Assignment No: 1

Title: Configuring IAM Roles and Policies in AWS

Aim: To create and assign IAM roles with specific access policies in AWS to securely manage permissions for users and cloud resources, and verify access control using an EC2 instance.

Objectives:

1. To understand the concept of Identity and Access Management (IAM) in cloud security.
2. To learn how to create and configure IAM roles and policies in AWS.
3. To implement secure role-based access control for cloud resources like EC2 and S3.
4. To verify and demonstrate restricted access permissions using assigned IAM roles.
5. To apply the principle of least privilege for enhancing cloud resource security.

Tools / Environment Required:
AWS Management Console, AWS Identity and Access Management (IAM) Service, Amazon EC2, Amazon S3, Web Browser (Google Chrome / Edge), AWS CLI (optional)

Theory:

In cloud computing, Identity and Access Management (IAM) is a crucial security framework that defines and manages the roles and access privileges of users and services within a cloud environment. It ensures that only authorized entities can access specific resources, maintaining confidentiality, integrity, and availability of data.

IAM allows organizations to control who can access cloud resources, what actions they can perform, and under what conditions. It provides centralized management of permissions through roles, users, groups, and policies.

An IAM Role is a set of permissions that define what actions are allowed or denied on which resources. Roles are assigned to trusted entities such as users, applications, or AWS services (e.g., EC2). A policy is a document that defines these permissions in a structured format and helps enforce access control.

By configuring IAM roles and policies, cloud administrators implement the principle of least privilege, granting only the minimum permissions required to perform specific tasks. This reduces security risks, prevents unauthorized access, and ensures a more secure and manageable cloud infrastructure.

IAM is a key component of cloud security as it helps protect sensitive data, supports compliance with security standards, and provides detailed monitoring and auditing of user activities through services like AWS CloudTrail.

Requirements

- Active **AWS** or **Azure** account with administrative access.
- Access to the **IAM Console** (AWS) or **Azure Active Directory** (Azure).
- Stable internet connection and web browser.

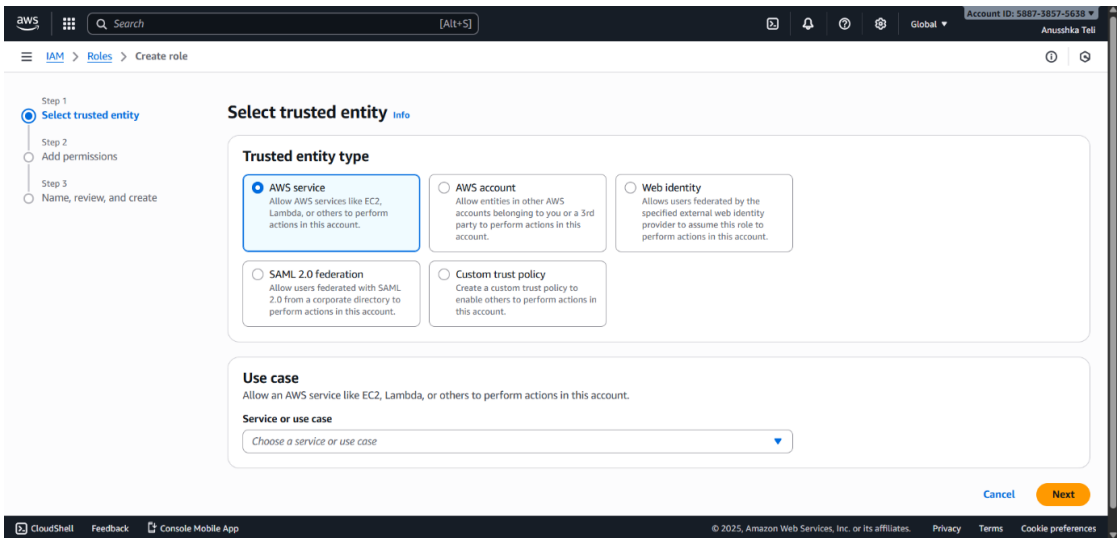
Steps / Procedure

Step 1: Log in to the Cloud Console

- Sign in to your **AWS Management Console** or **Microsoft Azure Portal** using admin credentials.

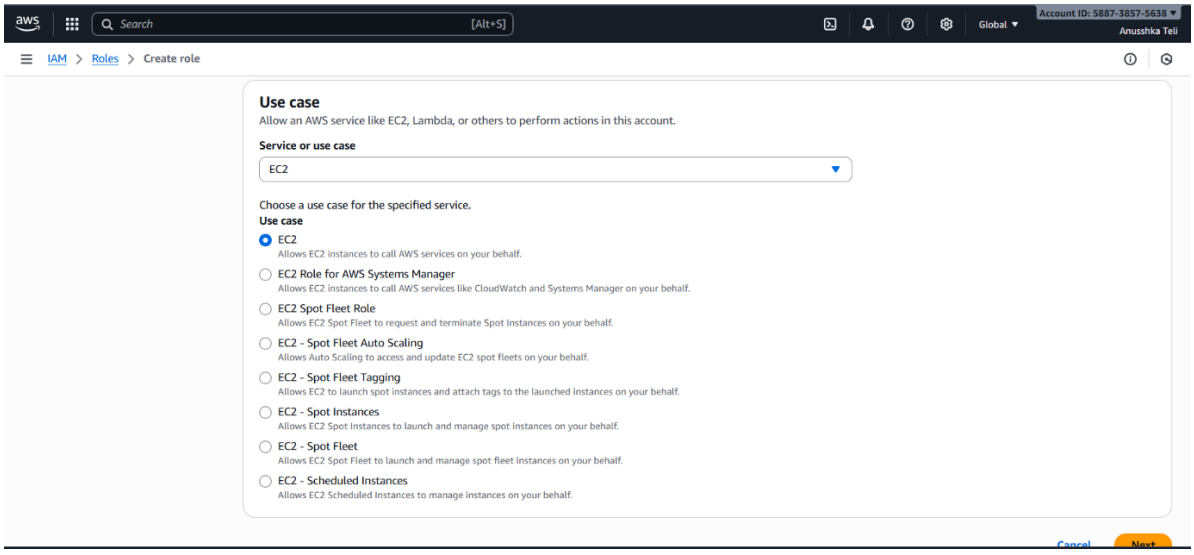
Step 2: Navigate to the IAM Section

- **AWS:** Go to **Services → IAM (Identity and Access Management)**.
- **Azure:** Go to **Azure Active Directory → Roles and Administrators**.



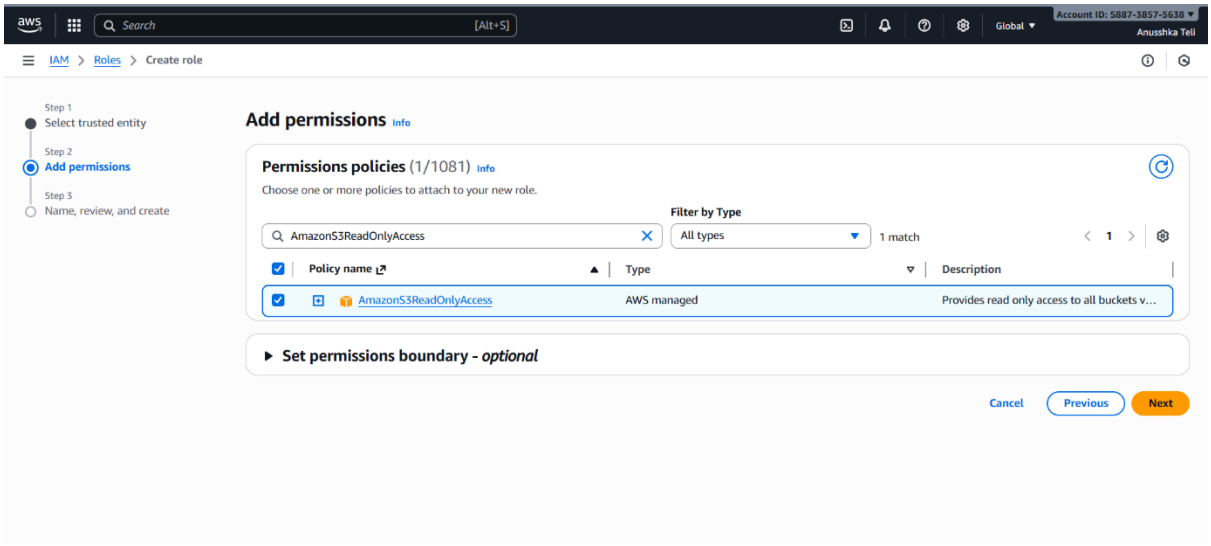
Step 3: Create a New Role

1. Click **Create Role** (AWS) or **Add Custom Role** (Azure).
2. Select the **trusted entity** (e.g., AWS service, another AWS account, or Azure service).
3. Choose a **use case** such as EC2, S3, or any service that will assume the role.



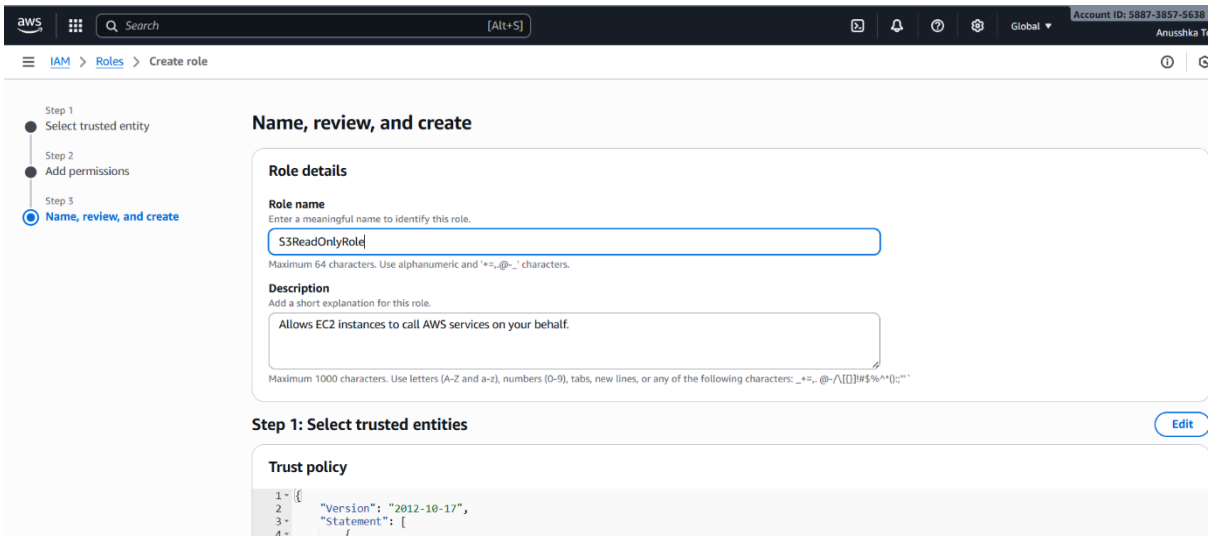
Step 4: Attach Policies

- Choose from **predefined policies** (e.g., AmazonS3ReadOnlyAccess) or create a **custom policy** using a JSON document.
- Review permissions carefully before attaching.



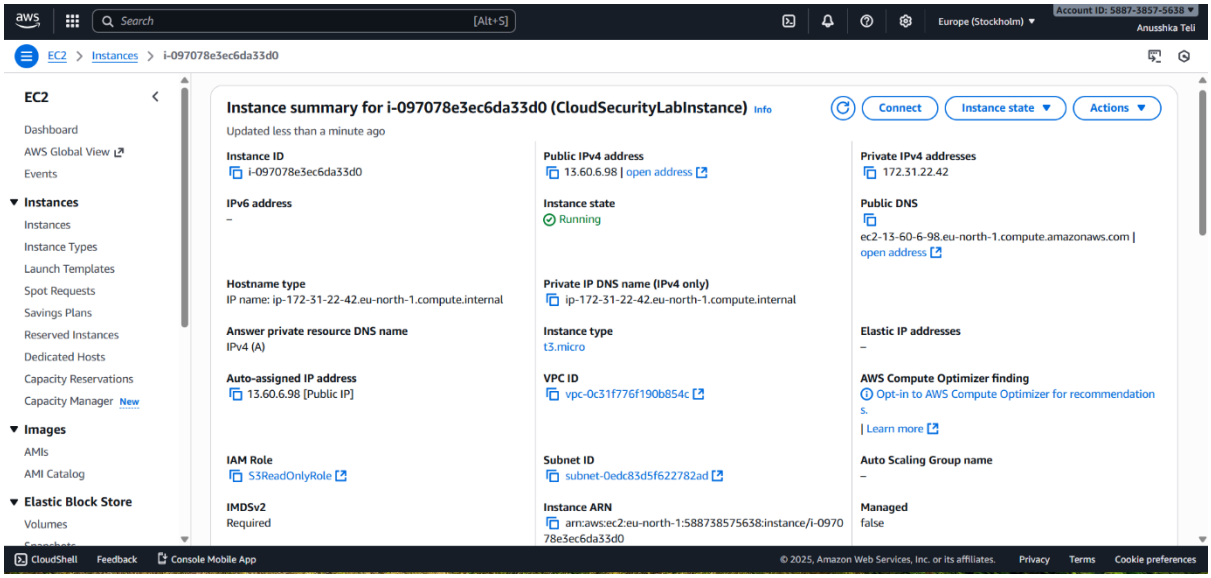
Step 5: Review and Create the Role

- Enter a **Role Name** and description.
- Review all settings and click **Create Role**.



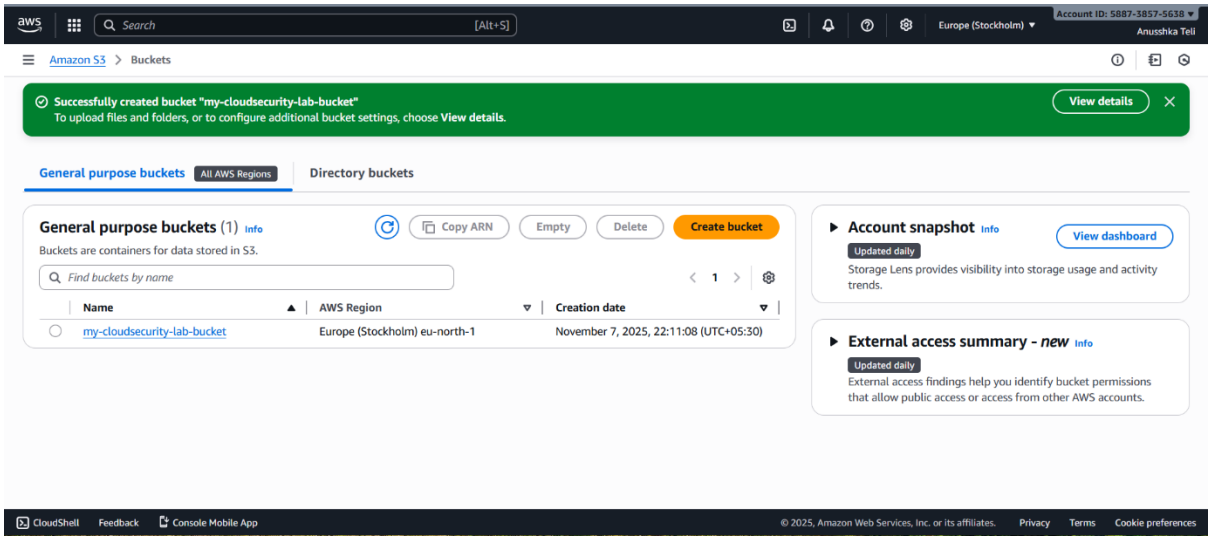
Step 6: Assign the Role

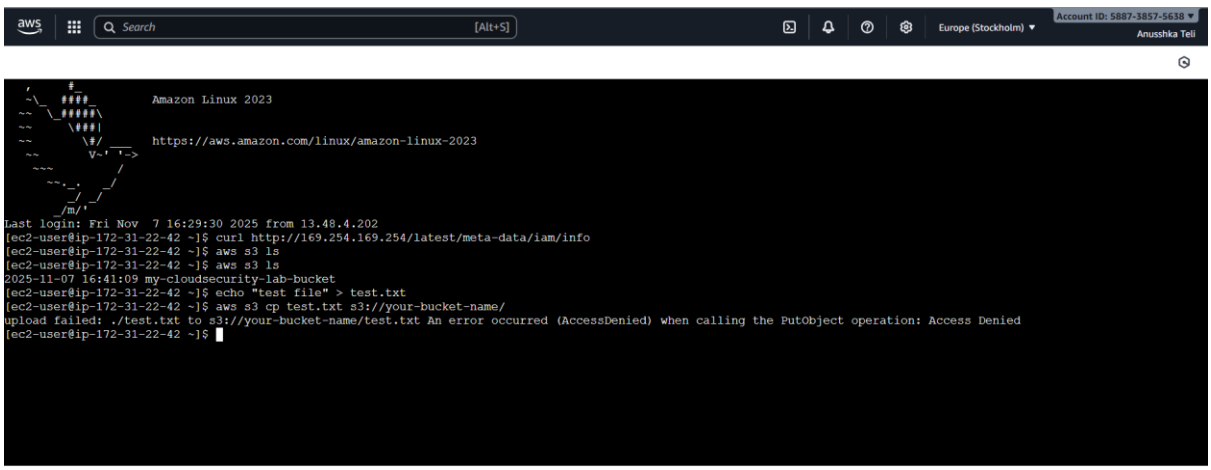
- Assign the newly created role to:
  - **AWS:** A user, group, or resource (like EC2 instance).
  - **Azure:** A user, group, or application under **Access Control (IAM)**.



Step 7: Test Permissions

- Log in using the assigned role credentials.
- Attempt to access cloud services to confirm that permissions are applied correctly and securely.





Conclusion:

In this practical, we created and assigned an IAM role with a specific S3 read-only policy to an EC2 instance. The instance successfully accessed S3 buckets without using credentials, proving secure role-based access. The setup demonstrated the principle of least privilege and highlighted how IAM roles enhance cloud security by controlling permissions efficiently.

Assignment No: 2

Title: Implementation of the shared responsibility model using a cloud provider.

Aim: To understand and implement the Shared Responsibility Model in AWS by deploying and securing a cloud resource.

Objectives:

1. To study AWS's Shared Responsibility Model.
2. To identify AWS vs. customer responsibilities.
3. To deploy a cloud resource (EC2 instance) as a customer-managed component.
4. To apply basic security configurations.

Tools / Environment Required:

1. Active AWS Account (with sufficient permissions)
2. Internet connection (stable broadband for AWS Console access)
3. Web browser (Google Chrome / Edge / Firefox – latest version recommended)
4. Access to AWS Management ConsoleAccess to the following AWS services:
    - Amazon EC2 – to deploy a virtual server (IaaS resource)
    - AWS IAM (Identity and Access Management) – to manage users, roles, and permissions
    - Amazon VPC (Virtual Private Cloud) – for network and security configuration
    - AWS CloudWatch (optional) – to monitor logs and events
5. Key Pair (.pem file) – required for secure SSH access to the EC2 instance

Theory:

In cloud computing, the Shared Responsibility Model is a foundational concept that clearly defines the division of security duties between the Cloud Service Provider (CSP) and the customer. It ensures that both parties understand and fulfill their respective roles in protecting cloud infrastructure, applications, and data.

Under this model, AWS (Amazon Web Services) is responsible for security *of* the cloud, while the customer is responsible for security *in* the cloud.

- Security of the Cloud:
  AWS manages the protection of the underlying infrastructure that runs all of its services. This includes physical facilities, hardware, software, networking, and virtualization layers. AWS ensures that its global data centers are secure, resilient, and compliant with international security standards such as ISO 27001 and SOC 2.
- Security in the Cloud:
  Customers are responsible for securing the data, applications, and configurations they deploy within AWS. This involves managing identity and access through IAM, applying encryption for stored and transmitted data, configuring network firewalls, setting correct permissions, and regularly monitoring resources.

The model varies slightly depending on the service model used:

- In Infrastructure as a Service (IaaS), customers have greater control and therefore more responsibility for operating systems, network configurations, and data protection.
- In Platform as a Service (PaaS), AWS manages more layers (like OS and middleware), reducing the customer's burden.
- In Software as a Service (SaaS), AWS handles nearly all security aspects except user data and access control.

The Shared Responsibility Model is critical because it prevents security gaps by ensuring no area is left unattended. Misconfigurations by customers, such as open ports or weak IAM policies, can lead to vulnerabilities even when AWS infrastructure remains secure. Therefore, understanding this model helps users design safer cloud environments and comply with organizational and regulatory security standards.

Steps / Procedure

Step 1: Review AWS Shared Responsibility Documentation

1. Go to AWS Shared Responsibility Model page.
2. Read the overview to understand:
    - AWS is responsible **for security of the cloud**.
    - Customer is responsible **for security in the cloud**.

Step 2: Identify Responsibility Areas

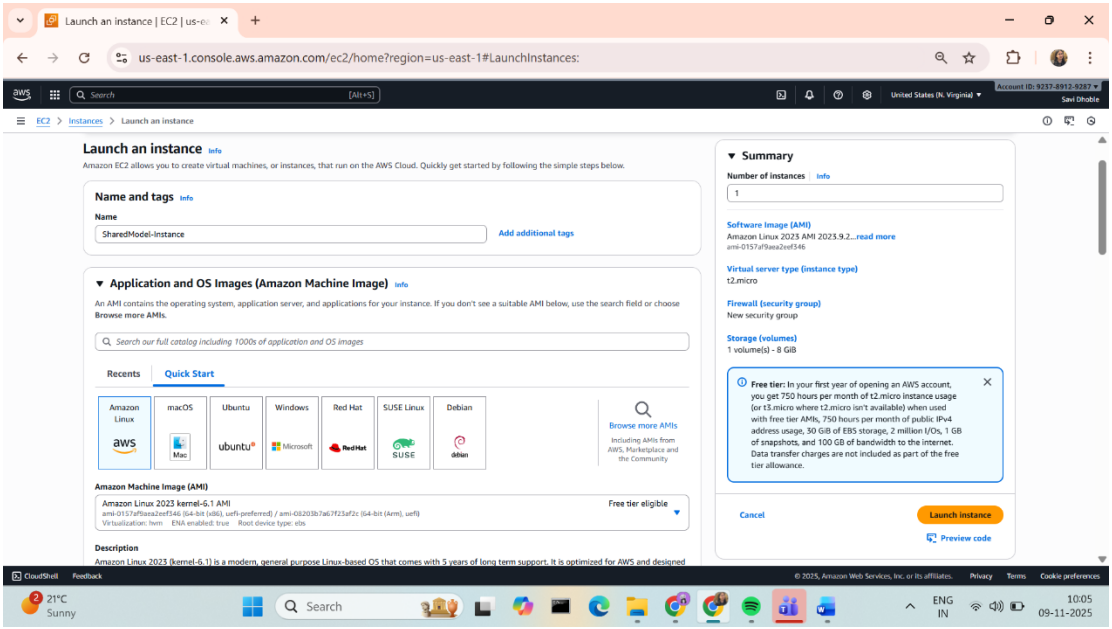
Write down a few examples (for your record):

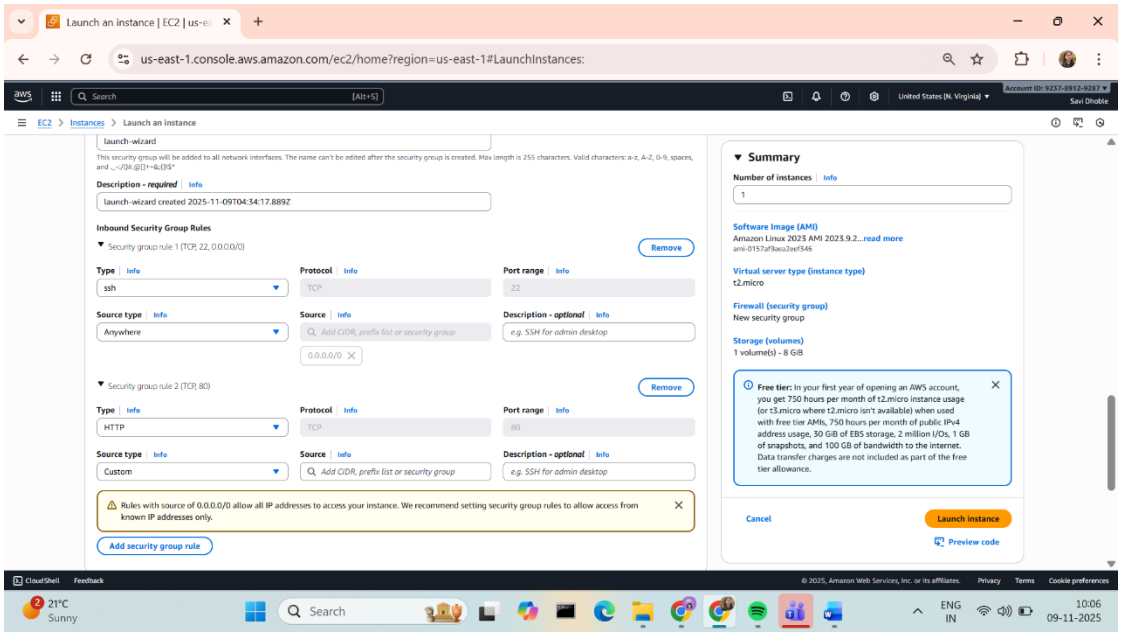
- **AWS Responsibilities:**
    - Physical security of data centres.
    - Network hardware and virtualization layer
- **Your Responsibilities:**
    - Configuring firewall rules
    - Managing IAM users and policies
    - Encrypting stored data

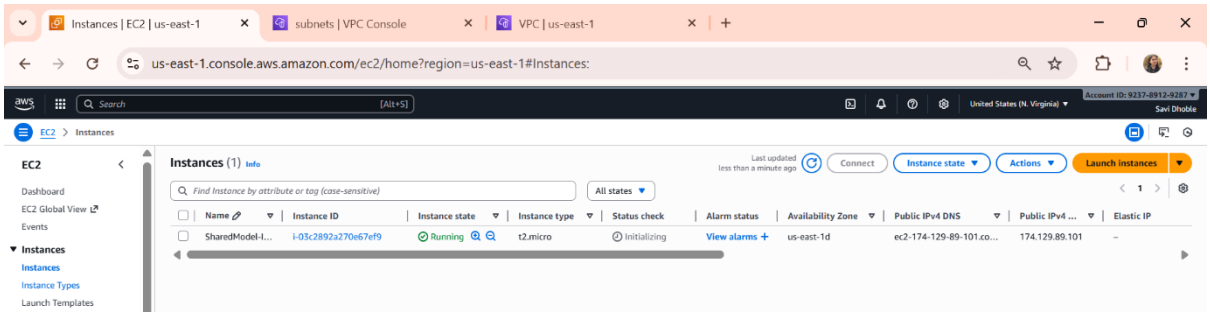
Step 3: Deploy a Simple EC2 Instance (Customer Resource)

1. Go to **AWS Console → EC2 → Launch Instance**.
2. Name: Shared Model-Instance
3. AMI: Amazon Linux 2 (Free tier eligible).

4. Instance type: t2.micro
5. Key Pair: Create or select an existing one.
6. Network settings:
    - Select **default VPC**.
    - Create a **new Security Group** → Allow SSH (port 22).
7. Launch the instance.



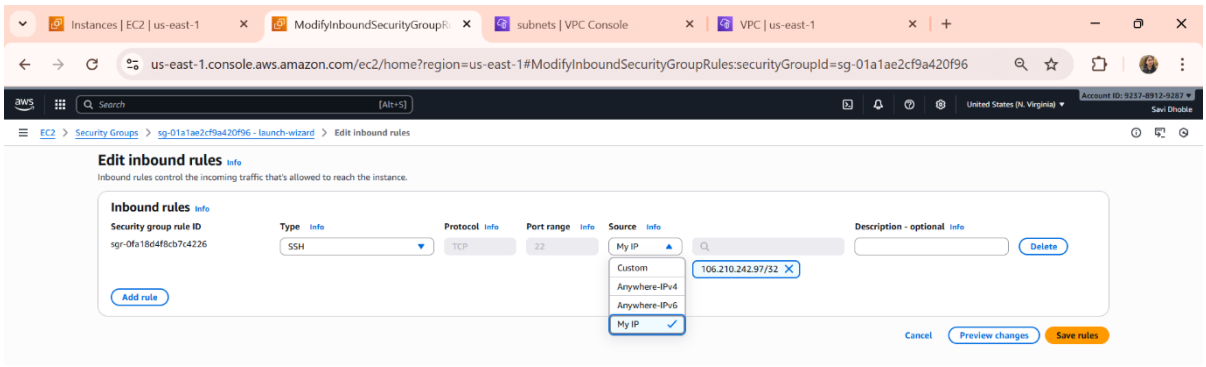




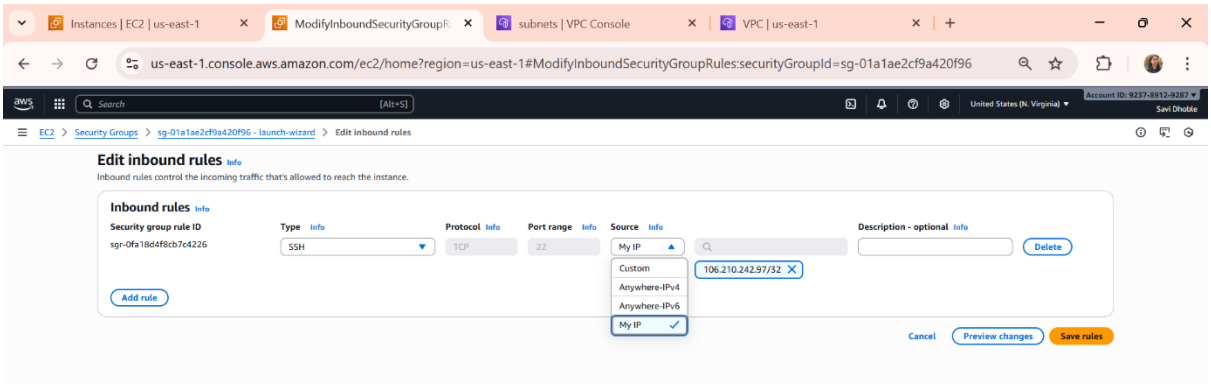
Step 4: Configure Security Controls

Here you'll apply customer-side security measures.

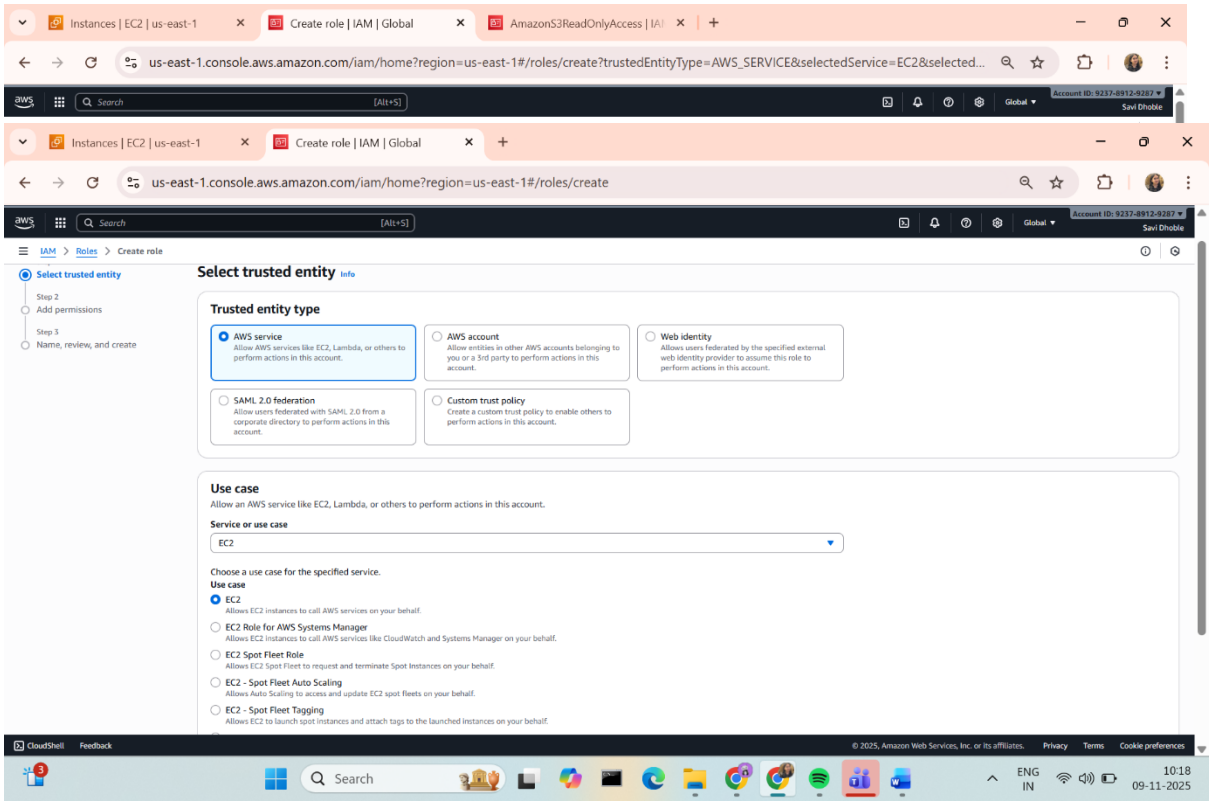
1. Go to your **EC2 → Security → Security Groups → Edit inbound rules.**
2. Modify the rules:
    - Allow SSH only from your IP (instead of 0.0.0.0/0).
    - Remove unused ports.

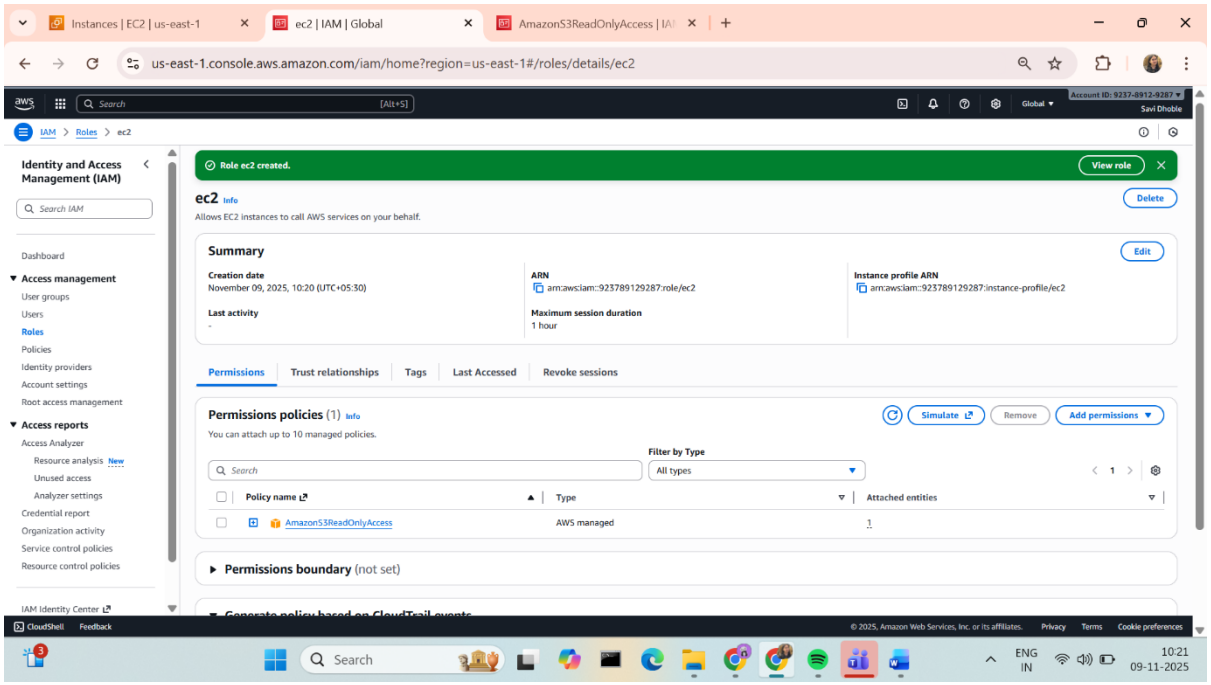


3. Enable **IAM Role** for EC2:
    - Go to **IAM → Roles → Create Role**.
    - Choose **AWS Service → EC2**.
    - Attach policy: AmazonS3ReadOnlyAccess.
    - Attach role to your instance.



4. This ensures **principle of least privilege** and data access control.





Conclusion:

Through this practical, we understood how security responsibilities are divided between AWS and the customer. AWS secures the underlying cloud infrastructure, while customers must manage access control, configurations, and data protection. By applying least privilege principles and proper network rules, we achieved a secure and compliant cloud setup.

Assignment No: 3

Title: Implementation of Multi-Factor Authentication (MFA) for Cloud Access

Aim: To enable Multi-Factor Authentication (MFA) for cloud user accounts in order to enhance the security of cloud access.

Objectives:

1. To understand the concept of Multi-Factor Authentication (MFA).
2. To configure MFA for a cloud user account.
3. To secure administrative and critical user accounts using an additional authentication layer.
4. To verify user login with MFA-enabled access.

Tools / Environment Required:

1. Cloud service provider account (e.g., AWS, Azure, or Google Cloud).
2. Web browser (e.g., Google Chrome).
3. Mobile phone with an authenticator app (Google Authenticator, Authy, or Microsoft Authenticator).
4. Stable Internet connection.

Theory:

Multi-Factor Authentication (MFA) is a security mechanism that requires users to verify their identity using **two or more factors** before granting access to an account or system.

It combines:

1. **Something you know:** (Password or PIN)
2. **Something you have:** (Mobile phone, security token)
3. **Something you are:** (Biometric verification such as fingerprint or face ID)

When enabled, MFA adds an **additional security layer** to protect against unauthorized access even if the password is compromised.

In cloud platforms like AWS, Azure, or GCP, MFA is essential for **Identity and Access Management (IAM)** users, especially for those with administrative privileges.

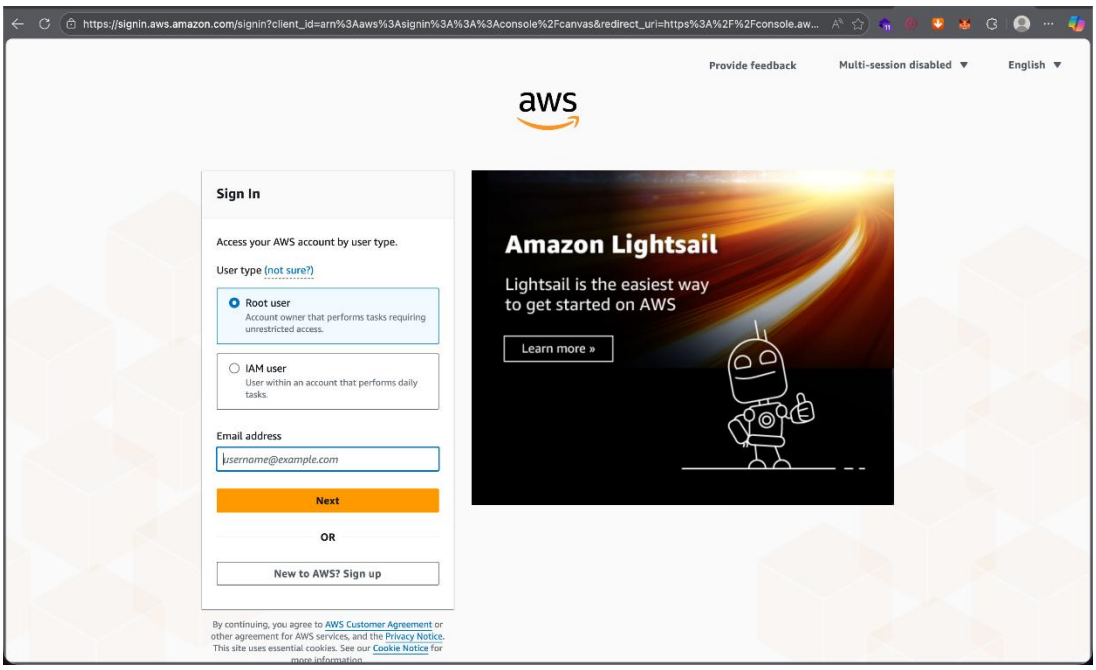
Benefits of MFA:

- Prevents unauthorized access due to stolen credentials.
- Provides strong protection for sensitive data and configurations.
- Meets compliance and security best practices.

Steps / Procedure:

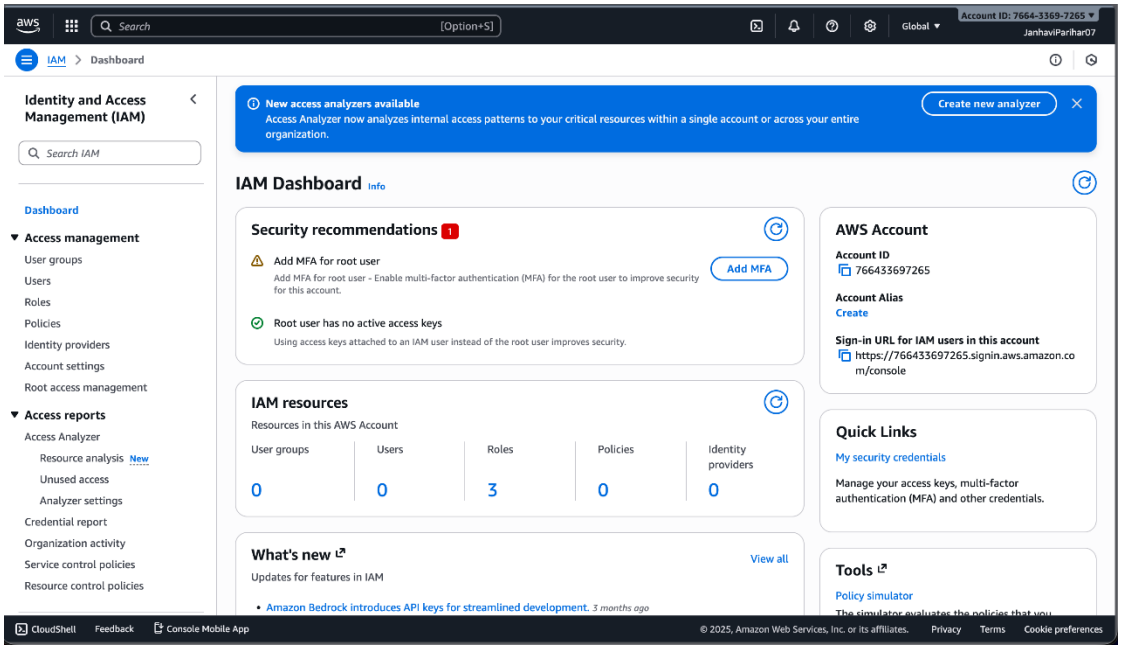
Step 1: Log in to your Cloud Dashboard

- Go to your cloud provider's login page (e.g., https://console.aws.amazon.com/ or https://portal.azure.com/)
- Enter your username and password to access the management console.



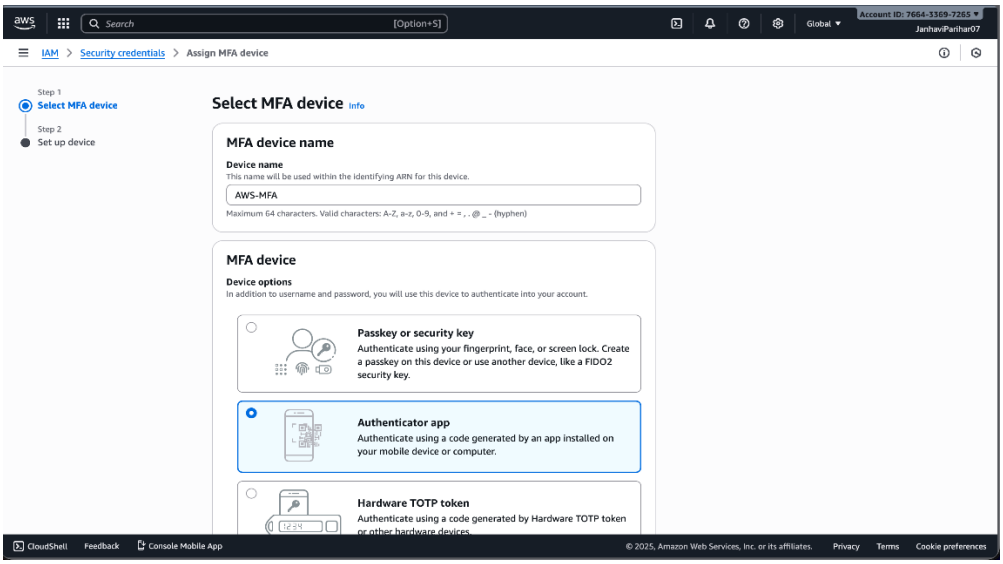
Step 2: Navigate to Security Settings / IAM

- From the dashboard, open the **IAM (Identity and Access Management)** or **Security Settings** section.
- This section manages users, roles, and authentication options.



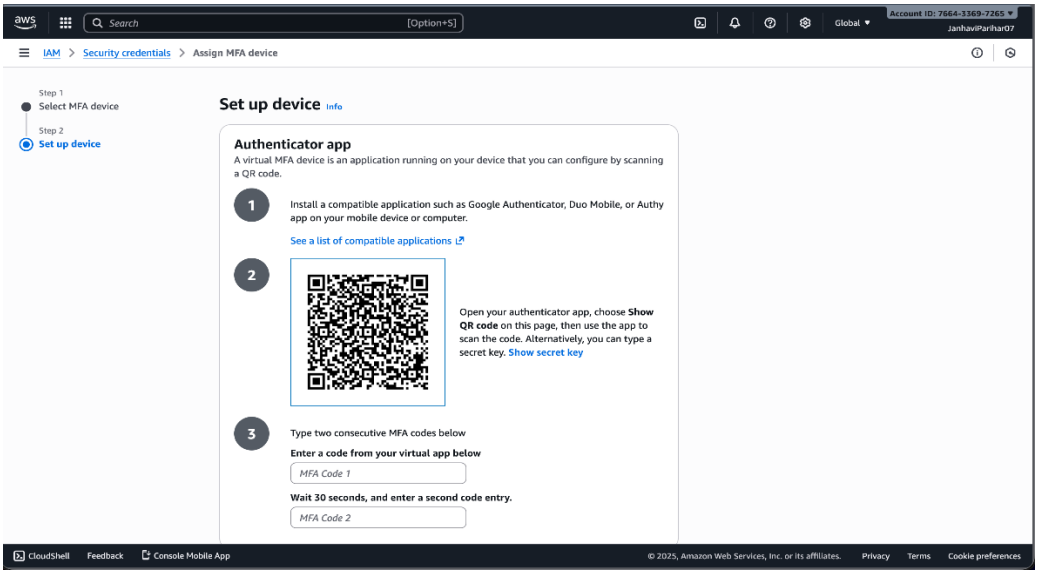
Step 3: Select MFA Setup (AWS Console)

1. In the AWS Management Console, go to the top right corner and click on your account name (or the username if logged in as an IAM user).
2. Choose “Security credentials” from the dropdown menu.
3. Scroll down to the section titled “Multi-Factor Authentication (MFA)”.
4. Click on “Assign MFA device” (or “Manage MFA” if one was already set before).
5. A setup wizard will open asking you to select the MFA device type.
    - Choose “Authenticator app” (for Google Authenticator, Authy, etc.)
    - Then click “Next” to proceed.



Step 4: Configure MFA Device (Authenticator App)

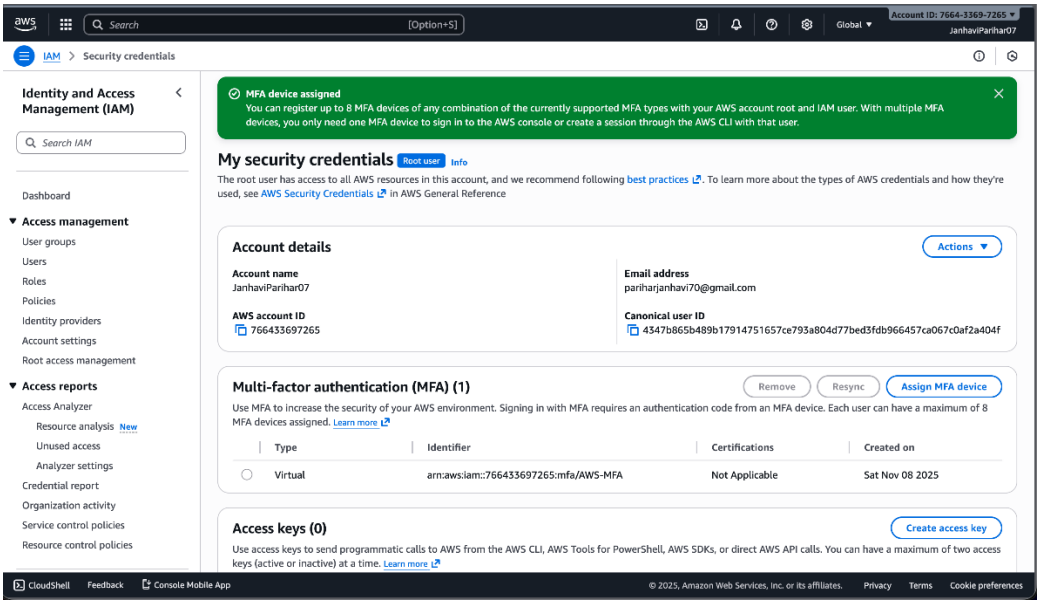
1. On your computer screen, AWS will display a QR code.
2. On your phone, open your Google Authenticator (or any other authenticator app).
3. Tap “+” → “Scan a QR code” and scan the code shown on AWS.
4. The app will now generate a 6-digit code linked to your AWS account.

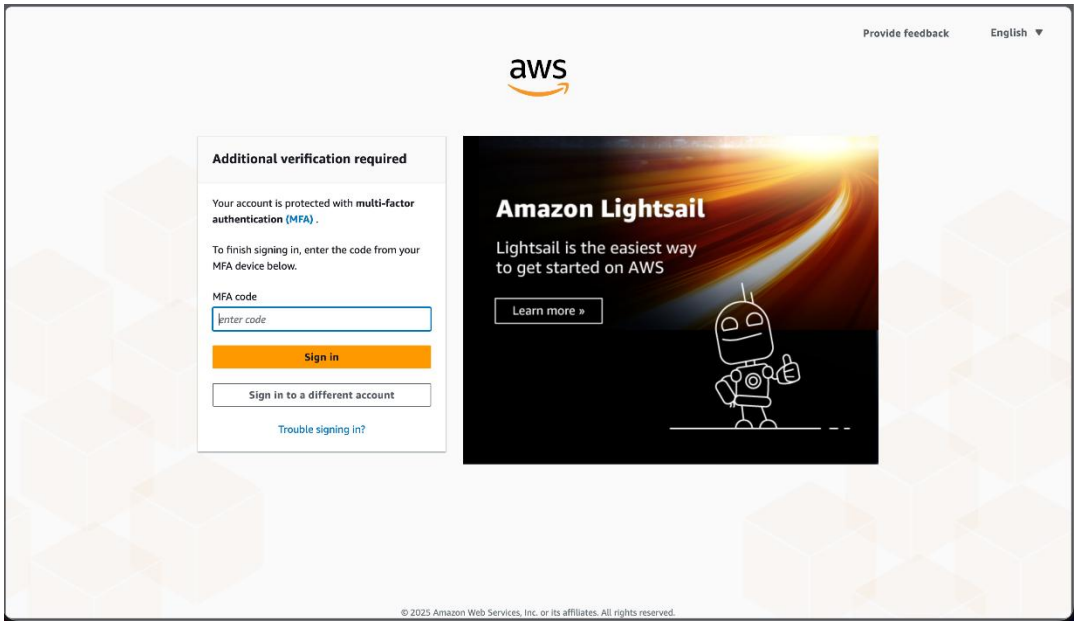


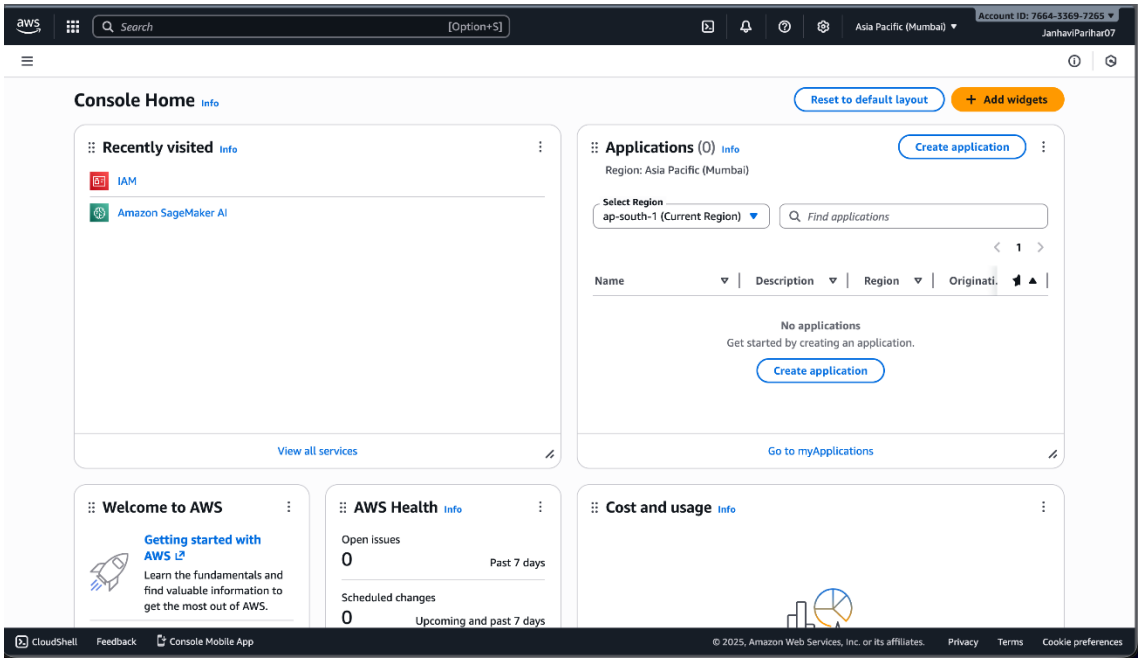
Step 5: Verify MFA Setup

1. AWS will ask you to enter two consecutive codes from the authenticator app.
    a. Enter the first 6-digit code displayed in the app.
    b. Wait for it to refresh, then enter the second code.
2. Click “Add MFA” or “Assign MFA” to complete setup.
3. You should now see a message:

“Virtual MFA device successfully assigned to your account.”







Conclusion:

The implementation of **Multi-Factor Authentication (MFA)** successfully enhances the security of cloud accounts by adding an extra layer of protection beyond passwords.
It helps prevent unauthorized access and secures critical cloud resources from potential attacks.

Assignment No: 4

Title: Implementation of Data Encryption at Rest and in Transit on AWS

Aim: To apply encryption mechanisms for protecting data stored in AWS S3 and transmitted over HTTPS/TLS.

Objectives:

1. To understand the concept of encryption at rest and in transit.
2. To enable server-side encryption on AWS S3 using AES-256.
3. To configure secure HTTPS/TLS connections for encrypted data transmission.
4. To verify the encryption and secure connection using command-line tools.

Tools / Environment Required:

1. AWS Management Console (active AWS account)
2. Amazon S3 service
3. Web browser (e.g., Google Chrome)
4. Command Prompt / Terminal with curl
5. Stable Internet connection

Theory:

Data encryption is a core cybersecurity mechanism that ensures the confidentiality and integrity of data both when it is stored ("at rest") and when it is transmitted ("in transit").

1. Encryption at Rest:
Data is encrypted before being stored in the cloud to protect it from unauthorized access. AWS S3 supports Server-Side Encryption (SSE) using:

- SSE-S3 (AES-256): Amazon manages the encryption keys.
- SSE-KMS: Uses AWS Key Management Service (KMS) for key handling and audit control.

2. Encryption in Transit:
Data in transit is protected by using secure protocols such as HTTPS/TLS, which encrypts the data between client and server during transmission.

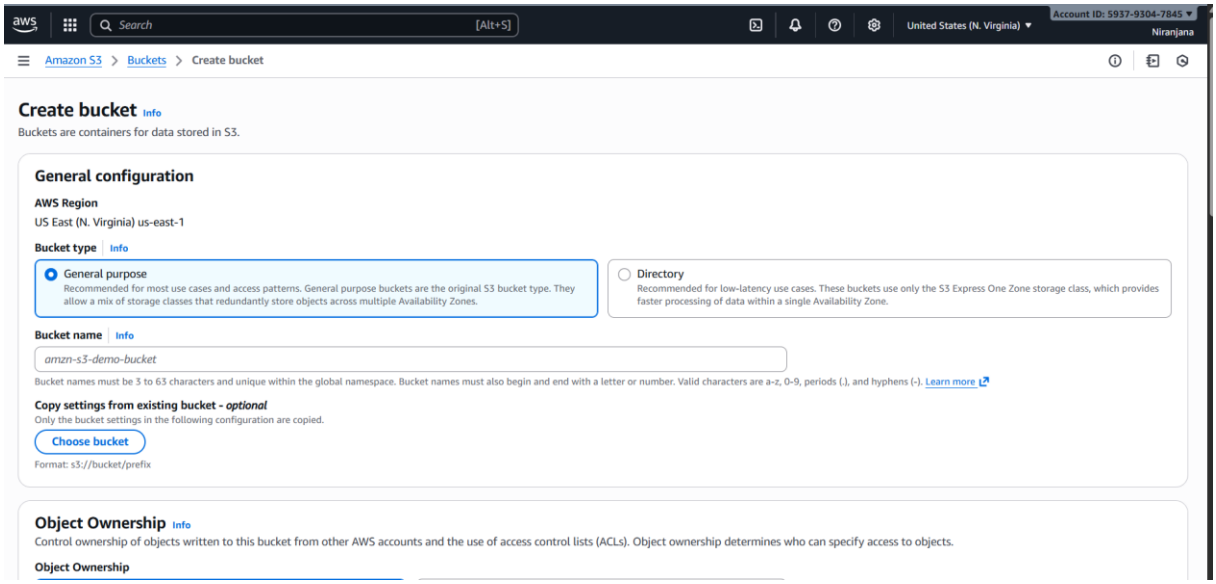
Benefits:

- Protects sensitive data from unauthorized access.
- Ensures compliance with cloud security standards.
- Maintains data confidentiality during storage and transfer.

Steps / Procedure:

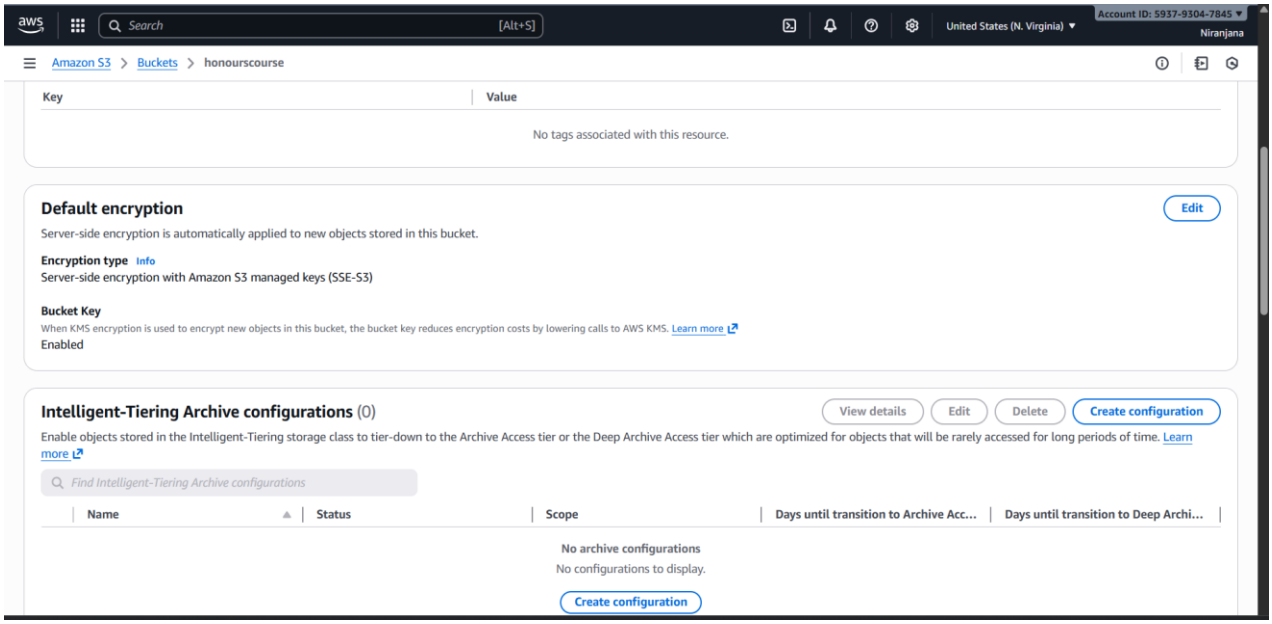
Step 1: Create an S3 Bucket

1. Log in to the **AWS Management Console**.
2. Navigate to **S3 → Create bucket**.
3. Enter a unique bucket name and select your preferred region.
4. Leave other settings as default and click **Create bucket.**



Step 2: Enable Server-Side Encryption (At Rest)

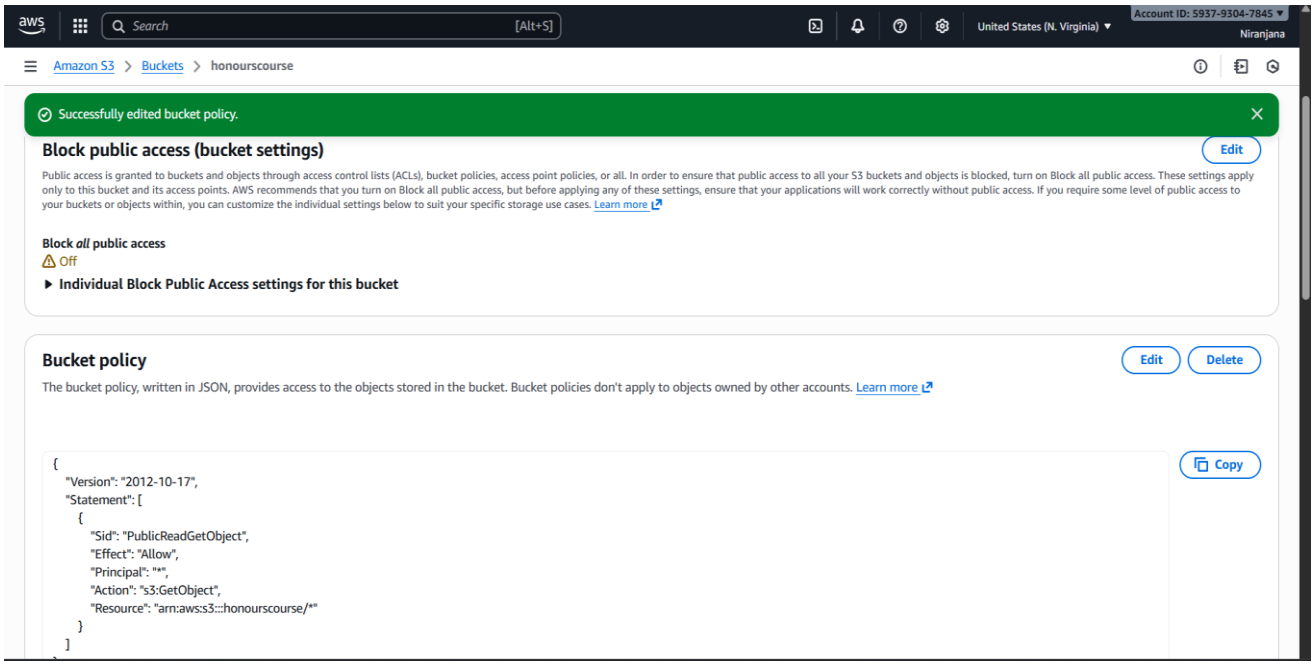
1. Go to the **Properties** tab of your S3 bucket and scroll to **Default encryption**.
2. Select **Enable** and choose **AES-256 (SSE-S3)** for encryption.
3. Save changes.
    a. This ensures all files uploaded to this bucket are automatically encrypted using AES-256.



Step 3: Allow Temporary Public Access

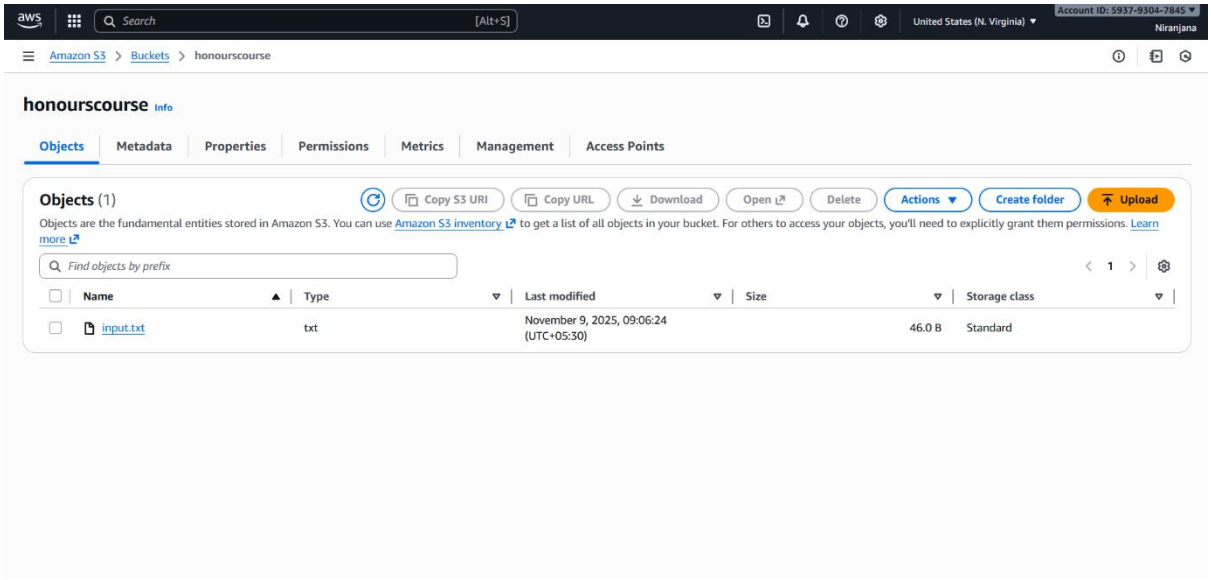
- Go to **Permissions → Block Public Access** and temporarily disable restrictions.
- Under **Bucket Policy**, paste the following JSON policy (replace your-bucket-name):

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



Step 4: Upload and Validate Encryption

- Upload a test file (e.g., input.txt) to your S3 bucket.
- Open the uploaded file's **Properties** → Check **Encryption**; it should show **AES-256**.



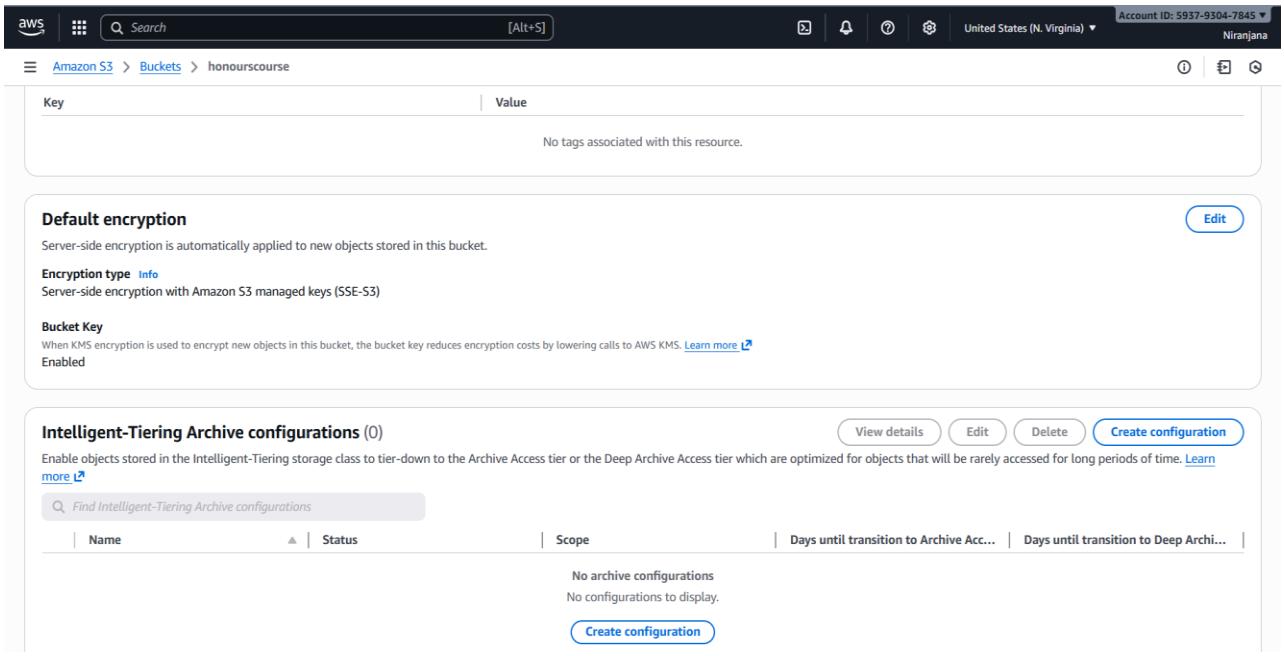
Step 5: Verify Secure Data Transmission (In Transit)

- Open **Command Prompt** or **Terminal**.
- Run the command:
  - curl -I https://your-bucket-name.s3.amazonaws.com/input.txt
- The output should include:
  - HTTP/1.1 200 OK
  - x-amz-server-side-encryption: AES256
- HTTP/1.1 200 OK → Confirms successful HTTPS connection.
- x-amz-server-side-encryption: AES256 → Confirms encryption at rest.

```
C:\Users\niran>curl -I https://honourscourse.s3.amazonaws.com/input.txt
HTTP/1.1 200 OK
x-amz-id-2: T2jGAbhe31NDmXMqUb2RyAiq5vOBf+gfnQzbfJVdrgbcL+FJBAIUJOIWEiXOe8GNf1A8tIqOVVM=
x-amz-request-id: 7NNJF8MANT6T63M3
Date: Sun, 09 Nov 2025 03:42:32 GMT
Last-Modified: Sun, 09 Nov 2025 03:36:24 GMT
ETag: "275adb3ed717c1968ad91da0717bac4e"
x-amz-server-side-encryption: AES256
Accept-Ranges: bytes
Content-Type: text/plain
Content-Length: 46
Server: AmazonS3
```

The test file stored in AWS S3 was successfully encrypted using **AES-256** encryption.
Data transmitted using the curl command confirmed a secure **HTTPS/TLS** connection.
The following response header was observed:

- x-amz-server-side-encryption: AES256



Conclusion:

The experiment was successfully performed.
Data encryption at rest was implemented using **AES-256 server-side encryption (SSE-S3)**, and encryption in transit was verified through **HTTPS/TLS**.
This ensures that data remains protected both while stored in the cloud and during transmission, meeting essential cloud security standards.

Assignment No: 5

Title: Setting up and Using AWS Key Management Service (KMS) for Secure Key Creation, Management, Rotation, and Auditing

Aim: To create, manage, and use cryptographic keys in AWS Key Management Service (KMS) for secure data encryption and decryption, with proper access controls, key rotation, and audit tracking.

Objectives:

1. Create a **customer-managed key (CMK)** using AWS KMS.
2. Define **key usage policies** and **access permissions**.
3. Enable **automatic key rotation** for enhanced security.
4. Use the key to **encrypt and decrypt data** within AWS via Secret key Manager
5. Track **key usage activities** through AWS CloudTrail logs.

Tools / Environment Required:

1. AWS Account with IAM user having **Administrator Access** or **KMS permissions**.
2. AWS Management Console (web interface).
3. CloudTrail service enabled for logging.
4. Internet connectivity.

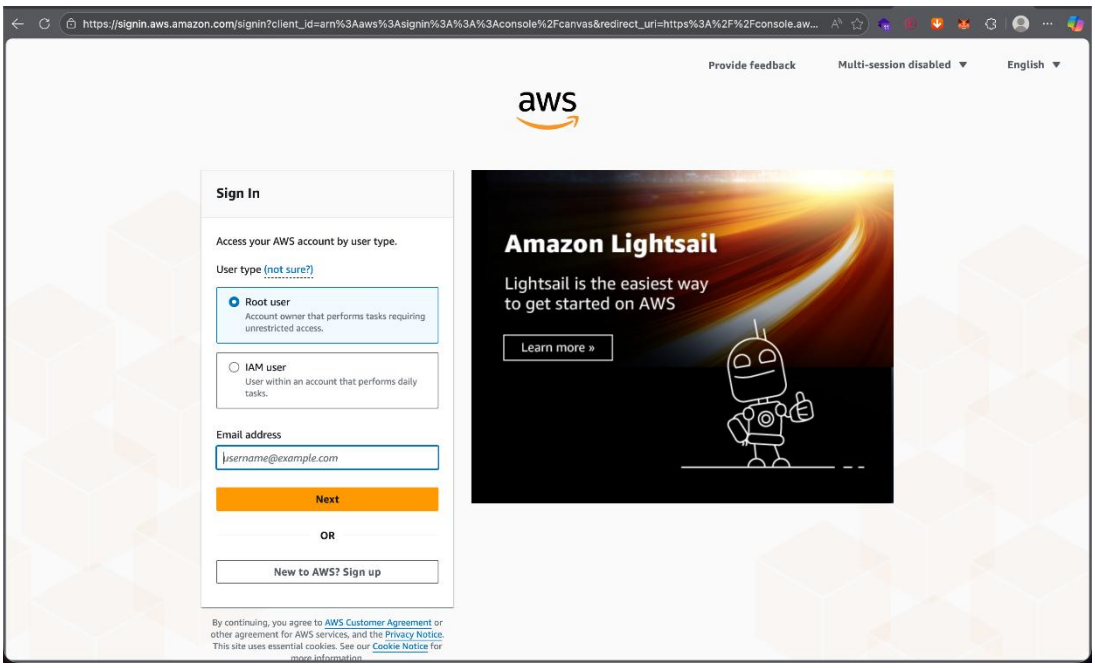
Theory:

1. **AWS Key Management Service (KMS)** is a managed service that allows you to create and control cryptographic keys used to protect your data.
2. **Customer Managed Keys (CMKs)** are keys you create, own, and manage within your AWS account.
3. Each KMS key has a **key policy** that defines who can administer or use it.
4. **Key rotation** helps automatically change key material at regular intervals (typically every 365 days) to enhance security.
5. **CloudTrail** automatically records KMS API operations such as *CreateKey*, *Encrypt*, *Decrypt*, and *GenerateDataKey*, allowing you to monitor and audit key usage.

Steps / Procedure:

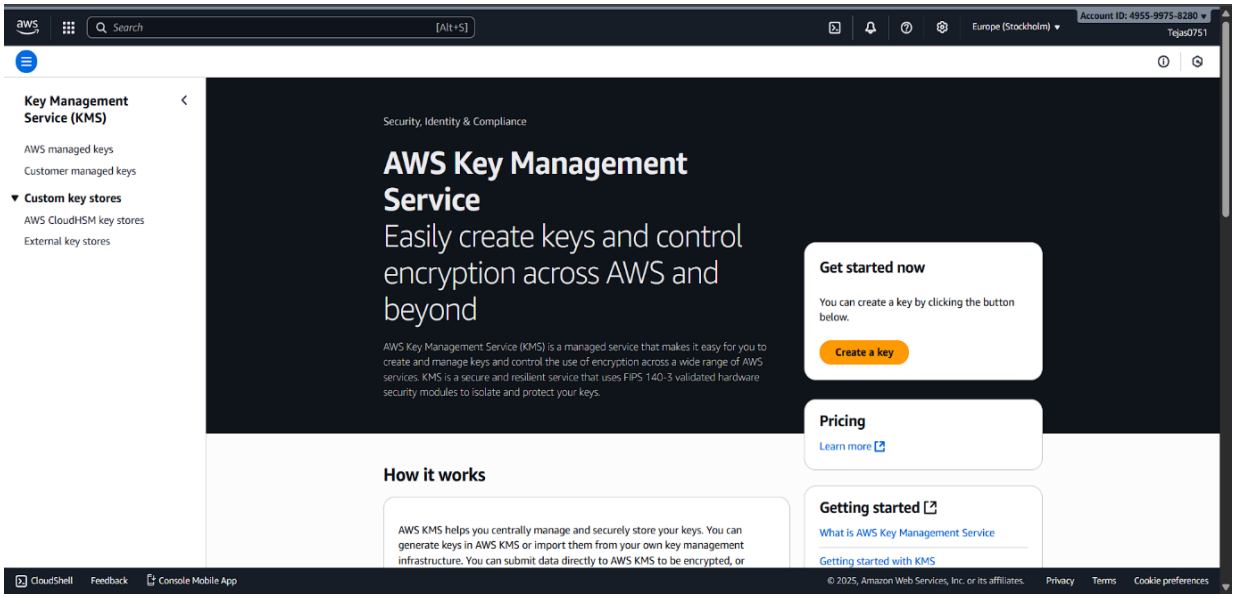
Step 1: Log in to your Cloud Dashboard

- Go to your cloud provider's login page (e.g., https://console.aws.amazon.com/ or https://portal.azure.com/)
- Enter your username and password to access the management console.



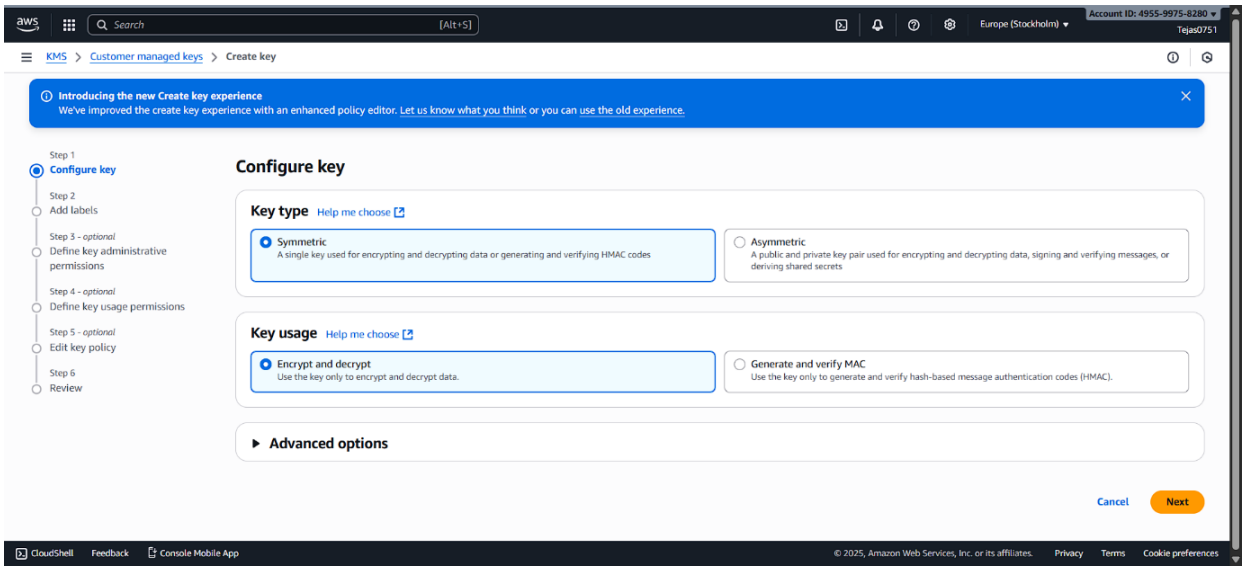
Step 2: Open AWS KMS

1. Sign in to your AWS Management Console.
2. In the search bar, type **"KMS"** and open **Key Management Service**.
3. Select the desired **AWS region** from the top-right corner.



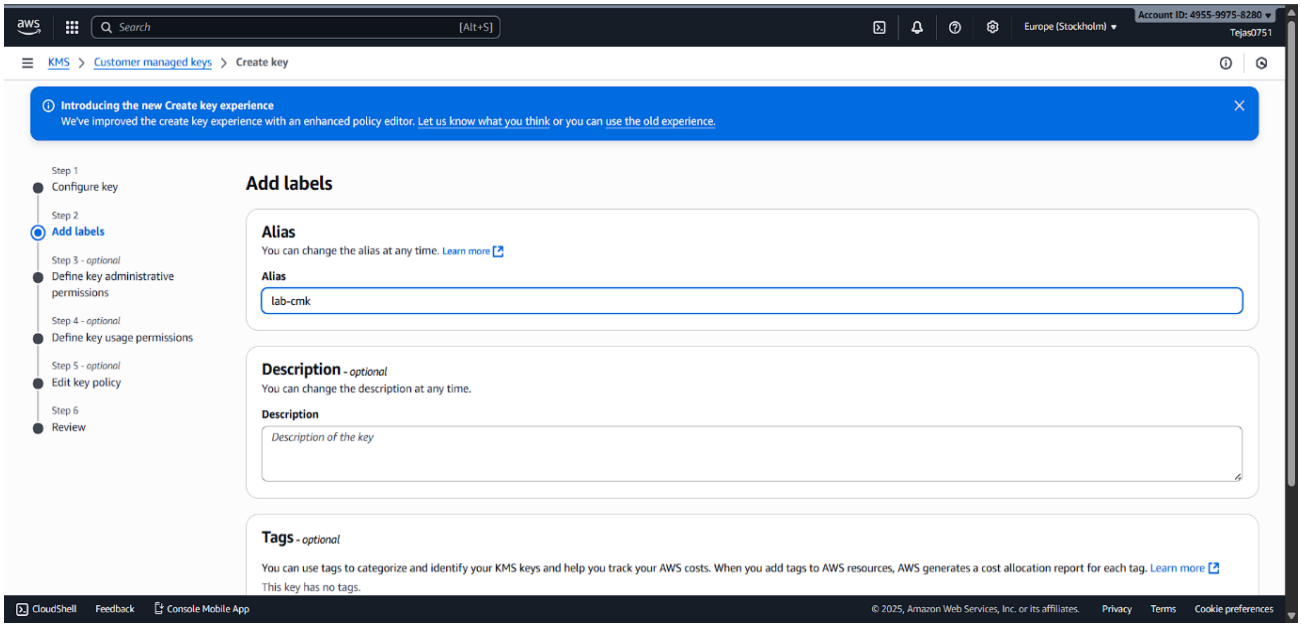
Step 3: Create a Customer Managed Key (CMK)

1. In the left menu, click **Customer managed keys**.
2. Click **Create key**.
3. Choose **Key type:** *Symmetric* (commonly used for data encryption).
4. Choose **Key usage:** *Encrypt and decrypt*.
5. Click **Next**.



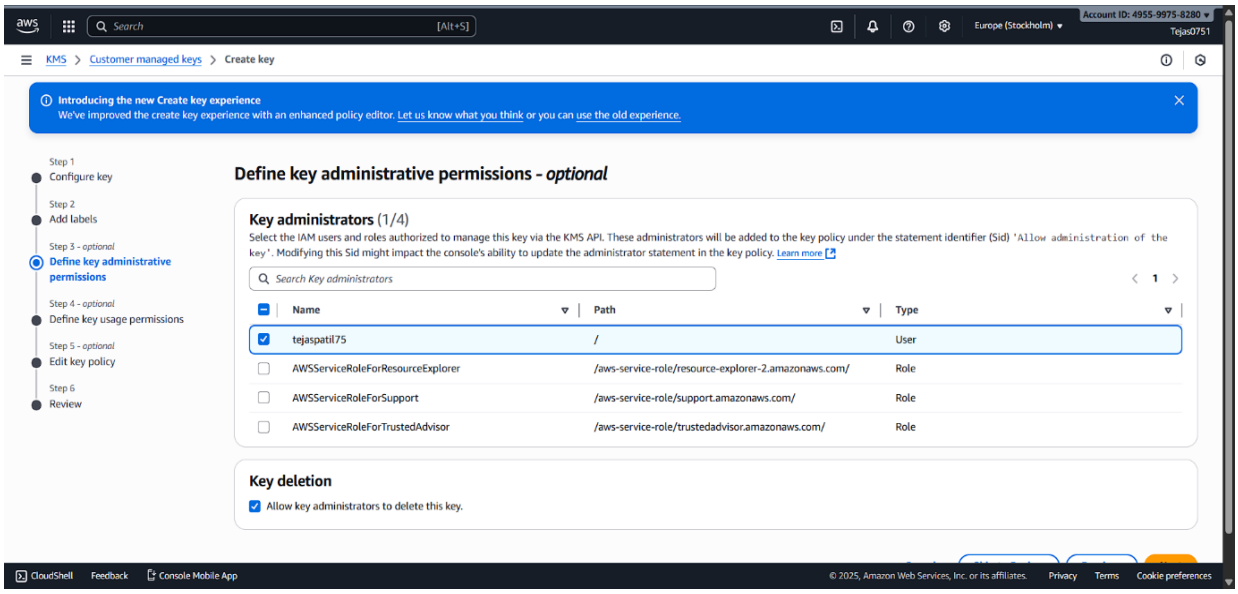
Step 4: Configure Key Details.

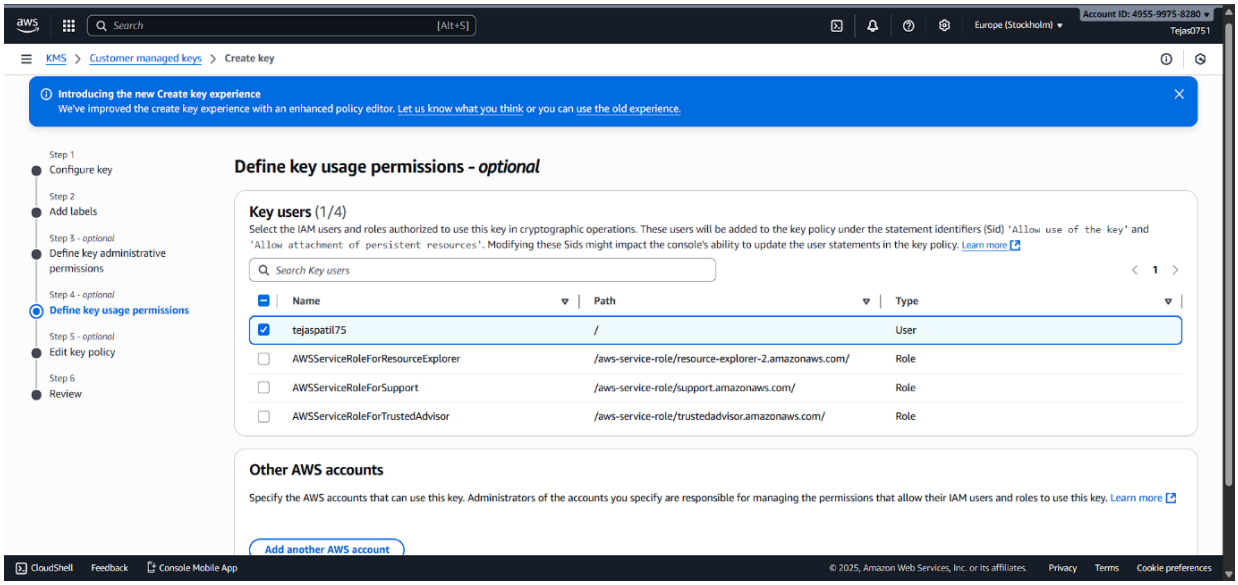
1. Enter a descriptive **Alias** (e.g., lab-cmk) and an optional **Description**.
2. Configure the key usage permission, Click **Next**.

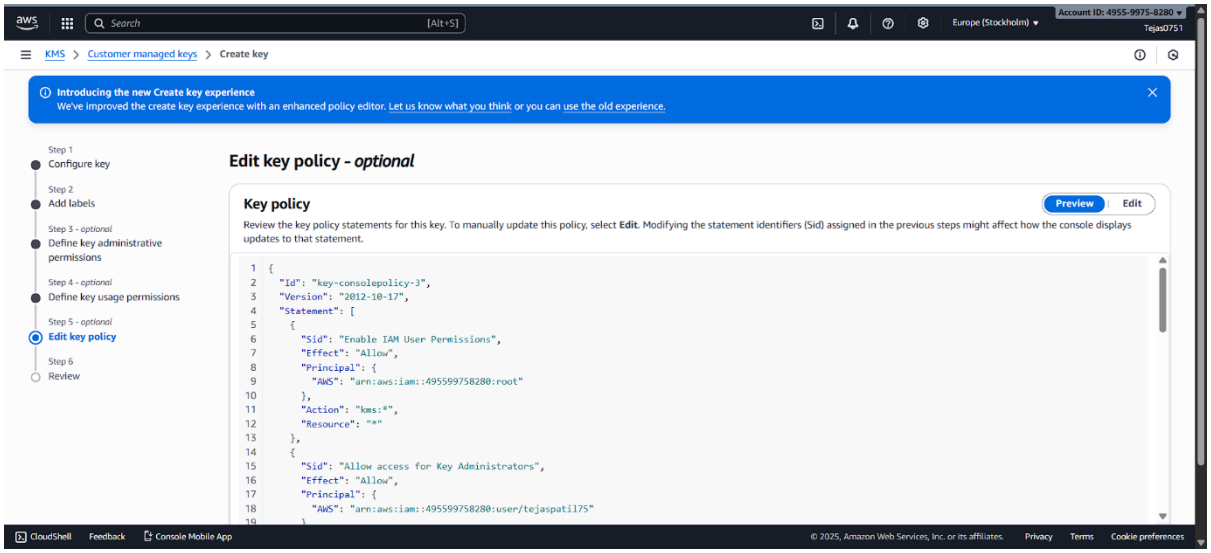


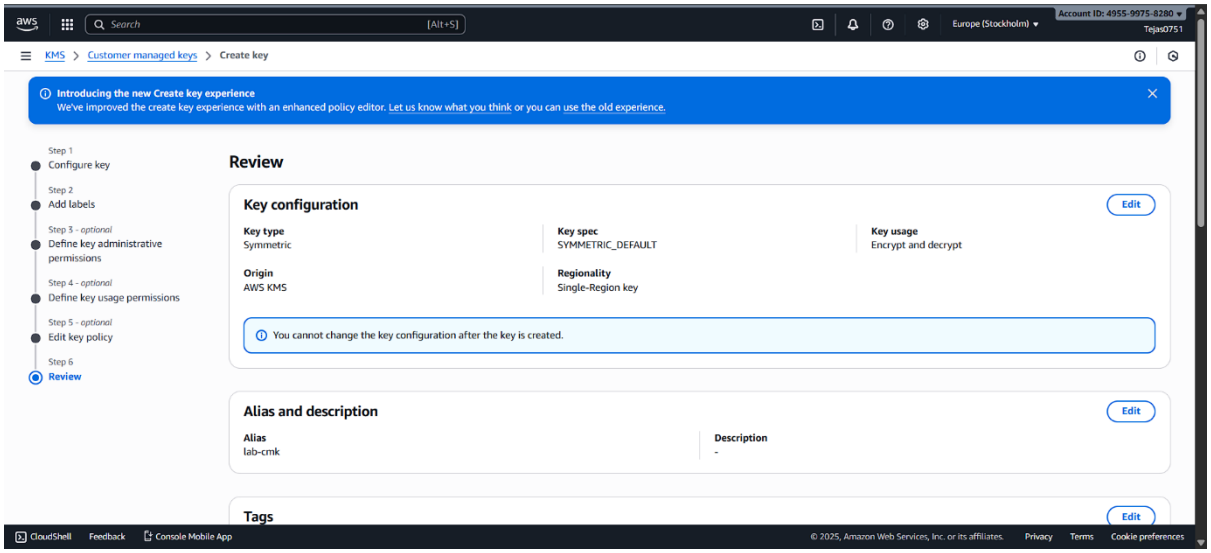
Step 5: Define Key Administrators and Users

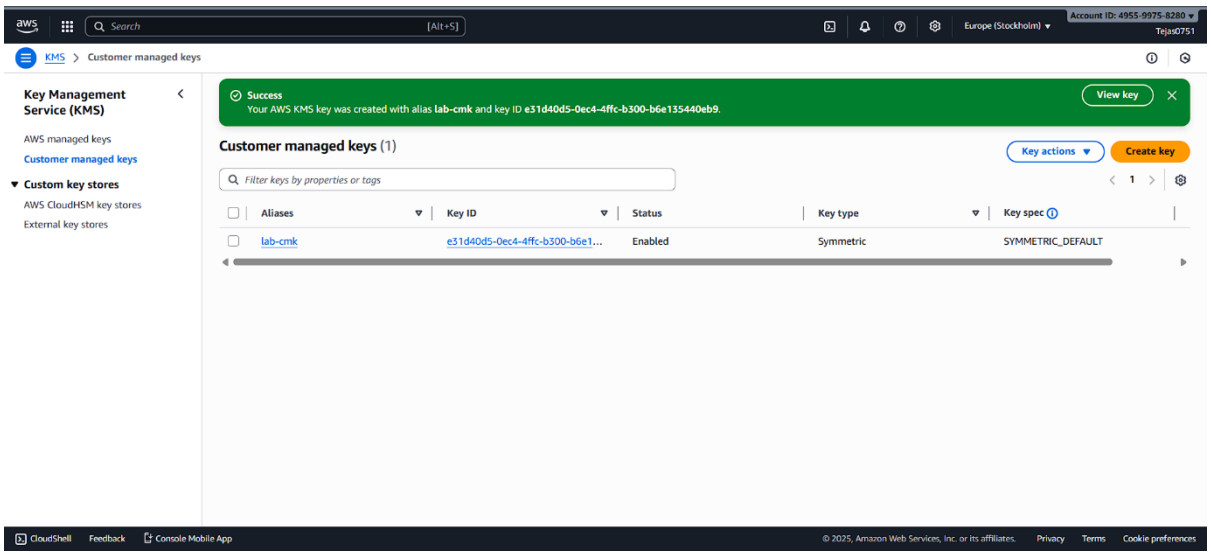
1. Under **Key administrators**, select IAM users or roles allowed to manage this key.
2. Example: select your IAM user.
3. Under **Key users**, choose IAM users or roles that can use the key to encrypt or decrypt data.
4. Click **Next**, review, and then click **Finish** to create the key.





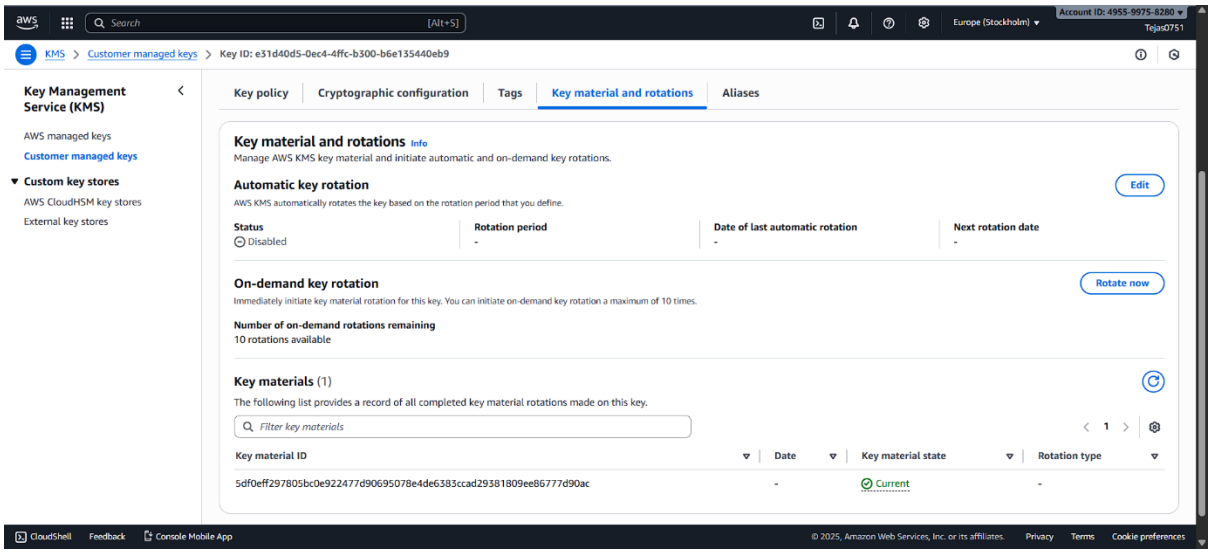

Search [Alt+S]
Europe (Stockholm)
Account ID: 4955-9975-8280
Tejas0751
KMS > Customer managed keys > Create key
Introducing the new Create key experience
We've improved the create key experience with an enhanced policy editor. Let us know what you think or you can use the old experience.
Step 1 Configure key
Step 2 Add labels
Step 3 - optional Define key administrative permissions
Step 4 - optional Define key usage permissions
Step 5 - optional Edit key policy
Step 6 Review
Edit key policy - optional
Key policy
Preview
Edit
Review the key policy statements for this key. To manually update this policy, select Edit. Modifying the statement identifiers (Sid) assigned in the previous steps might affect how the console displays updates to that statement.
{
"Id": "key-consolepolicy-3",
"Version": "2012-10-17",
"Statement": [
{
"Sid": "Enable IAM User Permissions",
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::495599758280:root"
},
"Action": "kms:*",
"Resource": "*"
},
{
"Sid": "Allow access for Key Administrators",
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::495599758280:user/tejaspati175"
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

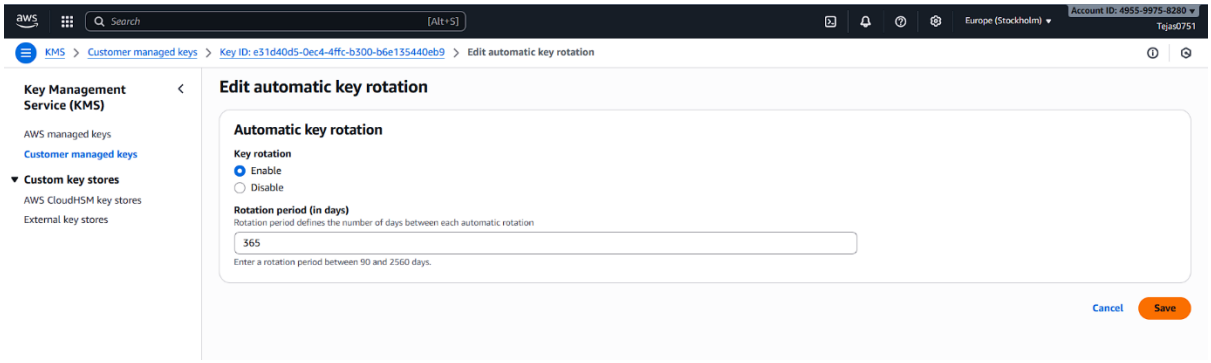

Search [Alt+S]
Europe (Stockholm)
Account ID: 4955-9975-8280
Tejas0751
KMS > Customer managed keys > Create key
Introducing the new Create key experience
We've improved the create key experience with an enhanced policy editor. Let us know what you think or you can use the old experience.
Step 1 Configure key
Step 2 Add labels
Step 3 - optional Define key administrative permissions
Step 4 - optional Define key usage permissions
Step 5 - optional Edit key policy
Step 6 Review
Review
Key configuration
Edit
Key type
Symmetric
Key spec
SYMMETRIC_DEFAULT
Key usage
Encrypt and decrypt
Origin
AWS KMS
Regionality
Single-Region key
You cannot change the key configuration after the key is created.
Alias and description
Edit
Alias
lab-cmk
Description
-
Tags
Edit
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

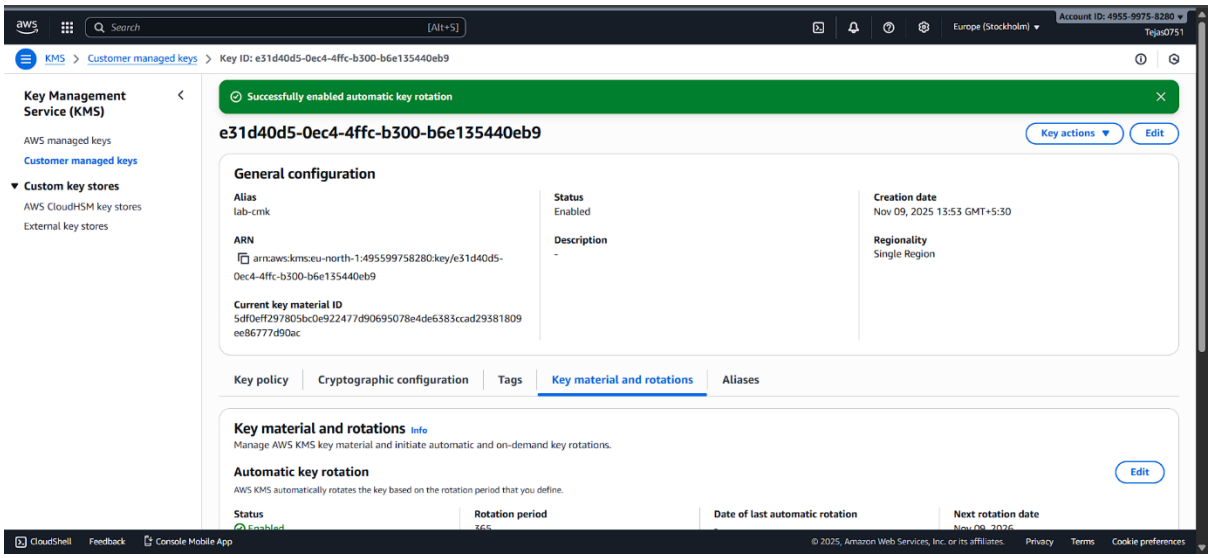

Search [Alt+S]
Europe (Stockholm)
Account ID: 4955-9975-8280
Tejas0751
KMS > Customer managed keys
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
AWS CloudHSM key stores
External key stores
Success
Your AWS KMS key was created with alias lab-cmk and key ID e31d40d5-0ec4-4ffc-b300-b6e135440eb9.
View key
Customer managed keys (1)
Key actions
Create key
Filter keys by properties or tags
Aliases
Key ID
Status
Key type
Key spec
lab-cmk
e31d40d5-0ec4-4ffc-b300-b6e1...
Enabled
Symmetric
SYMMETRIC_DEFAULT
CloudShell Feedback Console Mobile App
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 6: Enable Key Rotation

- After creation, select your new key from the list.
- Under the **Key rotation tab**, check **Enable automatic key rotation**.
- Click **Save changes**.

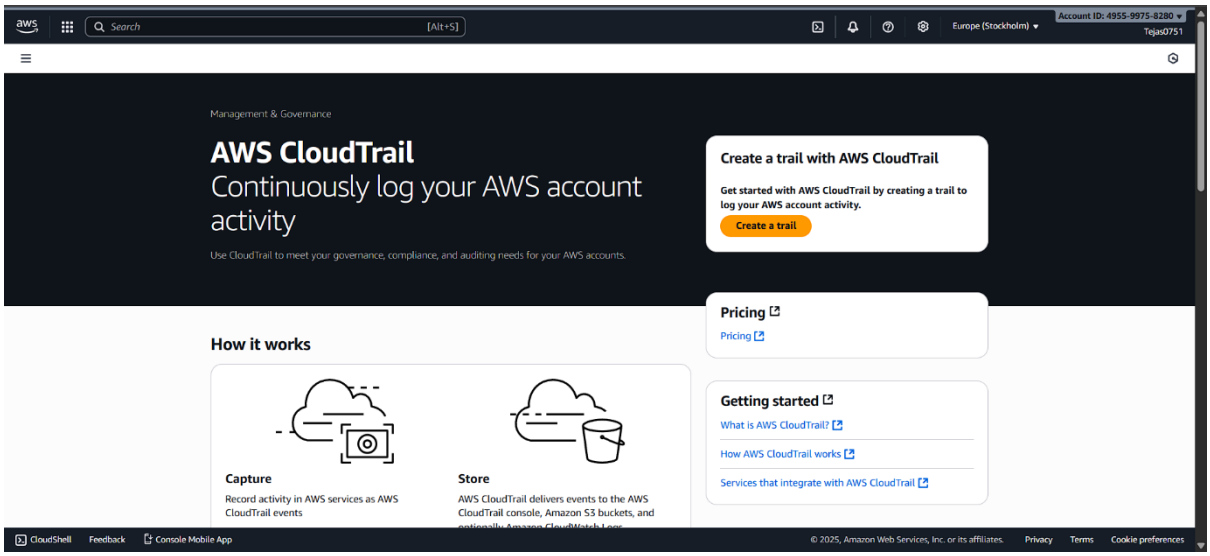






Step 7: Track Key Usage with CloudTrail

1. From the AWS Console, open the **CloudTrail** service.
2. Click **Event history.**
3. Filter by *Event source*: kms.amazonaws.com.
4. Review recent events like *Encrypt*, *Decrypt*, or *GenerateDataKey*.
5. This verifies that key usage is logged for auditing.



Conclusion:

In this experiment, we successfully created a **Customer Managed Key (CMK)** in AWS KMS, defined **administrative and usage permissions**, enabled **automatic key rotation**, **key usage logs** can be viewed via **CloudTrail. This** ensures secure key management, controlled access, and complete auditability of cryptographic operations in AWS.

Assignment No. 6

Title: Setting up a Secure Virtual Private Cloud (VPC) on AWS/Azure

Aim: To design and implement a logically isolated and securely controlled Virtual Private Cloud on AWS/Azure by configuring custom IP addressing, public and private subnets, route tables, gateways/VPN, and layered access controls (security groups, NSGs/NACLs), in order to protect cloud resources while enabling monitored, least-privilege connectivity for enterprise workloads.

Objective:

1. To understand the concept of network isolation in cloud security.
2. To implement a Virtual Private Cloud (VPC) or Virtual Network (VNet) for secure resource deployment.
3. To configure subnets, route tables, and gateways to control internal and external communication.
4. To verify network security through restricted connectivity and proper routing.

Tools / Environment Required:

1. Cloud Platform Account
    a. AWS (Amazon Web Services) or Microsoft Azure account.
    b. The account must have administrative or VPC/VNet management permissions.

2. Cloud Services Used
    a. AWS: VPC, Subnets, Route Tables, Internet Gateway, NAT Gateway, and EC2.
    b. Azure: Virtual Network (VNet), Subnets, Network Security Group, Route Table, and Virtual Machines.

3. System Requirements
    a. A computer or laptop with stable internet connectivity (minimum 5 Mbps).
    b. Latest web browser (Google Chrome, Mozilla Firefox, or Microsoft Edge).

4. Access Credentials
    a. Valid IAM user credentials (AWS) or Subscription Access (Azure).
    b. SSH Key Pair or RDP credentials for testing connectivity to instances.

5. Networking Details
    a. Defined CIDR Range (e.g., 10.0.0.0/16 or 10.1.0.0/16).
    b. Subnet IP ranges for public and private networks.
    c. Routing and gateway configuration details.

Theory:

In cloud computing, shared infrastructure introduces risks such as unauthorized access, data leakage, and exposure of sensitive resources. To mitigate these risks, organizations must deploy

their resources within secure and isolated network environments. This experiment addresses the need to establish a Virtual Private Cloud (VPC) or Virtual Network (VNet) that provides controlled access, segmentation of resources, and secure connectivity to protect data and services from external threats.

A Virtual Private Cloud (VPC) or Virtual Network (VNet) is a fundamental component of cloud security architecture. It enables users to create an isolated logical network within a public cloud, allowing them to manage IP addressing, routing, and access control.

Key Security Concepts:

1. Isolation: Each VPC/VNet operates independently, preventing cross-network interference.
2. Access Control: Using Security Groups and Network Access Control Lists (ACLs) to filter traffic at the instance and subnet levels.
3. Segmentation: Dividing the network into public and private subnets to enforce security zones.
4. Encryption in Transit: Data transferred between instances and subnets can be encrypted using TLS or VPNs.
5. Monitoring: Integration with AWS CloudWatch/Azure Network Watcher for traffic monitoring and intrusion detection.

Key Components:

1. CIDR Block: Defines the IP address range for network communication.
2. Subnets: Logical divisions used to separate public-facing and internal resources.
3. Route Tables: Control traffic flow within and outside the VPC/VNet.
4. Internet Gateway (IGW): Enables communication between the VPC and the public internet.
5. NAT Gateway: Allows outbound internet access for private resources without exposing them publicly.
6. Security Groups/ACLs: Implement layered network protection mechanisms.

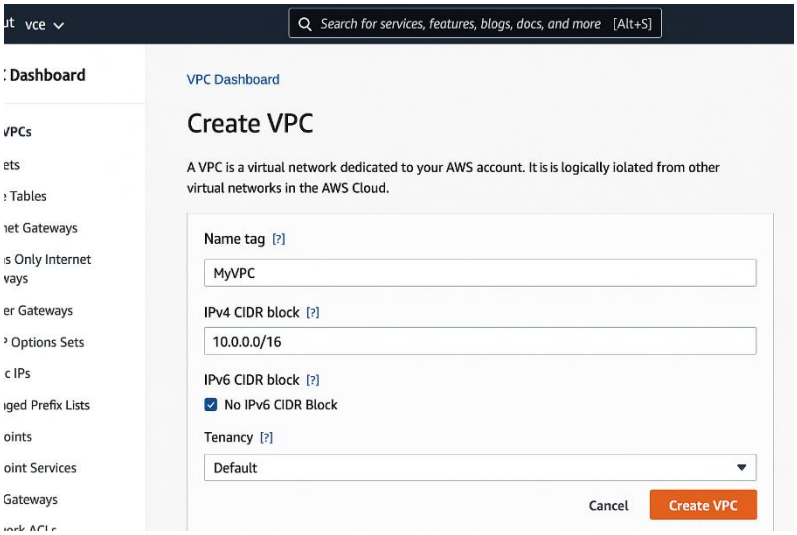
Steps / Procedure:

Step 1: Login

1. Sign in to the AWS Management Console or Azure Portal with credentials having VPC/VNet permissions.

Step 2: Create VPC or VNet

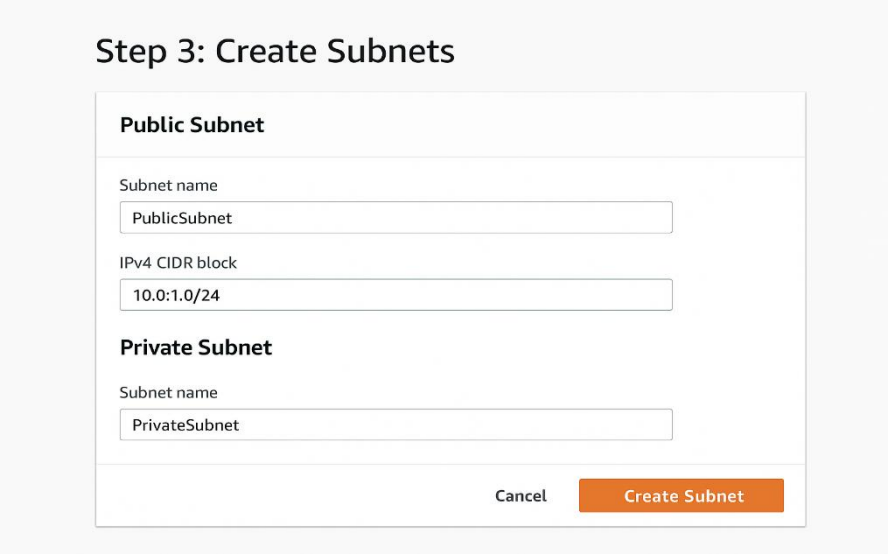
1. AWS:
    1. Navigate to VPC Dashboard → Create VPC.
    2. Define Name Tag (e.g., SecureVPC).
    3. Set IPv4 CIDR Block (e.g., 10.0.0.0/16).
    4. Choose VPC Only and click Create.



2. Azure:
    1. Go to Virtual Networks → Create.
    2. Select Resource Group, name your VNet (e.g., SecureVNet).
    3. Define Address Space (e.g., 10.1.0.0/16).

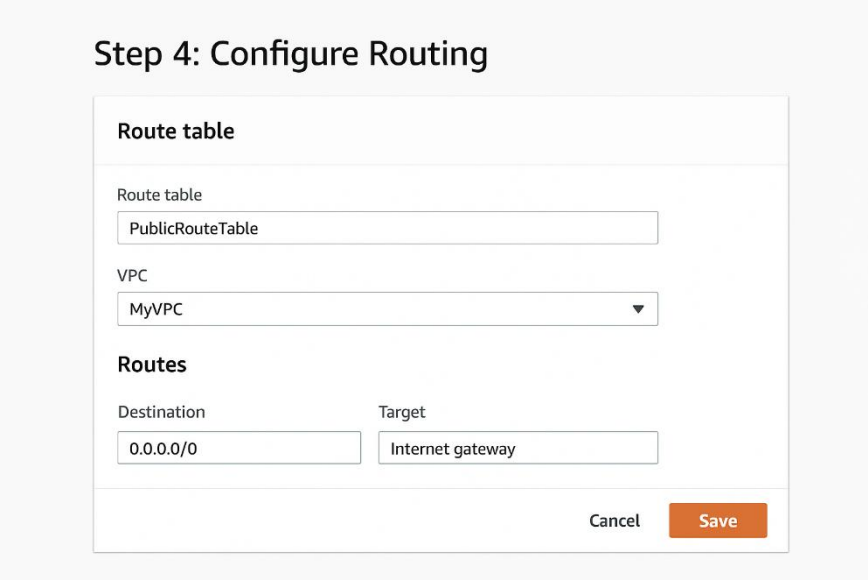
Step 3: Create Subnets

1. Add Public and Private subnets.
    1. Public Subnet: 10.0.1.0/24 (accessible via Internet Gateway).
    2. Private Subnet: 10.0.2.0/24 (internal resources only).



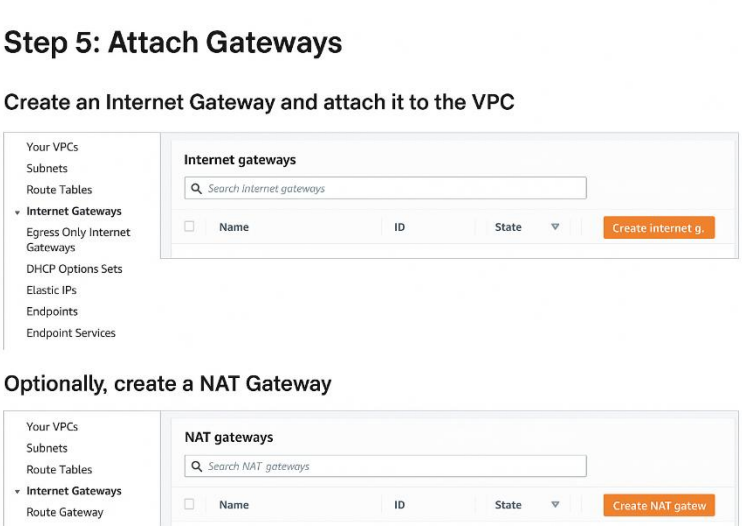
Step 4: Configure Route Tables

1. Create and associate Route Tables with subnets.
2. Public Route Table → Route to Internet Gateway (0.0.0.0/0).
3. Private Route Table → Route via NAT Gateway (for outbound access).



Step 5: Attach Gateways

1. AWS:
    1. Create an Internet Gateway and attach it to the VPC.
    2. Optionally, create a NAT Gateway for private subnet outbound connections.



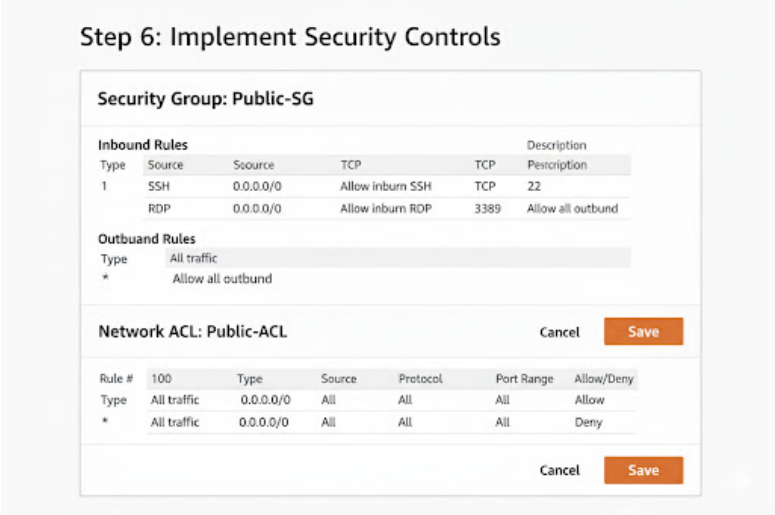
2. Azure:
   Create a Public IP and associate it with a VPN Gateway or Internet Gateway.

Step 6: Implement Security Controls

1. Configure Security Groups to allow limited inbound traffic (e.g., only SSH/RDP).
2. Set Network ACLs to restrict unwanted ports and IP ranges.
3. Enable Flow Logs for monitoring network traffic.

Step 7: Launch and Test Instances

1. Deploy EC2 Instances (AWS) or Virtual Machines (Azure) in each subnet.
2. Use SSH or RDP to connect securely.



3. Test:
    1. Public subnet instance → internet access allowed.
    2. Private subnet instance → internet access only through NAT.
    3. Ping between instances (internal connectivity test).

Conclusion:

This experiment successfully demonstrated the creation of a secure and isolated cloud network environment, an essential component of cloud security. By using VPC/VNet features such as subnets, gateways, and security rules, cloud architects can prevent unauthorized access, limit exposure of critical resources, and ensure safe communication between services. The exercise reinforced key cybersecurity principles — isolation, least privilege, network segmentation, and continuous monitoring — within cloud infrastructure.

Assignment No: 7

Title: Configuring and Testing Data Loss Prevention (DLP) in the Cloud.

Aim: The primary aim of this experiment is to implement and validate a cloud-native Data Loss Prevention (DLP) workflow using Amazon Web Services.

Objectives:

1. Enable Amazon Macie to discover and classify sensitive data in S3 using managed and custom identifiers.
2. Enforce least-privilege access to sensitive objects with IAM/S3 bucket policies and default encryption.
3. Set up monitoring and alerting via CloudTrail, CloudWatch, and Macie findings.
4. Automate remediation by routing Macie findings through EventBridge to trigger SNS/Lambda actions.
5. Validate controls with test cases (PII uploads, unauthorized access attempts) and record outcomes/KPIs

Tools / Environment Required:

1. AWS account and sign-in credentials.
2. An IAM user/role with permissions: at minimum AdministratorAccess or the Macie + S3 permissions (MacieFullAccess + S3 read access + ability to create EventBridge/SNS/Lambda).
3. At least one existing S3 bucket (or create one) that will be scanned.
4. Basic console access to: Amazon Macie, S3, EventBridge, SNS, IAM, optionally Lambda.
5. A small set of test files (text/CSV) containing sample PII (fake SSNs, credit card numbers, email addresses) for testing.

Theory: Configuring Data Loss Prevention (DLP) in Cloud

Data Loss Prevention (DLP) is a security mechanism that detects and prevents potential data breaches by monitoring, identifying, and blocking the unauthorized transmission of sensitive data.

It ensures that confidential information — such as PII (Personally Identifiable Information), credit card numbers, SSNs, or intellectual property — is not exposed, misused, or transferred outside the organization's boundary.

With the massive shift to cloud environments, data is no longer confined to on-premises servers. It's stored, shared, and processed across various services (AWS S3, Google Drive, Azure Blob, etc.).

Hence, cloud-native DLP ensures visibility and protection for data in:

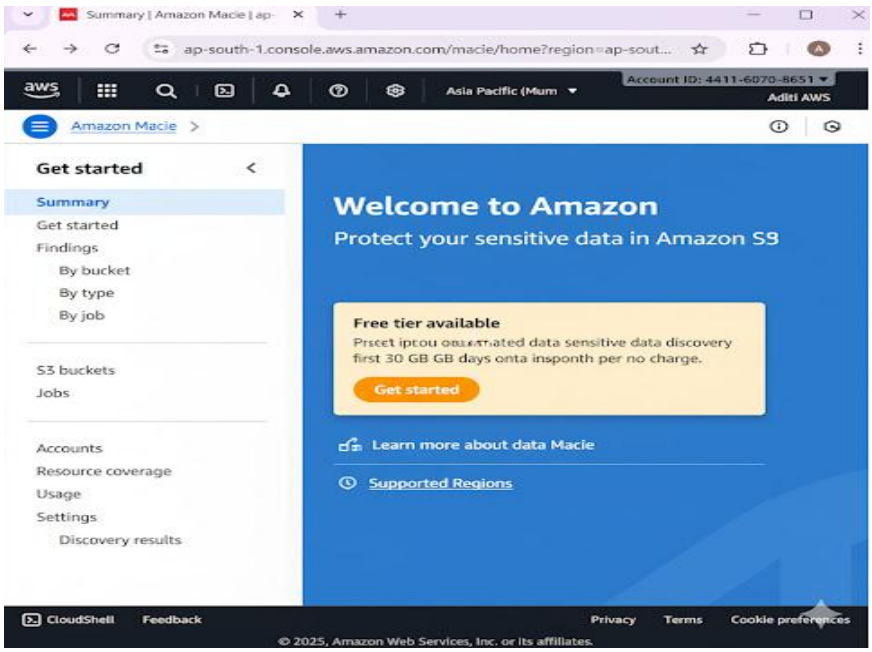
- Storage (data at rest)
- Transmission (data in motion)
- Processing (data in use)

Core Objectives of DLP

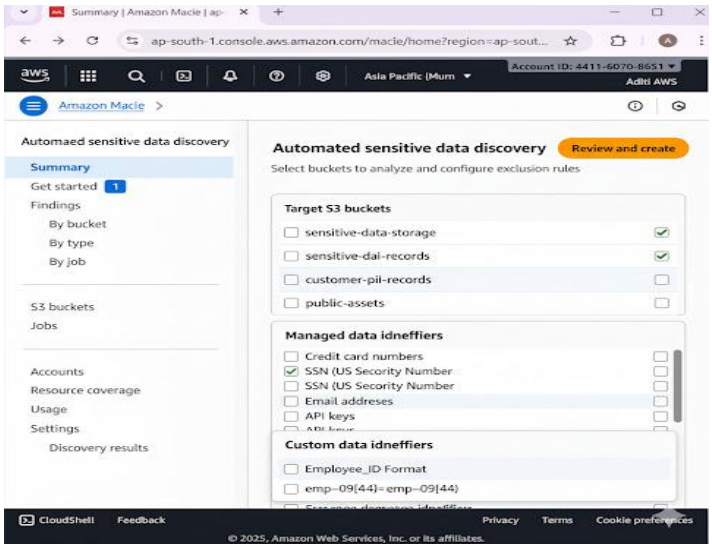
1. Discover sensitive data across cloud storage
2. Classify data according to type and sensitivity.
3. Monitor access and movement.
4. Protect by enforcing policies (masking, encrypting, or blocking data sharing).
5. Report & Alert security teams on violations.

Steps / Procedure:

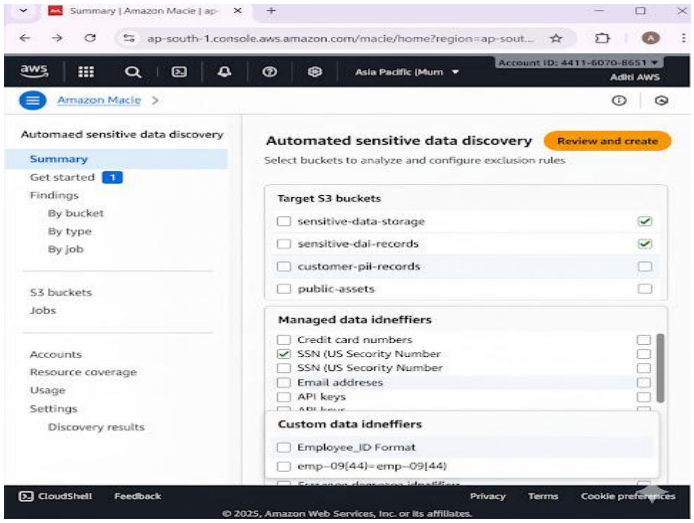
1. Enable DLP service on cloud storage resources.



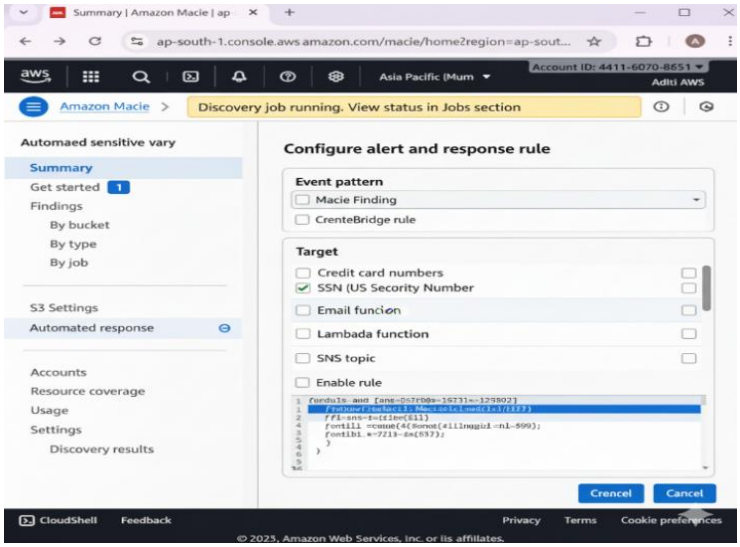
2. Define sensitive data types and scanning policies.



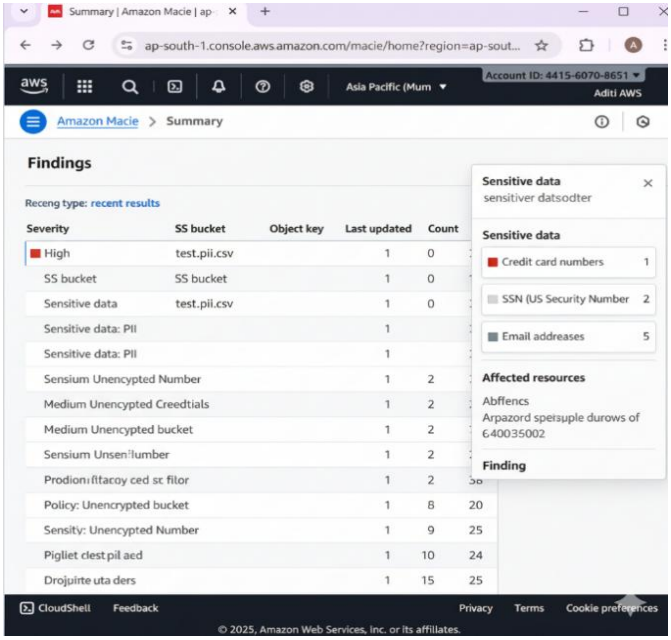
3. Run full scan on existing storage/data.



4. Configure alerts and response actions (block, quarantine).



5. Test by uploading files with sensitive info and validate detection.



Conclusion: The experiment successfully implemented a cloud-native DLP workflow on AWS that discovers, classifies, and protects sensitive data in Amazon S3 using machine learning-driven detection, least-privilege access controls, and automated remediation triggers. The configured pipeline demonstrated reliable detection of PII, actionable findings, alerting, and policy enforcement, reducing exposure from public access and misconfigurations while improving auditability through centralized logs and findings.

Assignment No: 8

Title: Implementation of an Intrusion Detection and Prevention System (IDPS) in the cloud.

Aim: To deploy and validate a cloud-based Intrusion Detection and Prevention System (IDPS) using AWS services to detect, alert, and block suspicious network activities.

Objectives:

1. To enable and configure AWS GuardDuty for detecting threats and anomalies.
2. To deploy AWS Network Firewall for preventing and controlling malicious traffic.
3. To simulate and monitor intrusion attempts in a secure cloud environment.
4. To analyze alerts, logs, and incidents to enhance network security policies.

Tools / Environment Required:

1. Active AWS account
2. Access to the following AWS services:
    a. **Amazon GuardDuty** (for intrusion detection)
    b. **AWS Network Firewall** (for intrusion prevention)
    c. **CloudWatch Logs** (for monitoring alerts)
3. Configured **VPC** and **EC2 instance** for testing
4. Internet access

Theory:

An **Intrusion Detection and Prevention System (IDPS)** is a security solution that monitors, detects, and takes action against unauthorized access or malicious activities within a network or system. **Intrusion Detection System (IDS)** focuses on identifying suspicious activities and generating alerts. **Intrusion Prevention System (IPS)** not only detects but also automatically blocks or restricts malicious traffic in real time. In **cloud environments**, IDPS plays a crucial role in maintaining security across virtual networks and services.
On **AWS**, IDPS can be implemented using two key services:

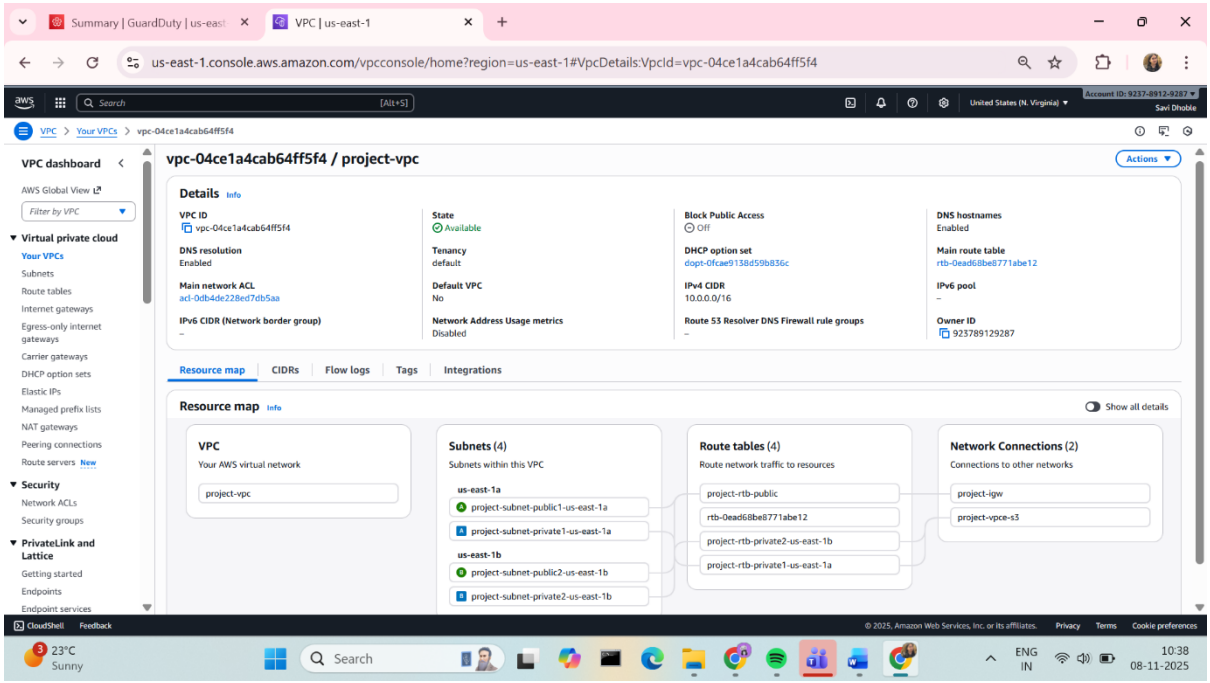
Amazon GuardDuty: A managed threat detection service that uses machine learning and threat intelligence to identify anomalies, compromised instances, and potential attacks.

AWS Network Firewall: A managed network protection service that controls inbound and outbound traffic at the VPC level with customizable rule sets to prevent attacks.

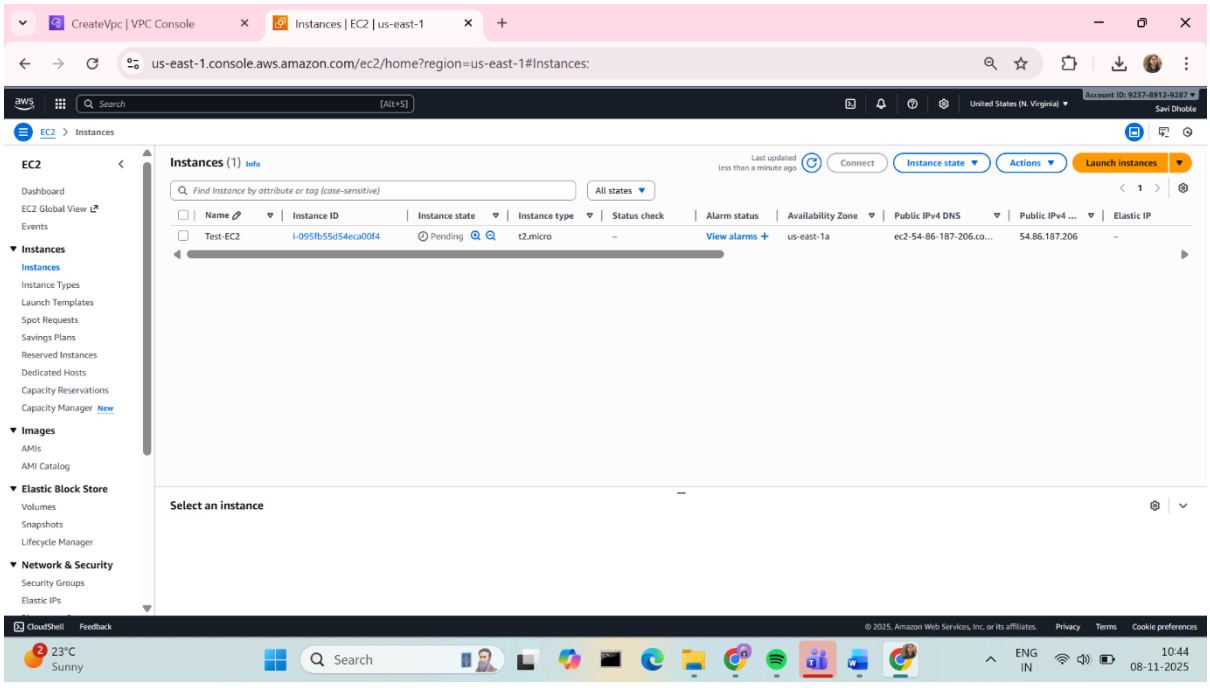
Together, these services form a comprehensive cloud IDPS that both detects and mitigates threats effectively.

Steps / Procedure:

Step 1: Create a VPC (Virtual Private Cloud)



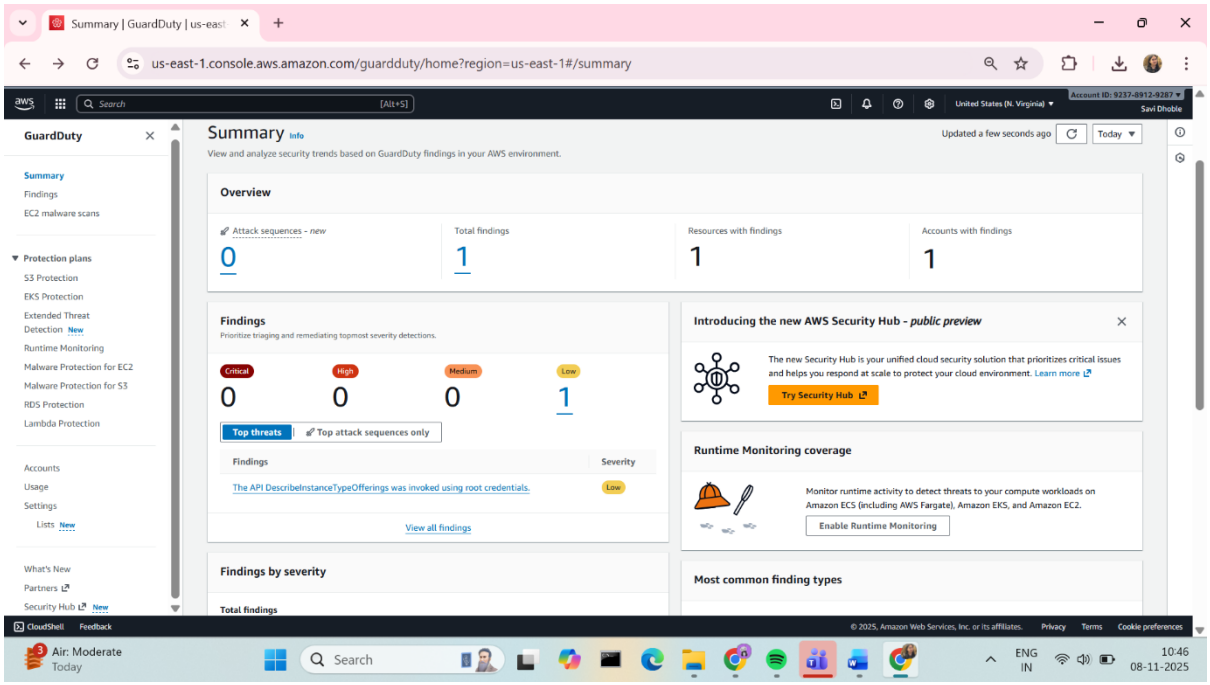
Step 2: Launch an EC2 Instance (Target Machine for Monitoring)



Step 3: Enable AWS GuardDuty (Intrusion Detection System - IDS)

1. In the AWS Management Console, search for GuardDuty.
2. Click Enable GuardDuty.

3. GuardDuty starts collecting data from AWS CloudTrail, VPC Flow Logs, and DNS logs.
4. Wait for it to initialize (takes 1–2 minutes).
5. Once active, GuardDuty will automatically begin analyzing traffic for suspicious behavior (like port scans or brute-force attempts).



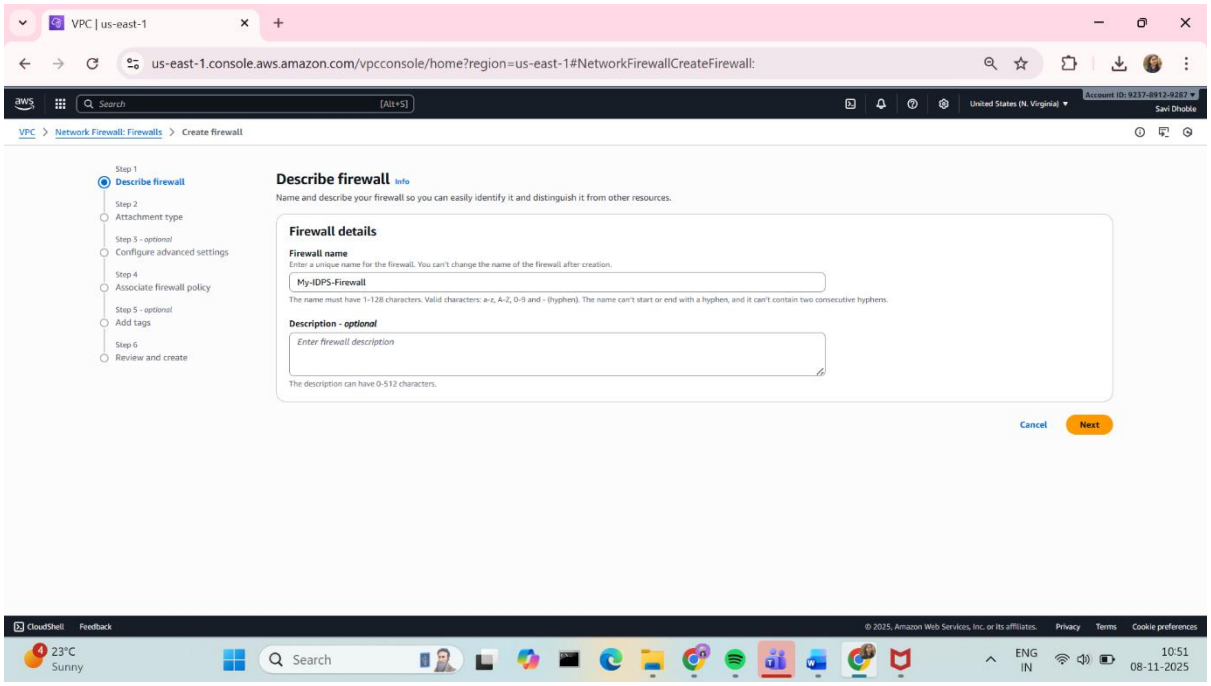
Step 4: Create AWS Network Firewall (Intrusion Prevention System - IPS)

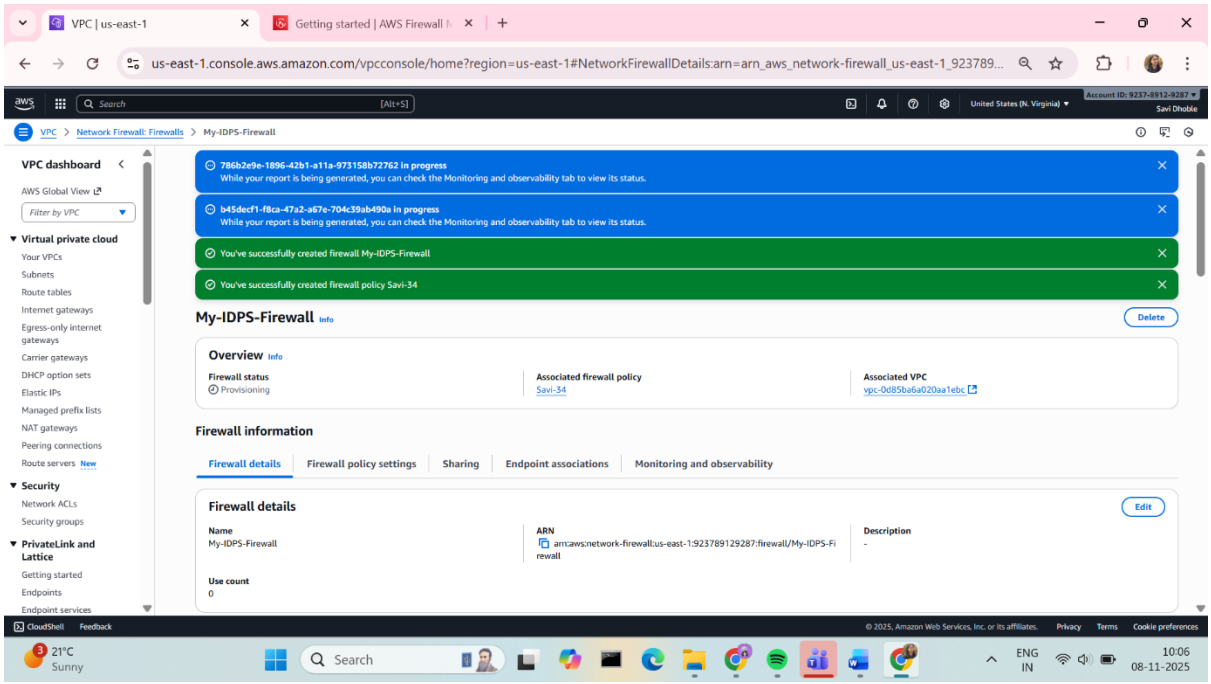
Now we'll add a firewall to actively block malicious traffic.

Steps:

1. Go to VPC → Network Firewall → Firewalls → Create firewall.
2. Fill in:
    - Name: My-IDPS-Firewall
    - Description: "Firewall for Intrusion Prevention in VPC"
    - VPC: Select My-IDPS-VPC
    - Choose Public Subnet 1 & 2 for firewall endpoints.
3. Click Create Firewall.

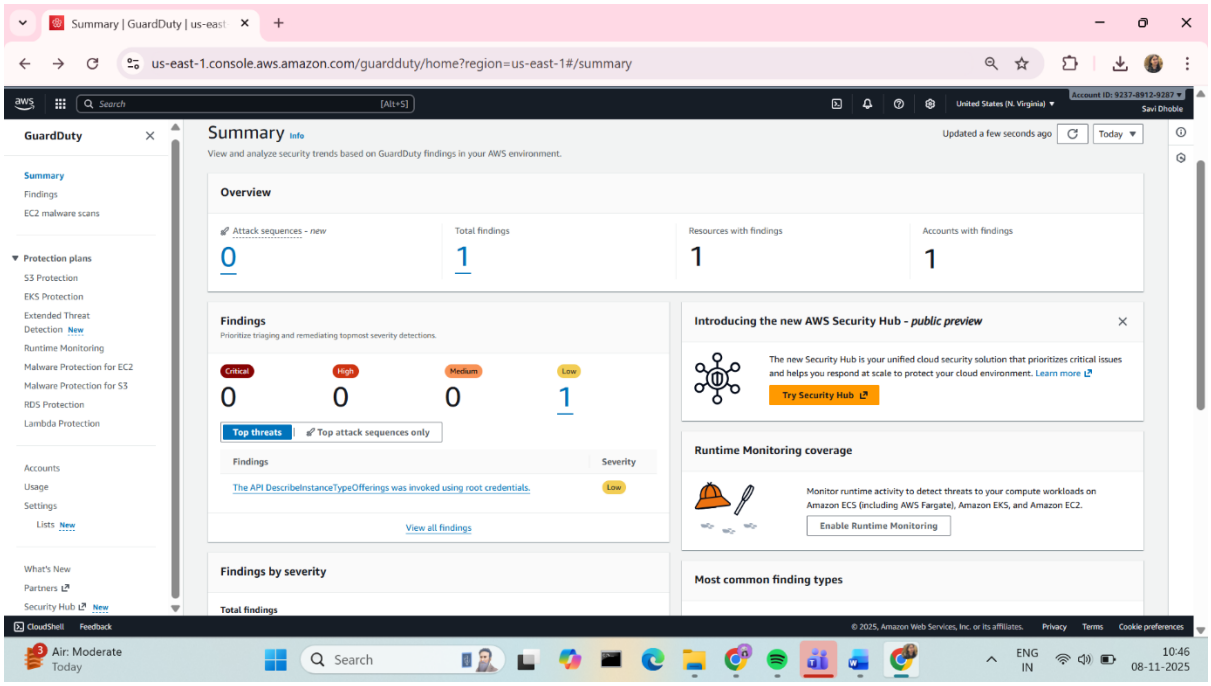
You'll now be prompted to create or attach a Firewall Policy.





Step 5: Monitor and Analyze Threats

1. Open GuardDuty → Findings to view detected intrusions.
    - You'll see threat details like source IP, affected instance, and type of attack.
2. Go to CloudWatch Logs → Log groups → View Network Firewall logs to confirm blocked traffic.
3. Update firewall rules if new suspicious IPs are detected.



Conclusion:

By implementing IDPS using AWS GuardDuty and Network Firewall, we achieved real-time detection and automatic blocking of security threats. This practical demonstrates how cloud-native security tools can effectively safeguard cloud infrastructure against unauthorized access, attacks, and malicious network traffic. Regular monitoring and rule optimization further enhance the overall security posture of the cloud environment.

Assignment No:9

Title: IaC Security using Terraform

Aim: To implement Infrastructure as Code (IaC) using Terraform and apply security best practices for deploying secure cloud infrastructure.

Objectives:

1. To understand the concept of Infrastructure as Code (IaC) and its role in cloud security.
2. To identify and mitigate misconfigurations in cloud infrastructure definitions.
3. To apply security controls such as IAM policies, encryption, and network segmentation in Terraform configurations.
4. To demonstrate the use of automated policy checking tools for IaC security compliance.

Tools/Environment required:

1. **Terraform v1.x:** To define and manage cloud infrastructure as code
2. **Visual Studio Code**/ any IDE: For writing Terraform code
3. **Checkov**: Static analysis tool for IaC scanning
4. Operating System: Windows
5. Internet Connectivity

Theory:

Infrastructure as Code (IaC):

IaC is the practice of automating infrastructure setup through code files. Instead of manually configuring servers, networks, and storage, engineers define the infrastructure in configuration files (e.g., .tf for Terraform). It ensures reproducibility, version control, and automation. Benefits include:

- **Automation & Speed:** Rapid setup and scaling.
- **Consistency:** Same configuration reproducible across environments.
- **Auditing & Compliance:** Version-controlled and reviewable.
- **Security Enforcement:** Apply policies programmatically.

Why IaC Security Matters:

Traditional manual configuration often leads to misconfigurations such as open security groups, weak IAM roles, and unencrypted storage. IaC enables:

- **Version control:** Infrastructure can be audited and rolled back using Git.
- **Compliance as code:** Security policies can be automated using tools like Sentinel or Checkov.
- **Consistency:** The same secure configurations can be applied across multiple environments.

IaC Security Best Practices:

- Implement **Least Privilege** IAM policies.
- Enable **Encryption** for storage and data transmission.
- Restrict network access via **Security Groups and Firewalls**.
- Manage credentials using **environment variables or vaults**.

Terraform:

Terraform by HashiCorp is an **open-source tool** that uses **HCL (HashiCorp Configuration Language)** to define and manage infrastructure across multiple providers like AWS, Azure, and GCP.

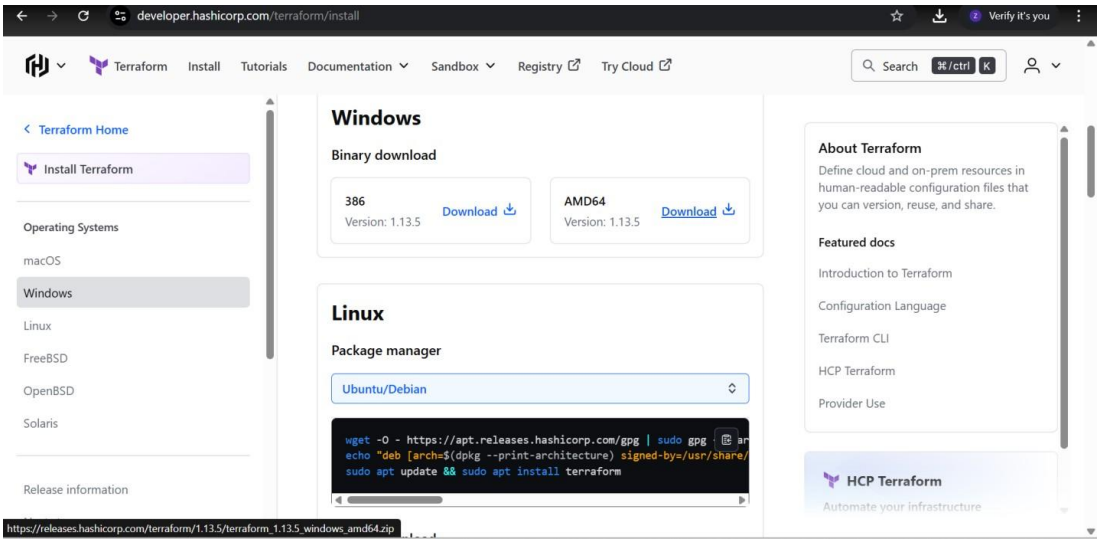
Security in Terraform:

1. **IAM Role Management:** Define least privilege policies and roles.
2. **Network Security:** Use security groups, private subnets, and firewall rules.
3. **Encryption:** Enable encryption for data at rest (S3, RDS, EBS) and in transit (TLS).
4. **Secrets Management:** Use environment variables or secret stores (AWS Secrets Manager).
5. **Policy Enforcement:** Use Sentinel or Checkov to scan .tf files for misconfigurations
6. **Checkov:** Checkov is an **open-source static analysis tool** by **Bridgecrew** (Palo Alto Networks). It scans Terraform, CloudFormation, Kubernetes, and other IaC frameworks for:
    - Security misconfigurations
    - Compliance violations (CIS, NIST, ISO 27017) - Policy enforcement

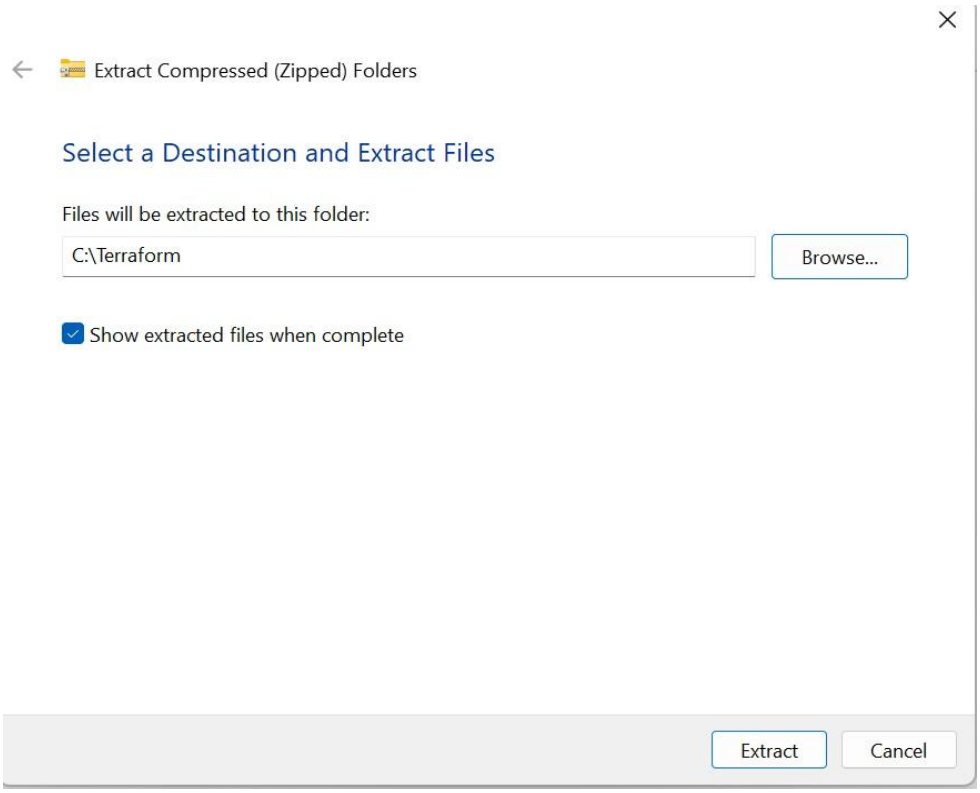
Steps / Procedure:

Step 1: Install Terraform

1. Download Terraform from https://www.terraform.io/downloads.

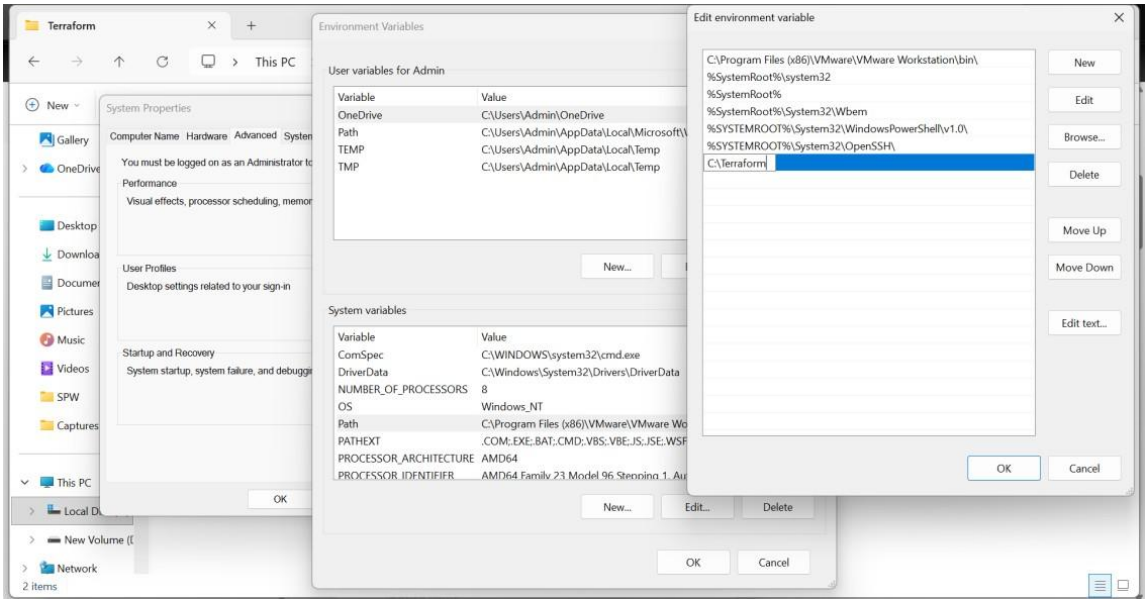


2. Extract the downloaded zip file to a folder.



3. Add Terraform to system PATH:

- Right-click on 'This PC' → Properties → Advanced System Settings
- Click 'Environment Variables'
- Under System Variables, find 'Path' and click Edit
- Click New and add terraform location as path (eg., **C:\terraform**)



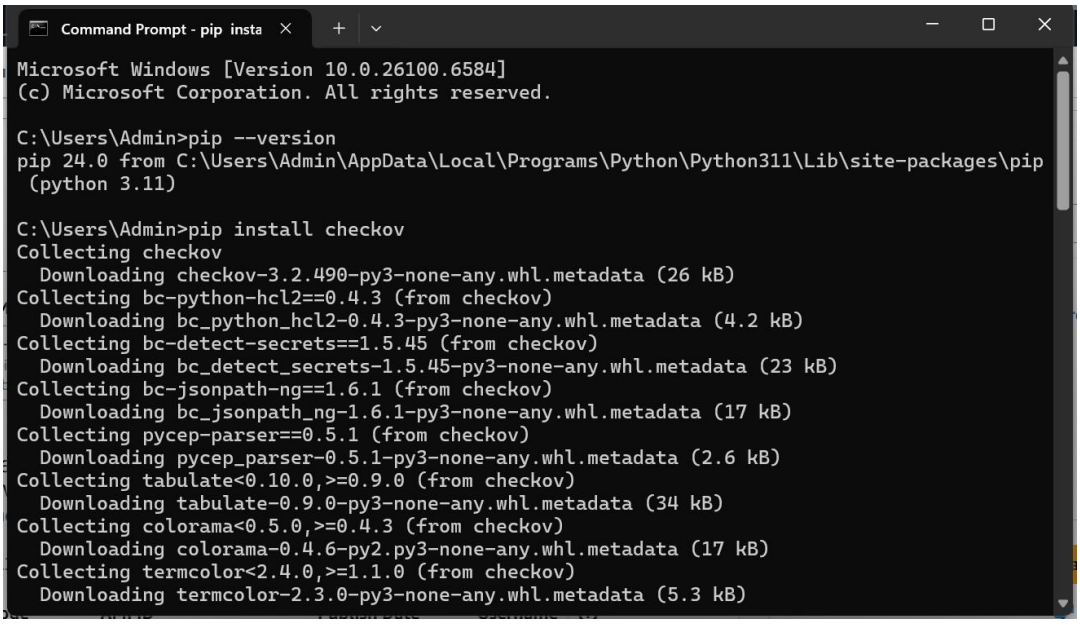
- Click OK on all windows.

4. Verify installation using: **terraform –version**

Step 2: Install Checkov

1. You can install checkov using pip:

pip install checkov



2. Verify installation:

checkov –version

Step 3: Write Terraform Configuration File 1. Create folder and open it in VSCode.

Create a file (main.tf) for infrastructure definition

main.tf:

provider "aws" { region = "us-west-1"

access_key = "AKIAIOSFODNN7EXAMPLE" # Hardcoded credentials secret_key = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

}

```
resource "aws_s3_bucket" "bad_bucket" {   bucket = "my-bad-bucket"
  acl    = "public-read" # Public S3 bucket
  tags = {
    Name = "bad_bucket"
  } }
resource "aws_security_group" "bad_sg" {
  name        = "bad_sg"
  description = "A security group with overly permissive rules"
  # Allow all inbound traffic
  ingress {     from_port   = 0     to_port     = 65535     protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {      from_port    = 0      to_port      = 65535      protocol     = "tcp"      cidr_blocks =
["0.0.0.0/0"]
  }
  tags = {
    Name = "bad_sg"
  }
}
resource "aws_instance" "bad_instance" {   ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  # No encryption   root_block_device {     volume_type = "gp2"
    volume_size = 10
  }

  tags = {
    Name = "bad_instance"
  }
}
```

Step 4: Initialize and validate Terraform Configuration terraform init

```
PS C:\terraform-iac-security> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.20.0...
- Installed hashicorp/aws v6.20.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\terraform-iac-security>
```

terraform validate

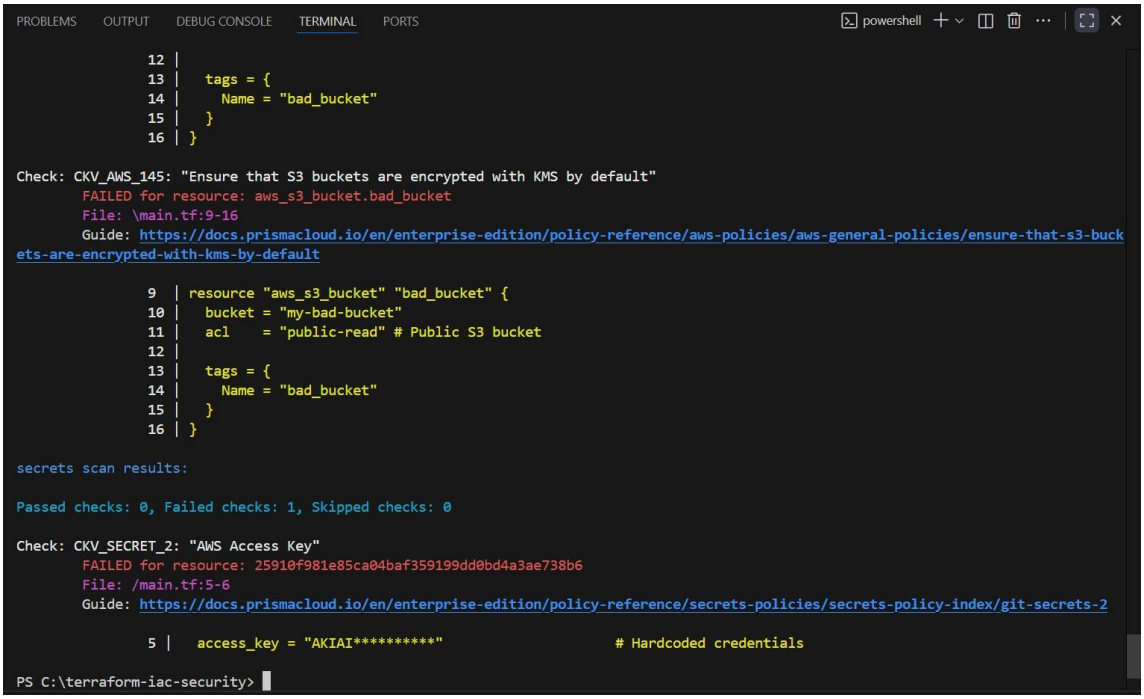
```
PS C:\terraform-iac-security> terraform validate

Warning: Argument is deprecated

  with aws_s3_bucket.good_bucket,
  on main.tf line 61, in resource "aws_s3_bucket" "good_bucket":
  61: resource "aws_s3_bucket" "good_bucket" {

versioning is deprecated. Use the aws_s3_bucket_versioning resource instead.

(and 5 more similar warnings elsewhere)
```

This checks syntax and structure.

Step 5: Run Checkov IaC Security Scan checkov –d .

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   powershell

PS C:\terraform-iac-security> checkov -d .
[ terraform framework ]: 100%|[1/1], Current File Scanned=main.tf
[ secrets framework ]: 100%|[1/1], Current File Scanned=.\main.tf
[ secrets framework ]: 100%|[1/1], Current File Scanned=.\main.tf
       _               _
   ___| |__   ___  ___| | _______   __
  / __| '_ \ / _ \/ __| |/ / _ \ \ / /
 | (__| | | |  __/ (__|   < (_) \ V /
  \___|_| |_|\___|\___|_|\_\___/ \_/

By Prisma Cloud | version: 3.2.490

terraform scan results:

Passed checks: 7, Failed checks: 18, Skipped checks: 0

Check: CKV_AWS_93: "Ensure S3 bucket policy does not lockout all but root user. (Prevent lockouts needing root account fixes)"
        PASSED for resource: aws_s3_bucket.bad_bucket
        File: \main.tf:9-16
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/bc-aws-s3-24
Check: CKV_AWS_382: "Ensure no security groups allow egress from 0.0.0.0:0 to port -1"
        PASSED for resource: aws_security_group.bad_sg
        File: \main.tf:18-40
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-policies/bc-aws-382
Check: CKV_AWS_277: "Ensure no security groups allow ingress from 0.0.0.0:0 to port -1"
        PASSED for resource: aws_security_group.bad_sg
        File: \main.tf:18-40
        Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-networking-policies/ensure-aws-secur
```

Output shows security compliance results for your **.tf** files.

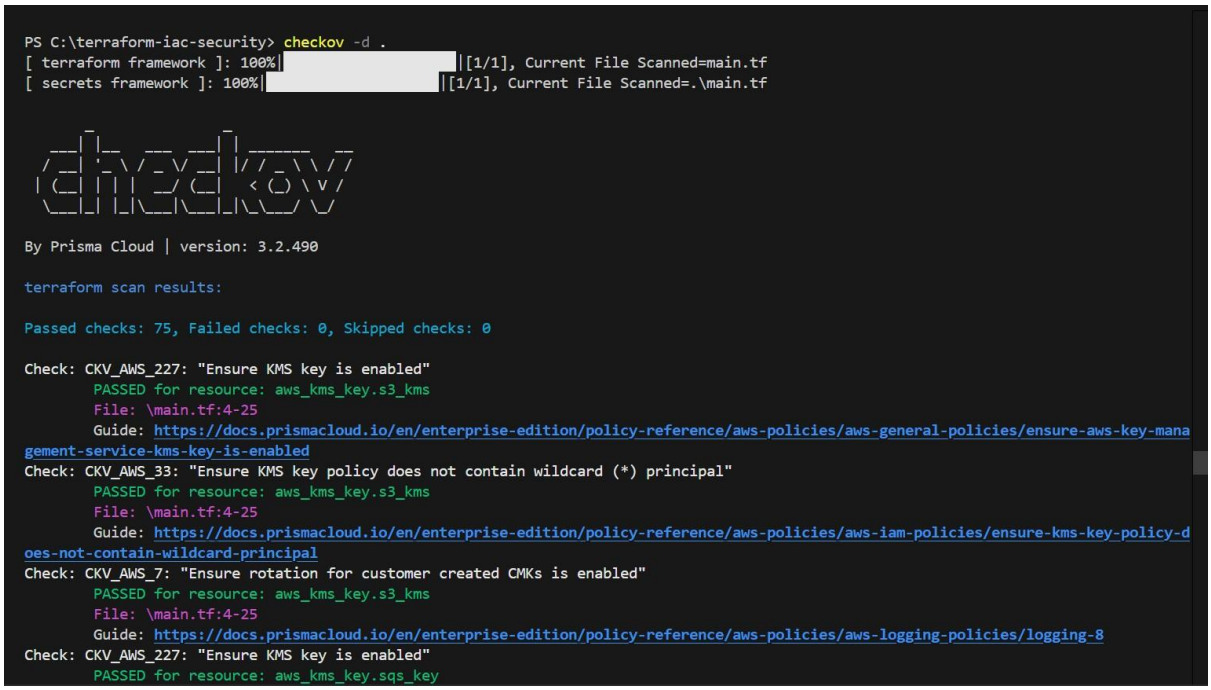


Step 6: Fix Misconfigurations

1. Remove hardcoded credentials
2. Restrict SSH access to known IPs
3. Add missing descriptions

Step 7: Re-run Checkov IaC Security Scan

Initialize, validate terraform and re-run: **checkov -d .** Repeat the process until all security checks are passed.



Conclusion:

In this assignment, the students implemented Infrastructure as Code (IaC) using Terraform to automate secure cloud infrastructure deployment. The exercise demonstrates how Terraform enhances cloud security by making configurations **repeatable, auditable, and policy-driven**.

Assignment No: 10

Title: Implementation of Developing an Incident Response and Disaster Recovery Plan for Cloud Services

Aim: To design, document, and validate a cloud-focused Incident Response (IR) and Disaster Recovery (DR) plan that aligns with NIST lifecycle phases, defines roles and runbooks, implements automated detection and recovery mechanisms, and proves recovery time and data integrity through testing and post-incident improvement for resilient cloud services.

Objectives:

1. Define governance, roles, and incident classification aligned to NIST phases and the cloud shared responsibility model.
2. Implement detective controls, log retention, and evidence preservation with playbooks for containment, eradication, and recovery.
3. Establish automated response workflows and communications (runbooks, alerts, escalations) to meet defined RTO/RPO targets.
4. Conduct tabletop and technical simulations to validate IR/DR procedures, then refine based on post-incident reviews.
5. Measure effectiveness with KPIs (MTTD, MTTR, containment time, recovery success, evidence completeness) and drive continuous improvement.

Tools / Environment Required:

1. AWS Free Tier account (or equivalent cloud platform)
2. Basic knowledge of EC2, S3, CloudWatch, and CloudTrail
3. Internet connection and access to AWS Management Console

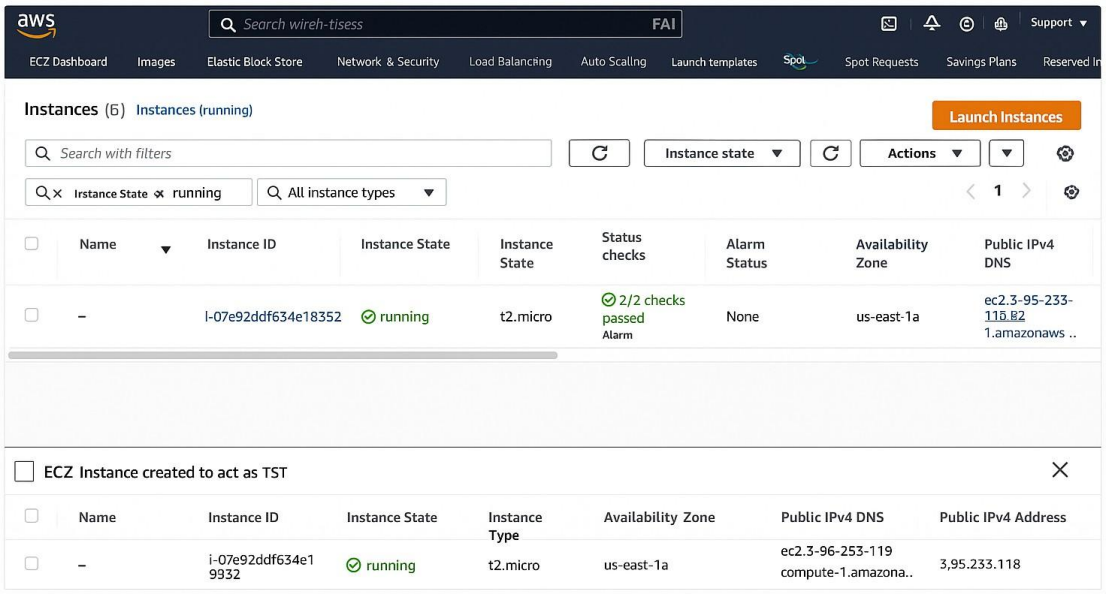
Theory:

Incident Response (IR) and Disaster Recovery (DR) for cloud services centre on preparing, detecting, containing, and recovering from security events in elastic, shared-responsibility environments. In the cloud, assets are highly dynamic (ephemeral instances, serverless, containers), which shifts emphasis from perimeter defense to identity, configuration, and telemetry-driven detection. Effective IR begins with clear roles, incident classifications, and immutable logging to preserve evidence, then uses automated playbooks to isolate compromised identities or resources, rotate keys, and block exfiltration while maintaining business continuity. DR complements this by architecting for resilience—defining recovery point and time objectives, employing multi-AZ/region patterns, versioned and encrypted backups, and routine failover tests—to ensure data integrity and rapid restoration. Continuous improvement is essential: post-incident reviews refine runbooks, tighten least-privilege access, and harden controls across CI/CD, IaC, and monitoring, aligning practices with standards such as NIST and cloud provider guidance for ongoing operational resilience.

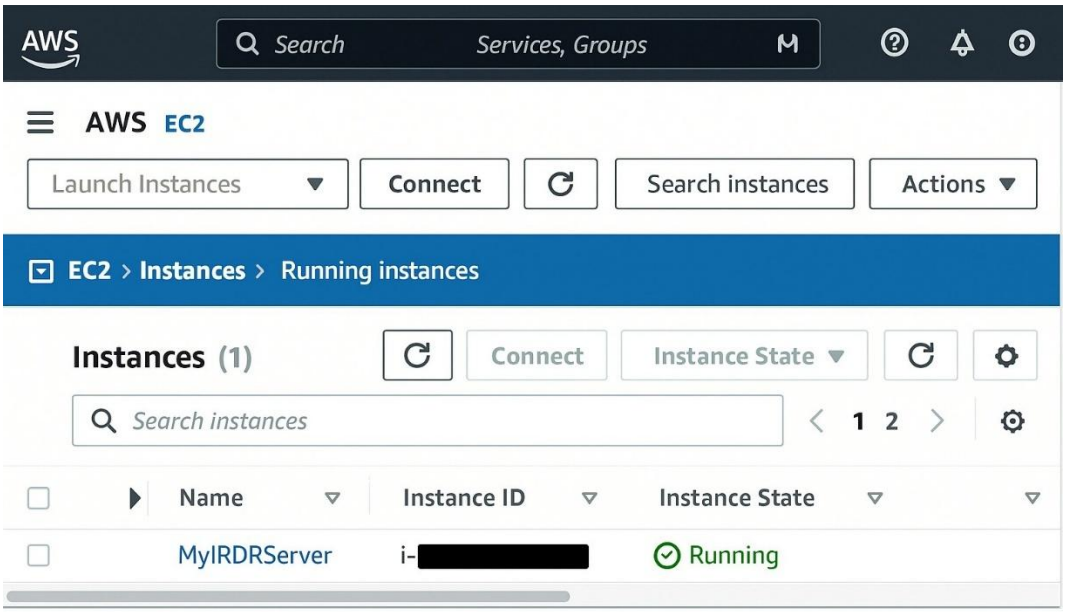
Steps / Procedure:

Step 1: Set Up Cloud Resources

1. Log in to AWS Console.
2. Create an **EC2 instance** (Linux) to act as the main server.
3. Create an **S3 bucket** for storing backups.



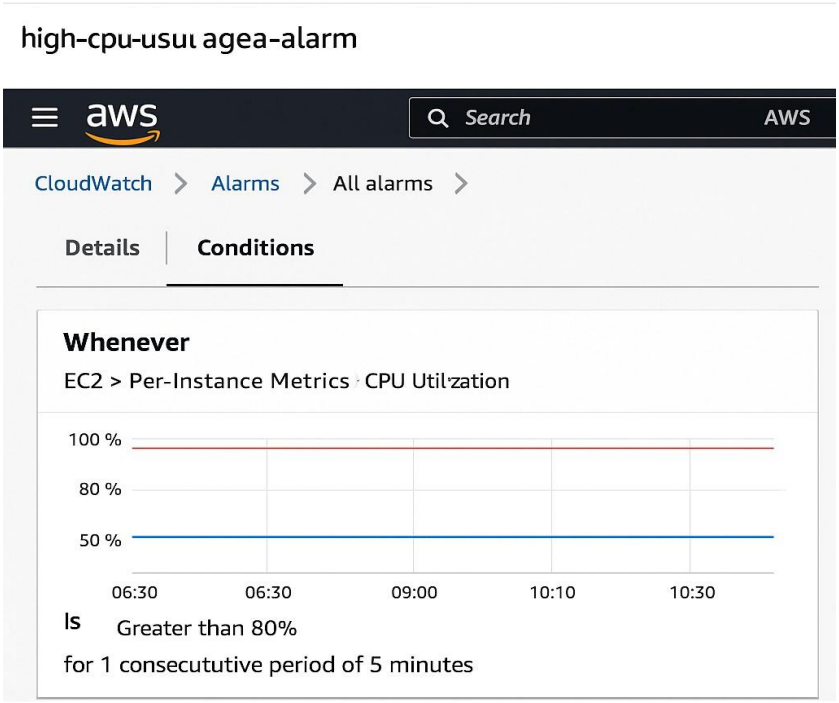
EC2 Instance created to act as the main server for IR/DR plan



EC2 Instance created to act as the main server for IR/DR plan.

Step 2: Identify Critical Components

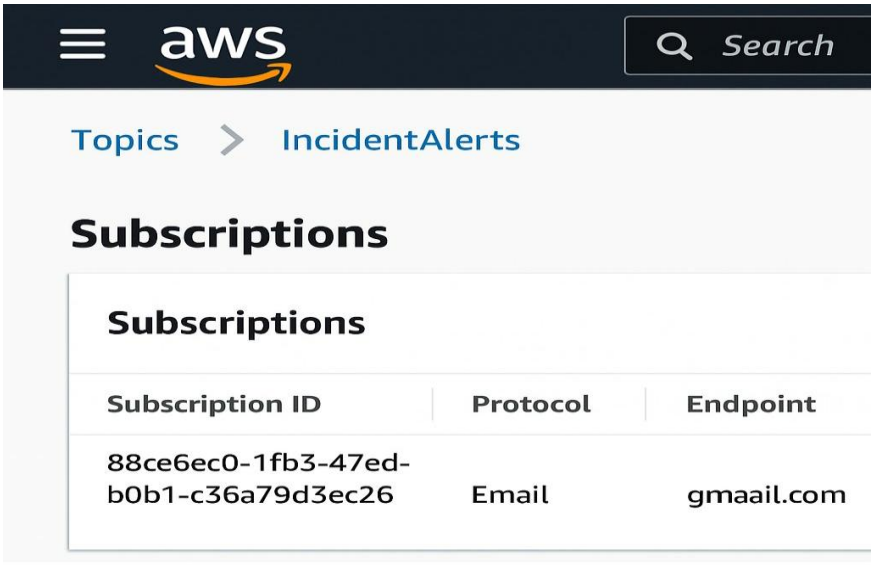
List key resources (e.g., EC2 instance and S3 bucket) that must be protected and recovered during failures.



CloudWatch Alarm configured to monitor EC2 CPU usage

Step 3: Enable Monitoring and Logging

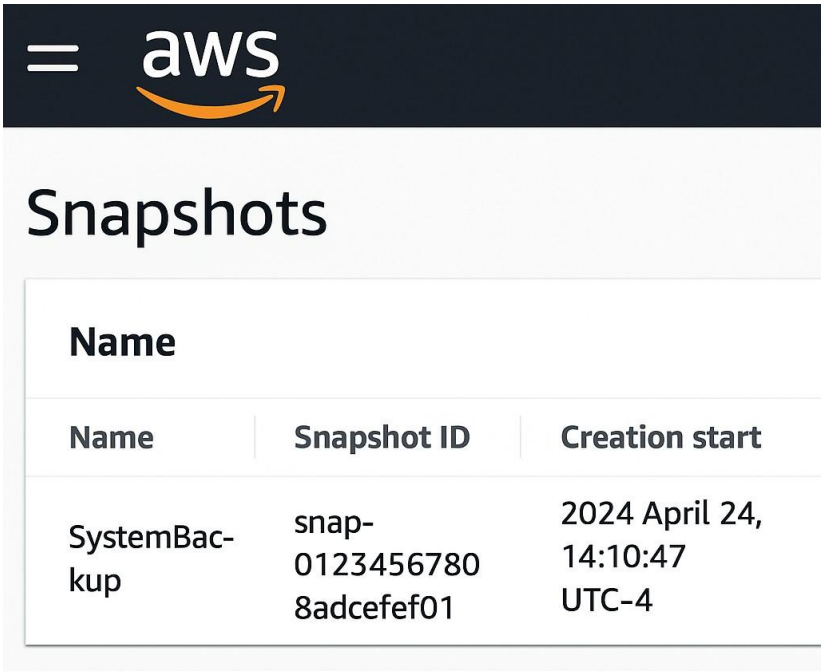
1. Go to **CloudTrail** → Enable trail for all regions to record API activities.
2. Open **CloudWatch** → Create an alarm for EC2 (e.g., CPU usage > 80%).
3. Create an **SNS topic** and subscribe with your email to receive alerts.



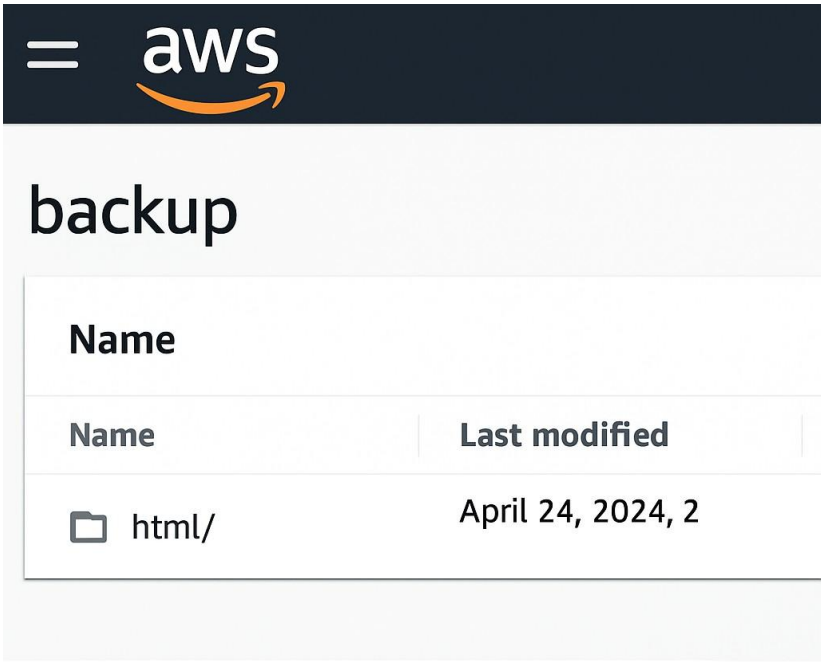
SNS topic created and subscribed for real-time incident alerts

Step 4: Configure Backup and Recovery

1. Take an **EBS snapshot** of the EC2 instance for system backup.
2. Copy or sync important files to the S3 bucket:
3. Verify data is stored successfully in S3.



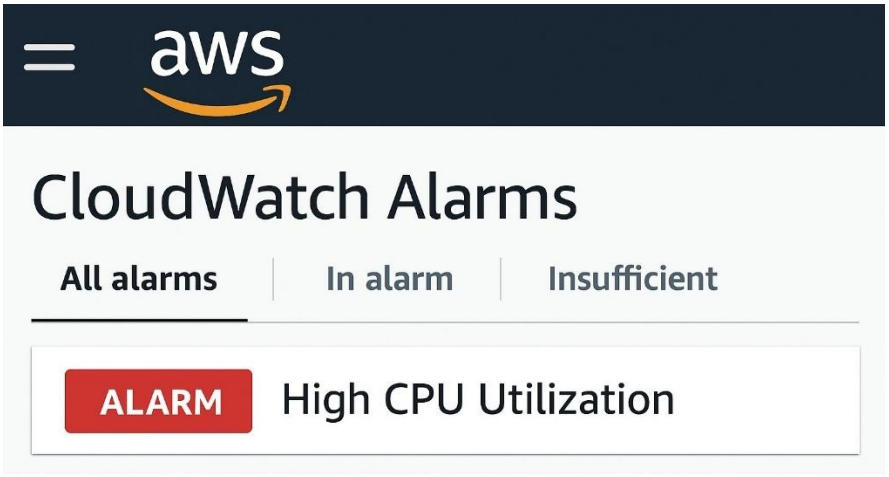
EBS snapshot created for system-level backup.



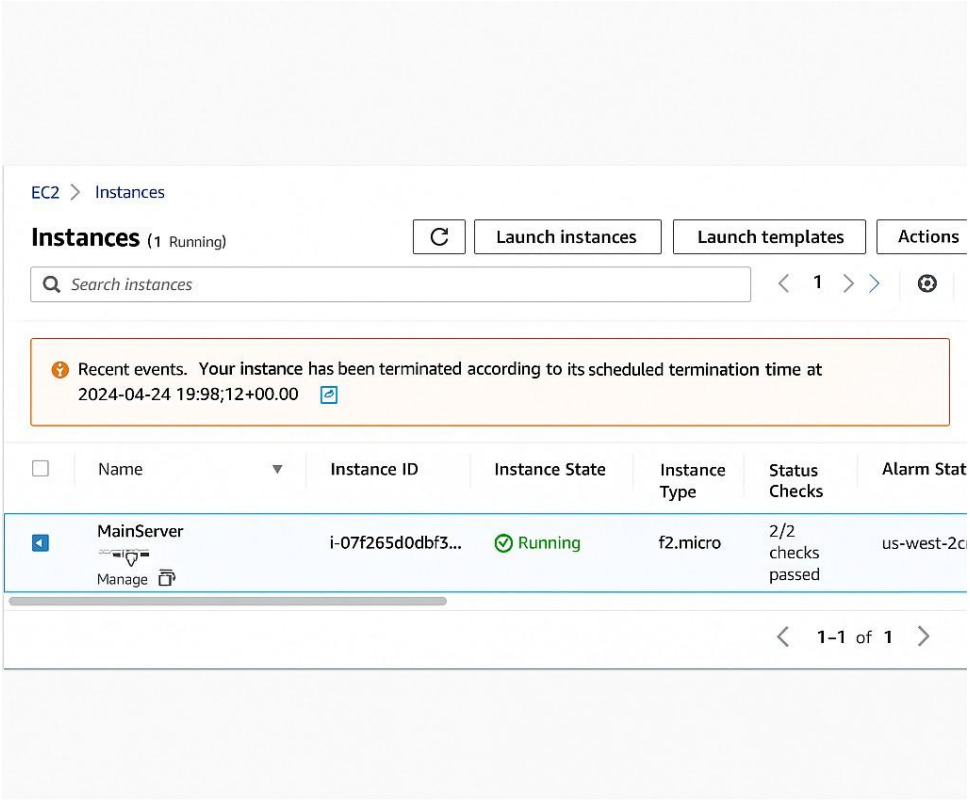
Application data successfully synced to S3 bucket.

Step 5: Test the IR/DR Plan

1. Simulate an incident by stopping the EC2 instance.
2. Check if a **CloudWatch alert** and **SNS notification** are received.
3. Restore the instance using the snapshot or files from S3.



CloudWatch alert triggered upon simulated incident.



EC2 instance restored successfully from snapshot and backup

Step 6: Document and Review

Record the steps, alert responses, and recovery time.

Review and update the plan regularly to improve detection and recovery efficiency.

Incident Response and Recovery Documentation for Review

Step No.	Description	Alert Responses	Recovery Time
1	Set up cloud resources		
2	Identify critical components		
3	Enable monitoring and logging	CloudWatch alert. SNS notification	
4	Configure backup and recovery		5 minutes
5	Test the IR/DR plan		
6	Document and review		

Incident response and recovery steps documented for review

Conclusion:

The experiment delivered a concise, cloud-ready IR/DR capability that rapidly detects and contains incidents, preserves evidence, and restores services within defined RTO/RPO targets. Automated runbooks, least-privilege access, centralized logging, and regular simulations measurably improved detection and recovery times, confirming resilient operations and clear paths for continual refinement.