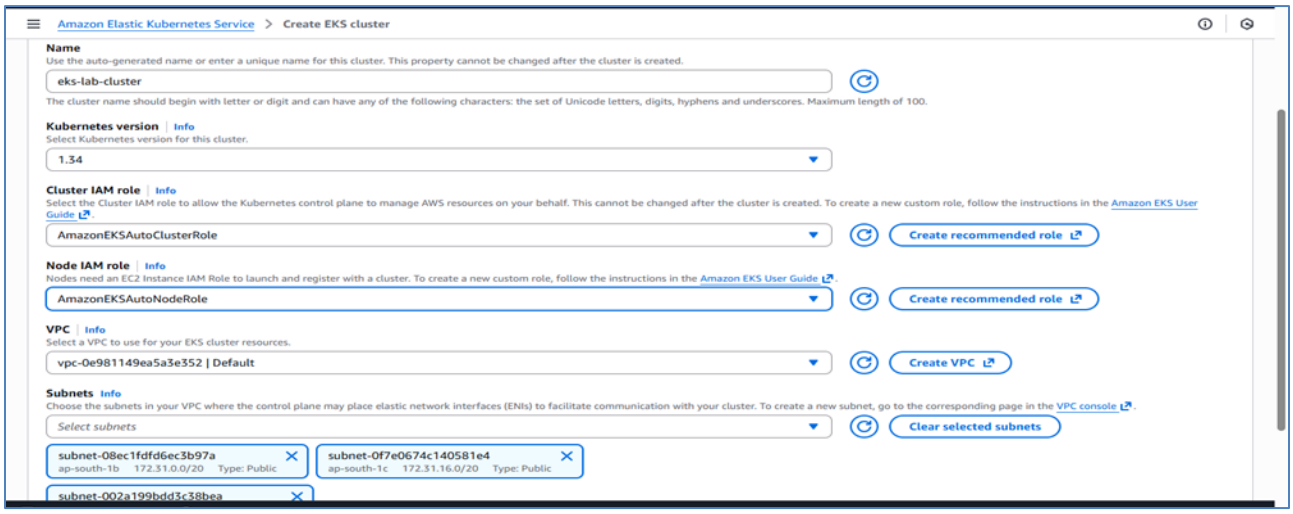
If eksctl asks / fails due to IAM (rare, but in case)

Run: aws sts get-caller-identity

If it shows:

- Your **root account ID**
- ARN ending with :root

Then IAM is **100% fine**.



This automatically:

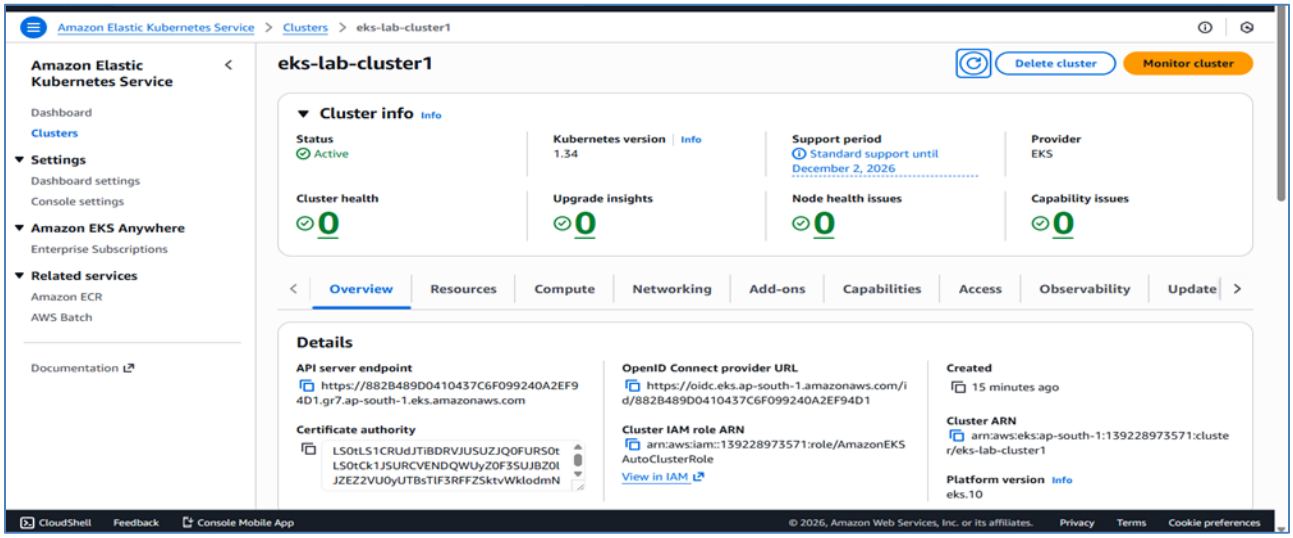
- Creates a **new VPC** with correct subnets/routes
- Creates the **EKS cluster**
- Creates the **Managed Node Group (worker nodes)**
- Configures kubeconfig for you

Wait until it completes (it will print “cluster created”)

Step A4: Verify nodes

kubectl get nodes

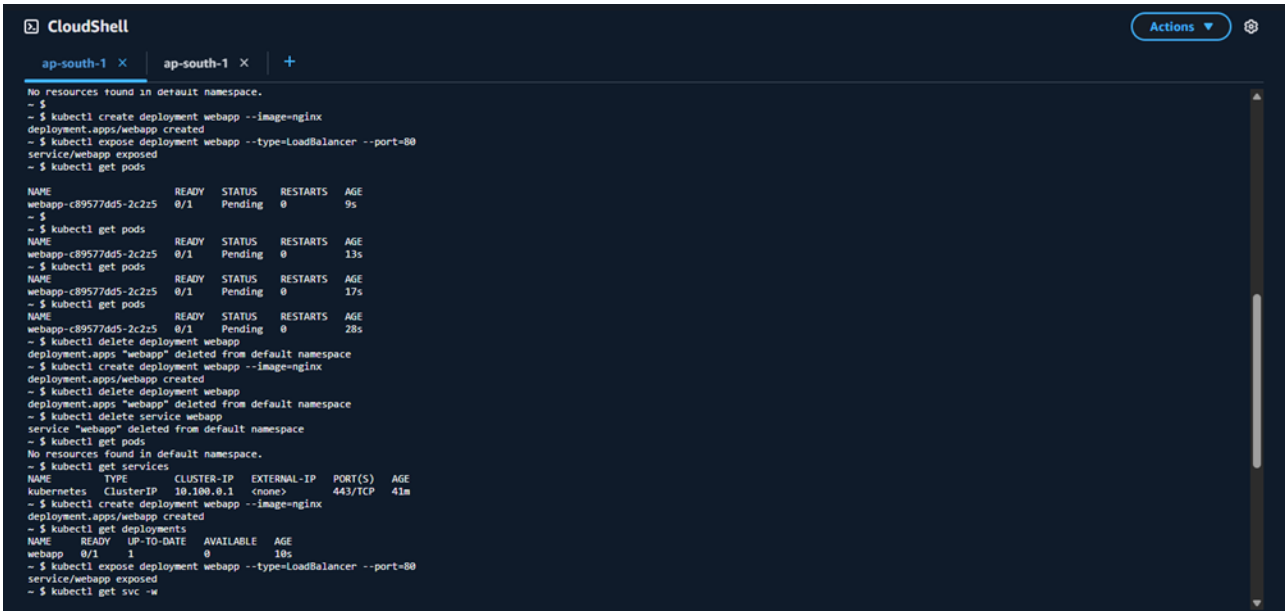
Expected: 2 nodes in **Ready** state.



Step A5: Deploy Nginx

kubectl create deployment webapp --image=nginx

kubectl get pods



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**