

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.

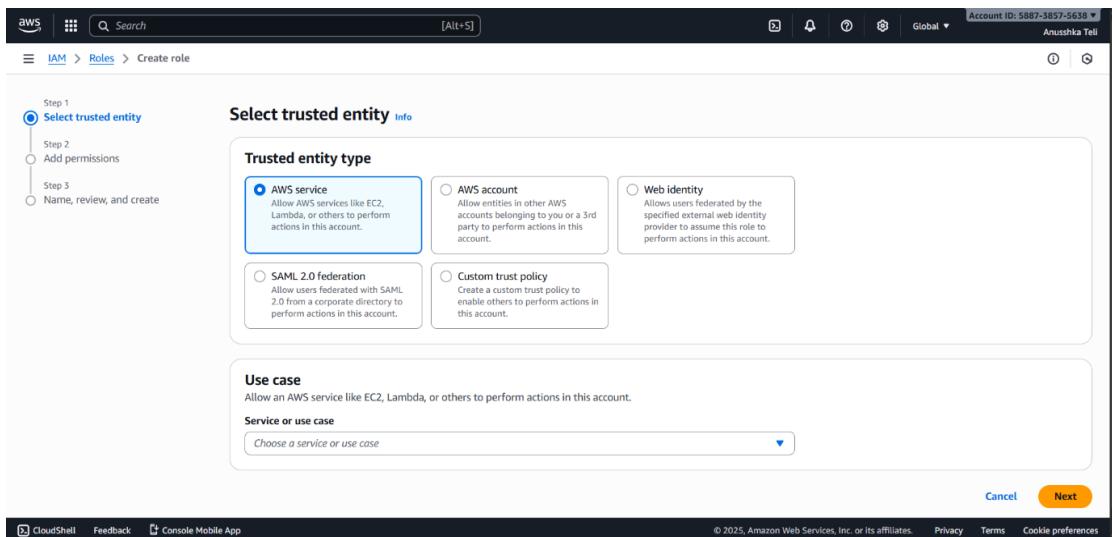
Steps / Procedure

Step 1: Log in to the Cloud Console

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

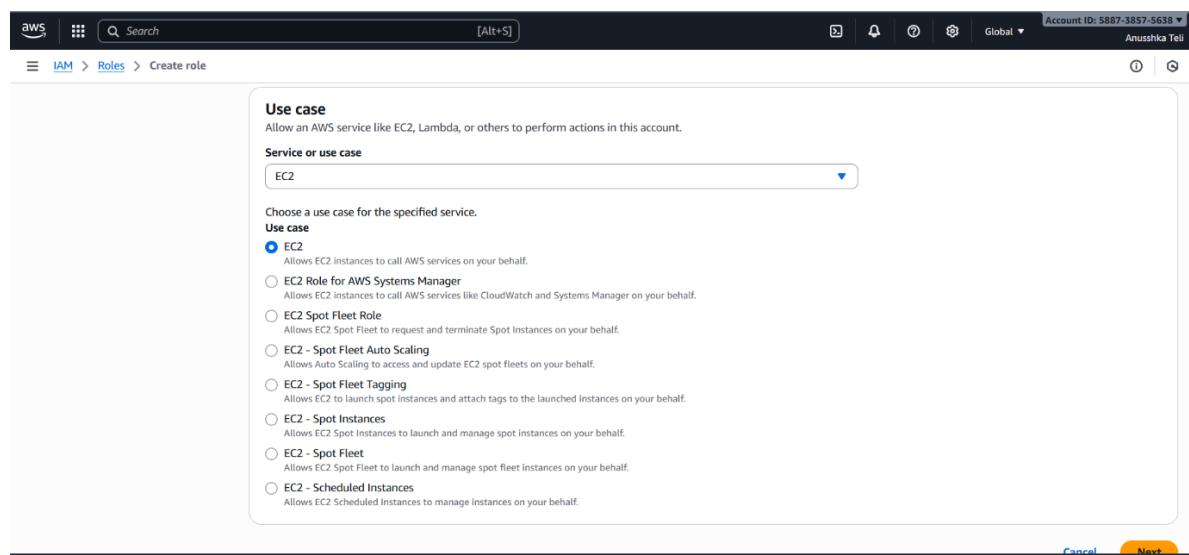
Step 2: Navigate to the IAM Section

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



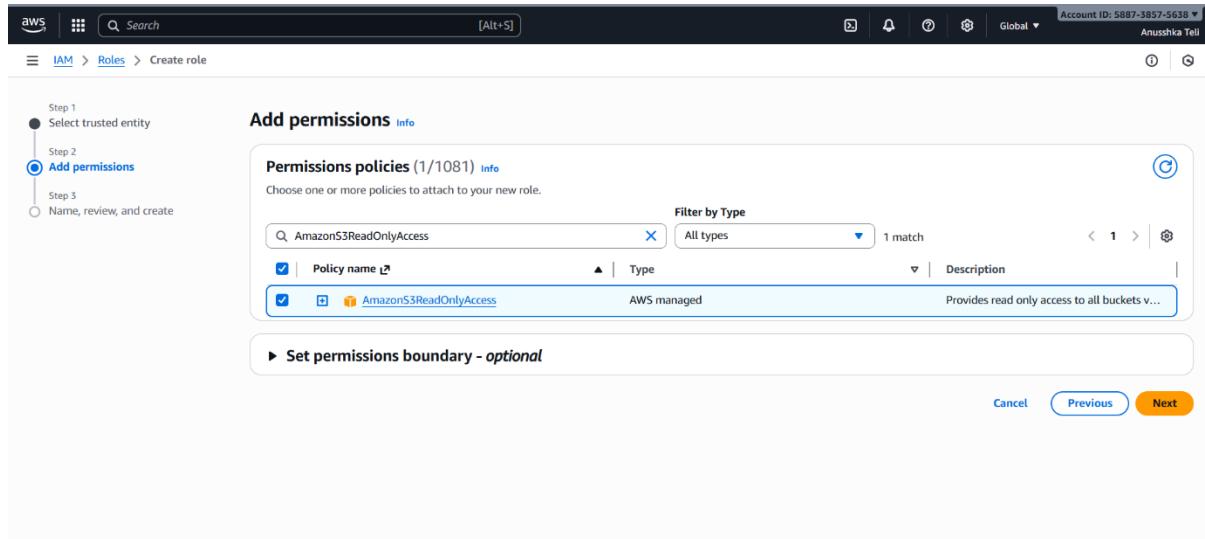
Step 3: Create a New Role

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



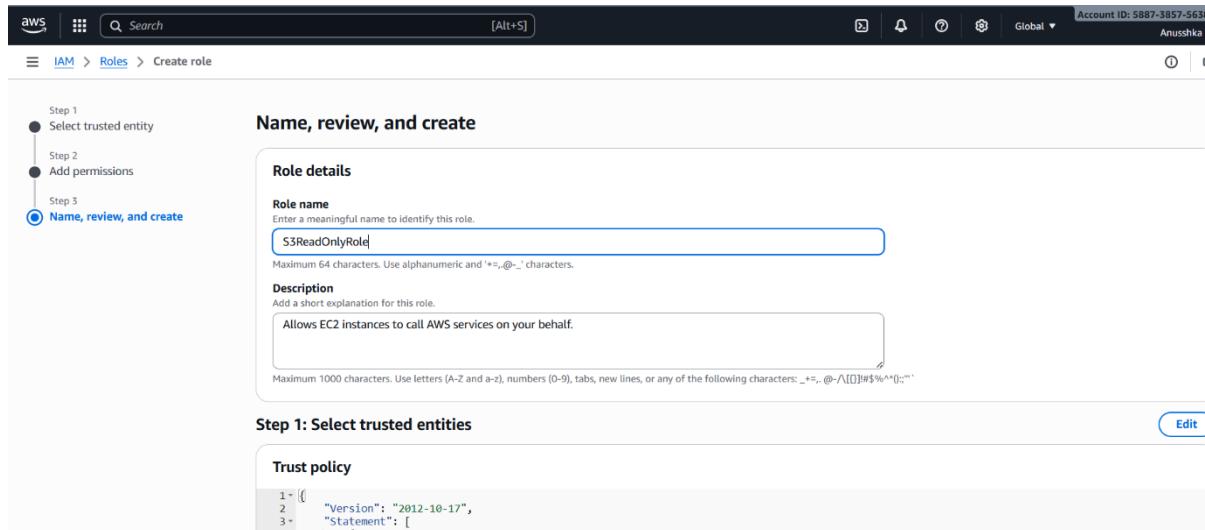
Step 4: Attach Policies

- 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)**.

Instance summary for i-097078e3ec6da33d0 (CloudSecurityLabInstance) [Info](#)

Updated less than a minute ago

Instance ID	i-097078e3ec6da33d0	Public IPv4 address	13.60.6.98 open address
IPv6 address	-	Instance state	Running
Hostname type	IP name: ip-172-31-22-42.eu-north-1.compute.internal	Private IP DNS name (IPv4 only)	ip-172-31-22-42.eu-north-1.compute.internal
Answer private resource DNS name	IPv4 (A)	Instance type	t3.micro
Auto-assigned IP address	13.60.6.98 [Public IP]	VPC ID	vpc-0c31f776f190b854c
IAM Role	S3ReadOnlyRole	Subnet ID	subnet-0edc83d5f622782ad
IMDv2	Required	Instance ARN	arn:aws:ec2:eu-north-1:588738575638:instance/i-097078e3ec6da33d0

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

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.

Successfully created bucket "my-cloudsecurity-lab-bucket"
To upload files and folders, or to configure additional bucket settings, choose [View details](#).

General purpose buckets All AWS Regions [View details](#)

Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
my-cloudsecurity-lab-bucket	Europe (Stockholm) eu-north-1	November 7, 2025, 22:11:08 (UTC+05:30)

Account snapshot [Info](#) [View dashboard](#)
Updated daily
Storage Lens provides visibility into storage usage and activity trends.

External access summary - new [Info](#)
Updated daily
External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

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

```

Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Fri Nov 7 16:29:30 2025 from 13.48.4.202
[ec2-user@ip-172-31-22-42 ~]$ curl https://169.254.169.254/latest/meta-data/iam/info
[ec2-user@ip-172-31-22-42 ~]$ aws s3 ls
[ec2-user@ip-172-31-22-42 ~]$ aws s3 ls
2025-11-07 16:41:09 my-cloudsecurity-lab-bucket
[ec2-user@ip-172-31-22-42 ~]$ echo "test file" > test.txt
[ec2-user@ip-172-31-22-42 ~]$ aws s3 cp test.txt s3://your-bucket-name/
upload failed: ./test.txt to s3://your-bucket-name/test.txt An error occurred (AccessDenied) when calling the PutObject operation: Access Denied
[ec2-user@ip-172-31-22-42 ~]$ 

```

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:
 - o Select **default VPC**.
 - o Create a **new Security Group** → Allow SSH (port 22).

7. Launch the instance.

Name and tags

Name: SharedModel-instance

Application and OS Images (Amazon Machine Image)

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose Browse more AMIs.

Recent AMIs: Amazon Linux, macOS, Ubuntu, Windows, Red Hat, SUSE Linux, Debian.

Description

Amazon Linux 2023 (kernel-6.1) is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed for the cloud.

Summary

Number of instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.9.2... read more

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with a new AMI, 750 hours of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet. Data transfer charges are not included as part of the free tier allowance.

Launch instance

Inbound Security Group Rules

Security group rule 1 (TCP, 22, 0.0.0.0/0)

Type: ssh, Protocol: TCP, Port range: 22, Source type: Anywhere, Description: e.g. SSH for admin desktop.

Security group rule 2 (TCP, 80)

Type: HTTP, Protocol: TCP, Port range: 80, Source type: Custom, Description: e.g. SSH for admin desktop.

Summary

Number of instances: 1

Software Image (AMI): Amazon Linux 2023 AMI 2023.9.2... read more

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with a new AMI, 750 hours of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet. Data transfer charges are not included as part of the free tier allowance.

Launch instance

A screenshot of the AWS EC2 Instances page. The top navigation bar shows tabs for Instances, subnets, VPC Console, and VPC. The main content area displays a table titled 'Instances (1) Info' with one row. The row details a single instance: Name is 'i-03c2892a270e67ef9', Instance ID is 'i-03c2892a270e67ef9', Instance state is 'Running', Instance type is 't2.micro', Status check is 'Initializing', View alarms button is present, Availability Zone is 'us-east-1d', Public IPv4 DNS is 'ex2-174-129-89-101.co...', and Public IPv4 IP is '174.129.89.101'. The left sidebar shows navigation links for EC2, Dashboard, Events, Instances, Instance Types, and Launch Templates.

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.

A screenshot of the AWS Security Groups Edit inbound rules page. The top navigation bar shows tabs for Instances, subnets, VPC Console, and VPC. The main content area displays a table titled 'Inbound rules' with one rule listed. The rule details: Security group rule ID is 'sgr-0fa1bd4f8cb7c4226', Type is 'SSH', Protocol is 'TCP', Port range is '22', Source is 'My IP' (with dropdown options: Custom, Anywhere-IPv4, Anywhere-IPv6, My IP), and Description is optional. At the bottom are 'Cancel', 'Preview changes', and 'Save rules' buttons.

3. Enable **IAM Role** for EC2:

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

A screenshot of the AWS Security Groups Edit inbound rules page, identical to the previous one but with a different URL in the address bar: 'us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-01a1ae2cf9a420f96'. The interface and rule details are the same as the first screenshot.

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

Select trusted entity

Trusted entity type

- AWS service
- AWS account
- Web identity
- SAML 2.0 federation
- Custom trust policy

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Use case
Choose a use case for the specified service.

- EC2
- EC2 Role for AWS Systems Manager
- EC2 Spot Fleet Role
- EC2 - Spot Fleet Auto Scaling
- EC2 - Spot Fleet Tagging

Identity and Access Management (IAM)

ec2

Role ec2 created.

ec2

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date
November 09, 2025, 10:20 (UTC+05:30)

Last activity
-

ARN
arn:aws:iam::923789129287:role/ec2

Instance profile ARN
arn:aws:iam::923789129287:instance-profile/ec2

Permissions

Permissions policies (1)

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonS3ReadOnlyAccess	AWS managed	1

Permissions boundary (not set)

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.

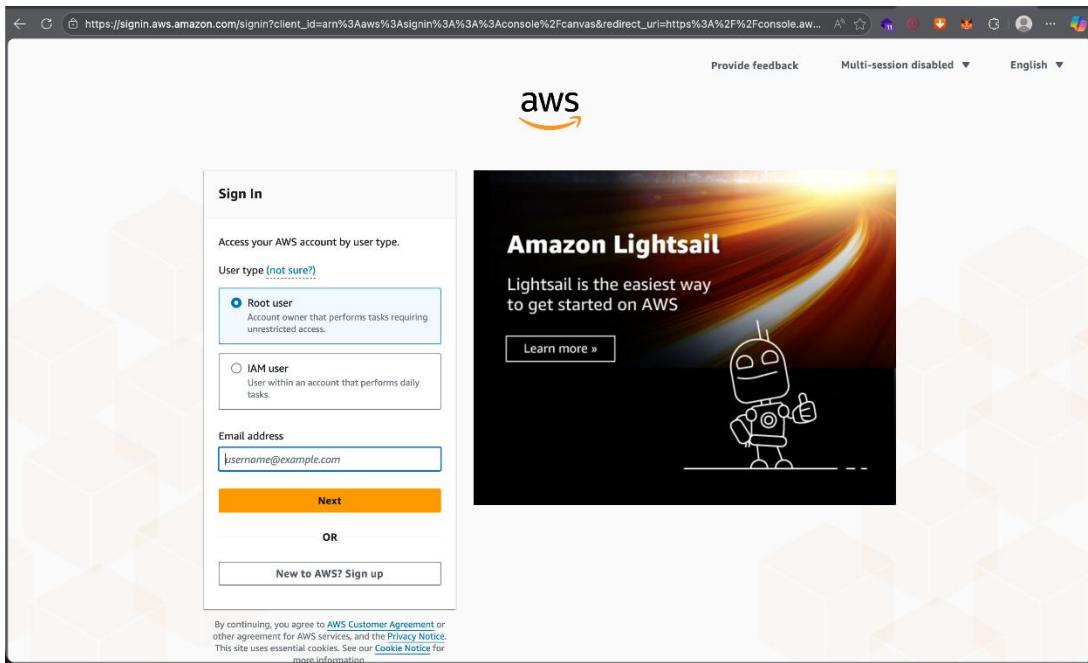
Benefits of MFA:

- 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.



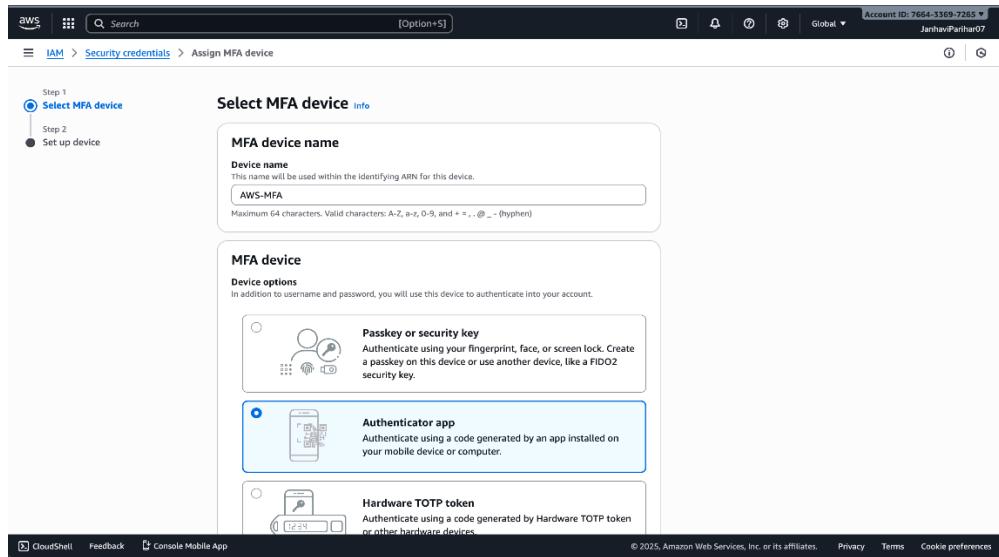
Step 2: Navigate to Security Settings / IAM

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

A screenshot of the AWS IAM Dashboard. The left sidebar shows navigation links for 'Identity and Access Management (IAM)', 'Dashboard', 'Access management', 'Access reports', and 'CloudShell'. The main content area has several sections: 'New access analyzers available' (with a 'Create new analyzer' button), 'IAM Dashboard Info', 'Security recommendations' (with a 'Add MFA for root user' button), 'AWS Account' (showing Account ID: 7664-3369-7265, Account Alias: Create, and a Sign-in URL: https://766433697265.siginin.aws.amazon.com/console), 'Quick Links' (with a 'My security credentials' link), 'Tools' (with a 'Policy simulator' link), and 'What's new' (with a 'View all' button). The 'IAM resources' section shows 0 User groups, 0 Users, 3 Roles, 0 Policies, and 0 Identity providers.

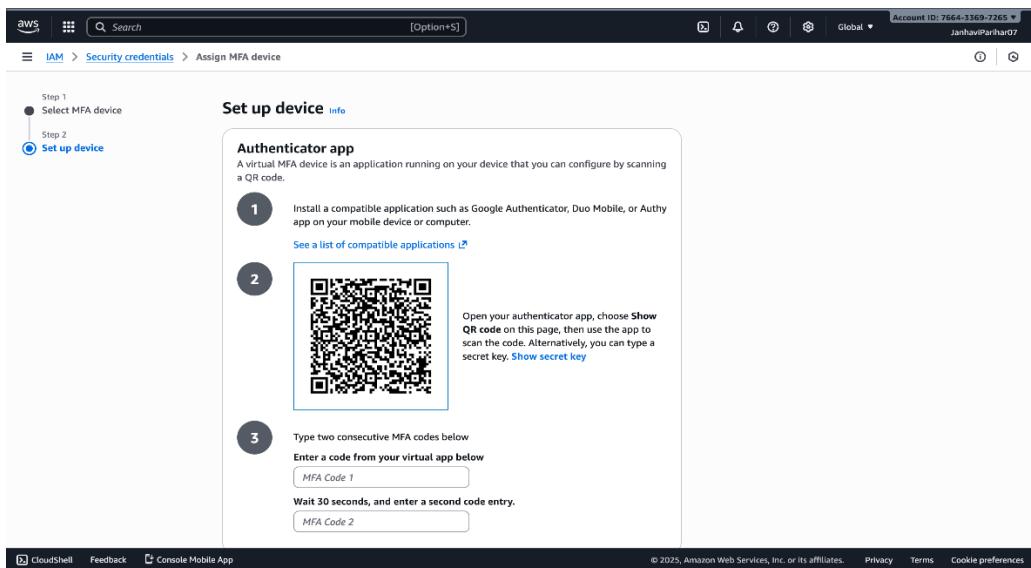
Step 3: Select MFA Setup (AWS Console)

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.



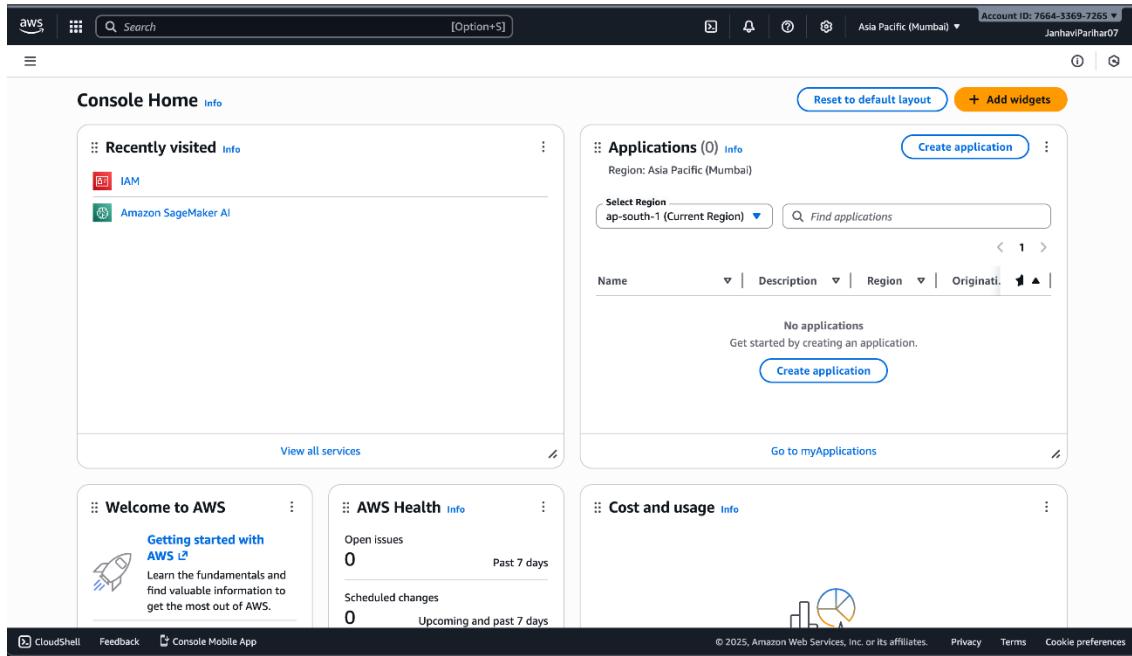
Step 5: Verify MFA Setup

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.”

The screenshot shows the AWS IAM 'My security credentials' page. A green success message box at the top right says 'MFA device assigned'. Below it, the 'Account details' section shows the account name 'Janhaviparihar07', AWS account ID '766433697265', email address 'pariharjanhav7@gmail.com', and canonical user ID '4347b865b489b17914751657ce793a804d77bed3fdb966457ca067c0af2a404f'. The 'Multi-factor authentication (MFA)' section shows one virtual MFA device assigned. The 'Access keys (0)' section shows no access keys. At the bottom, there are links for CloudShell, Feedback, and Console Mobile App.

The screenshot shows the AWS sign-in page. It displays a 'Provide feedback' link and an 'English' language selection. On the left, a 'Additional verification required' box asks the user to enter an MFA code from their device. On the right, there is an advertisement for 'Amazon Lightsail' featuring a cartoon robot character.



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**.

The screenshot shows the 'Create bucket' wizard in the AWS Management Console. The 'General configuration' tab is selected, displaying fields for 'Bucket name' (set to 'amzn-s3-demo-bucket'), 'AWS Region' (set to 'US East (N. Virginia) us-east-1'), and 'Bucket type' (set to 'General purpose'). The 'Object Ownership' tab is also visible below.

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

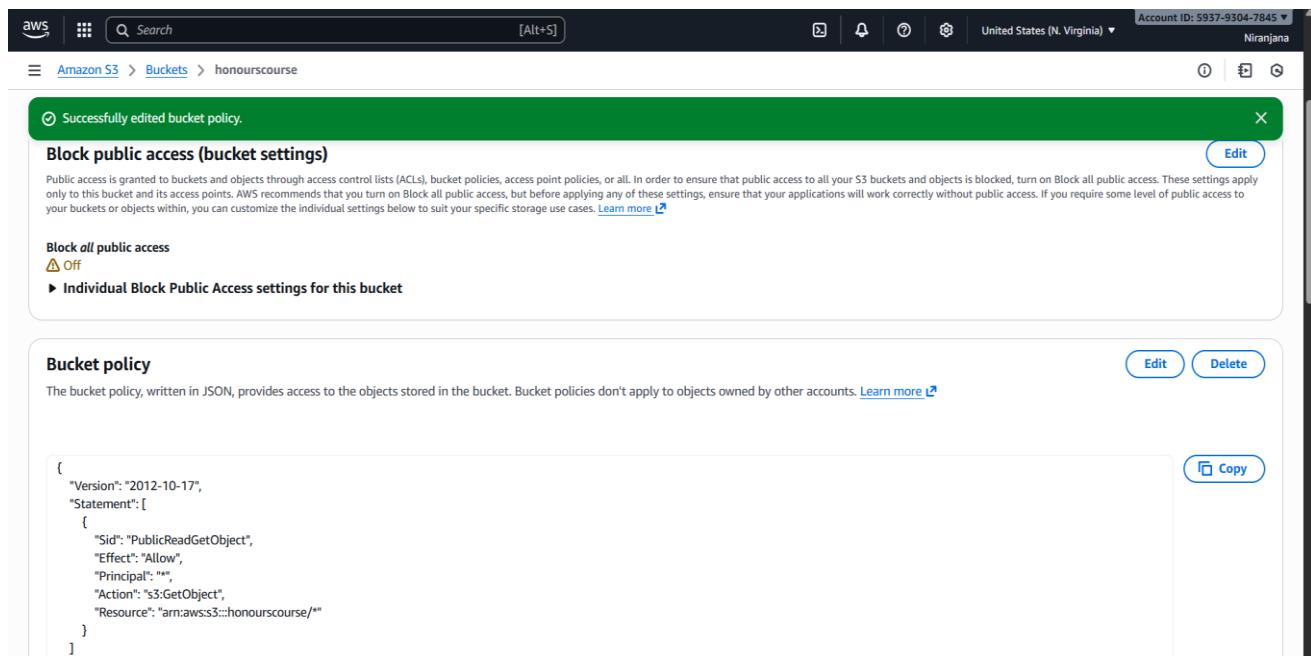
1. Go to the **Properties** tab of your S3 bucket and scroll to **Default encryption**.
2. Select **Enable** and choose **AES-256 (SSE-S3)** for encryption.
3. Save changes.
 - a. This ensures all files uploaded to this bucket are automatically encrypted using AES-256.

The screenshot shows the 'Properties' tab for the 'honourscourse' bucket. Under the 'Default encryption' section, 'AES-256 (SSE-S3)' is selected and enabled. In the 'Intelligent-Tiering Archive configurations' section, there are no configurations listed.

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/*"  
        }  
    ]  
}
```

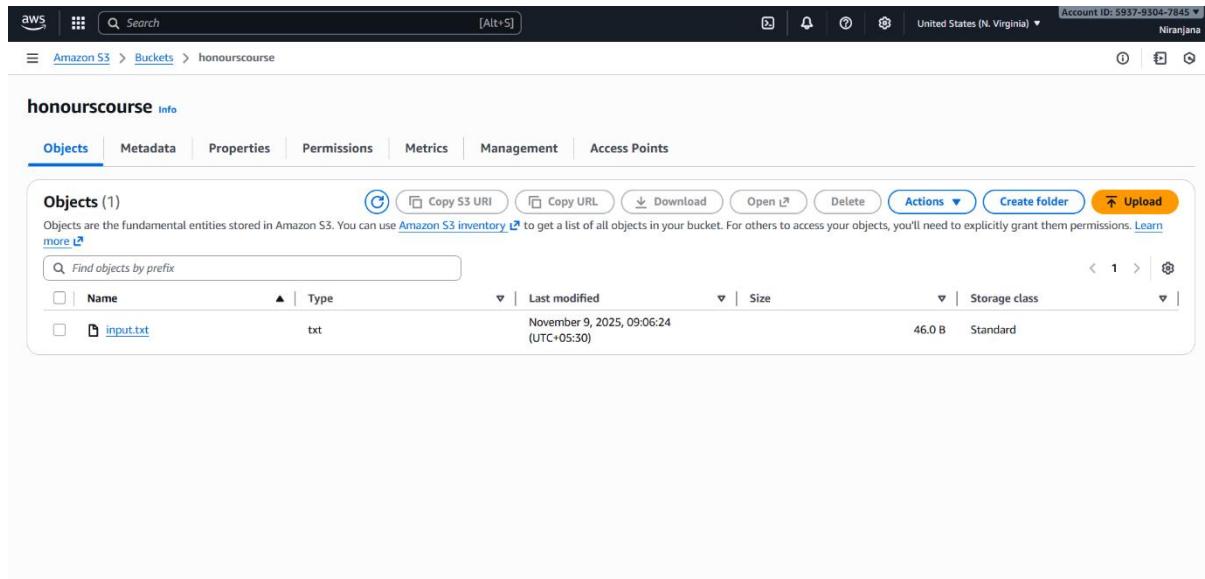


The screenshot shows the AWS S3 Bucket Settings page for a bucket named 'honourscourse'. A green success message at the top states 'Successfully edited bucket policy.' Below it, the 'Block public access (bucket settings)' section has 'Block all public access' set to 'Off'. The 'Individual Block Public Access settings for this bucket' section is collapsed. At the bottom, the 'Bucket policy' section displays the following JSON code:

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

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: T2jGAbhe31NDmXMqUb2RyAiq5v0Bf+gfnQzbFJVdrgbL+FJBAlUJ0IWElXo8GNf1A8tIq0VVMM=
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

The screenshot shows the AWS S3 Bucket Properties page for a bucket named 'honourscourse'. At the top, there's a navigation bar with the AWS logo, search bar, and account information (Account ID: 5937-9304-7845, United States (N. Virginia), Niranjan). Below the navigation, the bucket name 'honourscourse' is displayed. The main content area has several sections:

- Tags**: A table showing one tag: 'Key' (empty) and 'Value' (empty), with a note: 'No tags associated with this resource.'
- Default encryption**: A section explaining server-side encryption is automatically applied to new objects. It includes 'Encryption type' (Info: SSE-S3), 'Bucket Key' (Enabled), and a note about KMS encryption.
- Intelligent-Tiering Archive configurations (0)**: A section for managing archive configurations. It includes a 'Create configuration' button and a table with columns: Name, Status, Scope, Days until transition to Archive Acc..., and Days until transition to Deep Archi.... The table shows 'No archive configurations' and 'No configurations to display.'

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.

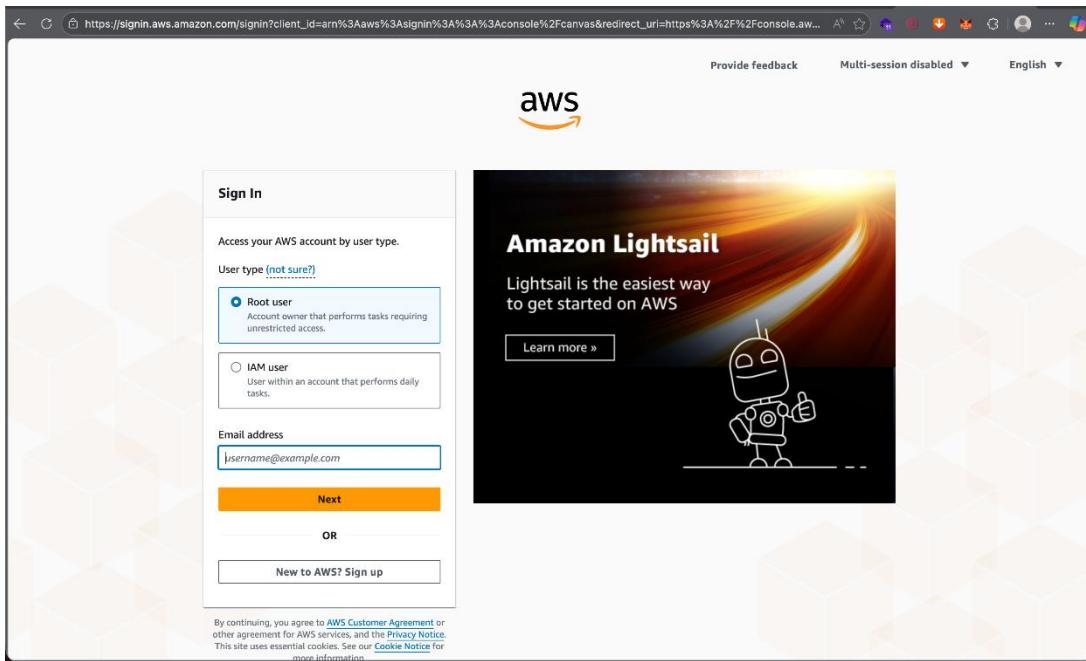
Theory:

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:

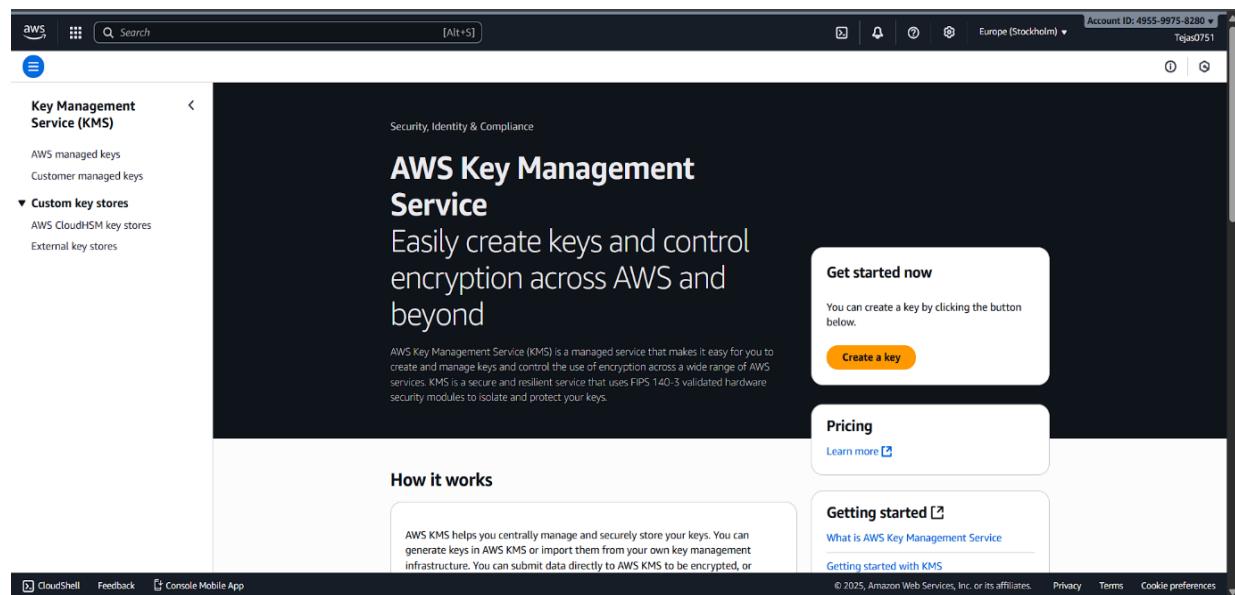
Step 1: Log in to your Cloud Dashboard

- 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.



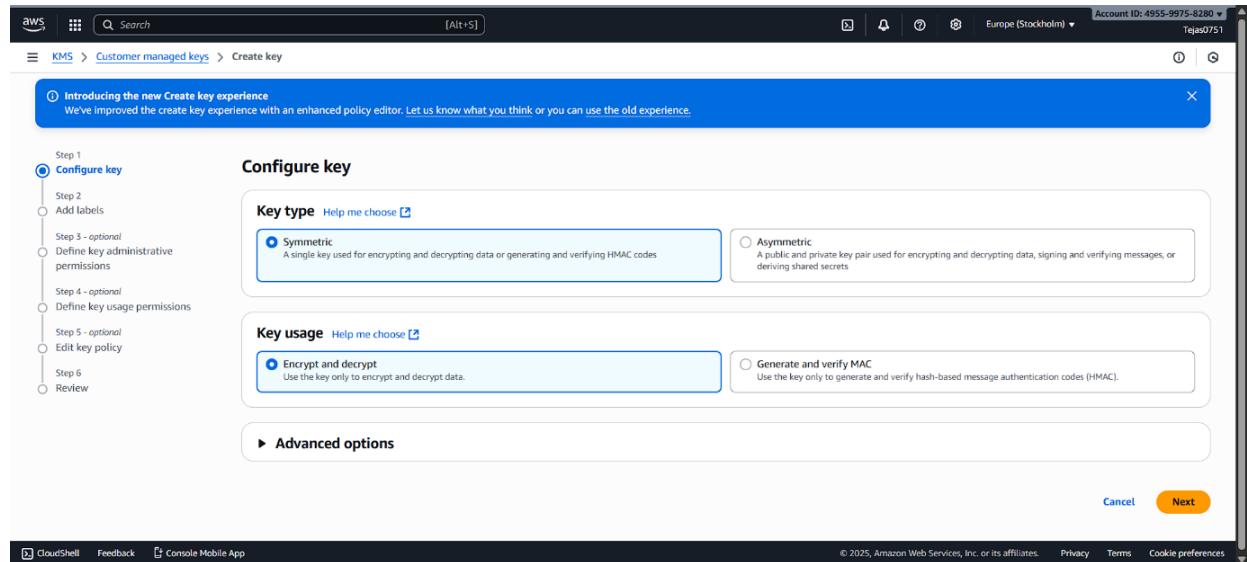
Step 2: Open AWS KMS

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.



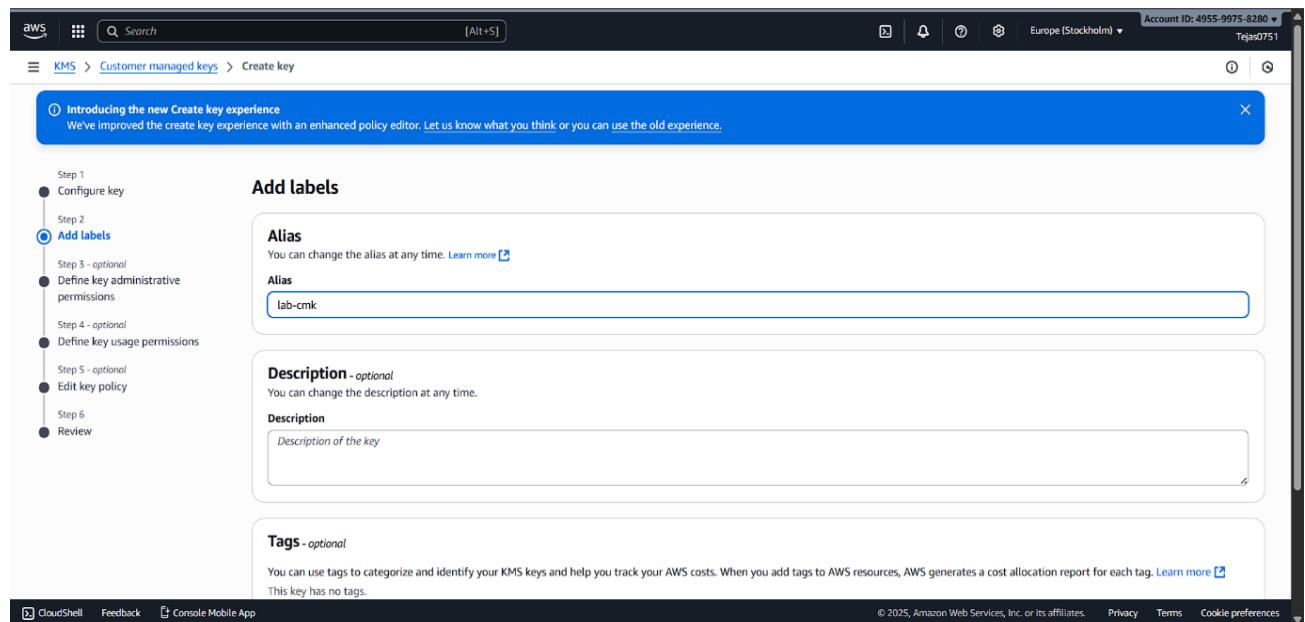
Step 3: Create a Customer Managed Key (CMK)

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.

Define key administrative permissions - optional

Key administrators (1/4)

Select the IAM users and roles authorized to manage this key via the KMS API. These administrators will be added to the key policy under the statement identifier (Sid) 'Allow administration of the key'. Modifying this Sid might impact the console's ability to update the administrator statement in the key policy. [Learn more](#)

Name	Path	Type
<input checked="" type="checkbox"/> tejaspatil75	/	User
<input type="checkbox"/> AWSServiceRoleForResourceExplorer	/aws-service-role/resource-explorer-2.amazonaws.com/	Role
<input type="checkbox"/> AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/> AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role

Key deletion

Allow key administrators to delete this key.

Define key usage permissions - optional

Key users (1/4)

Select the IAM users and roles authorized to use this key in cryptographic operations. These users will be added to the key policy under the statement identifiers (Sids) 'Allow use of the key' and 'Allow attachment of persistent resources'. Modifying these Sids might impact the console's ability to update the user statements in the key policy. [Learn more](#)

Name	Path	Type
<input checked="" type="checkbox"/> tejaspatil75	/	User
<input type="checkbox"/> AWSServiceRoleForResourceExplorer	/aws-service-role/resource-explorer-2.amazonaws.com/	Role
<input type="checkbox"/> AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/> AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

Add another AWS account

Screenshot of the AWS KMS 'Create key' wizard Step 5: Edit key policy - optional.

The sidebar shows steps 1-6: Configure key, Add labels, Define key administrative permissions, Define key usage permissions, Edit key policy (selected), and Review.

Key policy

```

1 {
2   "Id": "key-consolepolicy-3",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "Enable IAM User Permissions",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "arn:aws:iam::495599758280:root"
10      },
11      "Action": "kms:*",
12      "Resource": "*"
13    },
14    {
15      "Sid": "Allow access for Key Administrators",
16      "Effect": "Allow",
17      "Principal": {
18        "AWS": "arn:aws:iam::495599758280:user/tejaspatil75"
19      }
20    }
21  ]
22}

```

Preview | Edit

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS KMS 'Create key' wizard Step 6: Review.

The sidebar shows steps 1-6: Configure key, Add labels, Define key administrative permissions, Define key usage permissions, Edit key policy, and Review (selected).

Review

Key configuration

- Key type: Symmetric
- Key spec: SYMMETRIC_DEFAULT
- Origin: AWS KMS
- Regionality: Single-Region key

You cannot change the key configuration after the key is created.

Alias and description

- Alias: lab-cmk
- Description: -

Tags

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Screenshot of the AWS KMS 'Customer managed keys' page.

The sidebar shows: Key Management Service (KMS), AWS managed keys, Customer managed keys (selected), and Custom key stores (AWS CloudHSM key stores, External key stores).

Success

Your AWS KMS key was created with alias lab-cmk and key ID e31d40d5-0ec4-4ffcc-b500-b6e155440eb9.

Customer managed keys (1)

Aliases	Key ID	Status	Key type	Key spec
lab-cmk	e31d40d5-0ec4-4ffcc-b500-b6e155440eb9	Enabled	Symmetric	SYMMETRIC_DEFAULT

View key | Create key | Key actions | Filter keys by properties or tags | 1 | © 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**.

Key Management Service (KMS)

Key policy | Cryptographic configuration | Tags | **Key material and rotations** | Aliases

Key material and rotations Info

Manage AWS KMS key material and initiate automatic and on-demand key rotations.

Automatic key rotation

AWS KMS automatically rotates the key based on the rotation period that you define.

Status	Rotation period	Date of last automatic rotation	Next rotation date
Disabled	-	-	-

On-demand key rotation

Immediately initiate key material rotation for this key. You can initiate on-demand key rotation a maximum of 10 times.

Number of on-demand rotations remaining
10 rotations available

Key materials (1)

The following list provides a record of all completed key material rotations made on this key.

Filter key materials	Date	Key material state	Rotation type
5d0eff297805bc0e922477d90695078e4de6383ccad29381809ee86777d90ac	-	Current	-

Key Management Service (KMS)

Key policy | Cryptographic configuration | Tags | **Key material and rotations** | Aliases

Edit automatic key rotation

Automatic key rotation

Key rotation

Enable

Disable

Rotation period (in days)

Rotation period defines the number of days between each automatic rotation

365

Enter a rotation period between 90 and 2560 days.

Cancel **Save**

Key Management Service (KMS)

Key policy | Cryptographic configuration | Tags | **Key material and rotations** | Aliases

Key material and rotations Info

Manage AWS KMS key material and initiate automatic and on-demand key rotations.

Automatic key rotation

AWS KMS automatically rotates the key based on the rotation period that you define.

Status	Rotation period	Date of last automatic rotation	Next rotation date
Enabled	365	Nov 09, 2025	Nov 09, 2026

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.

The screenshot shows the AWS CloudTrail service page. At the top right, it displays 'Account ID: 4955-9975-8280' and 'Europe (Stockholm)'. Below the header, there's a dark banner with the text 'Management & Governance' and the 'AWS CloudTrail' logo. The main content area has a dark background with white text. It features a call-to-action button 'Create a trail with AWS CloudTrail' with the sub-instruction 'Get started with AWS CloudTrail by creating a trail to log your AWS account activity.' and a 'Create a trail' button. To the left, there's a section titled 'How it works' with two icons: 'Capture' (cloud with camera) and 'Store' (cloud with bucket). Below these icons, there's descriptive text: 'Record activity in AWS services as AWS CloudTrail events' for Capture, and 'AWS CloudTrail delivers events to the AWS CloudTrail console, Amazon S3 buckets, and optionally Amazon CloudWatch Logs' for Store. On the right side, there are sections for 'Pricing' and 'Getting started' with links to 'What is AWS CloudTrail?', 'How AWS CloudTrail works', and 'Services that integrate with AWS CloudTrail'. At the bottom, there are links for 'CloudShell', 'Feedback', 'Console Mobile App', and standard footer links for '© 2025, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

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.

VPC Dashboard

Create VPC

A VPC is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud.

Name tag [?]
MyVPC

IPv4 CIDR block [?]
10.0.0.0/16

IPv6 CIDR block [?]
 No IPv6 CIDR Block

Tenancy [?]
Default

Cancel **Create VPC**

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 3: Create Subnets

Public Subnet

Subnet name
PublicSubnet

IPv4 CIDR block
10.0.1.0/24

Private Subnet

Subnet name
PrivateSubnet

Cancel **Create Subnet**

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 4: Configure Routing

Route table

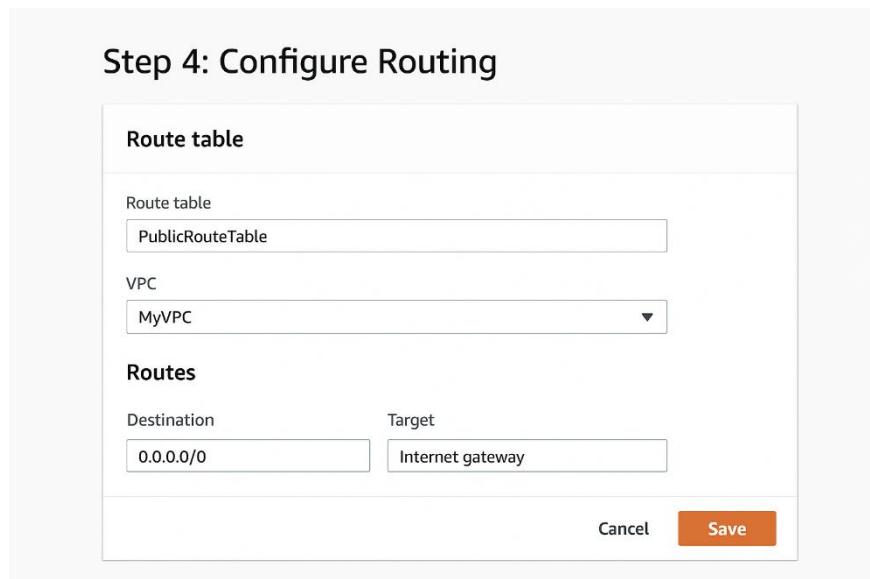
Route table
PublicRouteTable

VPC
MyVPC

Routes

Destination	Target
0.0.0.0/0	Internet gateway

Cancel **Save**



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.

Step 5: Attach Gateways

Create an Internet Gateway and attach it to the VPC

Your VPCs
Subnets
Route Tables
Internet Gateways
Egress Only Internet Gateways
DHCP Options Sets
Elastic IPs
Endpoints
Endpoint Services

Internet gateways

Name	ID	State

Search Internet gateways

Create internet g.



Optionally, create a NAT Gateway

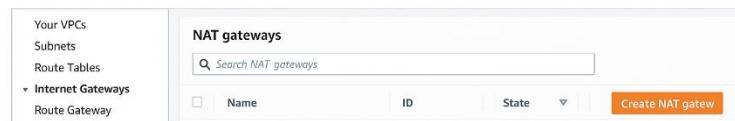
Your VPCs
Subnets
Route Tables
Internet Gateways
Route Gateway

NAT gateways

Name	ID	State

Search NAT gateways

Create NAT gateway



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.

The screenshot shows the AWS CloudFormation interface for Step 6: Implement Security Controls. It displays two main sections: 'Security Group: Public-SG' and 'Network ACL: Public-ACL'.

Security Group: Public-SG

Type	Source	Protocol	Description
SSH	0.0.0.0/0	TCP	Allow inbound SSH
RDP	0.0.0.0/0	TCP	Allow inbound RDP
		3389	Allow all outbound

Outbound Rules

Type	Destination
*	All traffic

Network ACL: Public-ACL

Rule #	Type	Source	Protocol	Port Range	Allow/Deny
100	All traffic	0.0.0.0/0	All	All	Allow
*	All traffic	0.0.0.0/0	All	All	Deny

Buttons: Cancel, Save

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:

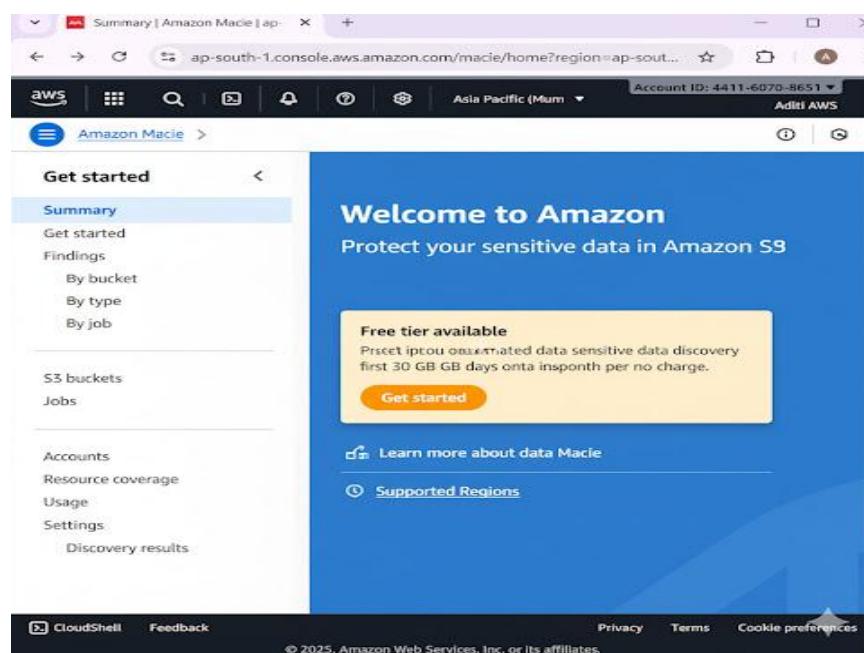
- Storage (data at rest)
- Transmission (data in motion)
- Processing (data in use)

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.

The screenshot shows the 'Automated sensitive data discovery' section of the Amazon Macie console. On the left, there's a sidebar with options like 'Summary', 'Findings', 'S3 buckets', 'Jobs', 'Accounts', 'Resource coverage', 'Usage', 'Settings', and 'Discovery results'. The main area is titled 'Automated sensitive data discovery' with a 'Review and create' button. It has two sections: 'Target S3 buckets' and 'Managed data identifiers'. In 'Target S3 buckets', 'sensitive-data-storage' and 'sensitive-dai-records' are checked. In 'Managed data identifiers', 'SSN (US Security Number)' is checked, while 'Credit card numbers', 'Email addresses', 'API keys', and 'AWS keys' are unchecked. A scroll bar is visible on the right side of the main content area.

3. Run full scan on existing storage/data.

This screenshot is identical to the one above, showing the 'Automated sensitive data discovery' section of the Amazon Macie console. The main difference is a vertical scroll bar is present at the bottom of the main content area, indicating more content is available below what is currently visible.

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

The screenshot shows the 'Configure alert and response rule' dialog box overlaid on the Amazon Macie console. The dialog has tabs for 'Event pattern' (set to 'Macie Finding') and 'Target' (with 'SSN (US Security Number)' checked). Below the target is a code editor containing a Lambda function. The code is as follows:

```
1 // Generated by AWS Lambda Function Generator
2 // For more information, see https://aws.amazon.com/lambda/functions/
3 // FTL=sn=1=t2ne(611)
4 // FTL=1111, count(4,format('1111ng11-n1-5999'));
5 // FTL=1111, count(4,format('1111ng11-n1-5999'));
6 // FTL=1111, count(4,format('1111ng11-n1-5999'));
7 // FTL=1111, count(4,format('1111ng11-n1-5999'));
8 // FTL=1111, count(4,format('1111ng11-n1-5999'));
9 // FTL=1111, count(4,format('1111ng11-n1-5999'))
```

At the bottom of the dialog are 'Cancel' and 'Create' buttons.

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

Severity	S3 bucket	Object key	Last updated	Count
High	test.pii.csv		1	0
Medium	SS bucket	SS bucket	1	0
Low	Sensitive data	test.pii.csv	1	0
Medium	Sensitive data: PII		1	1
Low	Sensitive data: PII		1	2
Medium	Sensitive Unencrypted Number		1	2
Medium	Medium Unencrypted Credentials		1	2
Medium	Medium Unencrypted bucket		1	2
Low	Sensitive Unencrypted Number		1	2
Medium	Prodioni fitacoy ced st filor		1	2
Medium	Policy: Unencrypted bucket		1	8
Low	Sensitivity: Unencrypted Number		1	9
Low	Pigliest dest pil aed		1	10
Low	Drojuitre uta ders		1	15

Sensitive data

- Credit card numbers: 1
- SSN (US Security Number): 2
- Email addresses: 5

Affected resources

Abfencs
Arpaord speisuple durows of
640035002

Finding

25

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)

The screenshot shows the AWS VPC console interface. On the left, there's a navigation sidebar with options like 'Virtual private cloud', 'Route tables', 'Subnets', and 'Route tables'. The main area displays the 'Details' tab for a VPC named 'vpc-04ce1a4cab64ff5f4'. It shows the VPC ID, state (Available), tenancy (default), and network ACL information. Below this is a 'Resource map' section with four boxes: 'VPC' (Your AWS virtual network), 'Subnets (4)' (Subnets within this VPC, including us-east-1a and us-east-1b), 'Route tables (4)' (Route tables connecting subnets to resources), and 'Network Connections (2)' (Connections to other networks). At the bottom, there's a toolbar with various icons and a status bar showing the date and time.

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

The screenshot shows the AWS EC2 Instances page. The left sidebar includes sections for 'Instances', 'Images', 'Elastic Block Store', and 'Network & Security'. The main content area shows a table for 'Instances (1)'. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, Public IPv4 DNS, Public IPv4 IP, and Elastic IP. One row is listed: 'Test-EC2' with Instance ID 'i-095fb55d54eca00f4', State 'Pending', Type 't2.micro', and AZ 'us-east-1a'. Below the table, there's a 'Select an instance' dropdown menu. The bottom of the screen features a toolbar and a status bar.

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).

The screenshot shows the AWS GuardDuty Summary page. On the left, a sidebar lists 'GuardDuty' under 'Summary', followed by 'Findings' and 'EC2 malware scans'. Below this is a section for 'Protection plans' which includes S3 Protection, EKS Protection, Extended Threat Detection (marked as 'New'), Runtime Monitoring, Malware Protection for EC2, Malware Protection for S3, RDS Protection, and Lambda Protection. Further down are sections for 'Accounts', 'Usage', 'Settings', 'Lists' (with a 'New' badge), 'What's New', 'Partners', 'Security Hub' (with a 'New' badge), and 'CloudShell' and 'Feedback' links. The main content area has a title 'Summary' with an 'Info' link. It displays an 'Overview' card with metrics: 0 attack sequences - new, 1 total finding, 1 resource with findings, and 1 account with findings. Below this is a 'Findings' card showing 0 Critical, 0 High, 0 Medium, and 1 Low severity finding. A button 'Top threats' is highlighted. To the right of the findings is a 'Runtime Monitoring coverage' card featuring a bell icon and a link to 'Enable Runtime Monitoring'. A modal window titled 'Introducing the new AWS Security Hub - public preview' explains the transition to AWS Security Hub, mentioning critical issues and runtime monitoring. At the bottom of the page, there are standard browser navigation buttons, a search bar, and a footer with copyright information and a date (08-11-2025).

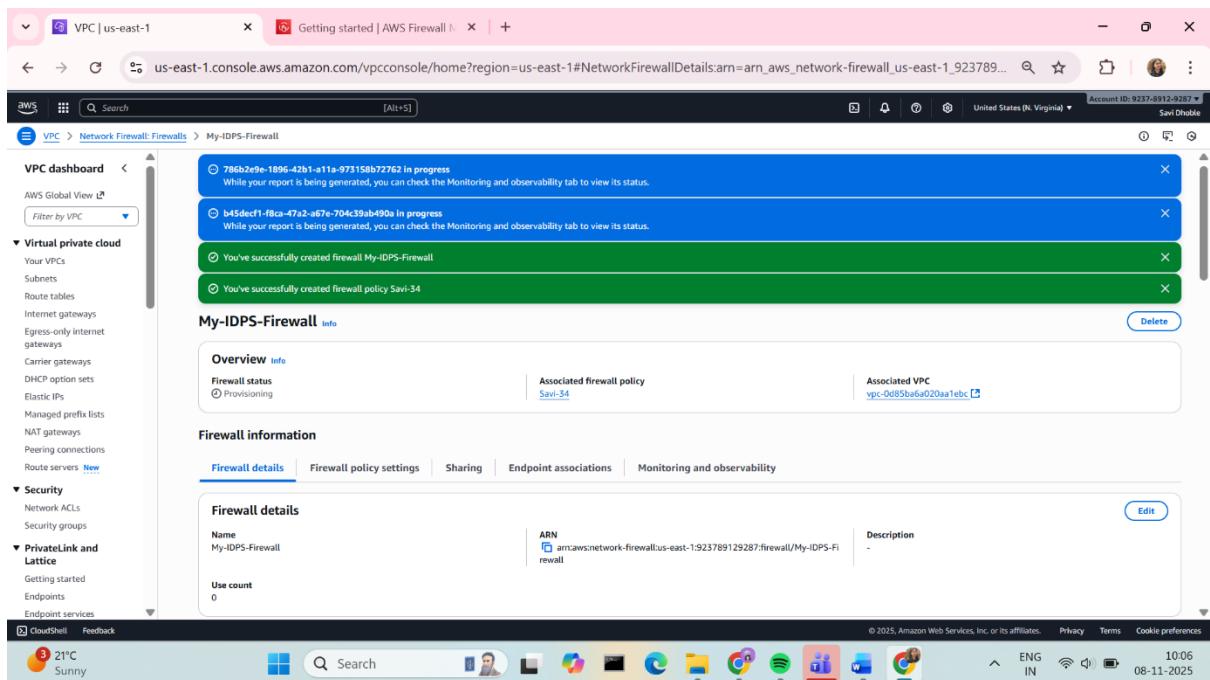
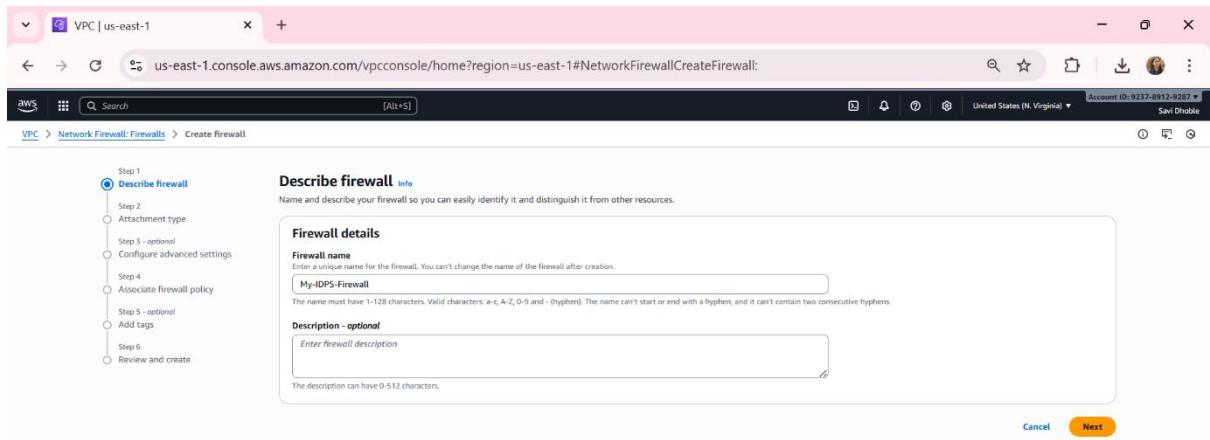
Step 4: Create AWS Network Firewall (Intrusion Prevention System - IPS)

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.

The screenshot shows the AWS GuardDuty Summary page for the us-east-1 region. The left sidebar includes sections for Protection plans (S3, EKS, Extended Threat Detection), Runtime Monitoring (Malware Protection for S3, RDS, Lambda), Accounts, Usage, Settings, and What's New. The main content area displays an Overview with 1 total finding, 1 resource with findings, and 1 account with findings. It also shows a Findings section with 0 Critical, 0 High, 0 Medium, and 1 Low severity finding, and a note about a root credential usage. A banner introduces the AWS Security Hub public preview. The bottom right corner shows the date as 08-11-2025.

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.

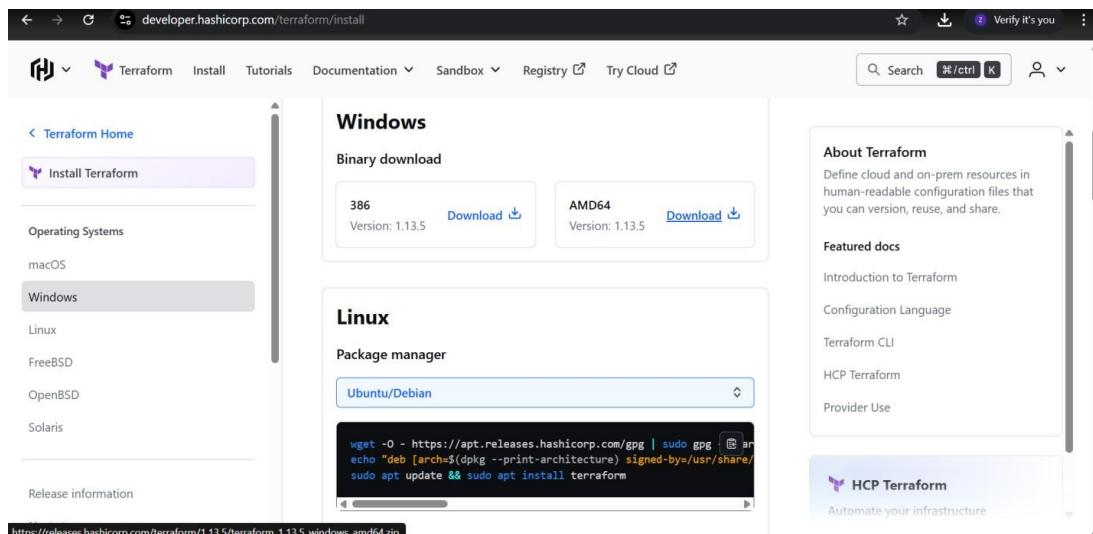
Security in Terraform:

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

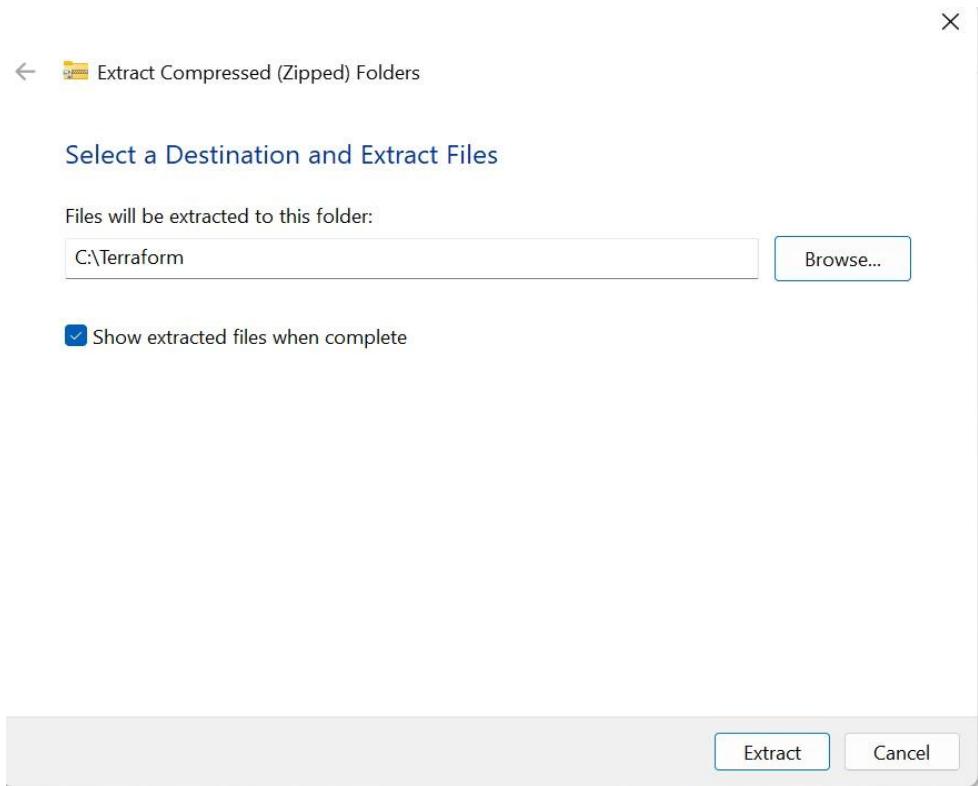
Steps / Procedure:

Step 1: Install Terraform

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

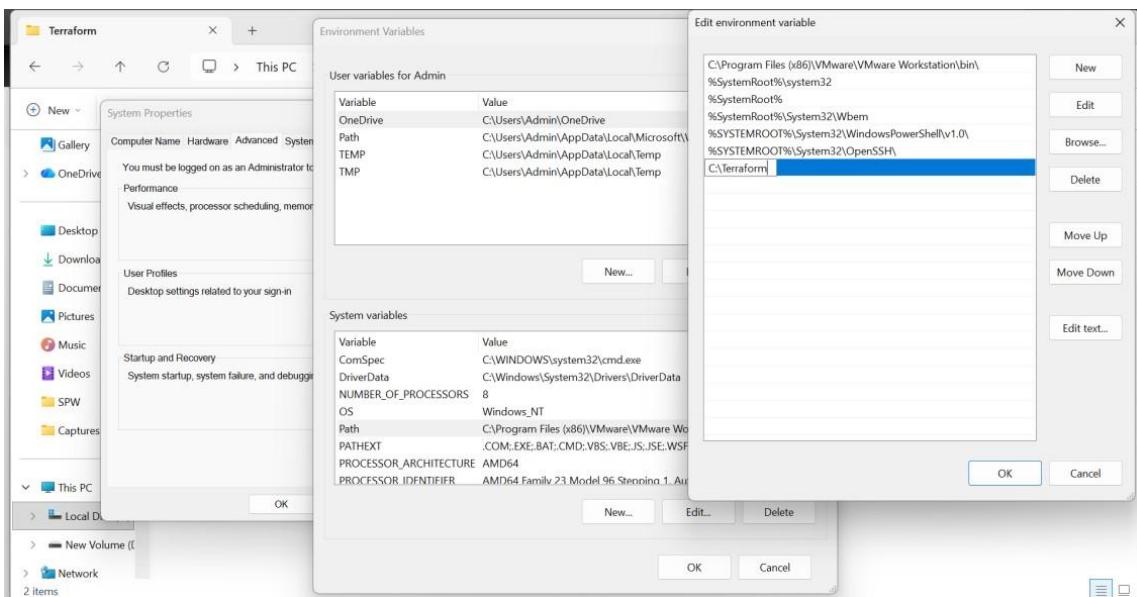


2. Extract the downloaded zip file to a folder.



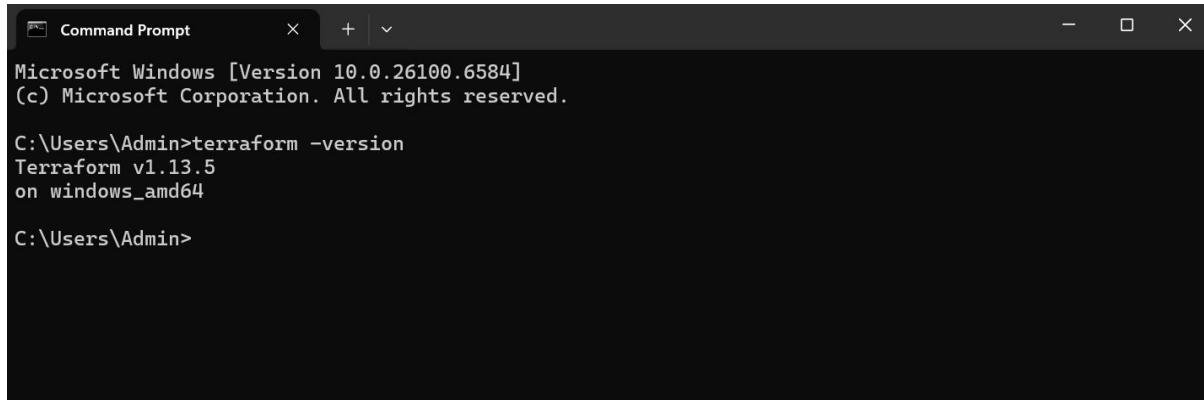
3. Add Terraform to system PATH:

- 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**



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

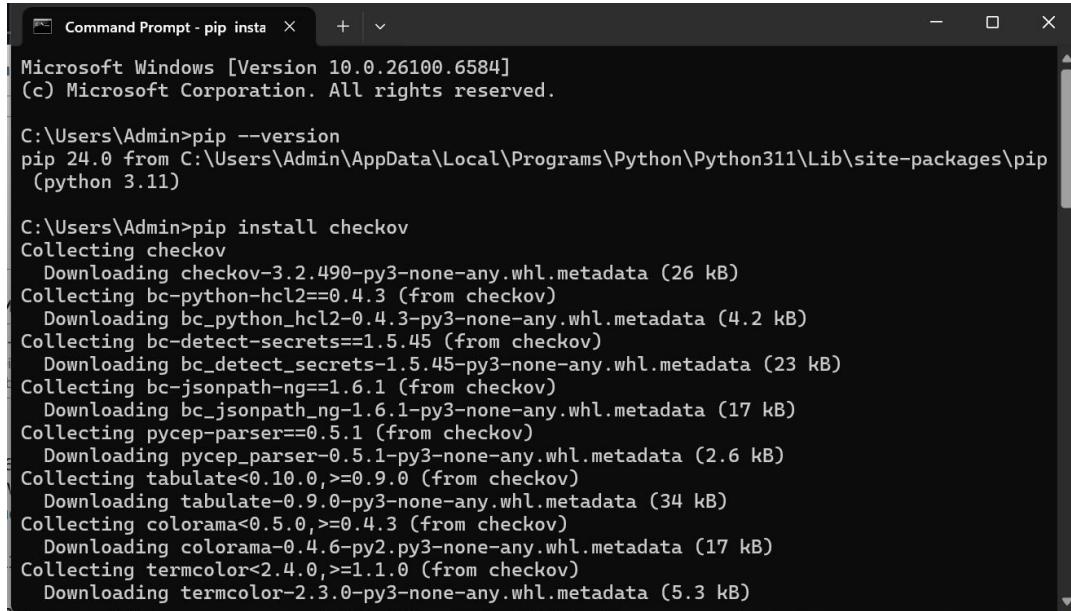
C:\Users\Admin>terraform -version
Terraform v1.13.5
on windows_amd64

C:\Users\Admin>
```

Step 2: Install Checkov

1. You can install checkov using pip:

pip install checkov



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>pip --version
pip 24.0 from C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip
(python 3.11)

C:\Users\Admin>pip install checkov
Collecting checkov
  Downloading checkov-3.2.490-py3-none-any.whl.metadata (26 kB)
Collecting bc-python-hcl2==0.4.3 (from checkov)
  Downloading bc_python_hcl2-0.4.3-py3-none-any.whl.metadata (4.2 kB)
Collecting bc-detect-secrets==1.5.45 (from checkov)
  Downloading bc_detect_secrets-1.5.45-py3-none-any.whl.metadata (23 kB)
Collecting bc-jsonpath-ng==1.6.1 (from checkov)
  Downloading bc_jsonpath_ng-1.6.1-py3-none-any.whl.metadata (17 kB)
Collecting pycep_parser==0.5.1 (from checkov)
  Downloading pycep_parser-0.5.1-py3-none-any.whl.metadata (2.6 kB)
Collecting tabulate<0.10.0,>=0.9.0 (from checkov)
  Downloading tabulate-0.9.0-py3-none-any.whl.metadata (34 kB)
Collecting colorama<0.5.0,>=0.4.3 (from checkov)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting termcolor<2.4.0,>=1.1.0 (from checkov)
  Downloading termcolor-2.3.0-py3-none-any.whl.metadata (5.3 kB)
```

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:\terrafrom-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 + ⌂ ⌂ ... | ⌂ x

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

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.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + ⌂ ⌂ ⌂

```
12 |
13 |   tags = {
14 |     Name = "bad_bucket"
15 |   }
16 | }
```

Check: CKV_AWS_145: "Ensure that S3 buckets are encrypted with KMS by default"
FAILED for resource: aws_s3_bucket.bad_bucket
File: \main.tf:9-16
Guide: <https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-general-policies/ensure-that-s3-buckets-are-encrypted-with-kms-by-default>

```
9 | resource "aws_s3_bucket" "bad_bucket" {
10 |   bucket = "my-bad-bucket"
11 |   acl    = "public-read" # Public S3 bucket
12 |
13 |   tags = {
14 |     Name = "bad_bucket"
15 |   }
16 | }
```

secrets scan results:

Passed checks: 0, Failed checks: 1, Skipped checks: 0

Check: CKV_SECRET_2: "AWS Access Key"
FAILED for resource: 25910f981e85ca04baef359199d0bd4a3ae738b6
File: /main.tf:5-6
Guide: <https://docs.prismacloud.io/en/enterprise-edition/policy-reference/secrets-policies/secrets-policy-index/git-secrets-2>

```
5 |   access_key = "AKIAI*****"                      # Hardcoded credentials
```

PS C:\terraform-iac-security> █

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

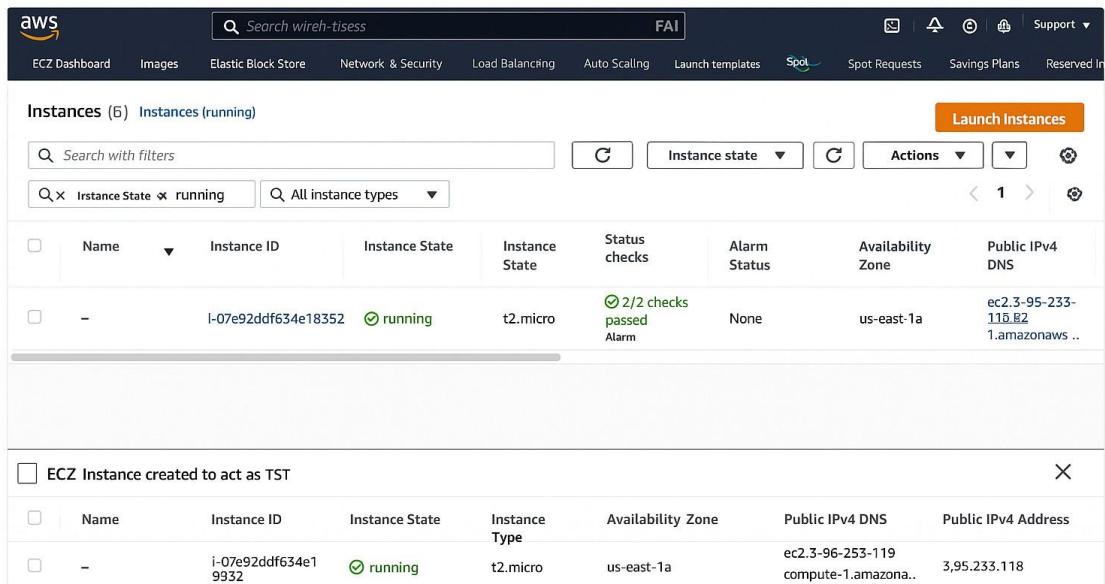
Theory:

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:

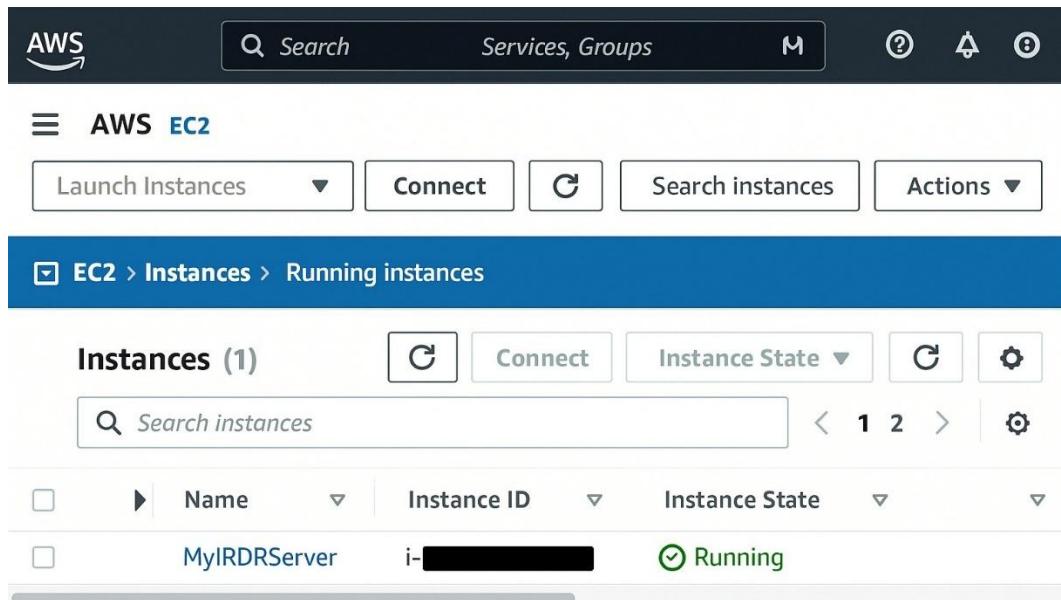
Step 1: Set Up Cloud Resources

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.



The screenshot shows the AWS EC2 Instances page. At the top, there's a search bar and a 'Launch Instances' button. Below the search bar, there are filters for 'Instance State' set to 'running' and 'All instance types'. A table lists one instance: Name - i-07e92ddf634e19932, Instance ID - i-07e92ddf634e19932, Instance State - running, Instance Type - t2.micro, Status checks - 2/2 checks passed, Alarm Status - None, Availability Zone - us-east-1a, and Public IPv4 DNS - ec2-3-96-253-119.compute-1.amazonaws.com. Below the table, a modal window displays the same instance information with a note: 'EC2 Instance created to act as TST'.

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

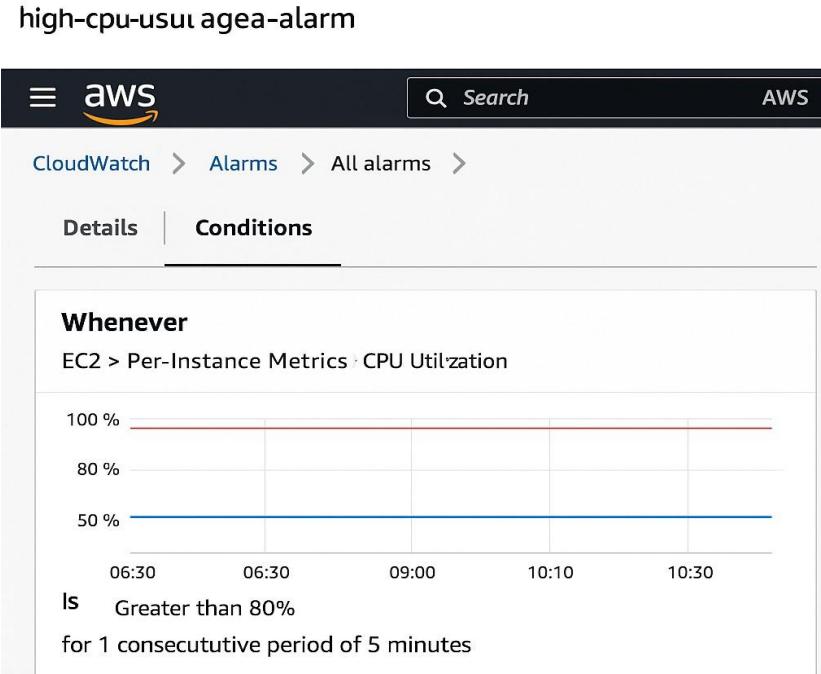


The screenshot shows the AWS EC2 Instances page. The navigation bar includes 'AWS', a search bar, and service links. Below the navigation, there are buttons for 'Launch Instances', 'Connect', 'Search instances', and 'Actions'. A blue header bar indicates the path: 'EC2 > Instances > Running instances'. The main area shows a table with one instance: Name - MyIRDRServer, Instance ID - i-[REDACTED], and Instance State - Running.

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

Step 2: Identify Critical Components

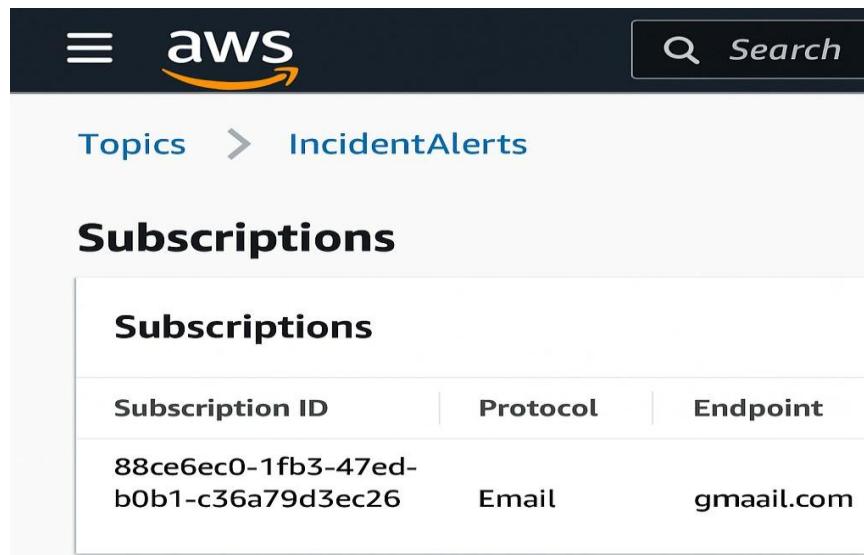
List key resources (e.g., EC2 instance and S3 bucket) that must be protected and recovered during failures.



CloudWatch Alarm configured to monitor EC2 CPU usage

Step 3: Enable Monitoring and Logging

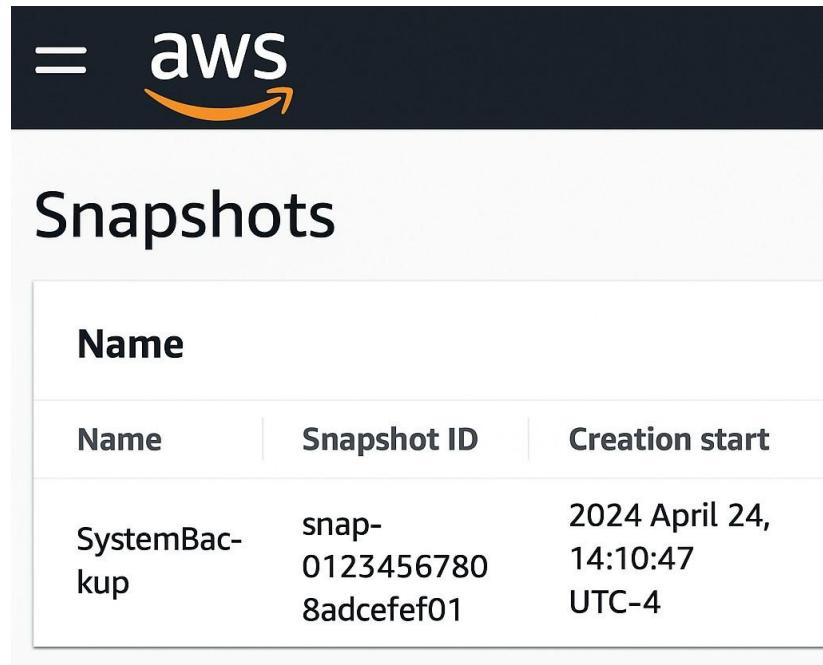
1. Go to **CloudTrail** → Enable trail for all regions to record API activities.
2. Open **CloudWatch** → Create an alarm for EC2 (e.g., CPU usage > 80%).
3. Create an **SNS topic** and subscribe with your email to receive alerts.



SNS topic created and subscribed for real-time incident alerts

Step 4: Configure Backup and Recovery

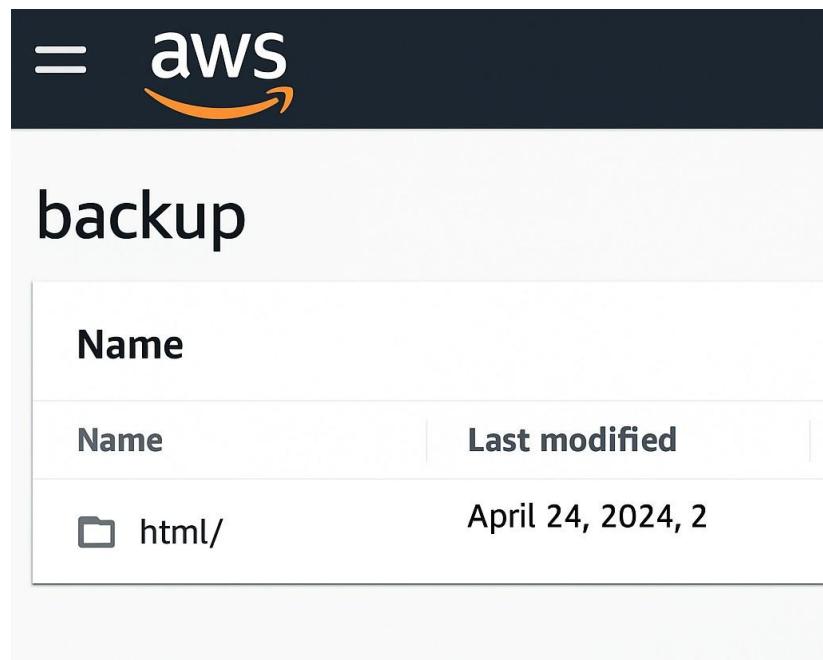
1. Take an **EBS snapshot** of the EC2 instance for system backup.
2. Copy or sync important files to the S3 bucket:
3. Verify data is stored successfully in S3.



The screenshot shows the AWS Snapshots interface. At the top, there's a dark header with the AWS logo. Below it, a large title says "Snapshots". A table lists one snapshot entry:

Name	Snapshot ID	Creation start
SystemBackup	snap-0123456780 8adcefef01	2024 April 24, 14:10:47 UTC-4

EBS snapshot created for system-level backup.



The screenshot shows the AWS backup interface. At the top, there's a dark header with the AWS logo. Below it, a large title says "backup". A table lists the contents of an S3 bucket:

Name	Last modified
html/	April 24, 2024, 2

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.

The screenshot shows the AWS CloudWatch Alarms interface. At the top, there's a navigation bar with tabs: 'All alarms' (which is selected), 'In alarm', and 'Insufficient'. Below the tabs, a red button labeled 'ALARM' is displayed next to the alarm name 'High CPU Utilization'. A large text overlay below the interface reads: 'CloudWatch alert triggered upon simulated incident.'

The screenshot shows the AWS EC2 Instances interface. At the top, it says 'EC2 > Instances'. There are buttons for 'Launch instances', 'Launch templates', and 'Actions'. A search bar and a page navigation area are also present. A message box displays: 'Recent events. Your instance has been terminated according to its scheduled termination time at 2024-04-24 19:98;12+00.00' with a link icon. The main table lists one instance:

<input type="checkbox"/>	Name	Instance ID	Instance State	Instance Type	Status Checks	Alarm Stat
	MainServer Manage	i-07f265d0dbf3...	Running	f2.micro	2/2 checks passed	us-west-2c

Pagination controls at the bottom indicate '1-1 of 1'.

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.