RAMAIAH
Institute of Technology

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

LABORATORY MANUAL

III Semester

Batch:2024-26

Name: Karthik Raghuram Shetty

USN: 1MS24MC042

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. Create AWS budget alert.
2.	AWS Identity Access Management (IAM) User and Group creation. Enable AWS IAM MFA. Create an AWS Account Alias (for Alternate Sign-in URL)
3.	Amazon S3 – Introduction, Bucket Creation and upload objects (files).
4.	Amazon S3 – Static Website Hosting (Multi-Page website), Versioning, Cross-Region Replication rule.
5.	Overview of EC2: To Launch a Windows EC2 Instance and Connect via RDP Client.
6.	- Launch a Linux EC2 Instance and Connect using SSH through PowerShell and PuTTY on Windows machine. - Launch a Linux EC2 Instance and Connect using SSH through Linux terminal on Linux machine. - 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 a Load-Balanced Web Application using Application Load Balancer (ALB), Auto Scaling Group (ASG), Custom AMI, and Target Group on AWS
13.	
14.	
15.	
16.	
17.	
18.	

19.	
20.	
21.	
22.	
23.	
24.	
25.	
26.	
27.	

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

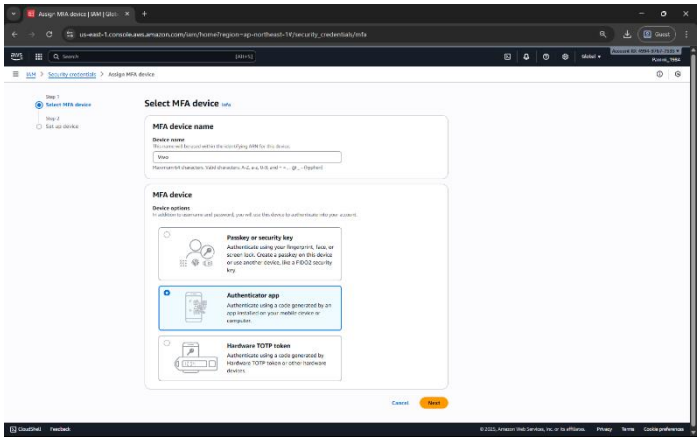
Notes

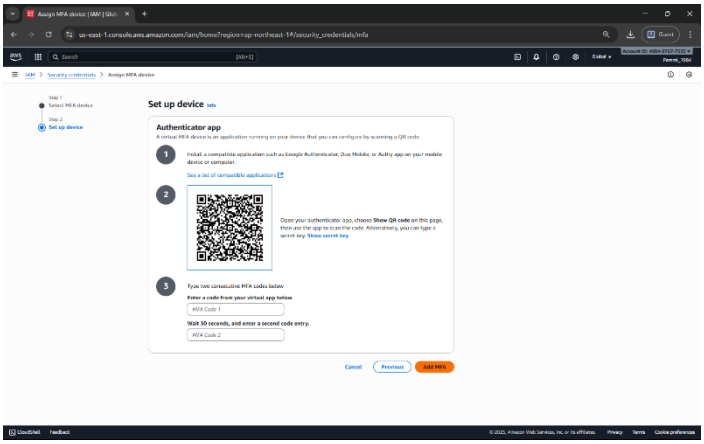
- Make sure your phone is unlocked, Bluetooth is on, and it uses a screen lock (fingerprint/PIN).

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.

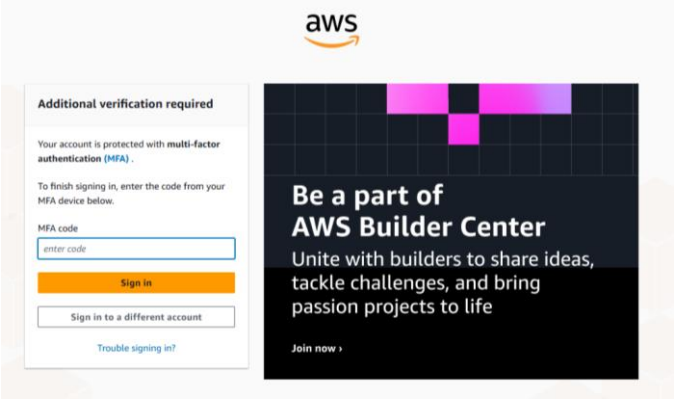




Next time you sign in, just choose **"Sign in with a passkey" → approve on phone**.

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.

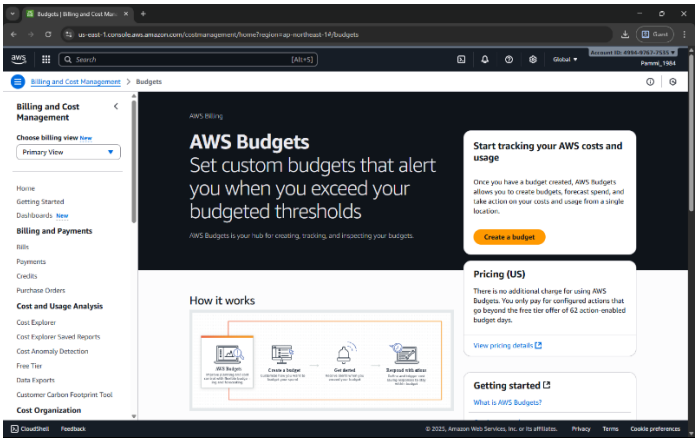
Step 2: After the ‘Budgets’ page loads

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

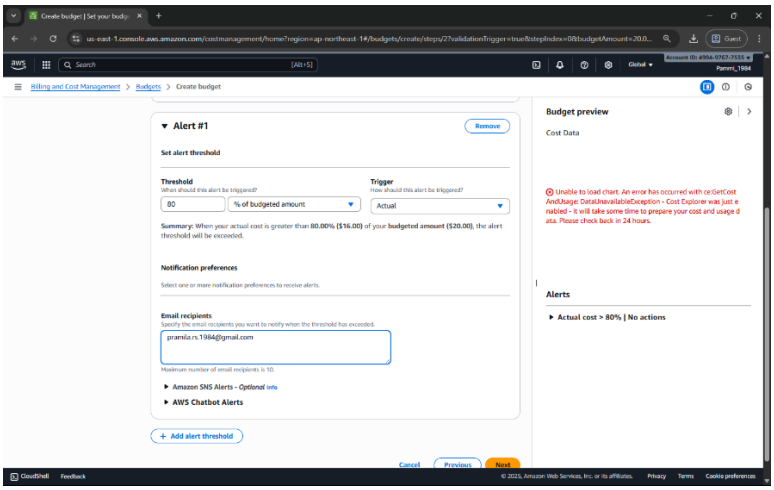
Click on Add an alert threshold

Under Set alert threshold

Set Threshold: 80 and Trigger: Actual

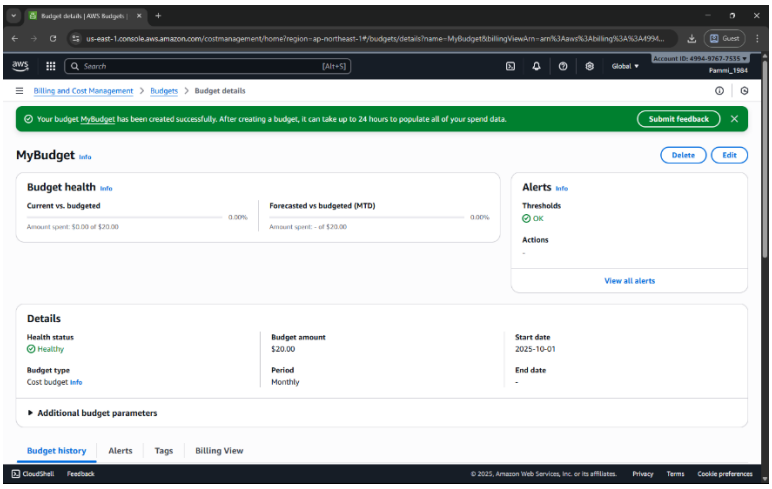
Email recipients: enter your email id.

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)

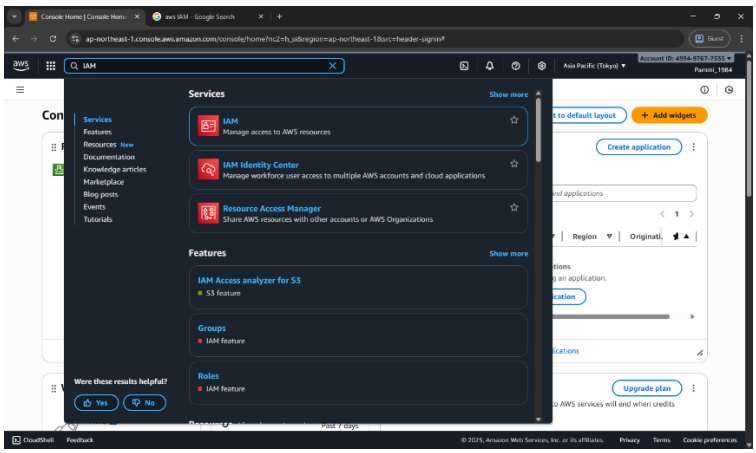
AWS Identity and Access Management (IAM) is a security service that helps you control who can access your AWS resources and what actions they can perform. It is a global AWS service.
It allows you to securely manage users, groups, roles, and permissions in your AWS account.

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

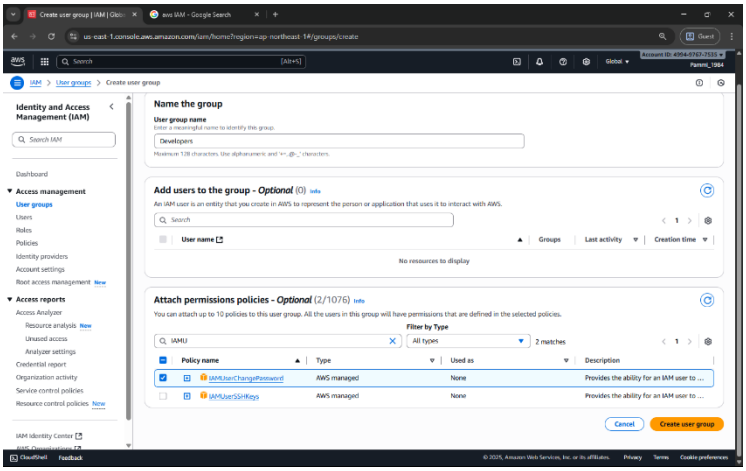
- In the IAM dashboard, look at the left sidebar.
- Click on "User groups" → then click "Create group".

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

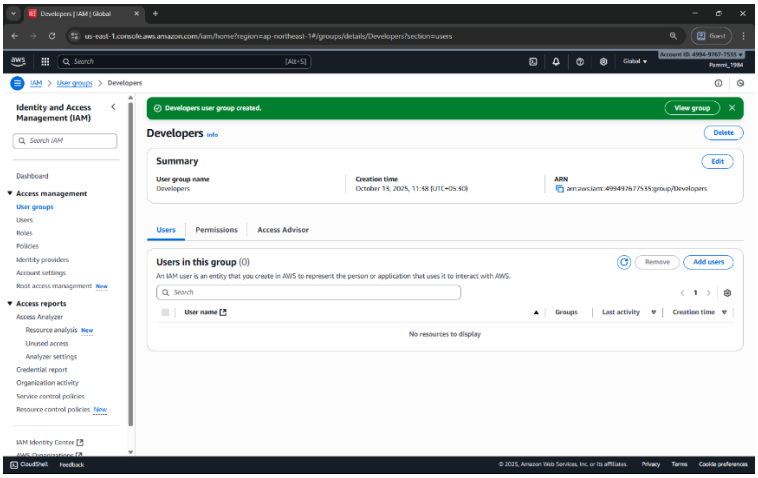


Step 5: Review and Create

- 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.



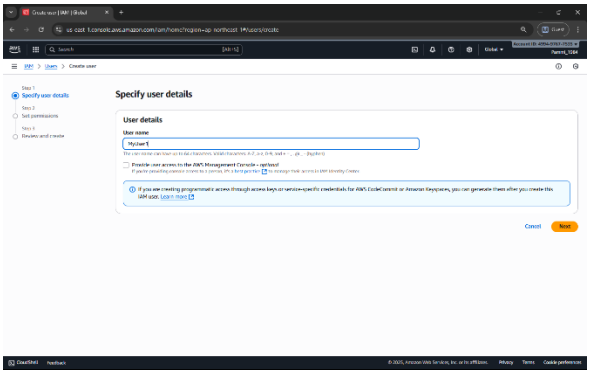
STEPS for AWS IAM User Creation

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.

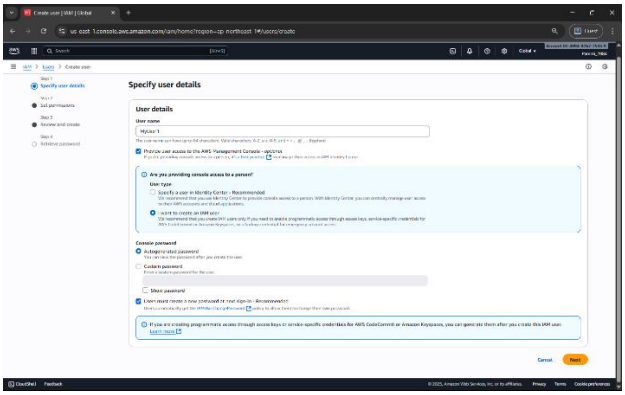
Step 2: Navigate to Users Section

- 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)

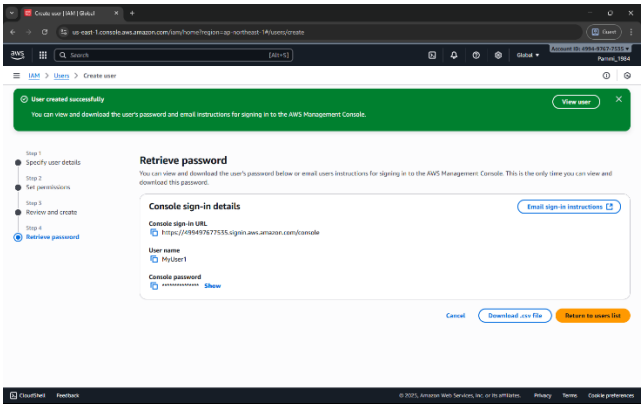
- Add tags like Department: MCA or Role: Faculty/Student for easy identification.
- Click Next to review.

Step 6: Review and Create User

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

Step 7: Save Credentials

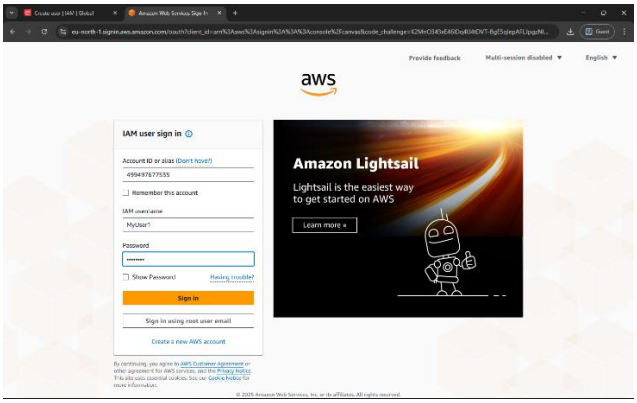
- After creation, AWS displays:
  - Console sign-in URL
  - Username
  - Password

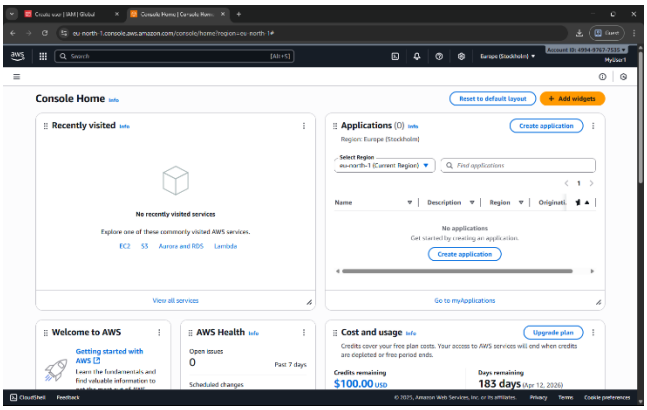


NOTE: Download or copy these credentials immediately — they cannot be retrieved later.

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

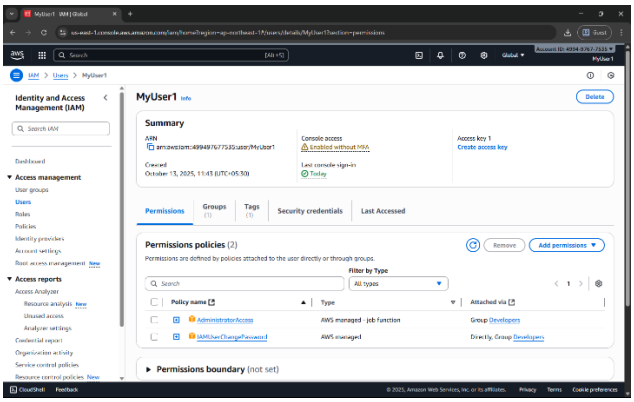
Step 1: Sign in to AWS Console as root user

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.

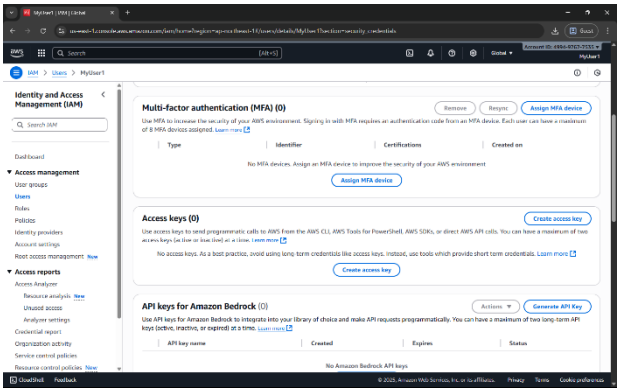
Step 3: Select a User

- 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)”.



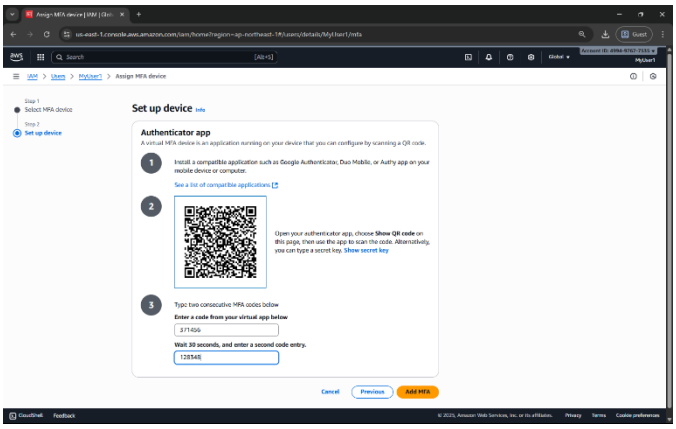
Step 5: Assign MFA Device

- 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.

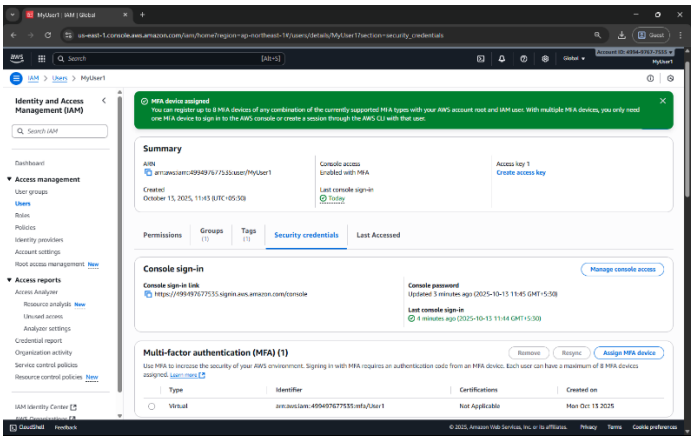


Step 7: Verify MFA

- 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)

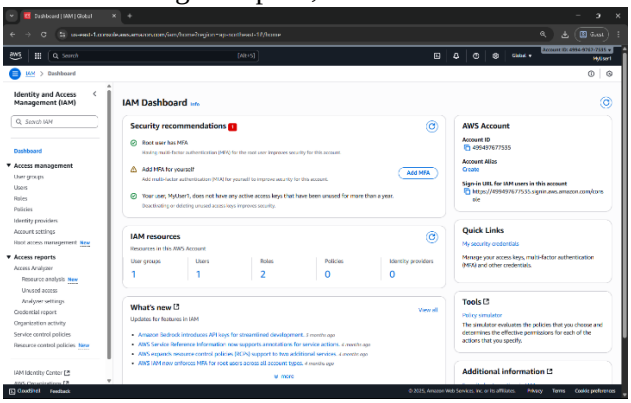
As an IAM user you can sign in using the default URL or create an account alias for it.
An Account Alias gives your AWS account a name instead of using the long numeric Account ID in your sign-in URL.
This makes it easier for IAM users to remember and log in.

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.

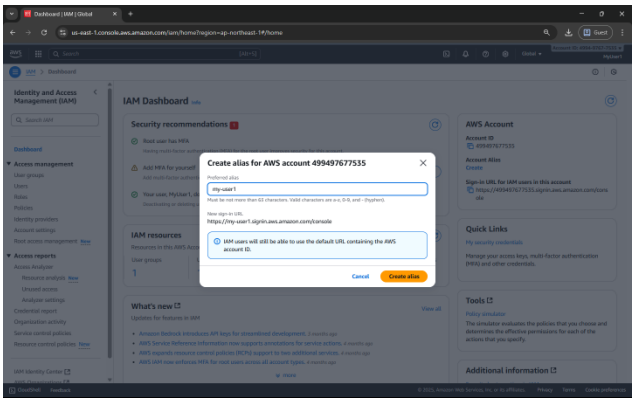
Step 2:

- 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).

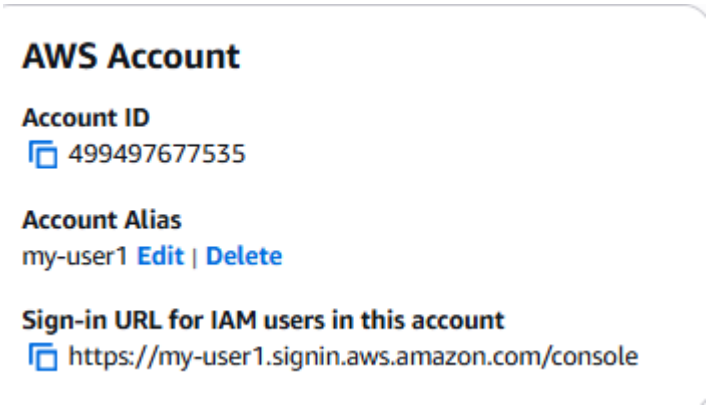


Step 4:

- 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:

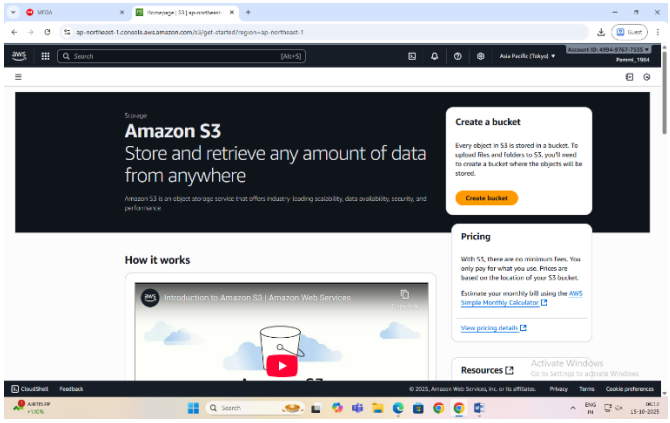
- Backup and archival
- Static website hosting
- Application data storage
- Media content delivery

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

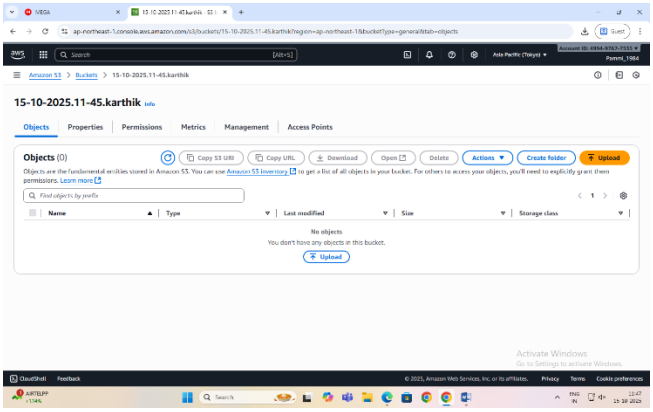
Step 1: Open the S3 Service

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



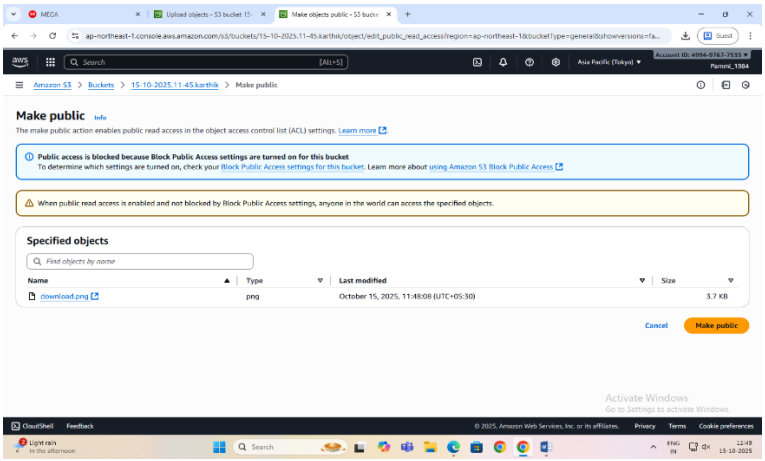
Step 2: Create a New Bucket

- 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.



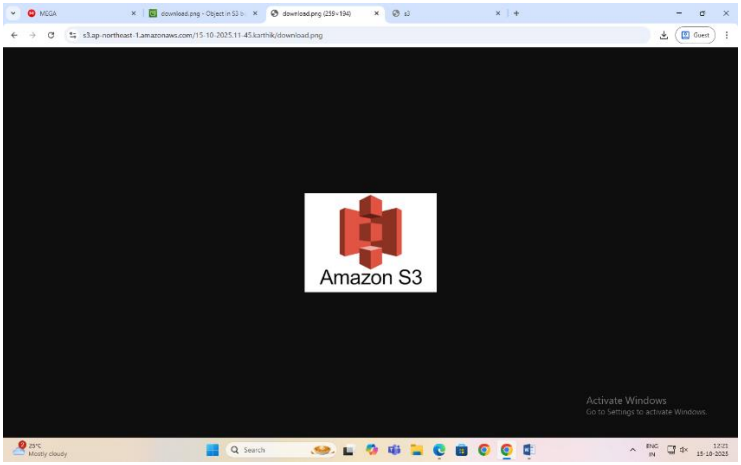
Step 4: Make the Object Public

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.

Date: 15-10-2025

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

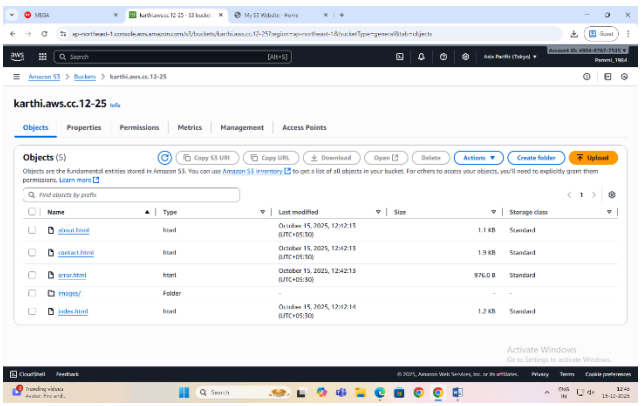
- Open the AWS Management Console → Navigate to S3.
- Click Create bucket.
- Select Bucket type: General purpose
- Enter a unique bucket name (e.g., my-static-web-demo).
- Uncheck "Block all public access."
- Click Create bucket.

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.

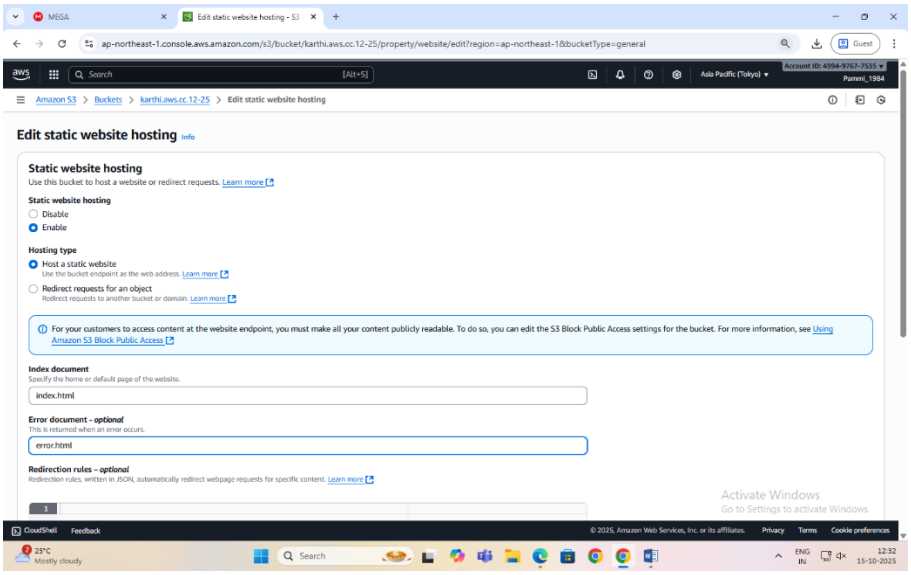


Step 3: Upload Website Files

- 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.

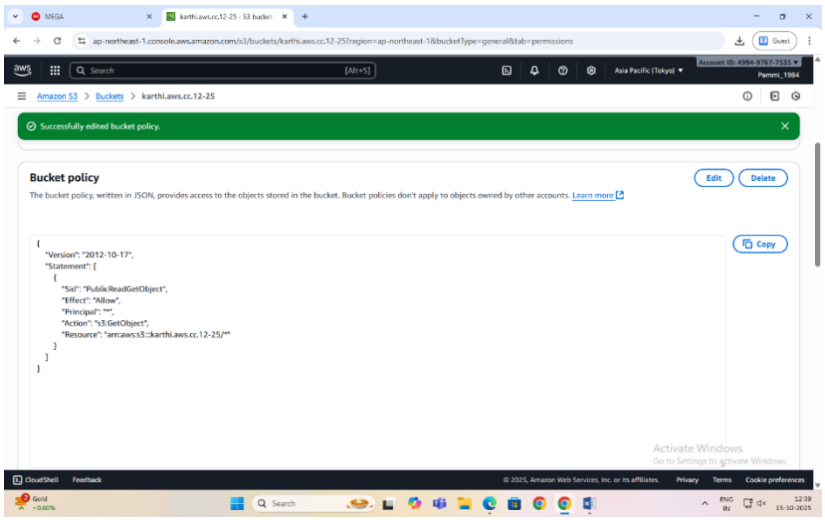


Step 5: Make Files Public (Bucket Policy)

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):

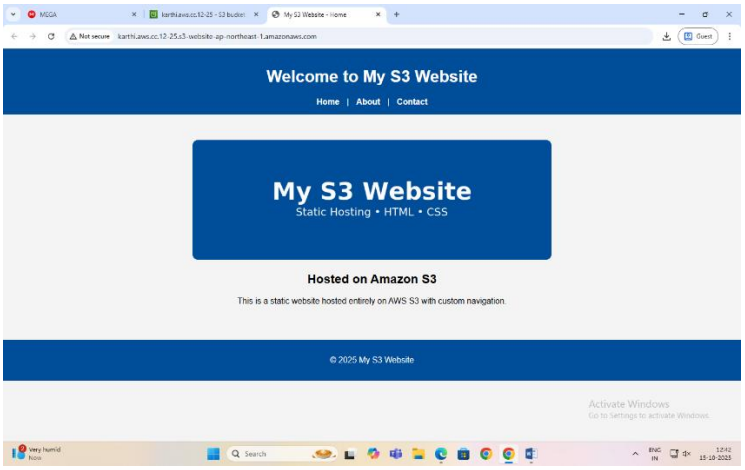
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-static-web-demo/*"
    }
  ]
}
```



- 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.

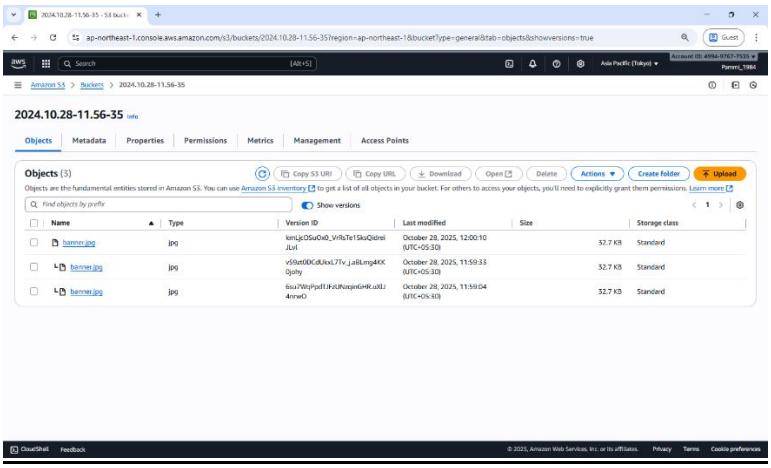
Steps to Enable Versioning

- 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

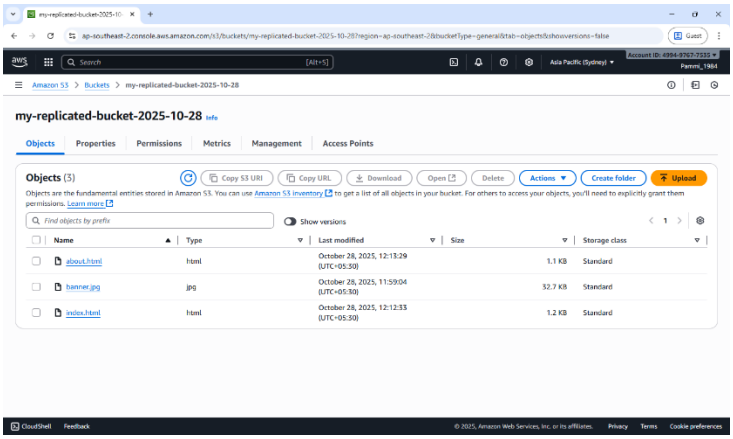
Step 4: Create Replication Rule
Choose Entire bucket or Prefix-based replication
Destination: Select the destination bucket

[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.

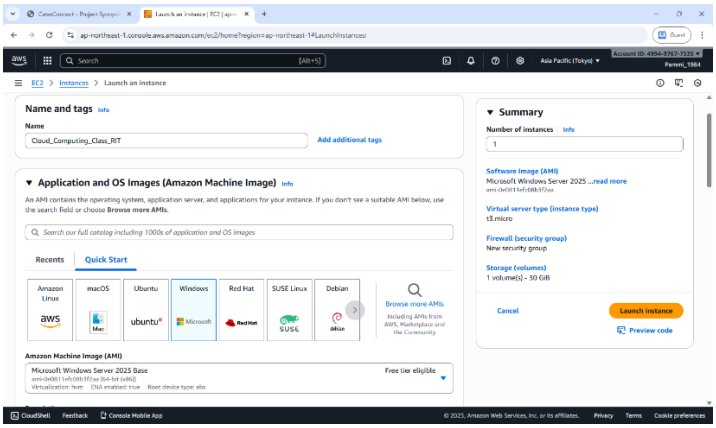
Steps to Launch a Windows EC2 Instance and Connect via RDP Client

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

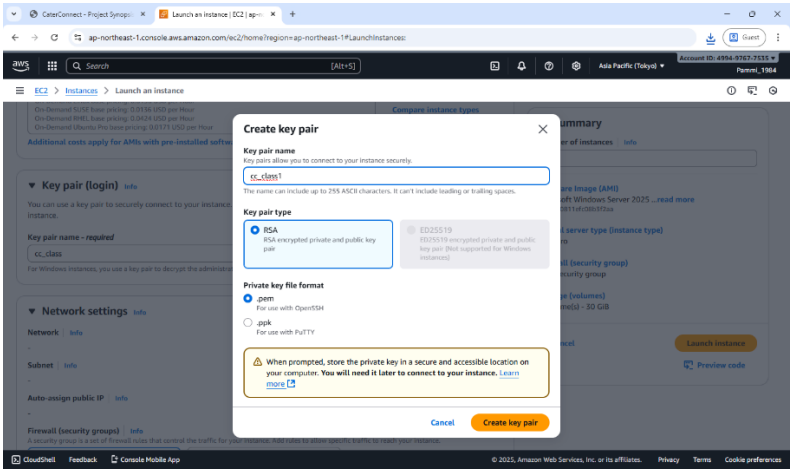
Under Key pair (login), choose an existing key pair or create a new one.
If creating a new key pair:

- 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).

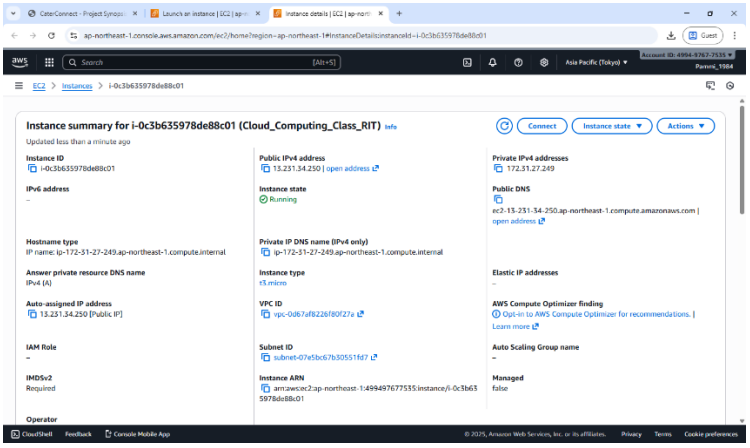


Step 7: Launch the Instance

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.

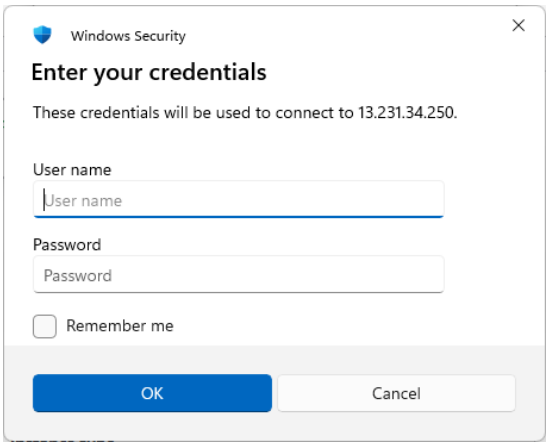
Copy the Public IPv4 address and Administrator Password shown.

Step 9: Connect Using RDP Client

On Windows system:

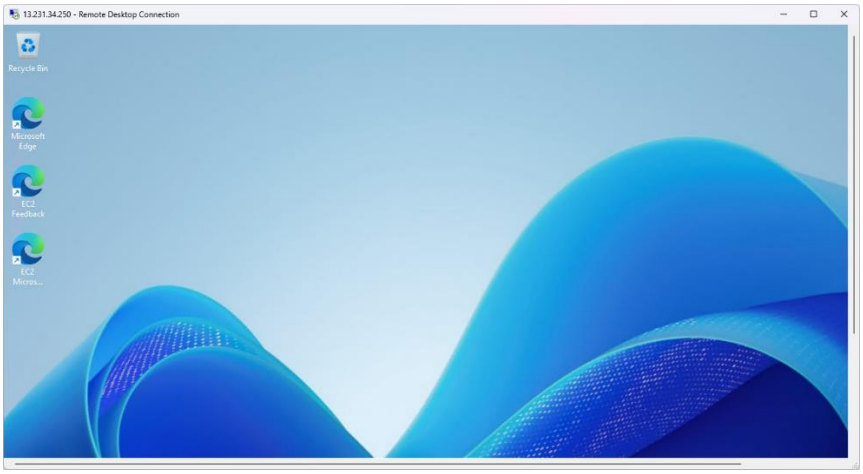
1. Open Remote Desktop Connection (from Start Menu).
2. Enter your instance's Public IPv4 address.
3. Click Connect → Enter:
    - Username: Administrator
    - Password: *(the decrypted password from AWS)*

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.

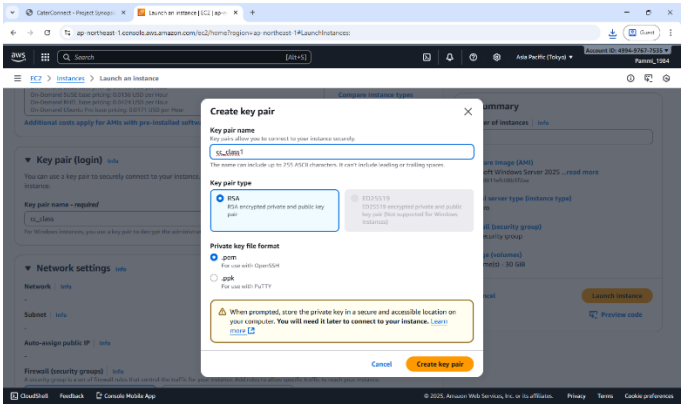
Scenario	Key File Format	Explanation
Using PuTTY on Windows	.ppk	The .ppk file is specific to the GUI based PuTTY application, a popular SSH client for Windows system.
Using PowerShell on Windows, Linux terminal on Linux	.pem	The .pem file is the default AWS key format used by the OpenSSH client available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

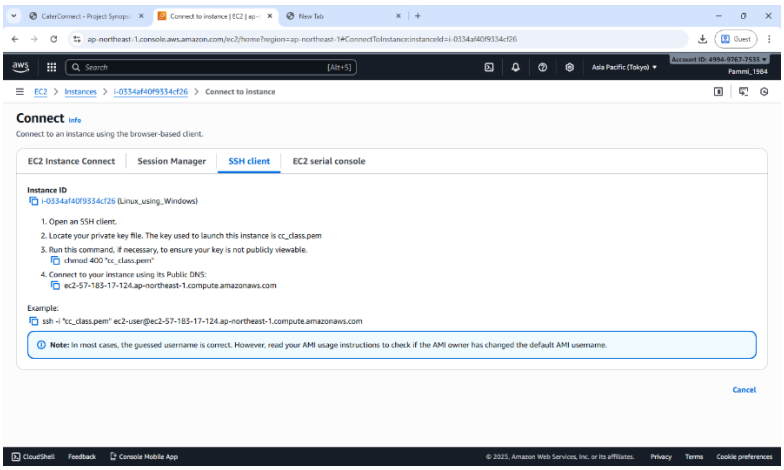
Step 6: Launch the Instance

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>
- When prompted, type **yes** to continue connecting.

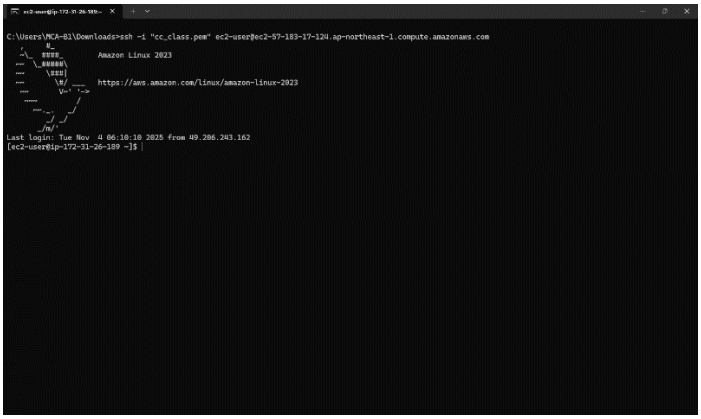
- You'll now be logged into your EC2 Linux terminal.

(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.



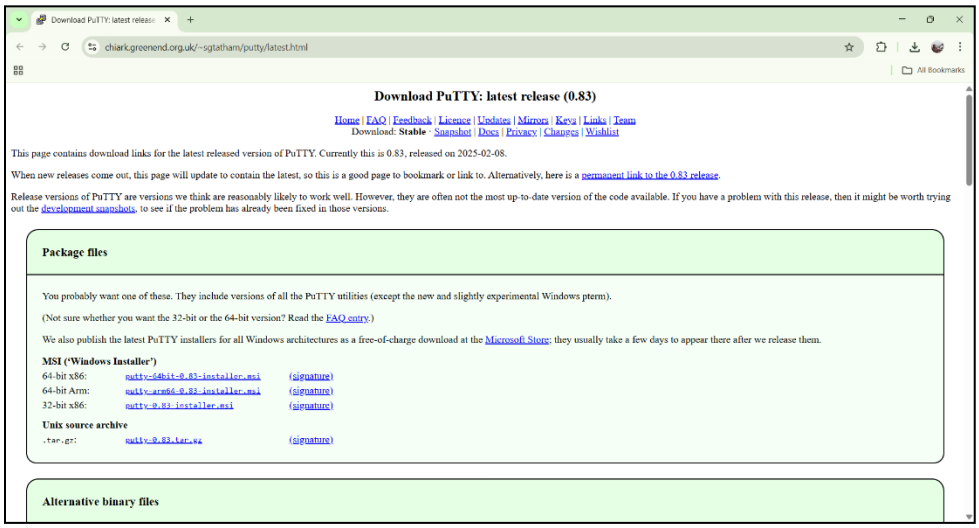
Step 10: Stop Instance after Use

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.

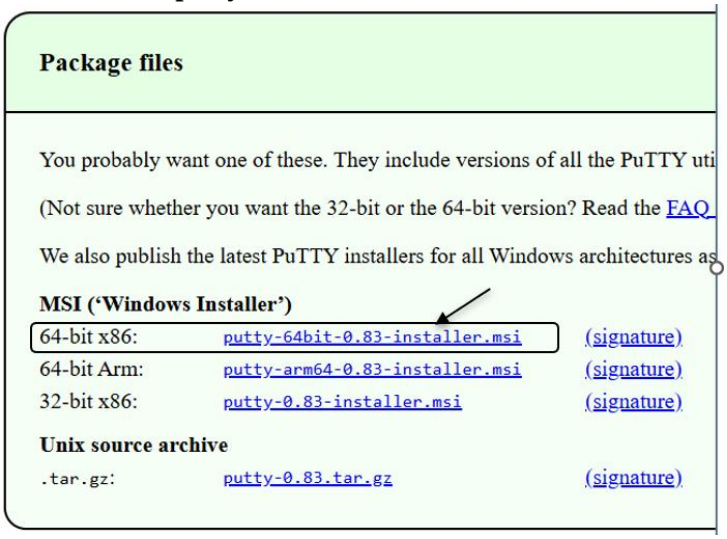
Use the correct, safe download link: Official download page: This is the only genuine PuTTY source.
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html



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

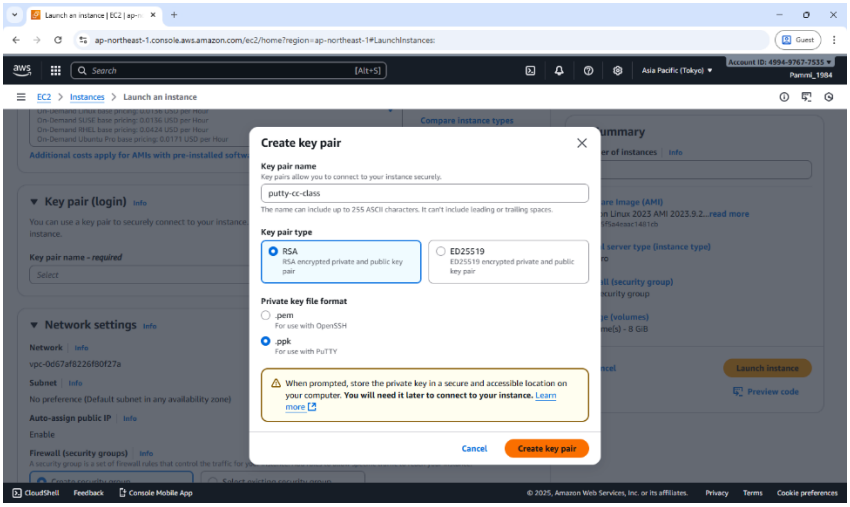
Under Name and Tags, enter an instance name, e.g., *Linux-PuTTY-Demo*.
Under Application and OS Images (AMI) → choose Amazon Linux 2 AMI (Free tier eligible).
Under Instance Type, select t2.micro (Free tier eligible).

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

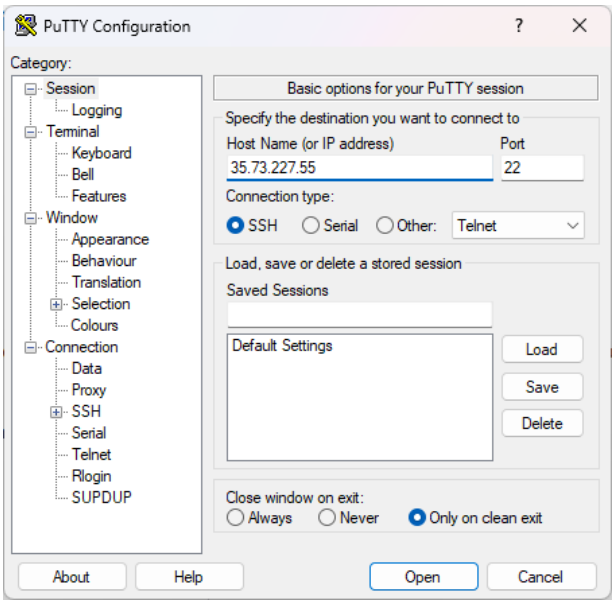
Review all configurations.
Click Launch Instance.
Wait until the Instance State changes to Running and Status Check = 3/3 passed.

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.

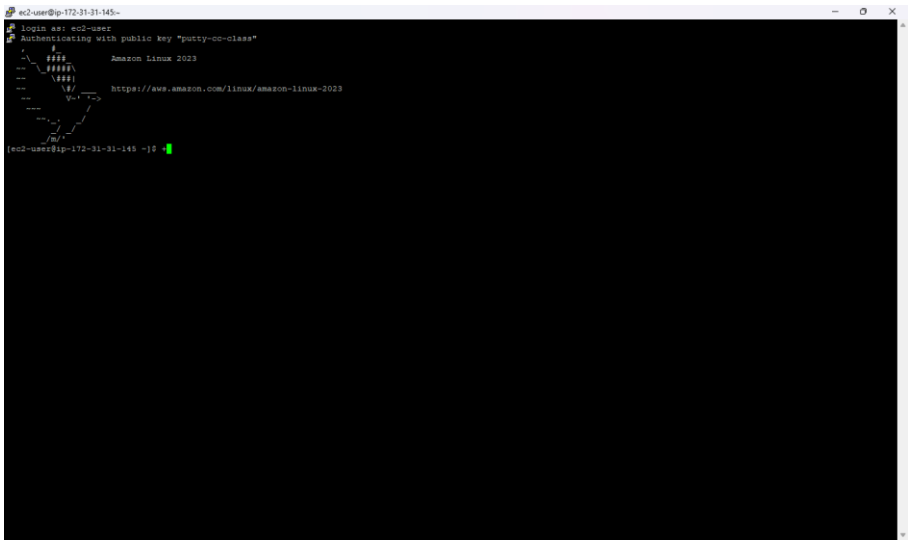
Step 8: Connect using PuTTY

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.

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>
- 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

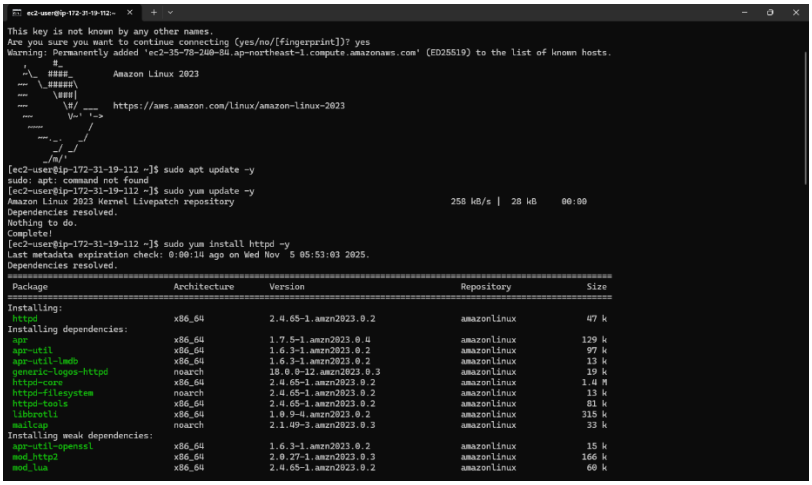
Step 2: Install Apache (httpd)
sudo yum install httpd -y
(This installs the Apache Web Server package.)

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.]

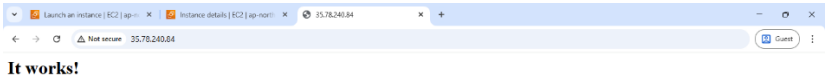


Step 7: Test Apache Server

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.

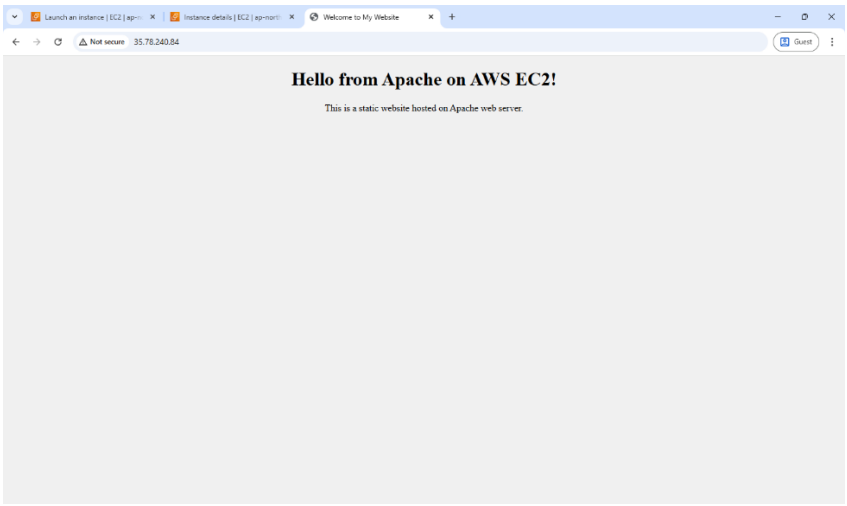
After Pressing Ctrl + O You'll see:
File Name to Write: index.html [Just press Enter to confirm saving with that name].

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:

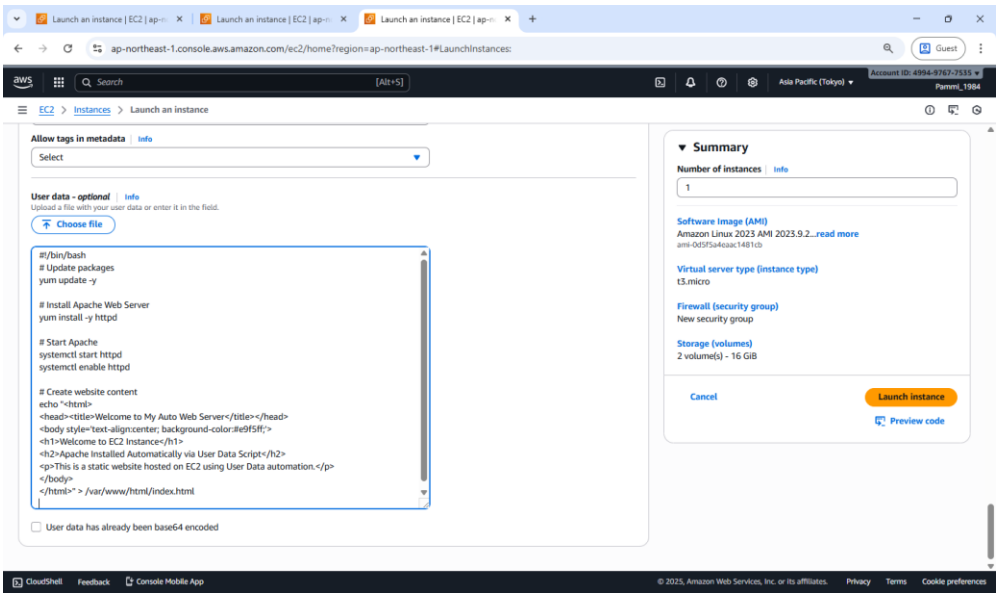
```
#!/bin/bash
# Update packages
yum update -y

# 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

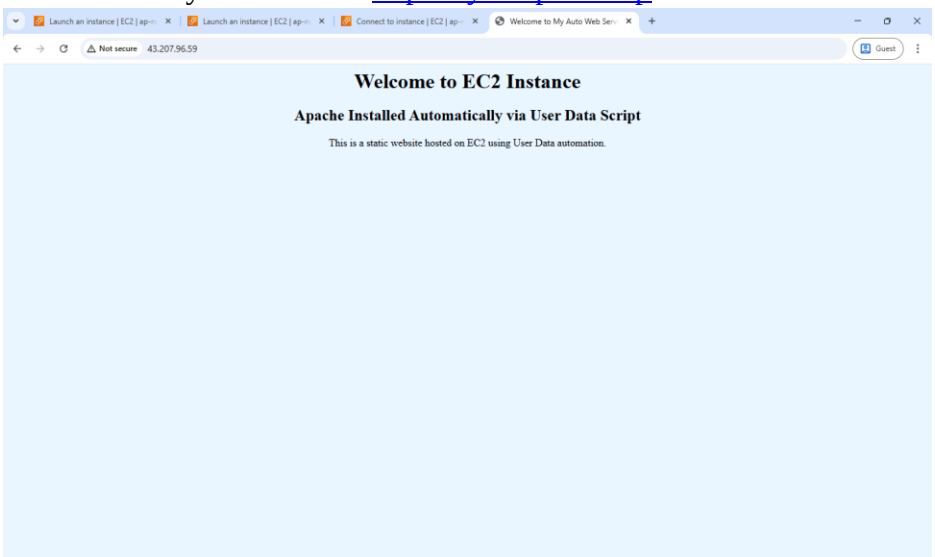
Click Launch Instance

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

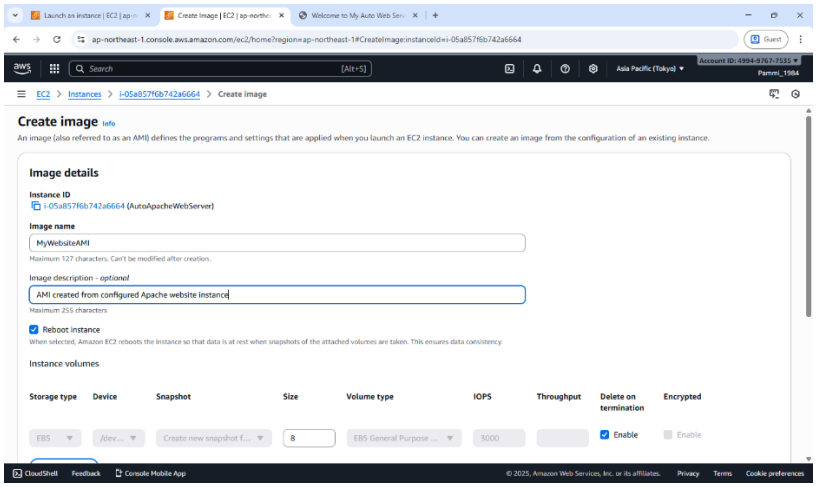
To capture the current configuration — installed packages, website files, and settings — into a reusable Amazon Machine Image (AMI).

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.

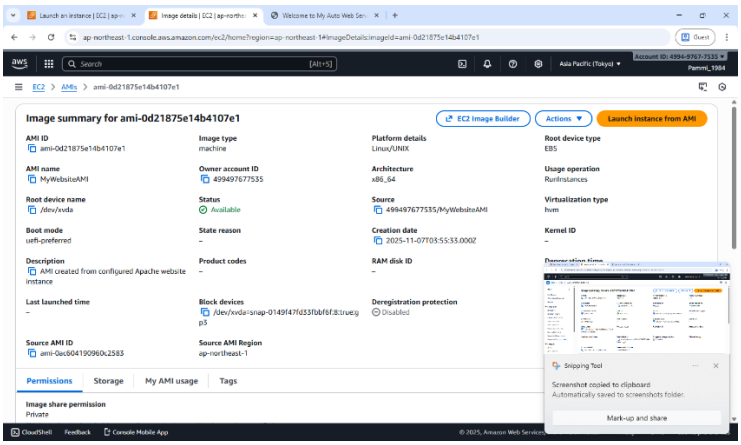
Step 5: Create image

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”).

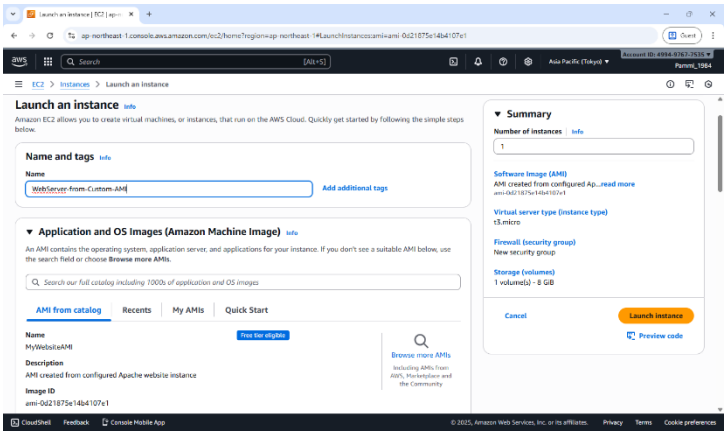
Step 2: Select Your Custom AMI

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

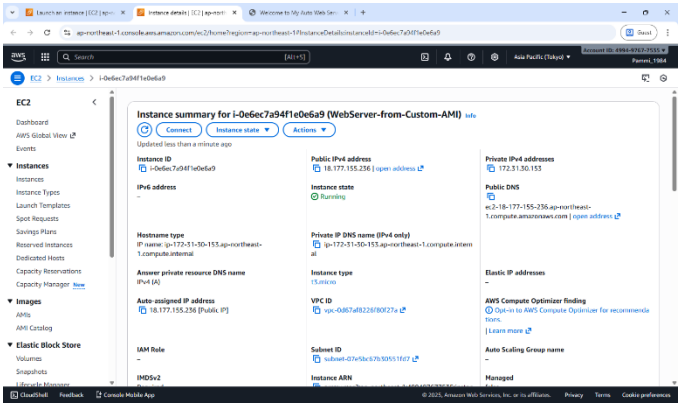
Name: WebServer-from-Custom-AMI
Instance type: t3.micro (Free Tier eligible)
Key pair: Choose an existing .pem key for SSH access.
Network settings:
VPC: Default
Subnet: Public
Security Group: Allow HTTP (80) and SSH (22)

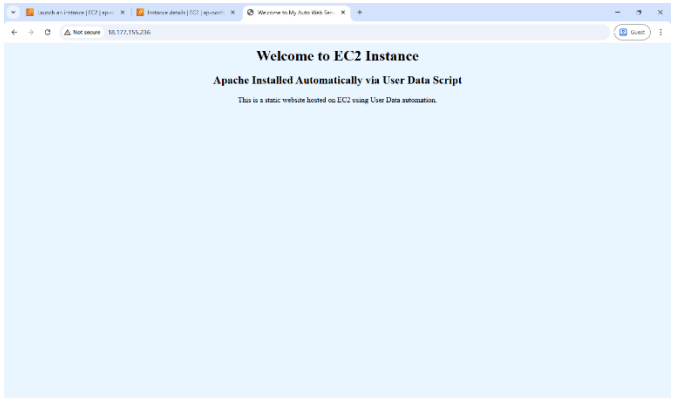
Step 5: Storage (EBS Volume)

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

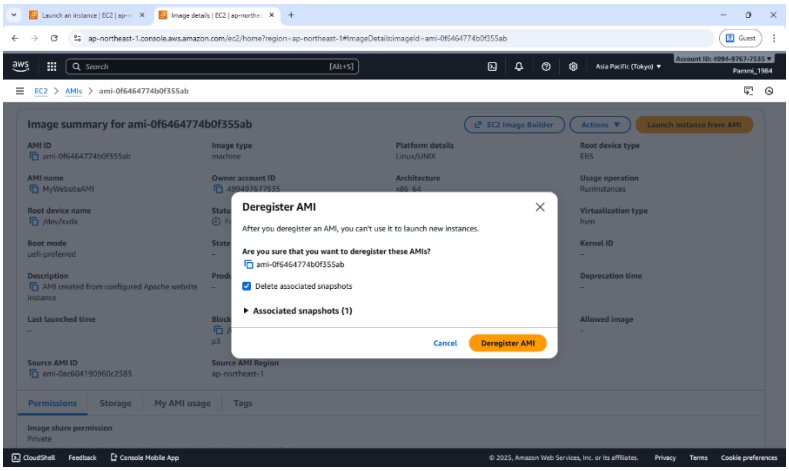
Sign in to the AWS Management Console.
Navigate to EC2 service.
In the left navigation pane, scroll down to Images → AMIs.

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.



- 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

9. Mini-Project: Hosting a Multi-Page Website using EC2 and S3

Website Structure:

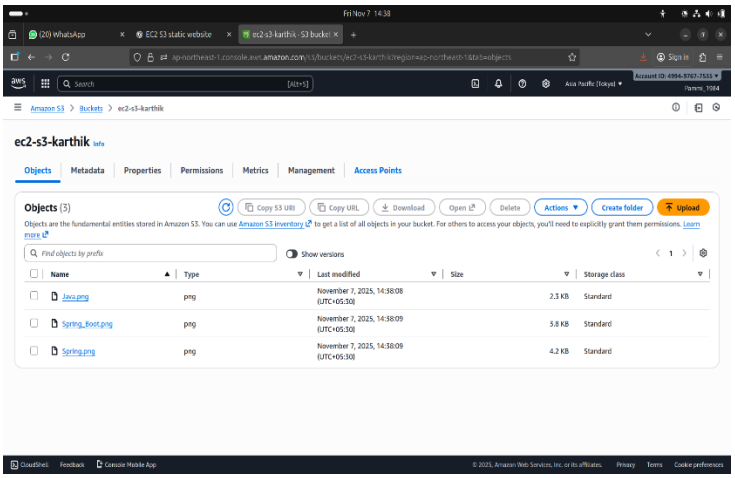
We'll make a **3-page static website**:

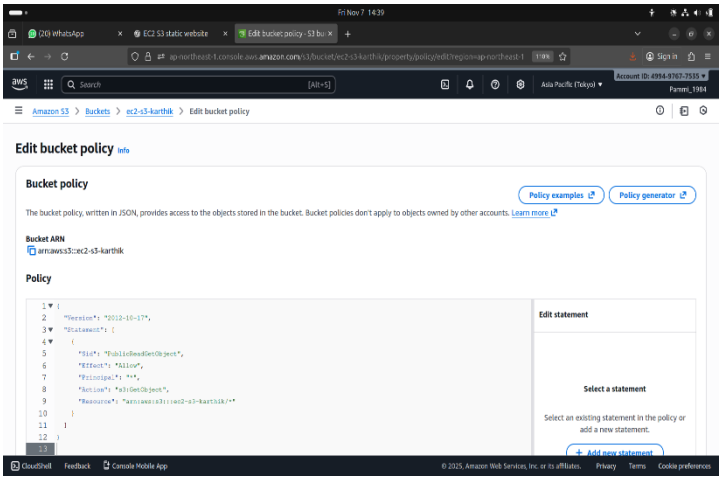
1. **Home Page (**index.html**)**
    - Welcome message
    - Navigation bar linking to other pages
    - Short intro about the site
2. **Gallery Page (**gallery.html**)**
    - Dynamically loads and displays images hosted in an S3 bucket
    - Uses public S3 URLs or signed URLs
3. **About Page (**about.html**)**
    - Information about the project and AWS services used

Step 1: Create a S3 Bucket and upload the images

1. Go to AWS Console → **S3** → **Create bucket**
    - Name: **<bucket-name>**
    - Region: same as your EC2 instance
    - Uncheck “Block all public access” (for demo)
2. Upload a few sample images.
3. Under each image → **Permissions** → **Make public** (or set a bucket policy like this):

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



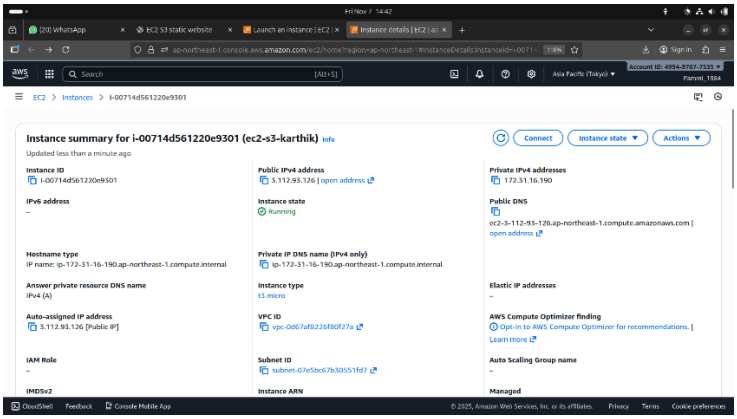


4. Note down the public URLs:

https://ec2-s3-karthik.s3.ap-northeast-1.amazonaws.com/Spring_Boot.png
https://ec2-s3-karthik.s3.ap-northeast-1.amazonaws.com/Java.png
https://ec2-s3-karthik.s3.ap-northeast-1.amazonaws.com/Spring.png

Step 2: Launch an EC2 Instance

1. **Launch an EC2 Instance**
    - Amazon Linux 2 or Ubuntu
    - Security group → open ports 22 (SSH) and 80 (HTTP)



2. **Connect via SSH**

    ssh -i your-key.pem ec2-user@<EC2-Public-IP>

3. **Install Apache**
    - sudo yum update
    - sudo yum install httpd -y
    - sudo systemctl start httpd
    - sudo systemctl enable httpd

4. **Check in browser:**

   http://<EC2-Public-IP> → should show Apache test page.

Step 3: Deploy Website Files

1. Change pwd to **/var/www/html** and create html files

   - **cd /var/www/html/**

➢ sudo nano index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CloudGallery - Home</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>Welcome to CloudGallery</h1>
    <nav>
      <a href="index.html">Home</a> |
      <a href="gallery.html">Gallery</a> |
      <a href="about.html">About</a>
    </nav>
  </header>
  <section>
    <h2>Your photos, served from the cloud ☁</h2>
    <p>This website is hosted on an AWS EC2 instance and fetches images from an AWS S3 bucket.</p>
  </section>

  <footer>
    <p>© 2025 CloudGallery | Powered by AWS</p>
  </footer>
</body>
</html>
```

➢ sudo nano gallery.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CloudGallery - Gallery</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>Gallery</h1>
    <nav>
      <a href="index.html">Home</a> |
      <a href="gallery.html">Gallery</a> |
      <a href="about.html">About</a>
    </nav>
```

```
  </header>
  <section>
    <h2>Images Fetched from S3</h2>
    <div class="gallery">
      <img src="https://cloudgallery-bucket-yourname.s3.amazonaws.com/img1.jpg" alt="Image 1">
      <img src="https://cloudgallery-bucket-yourname.s3.amazonaws.com/img2.jpg" alt="Image 2">
      <img src="https://cloudgallery-bucket-yourname.s3.amazonaws.com/img3.jpg" alt="Image 3">
    </div>
  </section>
  <footer>
    <p>Images served from AWS S3</p>
  </footer>
</body>
</html>
```

- sudo nano about.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>About - CloudGallery</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>About</h1>
<nav>
      <a href="index.html">Home</a> |
      <a href="gallery.html">Gallery</a> |
      <a href="about.html">About</a>
    </nav>
  </header>

  <section>
    <h2>About This Project</h2>
    <p>
      CloudGallery 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.
    </p>
    <p>
      This project highlights the simplicity of using AWS services together for scalable and cost-effective hosting.
    </p>
  </section>

  <footer>
    <p>© 2025 CloudGallery</p>
  </footer>
</body>
</html>
```

- sudo nano style.css

```
body {
  font-family: Arial, sans-serif;
  background: #f7f9fb;
  color: #333;
}

header {
  background: #0073bb;
```

```
  color: white;
  padding: 1rem;
  text-align: center;
}

nav a {
  color: white;
  margin: 0 10px;
}

section {
  padding: 2rem;
  text-align: center;
}

.gallery {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
  gap: 15px;
}

.gallery img {
  width: 300px;
  height: 200px;
  border-radius: 10px;
  box-shadow: 0 0 10px rgba(0,0,0,0.2);
}

footer {
  background: #222;
  color: white;
  text-align: center;
  padding: 1rem;
}

  box-shadow: 0 0 10px rgba(0,0,0,0.2);
}

footer {
  background: #222;
  color: white;
  text-align: center;
  padding: 1rem;
}
```

Step 4: Adjust Permissions & Test

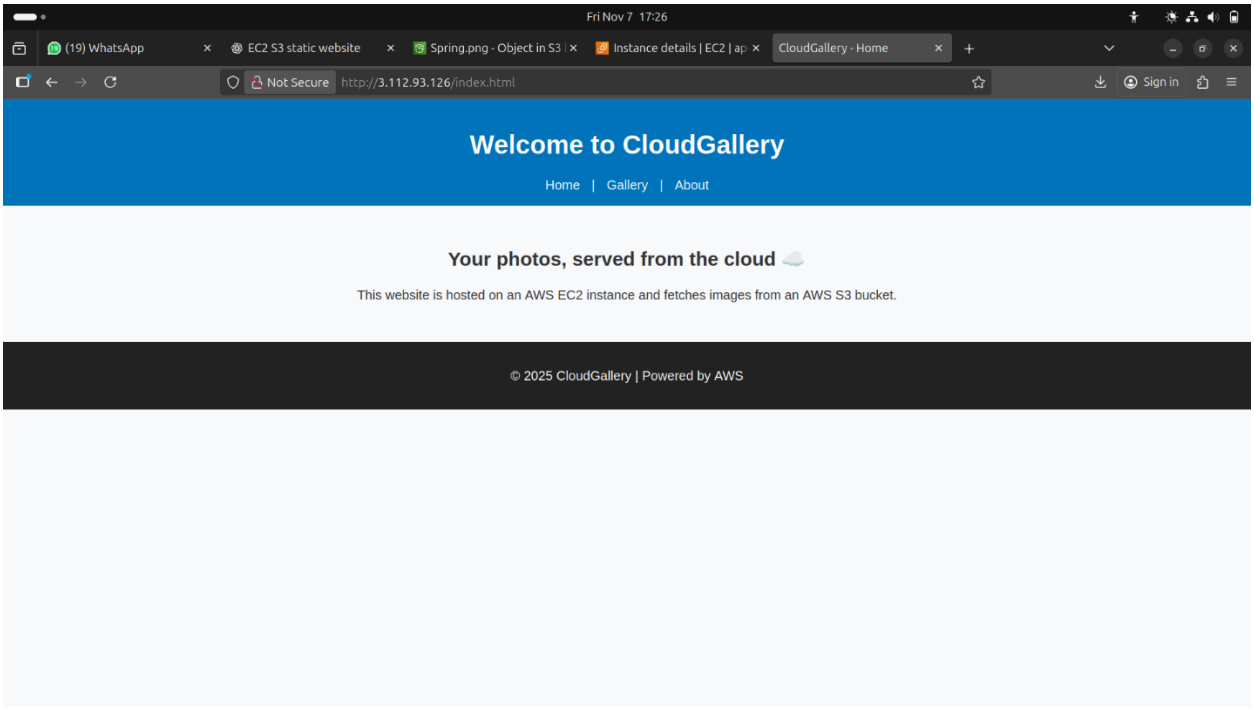
- sudo chmod -R 755 /var/www/html

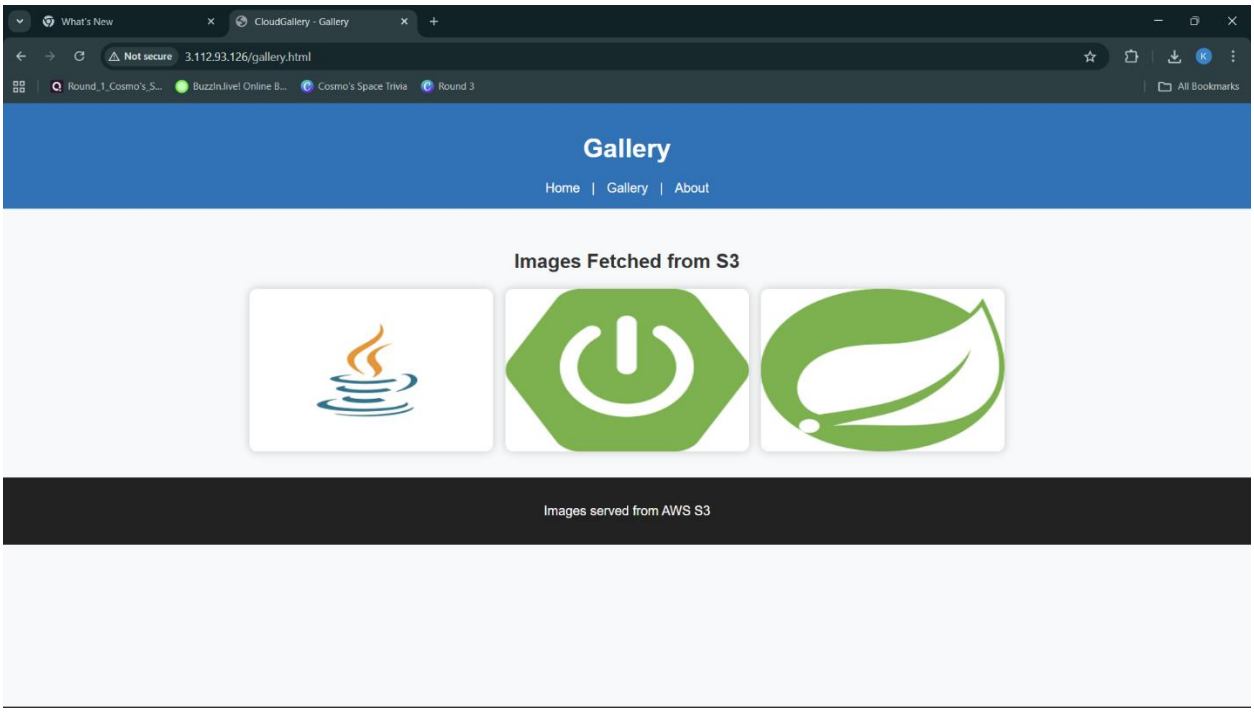
Open your browser and visit:

- **http://<EC2-Public-IP>/index.html**

You should see:

- Home page
- Gallery fetching S3 images
- About page





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.

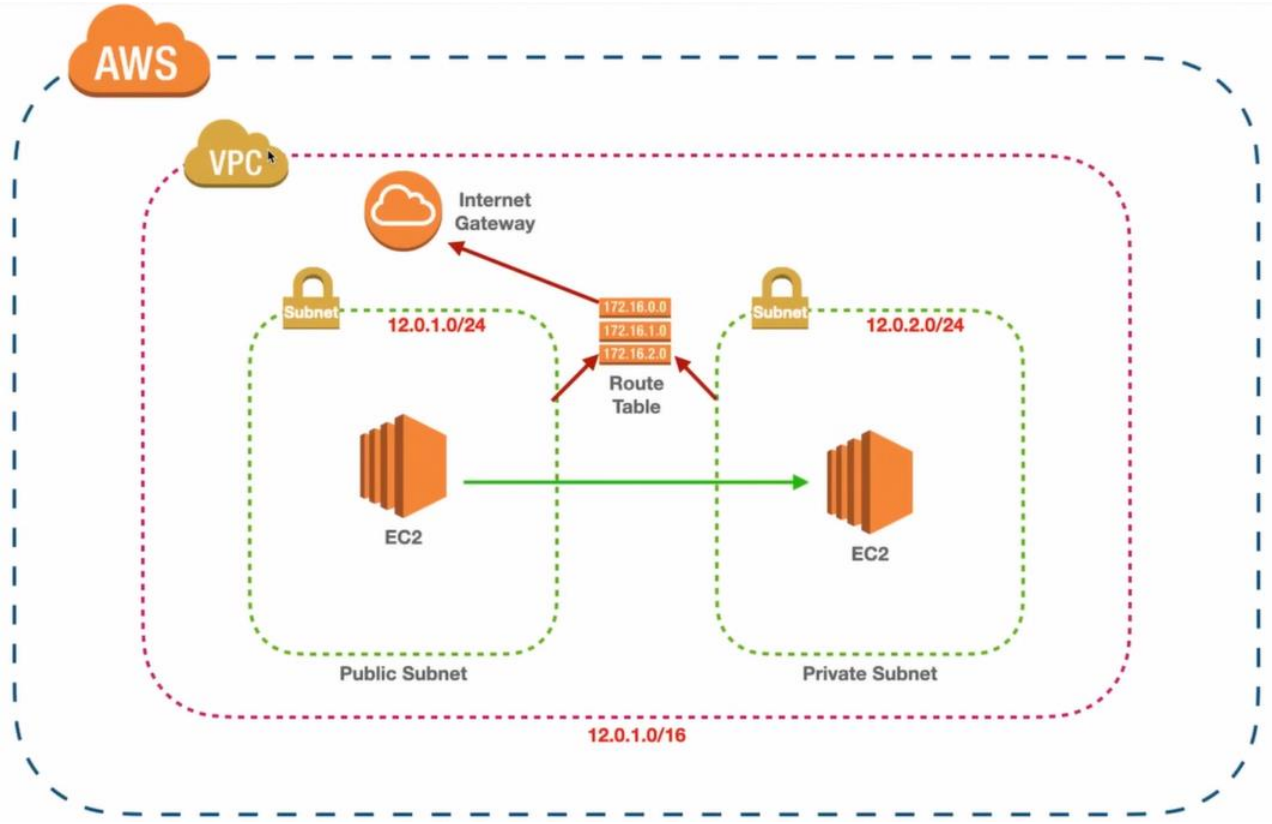
- Useful for **NAT Gateways**, load balancers, or fixed server access.

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
- Custom **Security Groups** to enforce network isolation

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 a Load-Balanced Web Application using Application Load Balancer (ALB), Auto Scaling Group (ASG), Custom AMI, and Target Group on AWS

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)
- Auto Scaling Group (ASG)
- CPU-based scaling policies
- Stress testing to validate scaling

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.

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.

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

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

1. EC2 → **Target Groups** → Create Target Group
2. Target type: **Instances**
3. Name: WebApp-TG
4. Protocol: HTTP
5. Port: 80
6. Health Check Path: /
7. 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)

1. EC2 → Load Balancers → Create Load Balancer
2. Choose Application Load Balancer
3. Name: WebApp-ALB
4. Scheme: Internet-facing
5. Listeners: HTTP (80)
6. Select two public subnets
7. Security Group: allow HTTP (port 80)
8. Forward to Target Group → WebApp-TG
9. Create

Copy the ALB DNS name for testing later.

STEP 6 — Create Auto Scaling Group (ASG)

6A. Create Launch Template

1. EC2 → Launch Templates → Create
2. Name: WebServer-LT
3. AMI: WebServer-AMI

4. Instance type: t3.micro
5. Security Group: allow HTTP + SSH
6. Create

6B. Create Auto Scaling Group

1. Go to Auto Scaling → Create Auto Scaling Group
2. Name: WebApp-ASG
3. Select Launch Template: WebServer-LT
4. Choose VPC + two public subnets
5. Load Balancing:
    - Attach to existing ALB
    - Choose WebApp-TG
6. Group size:
    - Desired: 1
    - Min: 1
    - Max: 3
7. Create

STEP 7 — Configure Auto Scaling Policies

Go to ASG → Automatic Scaling → Add Policy

Scale-Out Policy

- Metric: CPU Utilization
- Rule: > 60% for 2 minutes
- Action: Add +1 instance

Scale-In Policy

- Rule: < 20% for 5 minutes
- Action: Remove 1 instance

STEP 8 — Stress Test to Trigger Scaling

SSH into your running instance:
Install stress:
sudo amazon-linux-extras install epel -y
sudo yum install stress -y
Run CPU load:
stress --cpu 90 --timeout 300
Within 2–3 minutes:

- CPU becomes high
- ASG launches second instance
- Target Group shows both instances as healthy

STEP 9 — Test Load Balancing

Open ALB DNS Name in browser: http://<alb-dns-name>
Refresh many times — you will see:

Hello from Instance 1 — ip-XX-XX…
Hello from Instance 2 — ip-YY-YY…

- Confirms ALB working
- Confirms auto scaling working

After stress test ends → ASG scales back to 1 instance.

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.