Zero Trust Platform with least privilege platform

Step 1: AWS Organizations + Service Control Policies (SCPs)

A. Create AWS Organization (Root Account)

1. Log into AWS Management Console

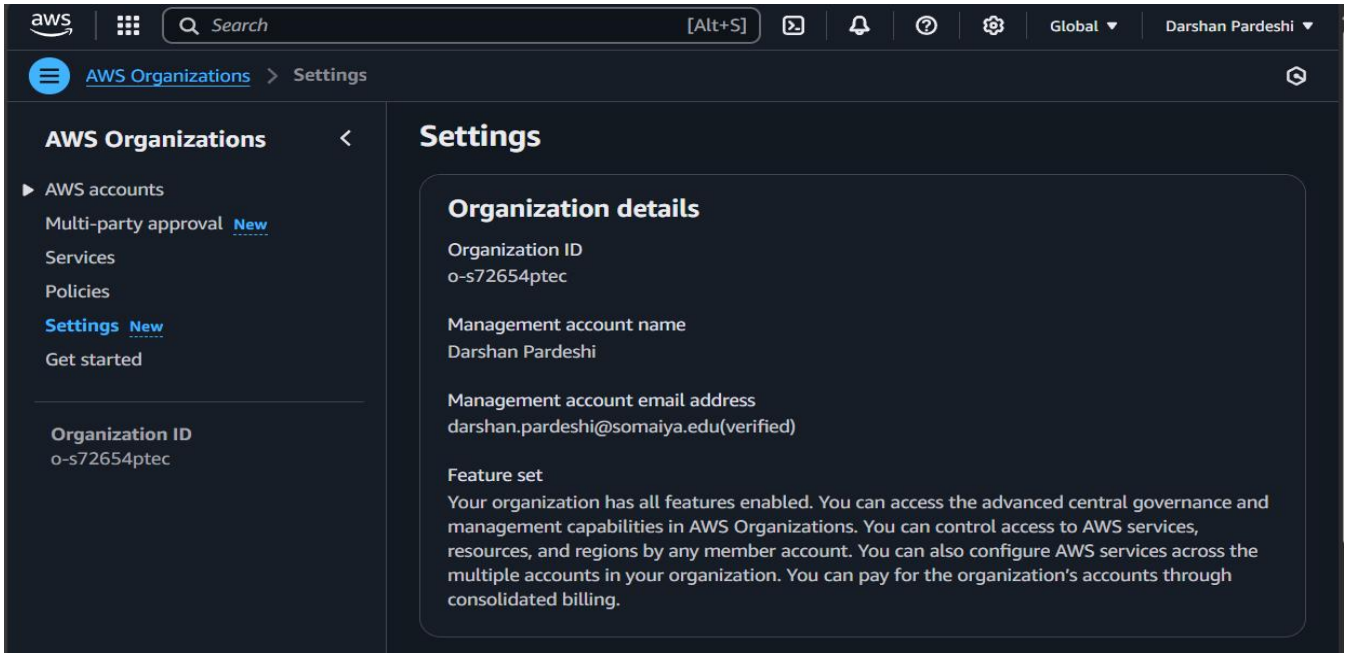
- Go to: https://console.aws.amazon.com
- Use the email/account you want to be the root of the organization

2. Go to AWS Organizations

- In the AWS Console search bar → type Organizations
- Click on **"AWS Organizations"**

3. Create Organization

- If prompted:
  -Click **“Create organization”**
  -Choose: **Enable all features** (this is required for SCPs and security tools)

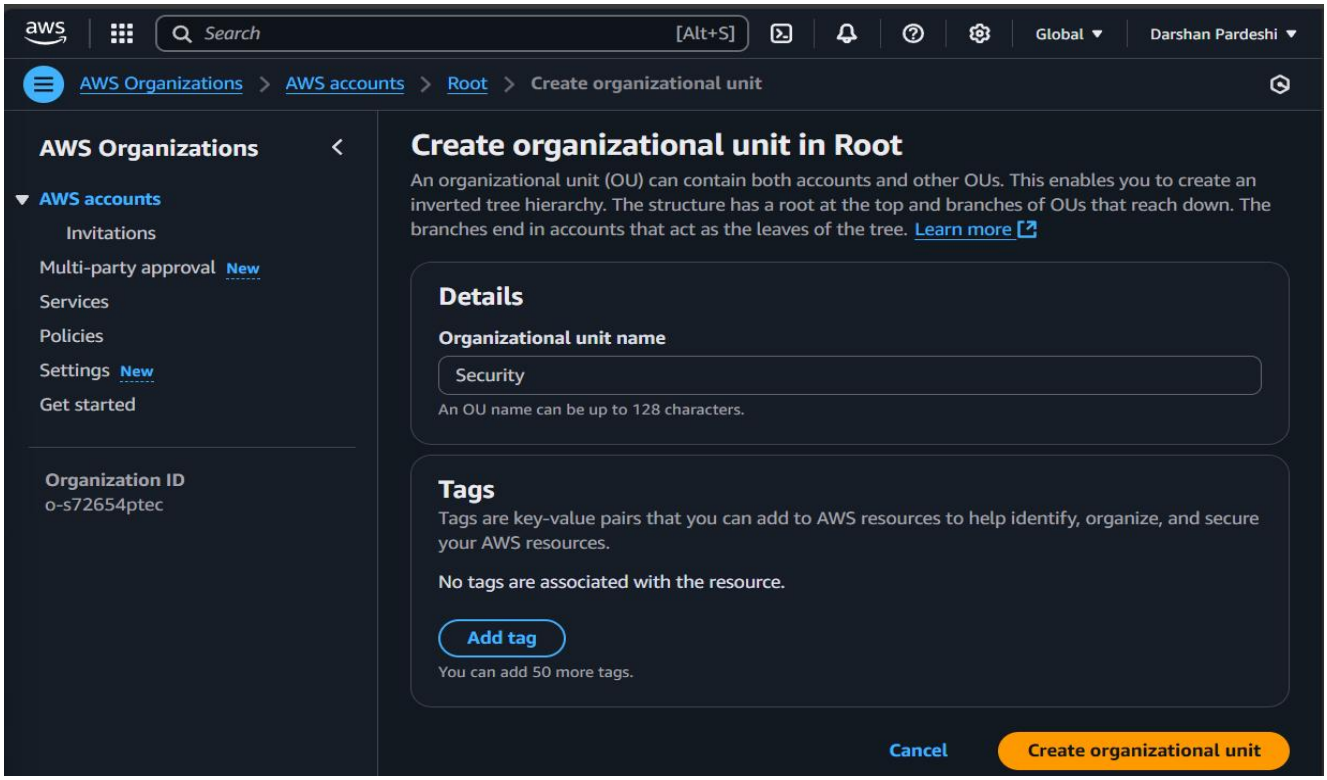


B. **Create Organizational Units (OUs) and Accounts**

4. Create OUs

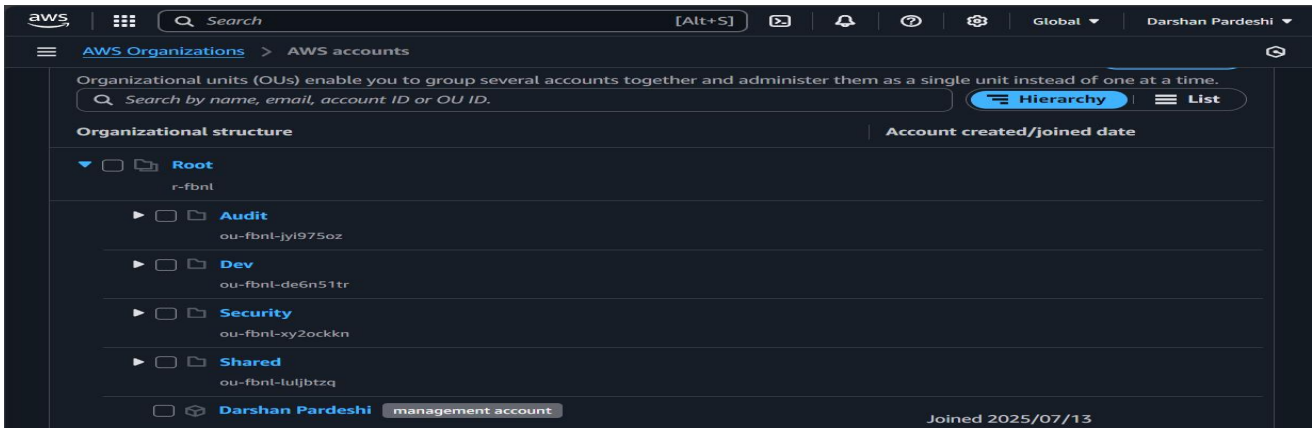
- Go to **“Organize accounts”** tab in Organizations.

- **Check the box** next to **“Root”**.
  Then click the **“Actions”** dropdown menu that appears above the tree.
- In the Actions menu, click :- Create new :- organizational unit
- Click **“Add an organizational unit”**
- Create these OUs:-



  - Security
  - Dev
  - Audit
  - Shared

Repeat this for each OU.



5. Create Accounts Under OUs

For each OU:

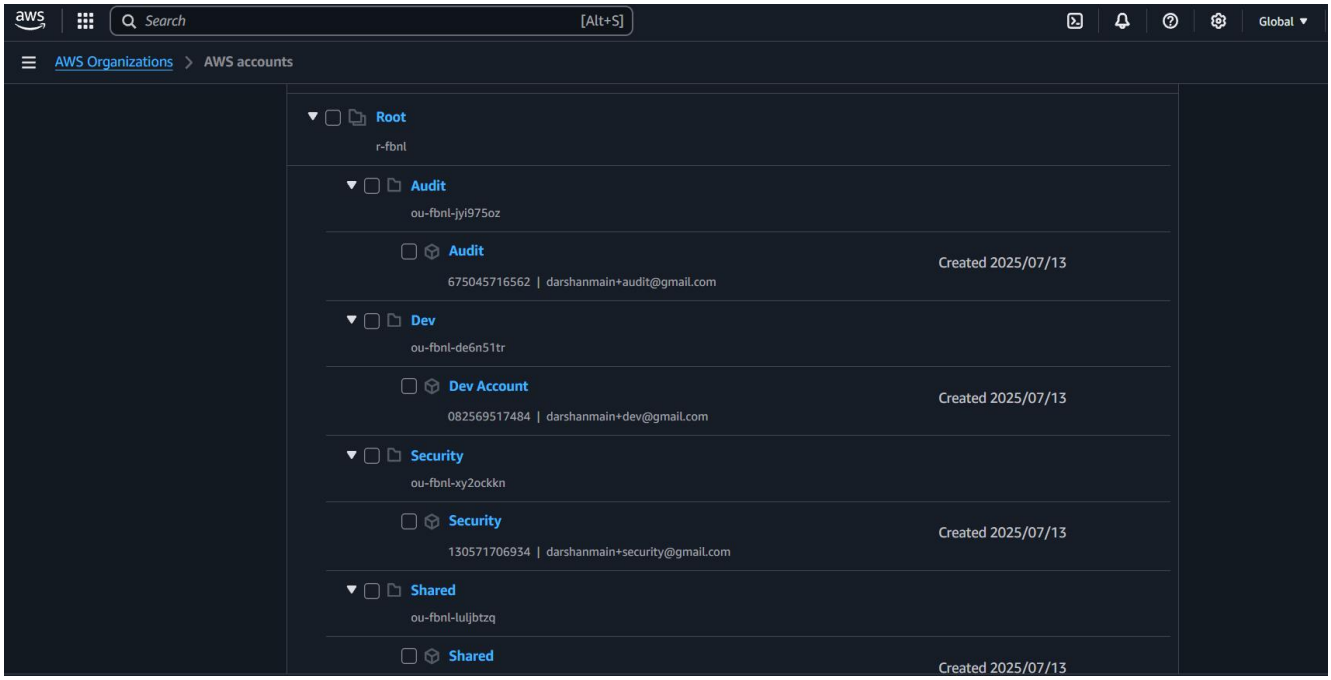
- Click **“Add account” > “Create an AWS account”**

Click **“Add account” > “Create an AWS account”**

For each account:

- Filled in **Account Name**
- Used unique **Gmail aliases** (e.g., darshanmain+security@gmail.com)
- Left IAM Role as default: OrganizationAccountAccessRole
- Chose(move) the correct **Parent OU**

Click create AWS Account .



C. **Create and Attach Service Control Policies (SCPs)**

6. Go to "Policies" → Service Control Policies

- From left panel, click **"Policies" → "Service control policies"**
- Click **"Create policy"**

(1)Create First SCP

1. **Name:** DenyFullAdminUnlessUsEast1
2. **Description:** Blocks full access outside of us-east-1 region
3. **Paste this JSON into the editor:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyFullAdmin",
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "aws:RequestedRegion": "us-east-1"
        }
      }
    }
  ]
}
```

Click Create policy.

(2)Create Second SCP

1. Click **“Create policy”** again
2. **Name:** DenyEC2UnlessSecurityAccount
3. **Description:** Denies EC2 unless requested by Security Account
4. Replace the account ID in this JSON:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Sid": "DenyEC2UnlessSecurityAccount",
    "Effect": "Deny",
    "Action": "ec2:*",
    "Resource": "*",
    "Condition": {
      "StringNotEquals": {
        "aws:PrincipalAccount": "YOUR_SECURITY_ACCOUNT_ID"
      }
    }
  }
 ]
}
```

Replace "YOUR_SECURITY_ACCOUNT_ID" with your **actual Security Account ID**
(You can find this from the **Organize accounts** tab)

5. Click **Create Policy**

(3)**Create Third SCP**

1. Click **“Create policy”** again
2. **Name:** EnforceMFA
3. **Description:** Denies all actions if user is not using MFA
4. Paste this JSON:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
    "Sid": "BlockActionsWithoutMFA",
    "Effect": "Deny",
```

```
    "Action": "*",
    "Resource": "*",
    "Condition": {
      "BoolIfExists": {
        "aws:MultiFactorAuthPresent": "false"
      }
    }
  }
 ]
}
```

5. Click **Create Policy**

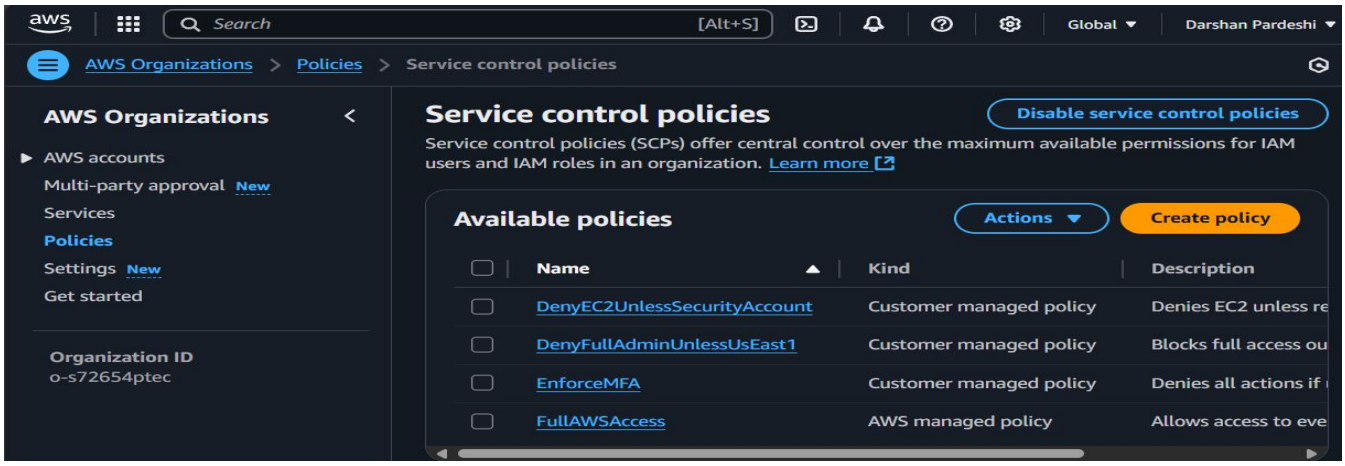
7. Attach SCPs to OUs or Accounts

- Go to the **OU** (e.g., Dev, Audit)
- Click **“Policies” tab > Attach policy**
- Attach the **SCPs you created**

Repeat for each OU as needed.

8. Enable SCPs Globally (IMPORTANT)

- Go to **"Settings"** tab in AWS Organizations
- Enable toggle for **"Service Control Policies"**
- Make sure **“FullAWSAccess”** policy is still attached unless replaced



STEP 2: IAM + IAM Access Analyzer + STS (Temporary Cross-Account Access)

A. Go to IAM Console

- **Sign in to AWS Console**
- **First Created policies then attached them to Roles.**
- **Go to IAM > Roles > Create Role**

B. Role 1:- DevOpsRole

Step-by-Step:

1. Trusted Entity:

- **Select: Another AWS account**
- **Enter: Dev account ID (Account-B)**

2. Permissions: Attach custom policy (create if needed)

Example: DevOpsPolicy

```
{
 "Version": "2012-10-17",
 "Statement": [
  { "Effect": "Allow", "Action": ["s3:*"], "Resource": "*" },
  { "Effect": "Allow", "Action": ["lambda:*"], "Resource": "*" },
  { "Effect": "Allow", "Action": ["cloudwatch:*"], "Resource": "*" }
 ]
}
```

4. Name: DevOpsRole

Done.

C. Role 2: AuditorRole

Step-by-Step:

1. Trusted Entity:

- **Another AWS account → Enter Audit account ID**

2. Permissions: Custom ReadOnlyAuditPolicy

{

"Version": "2012-10-17",

"Statement": [

{ "Effect": "Allow", "Action": ["cloudtrail:LookupEvents", "cloudtrail:Get*", "config:Get*", "iam:List*", "iam:Get*"], "Resource": "*" }

]

}

3. Boundary (Optional): Same as above

4. Name: AuditorRole

Done.

D. Role 3: IncidentResponderRole

Step-by-Step:

1. Trusted Entity:

- **Enter Security or Shared account ID**

2. Permissions: IncidentResponsePolicy

{

"Version": "2012-10-17",

"Statement": [

{ "Effect": "Allow", "Action": ["guardduty:Get*", "guardduty:List*", "detective:Get*", "detective:List*", "sns:Publish"], "Resource": "*" }

]

}

3. Boundary: (optional)

4. Name: IncidentResponderRole

Done.

E. Role 4: SecurityAdminRole

🛠️ Step-by-Step:

1. Trusted Entity:

- **Shared or Security account ID**

2. Permissions: SecurityAdminPolicy

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "IAMControl",
   "Effect": "Allow",
   "Action": [
    "iam:CreateRole",
    "iam:AttachRolePolicy",
    "iam:PassRole",
    "iam:GetRole",
    "iam:ListRoles"
   ],
   "Resource": "*"
  },
  {
   "Sid": "GuardDutyAdmin",
   "Effect": "Allow",
   "Action": [
    "guardduty:EnableOrganizationAdminAccount",
    "guardduty:Get*",
```

```
      "guardduty:List*"
    ],
    "Resource": "*"
  },
  {
    "Sid": "SCPManagement",
    "Effect": "Allow",
    "Action": [
      "organizations:AttachPolicy",
      "organizations:DetachPolicy",
      "organizations:ListPolicies",
      "organizations:ListRoots"
    ],
    "Resource": "*"
  },
  {
    "Sid": "BasicSecurityReadOnly",
    "Effect": "Allow",
    "Action": [
      "cloudtrail:LookupEvents",
      "config:GetComplianceSummaryByResourceType"
    ],
    "Resource": "*"
  }
 ]
}
```

3. Name: SecurityAdminRole

Done.

IAM Access Analyzer:-

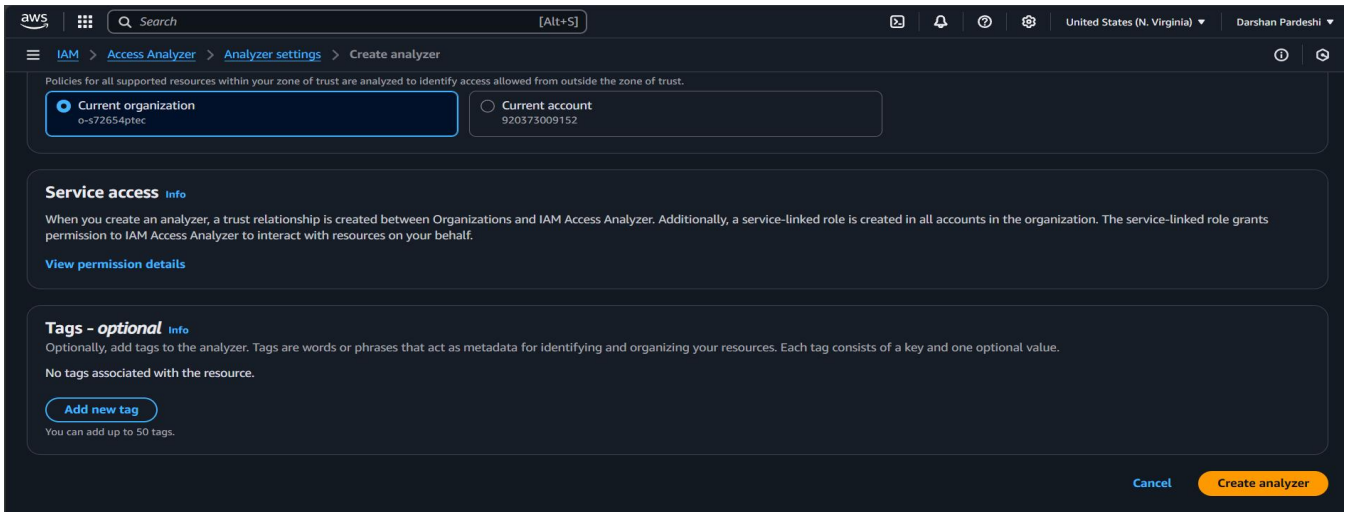
Steps to Enable Access Analyzer

Step 1: Go to IAM Console → Access Analyzer

Step 2: Click "Create Analyzer"

- **Analyzer name**: OrgAnalyzer
- **Zone of trust** : **"Organization"**

Step 3: Click **“Create”**



Part 2 — STS Cross-Account Role Access:

STEP 1: Edit Trust Policy (Target Account)

Example: Allow Dev account to assume DevOpsRole created in Shared account

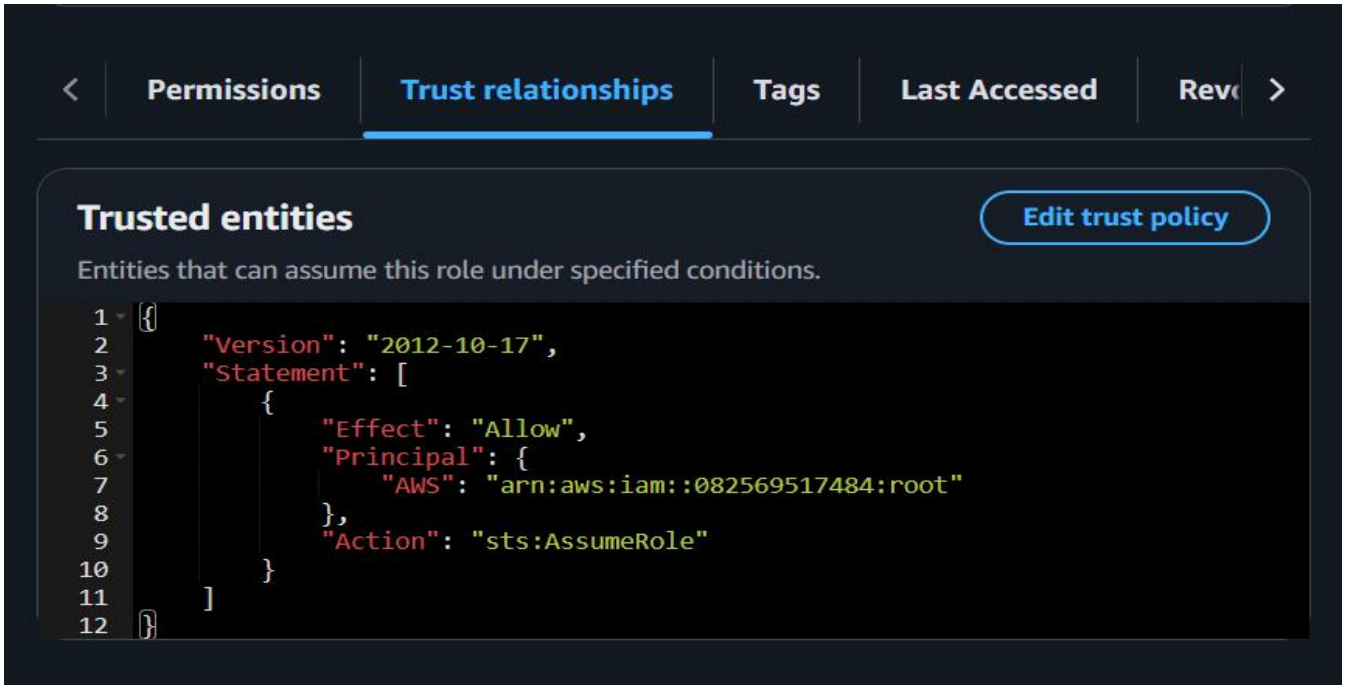
Location:

1. Go to Shared Account
2. Go to IAM → Roles
3. Find your role → e.g., DevOpsRole
4. Click on Trust relationships tab → Edit Trust Policy

Paste this (replace 111122223333 with your Dev Account ID):

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
```

```
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::DEV_ACCOUNT_ID:root"
    },
    "Action": "sts:AssumeRole"
  }
 ]
}
```

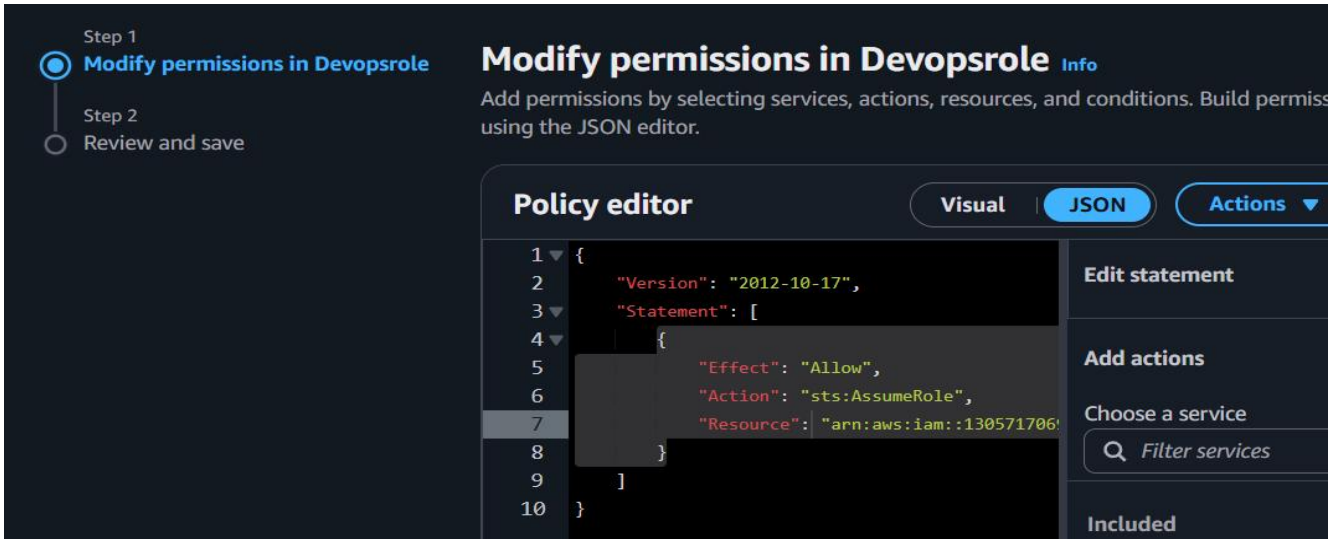


STEP 2: Attach AssumeRole Permission (Source Account)

Now go to the account that wants to assume the role (e.g., Dev account).

Location:

1. Login to Dev account
2. Go to IAM → Users
3. Choose the IAM user who should be allowed to switch roles
4. Click Add Permissions → Attach policies directly
5. Click Create policy → Use JSON tab

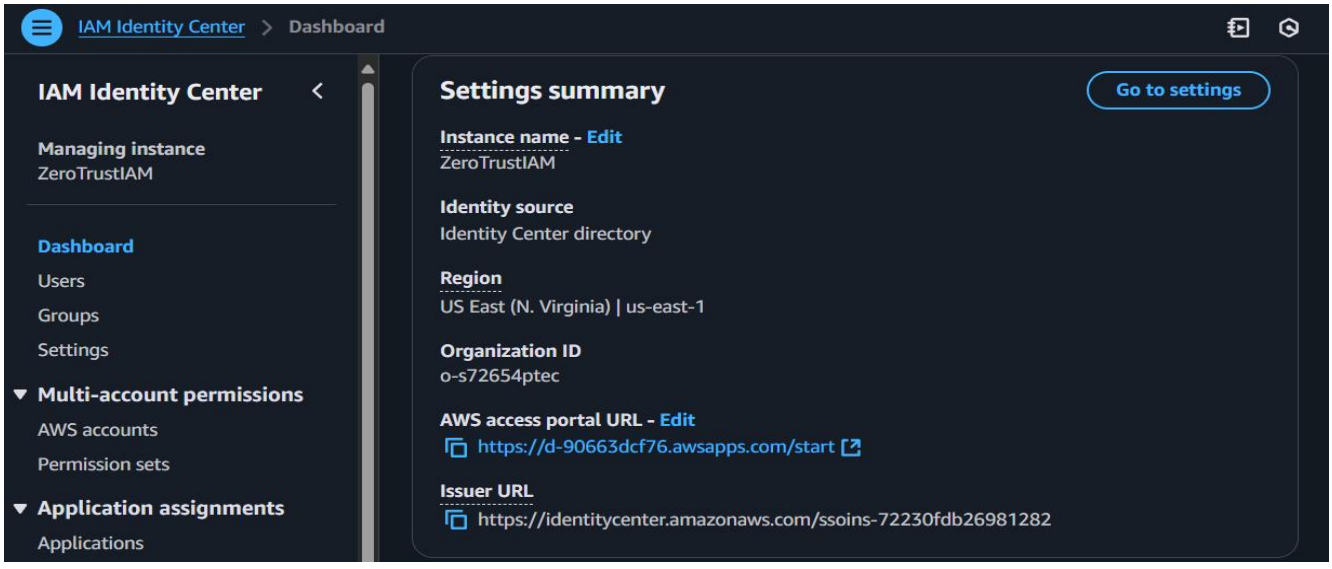


STEP 3: AWS SSO + Amazon Cognito

A. AWS SSO

Step 1: Enable Identity Center

1. Go to AWS Console → Search IAM Identity Center
2. Click “Enable” (if not enabled already)
3. Choose Identity Source:
    - Default Directory



Step 2: Create Users and Groups

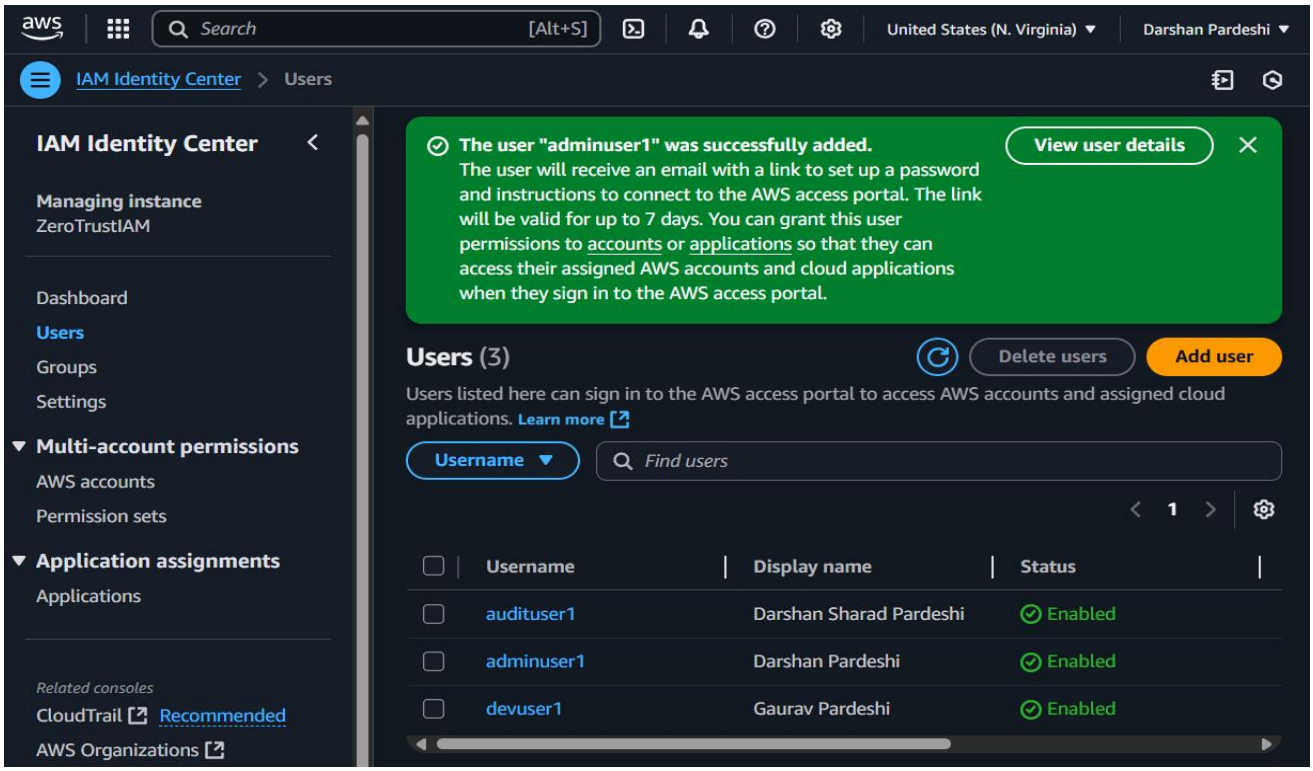
(1)Go to Users → Add User

- Example: devuser1, audituser1, adminuser1
- Set temporary password

(2) Go to Groups → Create Group

- Developers
- Auditors
- SecurityAdmins

(3) Add users to respective groups



Step 3: Assign Accounts & Permissions

1. Go to AWS Accounts tab (within Identity Center)
2. Click “Assign Users or Groups”
    - Choose: e.g., Developers, Auditors, etc.
3. Select AWS Account(s): Dev, Audit, Security
4. Choose or create Permission Sets:

Add Policies – Give set Name – Create.

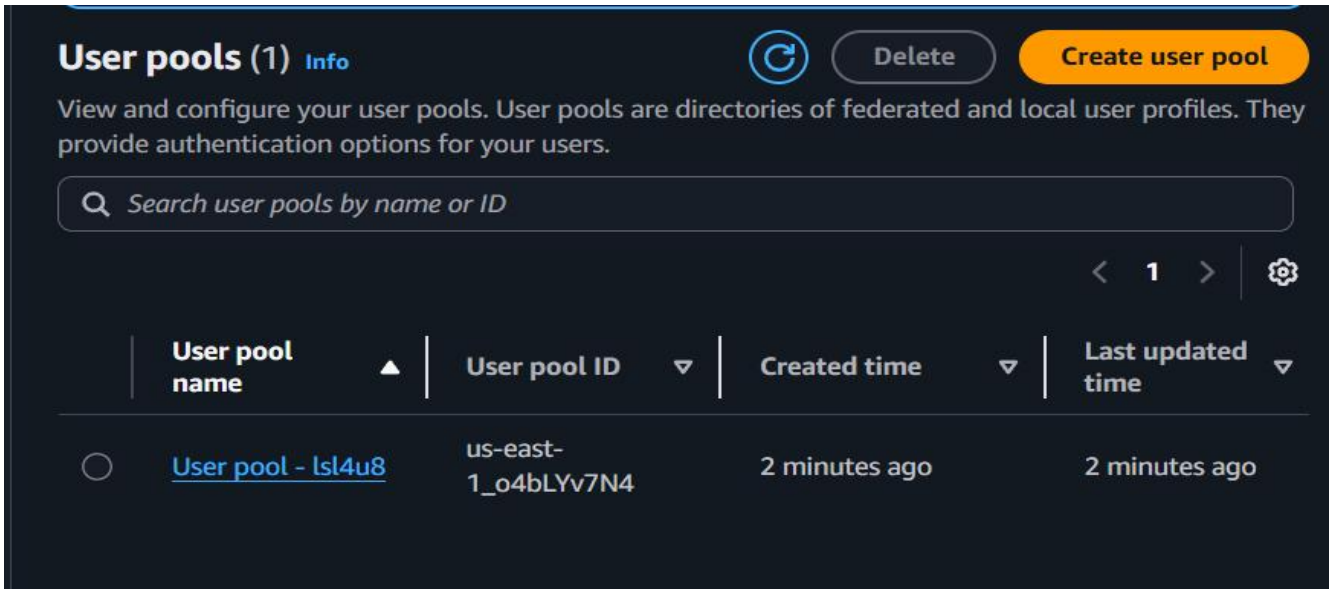
Repeat to create:

- Audit-ReadOnly
- Security-Admin

B. Amazon Cognito

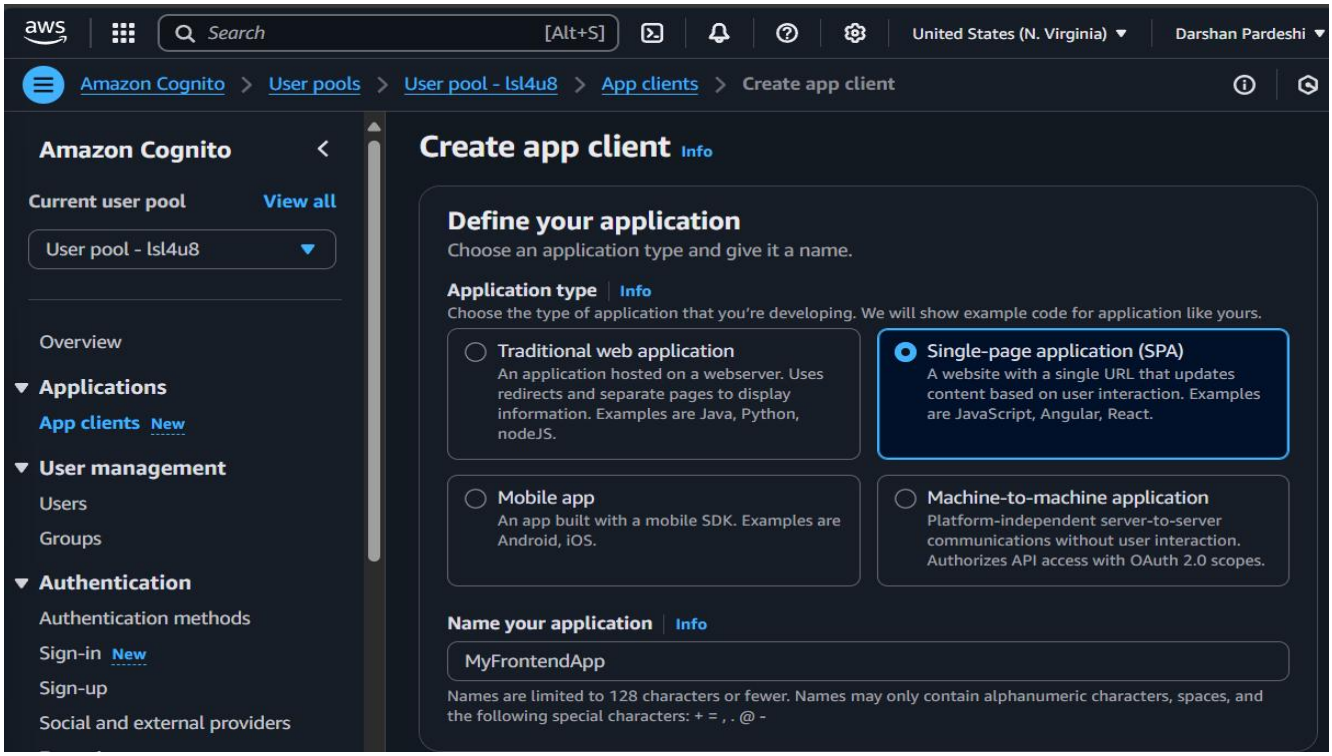
Step 1: Create User Pool

1. Go to AWS Console → Search for **Amazon Cognito**
2. Click **Create user pool**
3. Name: MyAppUsers
4. Choose **Standard creation**
5. Configure:
    - **Sign-in method**: Username, or email
    - **Password policy**: Set password strength rules



Step 2: Create App Client

1. Under **App Integration**, click **Create App Client**
2. Name: MyFrontendApp
3. Turn OFF client secret (for frontend apps)
4. Enable **OAuth 2.0** flows:
    - Authorization code grant
    - Set redirect URIs: e.g., https://myapp.com/callback
5. Enable Hosted UI (Cognito login page)



Step 3: Set Up Hosted UI Domain (if not done yet)

How to Set Cognito Domain:

Cognito Hosted UI – Final Configuration Guide

This guide outlines the essential configuration settings required to set up a secure and functional Hosted UI using Amazon Cognito.

1. Allowed Callback URLs (Required)

Purpose:
Defines the URLs where Cognito should redirect users after a successful login.

Recommendations:

- For testing:
  https://jwt.io
- For production:
  https://yourfrontend.com/callback

2. Allowed Sign-Out URLs (Recommended)

Purpose:
Specifies where users are redirected after logging out of the application.

Recommendations:

- Safe default:
  https://jwt.io
- Use the same value as the callback URL for simplicity, unless a separate logout page is desired.

3. Identity Providers (Required)

Purpose:
Defines the source(s) from which users can authenticate.

Configuration:

- Enable the **Cognito user pool** (this is selected by default).
- Leave third-party providers (e.g., Facebook, Google) disabled unless federated identity has been configured.

4. OAuth 2.0 Grant Types (Required)

Purpose:
Specifies how Cognito issues tokens to the application.

Configuration:

- Enable **Authorization code grant** (recommended for security and compatibility with frontend apps).
- Do not enable Implicit grant unless there is a specific use case requiring it.

5. OpenID Connect (OIDC) Scopes (Required)

Purpose:
Determines the user information included in the ID token.

Recommended Scopes:

- openid – Required to receive an ID token.
- email – Provides the user's email address.
- profile – (Optional) Includes additional attributes like name and profile picture.

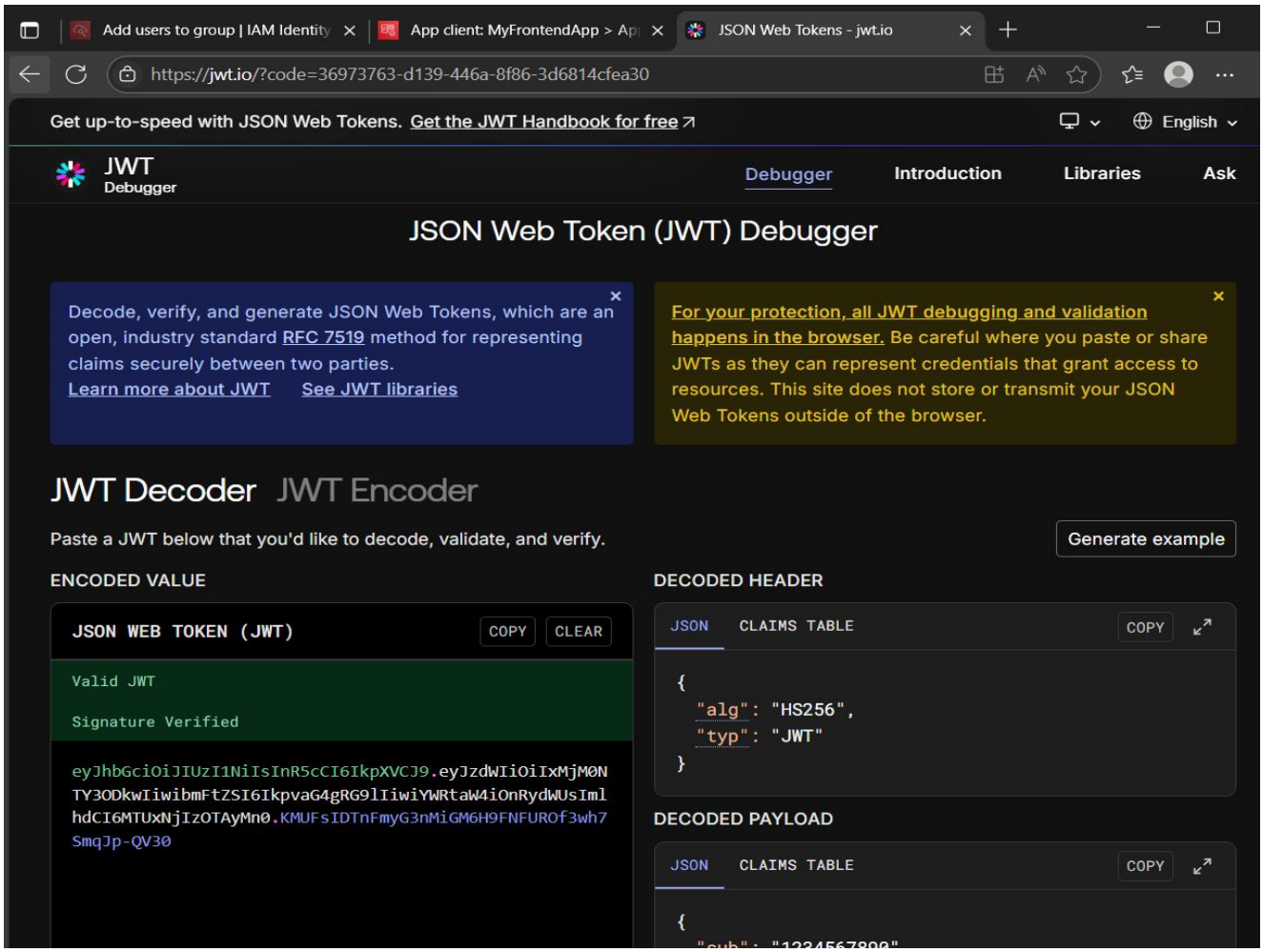
6. Custom Scopes (Optional – Skip for Now)

Purpose:
Used for advanced use cases like securing API access with custom scopes.

Recommendation:

- Skip this step during initial setup.

- Consider custom scopes only if integrating with a resource server or fine-grained access control is needed.



Step 5: Lambda + EventBridge + CloudWatch (Full Setup)

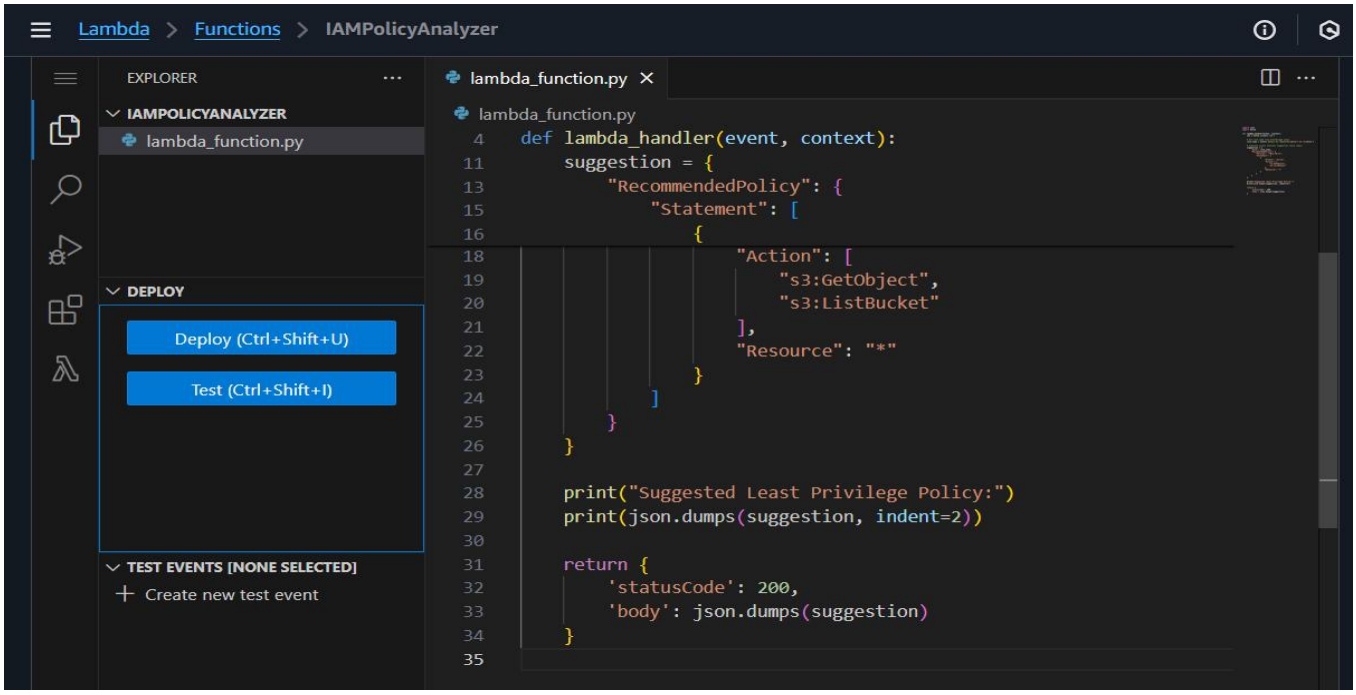
Step 1: Create the Lambda Function

1. Go to **Lambda Console → Create function**
2. Name: IAMPolicyAnalyzer
3. Runtime: Python 3.12 or Node.js 20.x (your choice)
4. Permissions: Create a new role with the following policies:
    - IAMReadOnlyAccess
    - AccessAnalyzerReadOnly
    - CloudWatchLogsFullAccess

Click **Create function**

Step 2: Add Code

```
import json
import boto3

def lambda_handler(event, context):
    iam = boto3.client('iam')

    # Get role name from EventBridge event
    role_name = event['detail']['requestParameters']['roleName']

    # Simulate access analyzer suggestion (mock logic)
    suggestion = {
        "Role": role_name,
        "RecommendedPolicy": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "s3:GetObject",
                        "s3:ListBucket"
                    ],
                    "Resource": "*"
                }
            ]
        }
    }
```



B. EventBridge Integration

Step 1: Create Rule in EventBridge

1. Go to Amazon EventBridge → Rules → Create Rule
2. Name: TriggerIAMPolicyAnalyzer
3. Event Bus: default
4. Define pattern manually:
    - Choose: AWS events
    - Service Name: IAM
    - Event Type: AWS API Call via CloudTrail

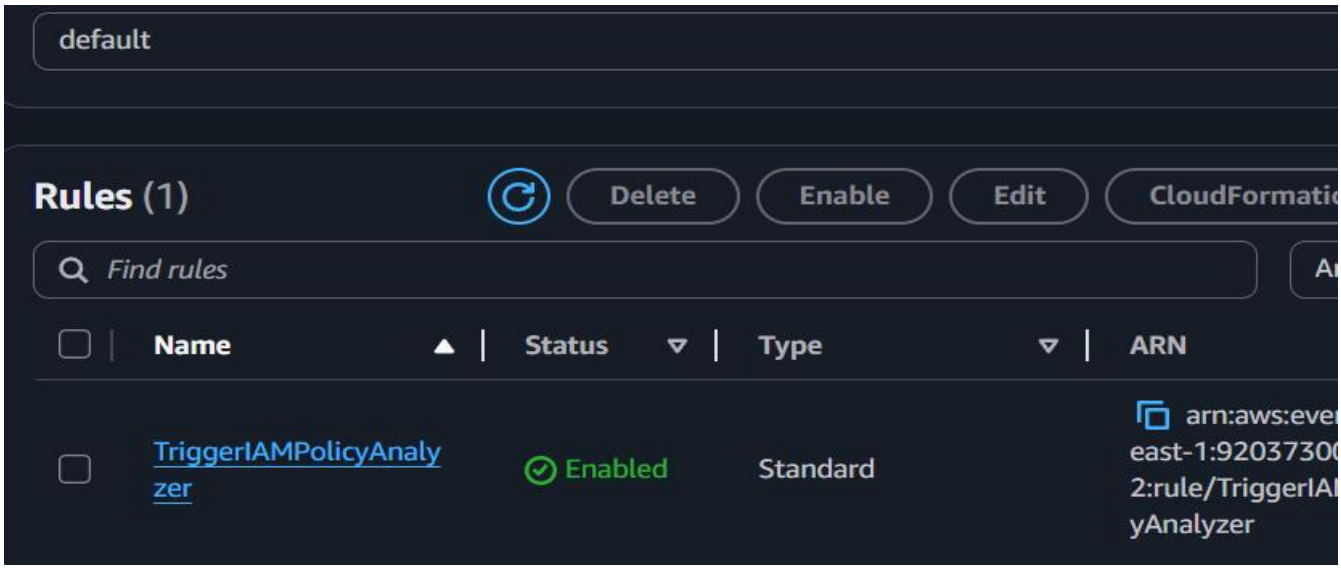
Event Pattern Example:

```
{
 "source": ["aws.iam"],
 "detail-type": ["AWS API Call via CloudTrail"],
 "detail": {
  "eventName": [
   "PutRolePolicy",
```

```
    "AttachRolePolicy",
    "CreateRole",
    "UpdateAssumeRolePolicy"
  ]
 }
}
```

5. **Target**: Select **Lambda function**
    - Choose: IAMPolicyAnalyzer

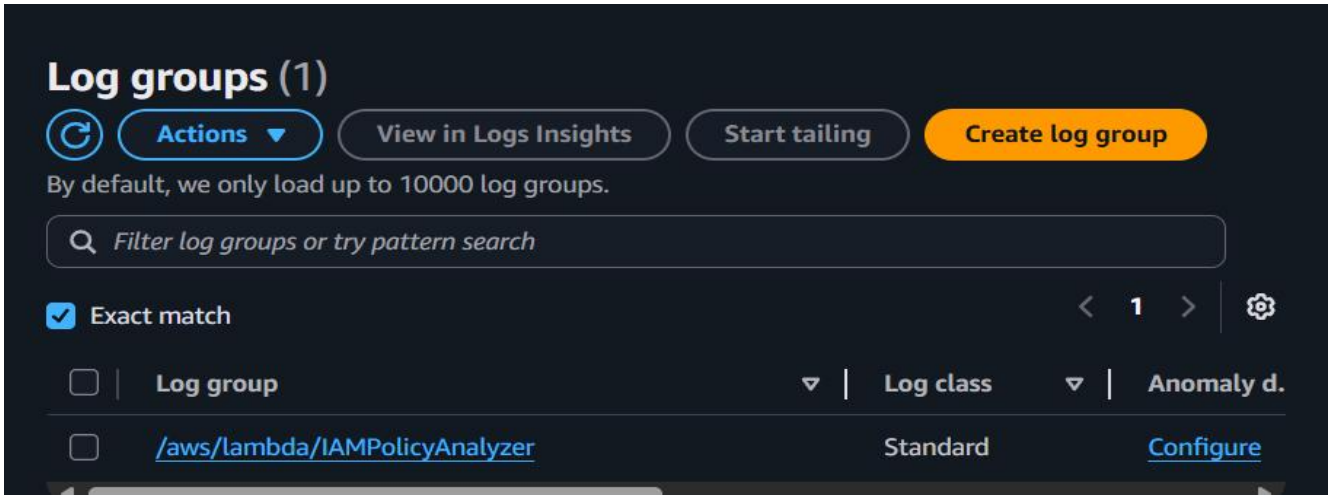
Click **Create Rule**



C. CloudWatch Monitoring

Step 1: Monitor Lambda Logs

1. Go to **CloudWatch → Logs → Log groups**
2. Find /aws/lambda/IAMPolicyAnalyzer
3. View logs for each invocation



Alarm: High Error Rate in Lambda

🔧 **Steps:**

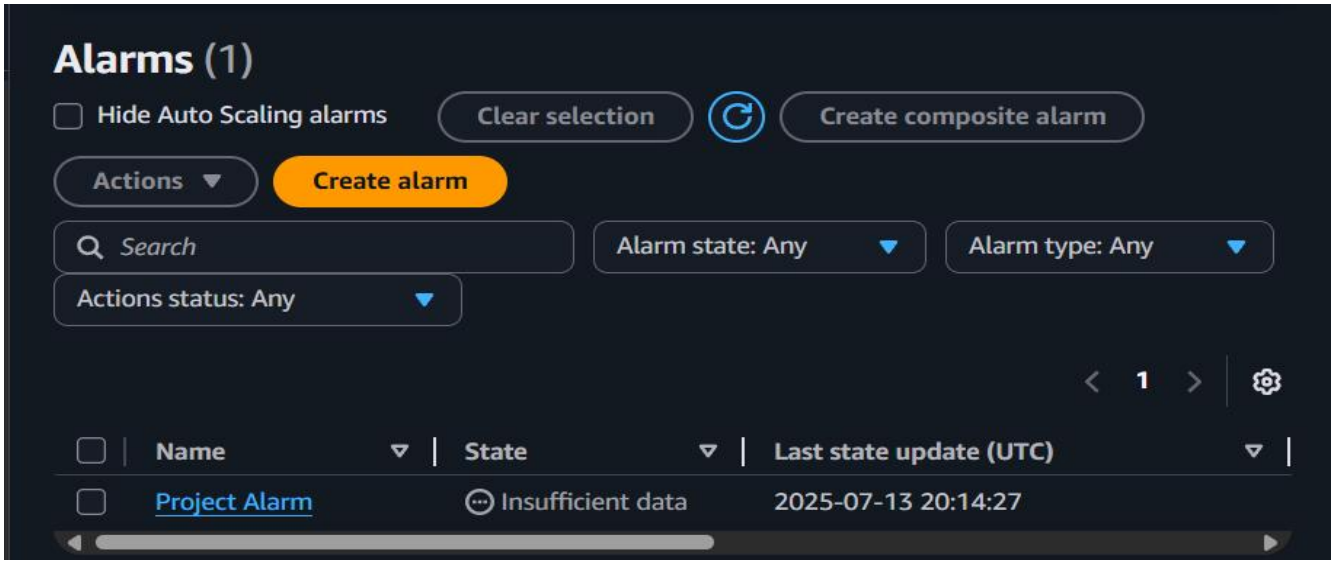
1. Go to **AWS Console → CloudWatch → Alarms → Create Alarm**
2. Click **"Select metric"**
3. Navigate:
   Browse → Lambda → By Function Name
4. Select your function: IAMPolicyAnalyzer
5. Check the box for **"Errors"**
6. Click **"Select metric"**

Configure the alarm:

7. Under **Conditions**, set:
   - **Threshold type**: Static
   - **Whenever Errors is...**: Greater than 1
   - **For...**: 1 out of 1 datapoints
   - **Period**: 5 minutes
8. Click **Next**

Notifications (Optional but recommended):

9. Choose/create an **SNS topic** to receive email alerts
   - If you haven't created one, do it now and **subscribe your email**
10. Click **Next → Next → Create alarm**



Step-by-Step: CloudTrail + S3 + KMS Logging Setup

A. CloudTrail Setup

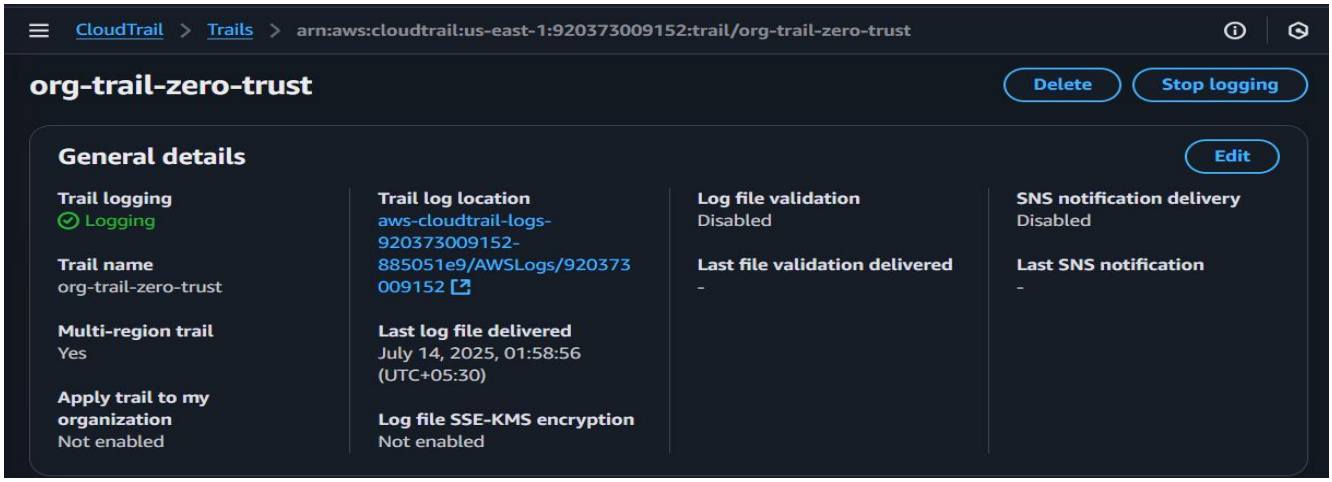
Step 1: Go to CloudTrail

Step 2: Create a Trail

1. Click **Create trail**
2. Trail name: org-trail-zero-trust

Step 3: Select Trail Options

1. Enable:
    - Management events
    - Data events:
        - For **S3** (All buckets)
        - For **Lambda** (All functions)
2. Enable:
    - Insights (optional, detects unusual API activity)



Step-by-Step: Create KMS Key (cloudtrail-log-key) and Use It in CloudTrail :-

◈ Step 1: Go to AWS KMS Console

1. Open AWS Console
2. Search or go to **Key Management Service (KMS)**

◈ Step 2: Create a New Customer Managed Key (CMK)

1. Click **Create key**
2. **Key type**: Symmetric
3. **Key usage**: Encrypt and decrypt
4. Click **Next**

◈ Step 3: Configure Key Settings

1. **Alias**: cloudtrail-log-key
2. **Description**: Used for encrypting CloudTrail logs
3. Enable K**ey rotation**
4. Click **Next**

◈ Step 4: Set Key Administrators

Add your IAM user or role that you're using now
(So you can manage the key)

Then click **Next**

◈ Step 5: Set Key Users

Here, we'll allow **CloudTrail** to use the key:

1. Under **Other AWS services**, select:
    - CloudTrail
2. Click **Next** → then **Finish**

Your KMS CMK is now ready

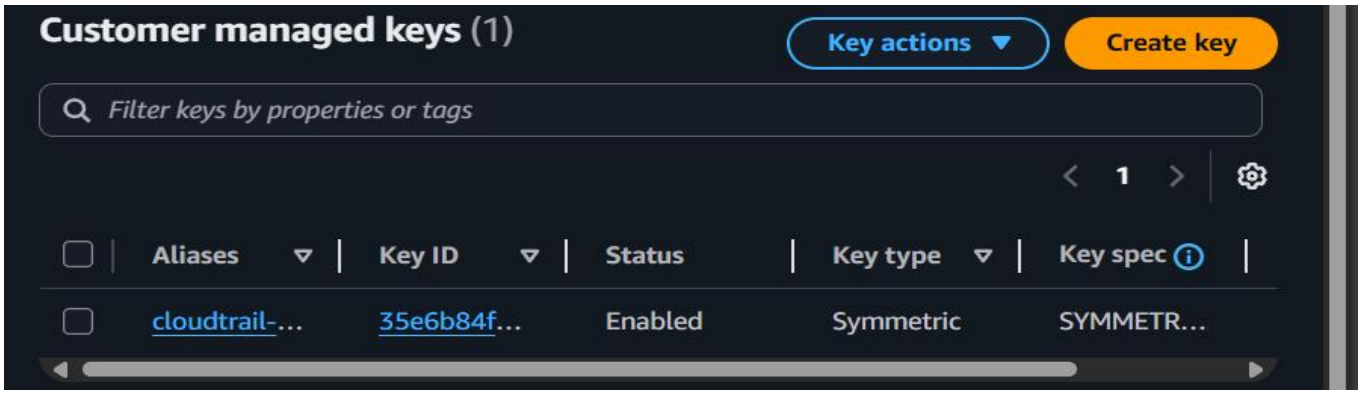
◈ Step 6: Go Back to CloudTrail & Attach the KMS Key

Now return to your **CloudTrail trail creation** screen:

1. On **Step 4: Configure Destination**
2. Under **Log file SSE-KMS encryption**:
    - Enable it
    - In the **KMS key** field, select:
      alias/cloudtrail-log-key

◈ Step 7: Complete Trail Setup

1. Review your full CloudTrail configuration
2. Click **Create trail**



Step 4: Configure Destination

1. **Storage location**:
   Choose "Create new S3 bucket" or select one if already created (see next section).
2. **Enable log file SSE-KMS encryption**
    - Choose your **KMS CMK**

Step 5: Finalize & Create

- Click **Create trail**

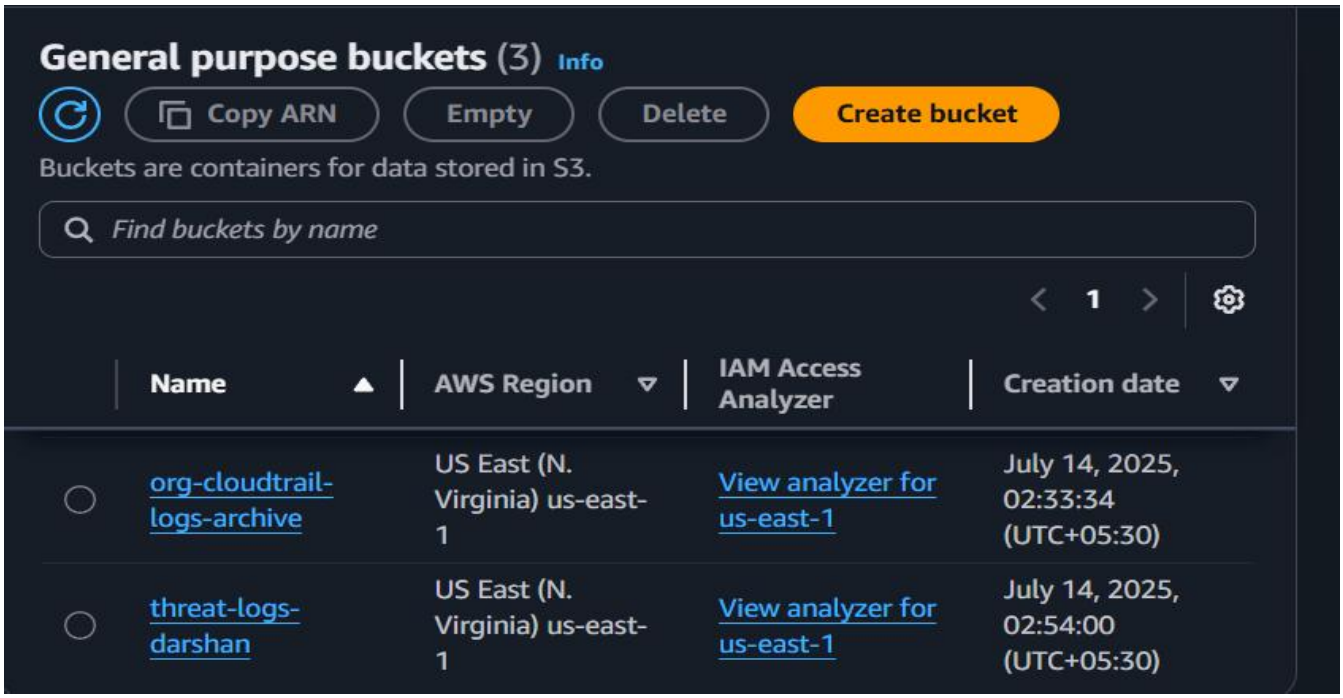
B. S3 Log Archive Bucket (Secure & Compliant)

Step 1: Create S3 Bucket

1. Go to **S3 Console → Create bucket**
2. Name: org-cloudtrail-logs-archive
3. Region: Choose central region (e.g. us-east-1)
4. Uncheck **Block all public access** (already on by default)

Step 2: Enable Versioning

- After creation → Go to bucket
- Click **Properties > Enable versioning**



7. AWS Config Rules

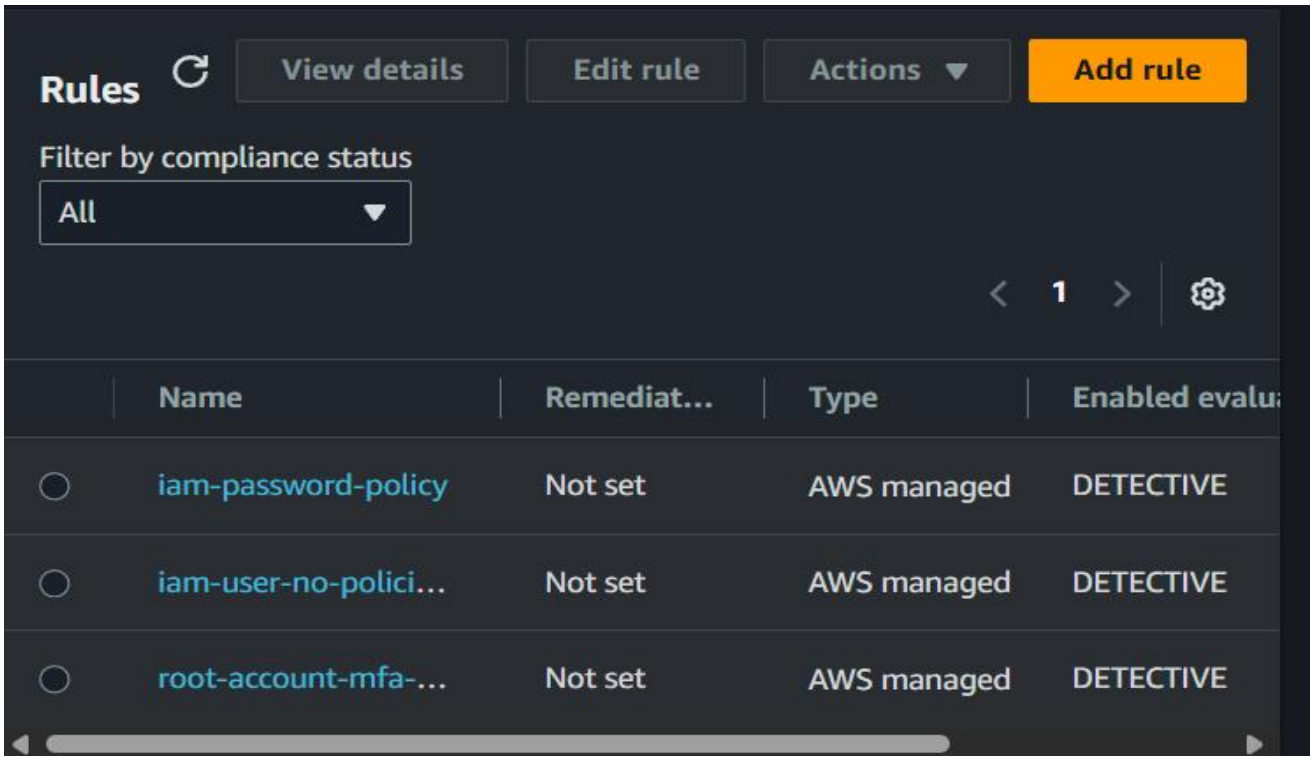
Setup:

- Go to **AWS Config > Setup**
    - Enable in **all regions**
    - Choose **global resource recording**
    - Store logs in S3 bucket (use same central bucket)

Use these IAM security rules:

- iam-user-no-policies-check

- iam-password-policy
- restricted-ssh
- cloud-trail-enabled
- root-account-mfa-enabled



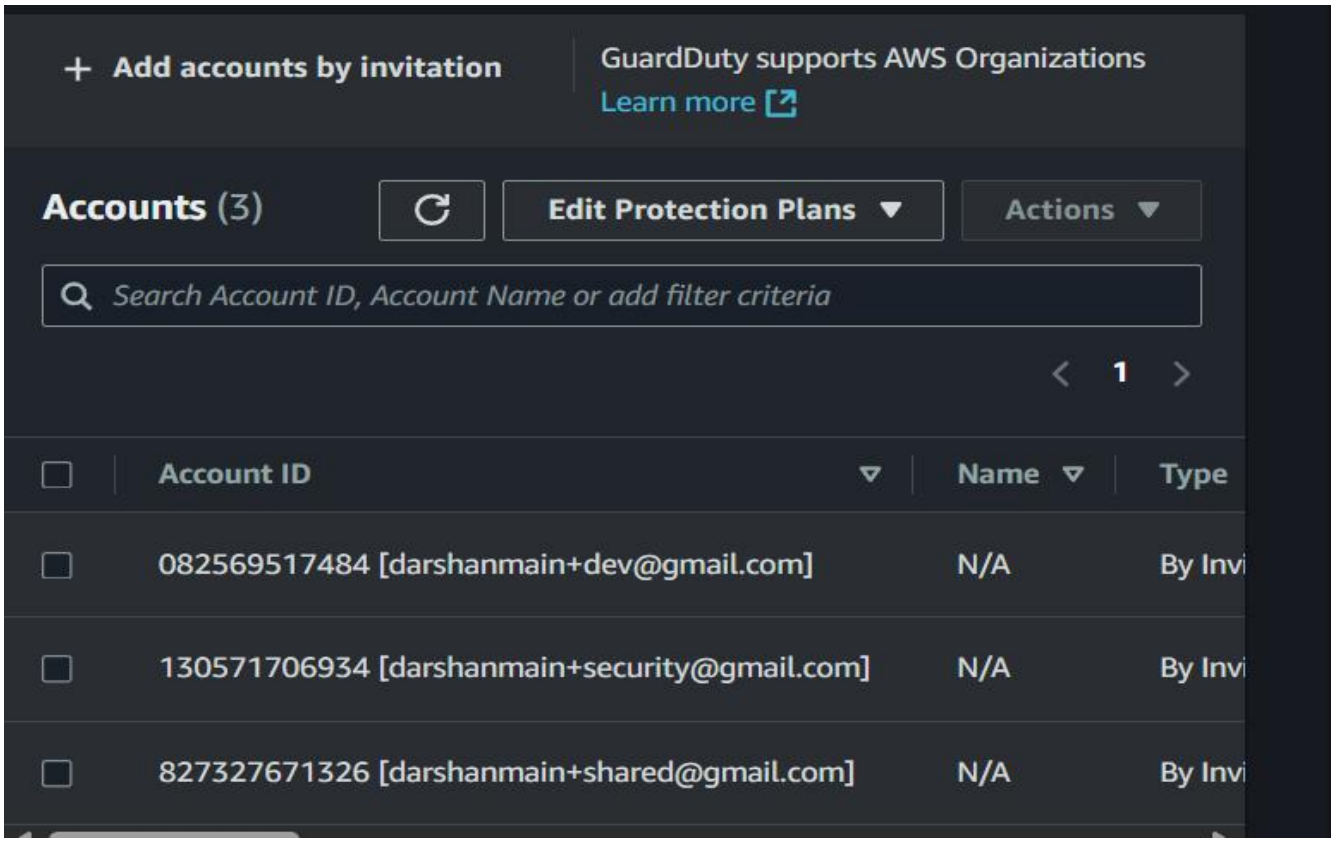
Step 8: Enable GuardDuty + Security Hub + Detective

🔷 **A. Enable GuardDuty in All Accounts & Regions**

1. **Go to GuardDuty Console:**
   https://console.aws.amazon.com/guardduty/
2. **Enable GuardDuty in your Management Account:**
   - Click **Get started** or **Enable GuardDuty** if not already enabled.
   - Choose **All regions** to enable multi-region detection.
   - This creates a **master GuardDuty account**
3. **Enable GuardDuty in Member Accounts:**

- Go to **Accounts** section in GuardDuty.
- Invite member accounts (Dev, Audit, Security).
- In member accounts, accept the invitation.

- This allows GuardDuty to aggregate findings across accounts.

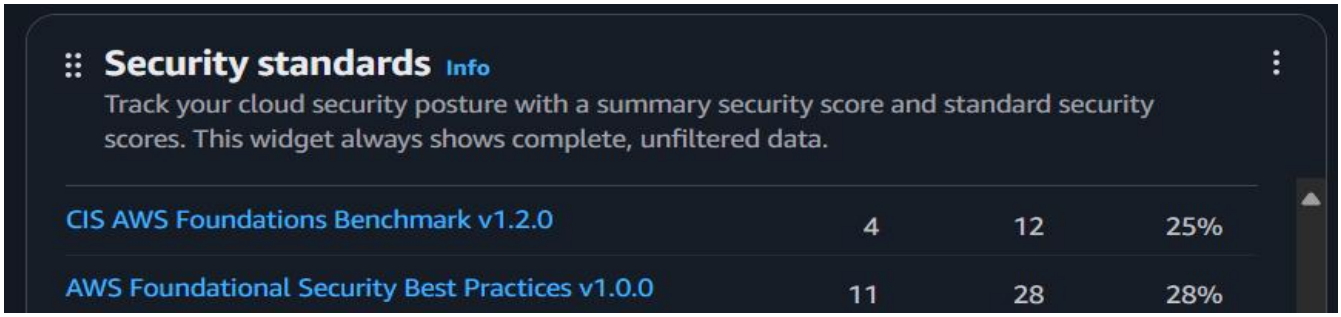


4. **Verify that GuardDuty is monitoring:**

- **CloudTrail events**
- **VPC Flow Logs**
- **DNS logs**

B. Enable AWS Security Hub and Integrate Findings

1. **Go to Security Hub Console:**
   https://console.aws.amazon.com/securityhub/
2. **Enable Security Hub:**
   - Click **Enable Security Hub**.
3. **Enable Security Standards:**
   - Turn on **CIS AWS Foundations Benchmark**.
   - Turn on **AWS Foundational Security Best Practices**.



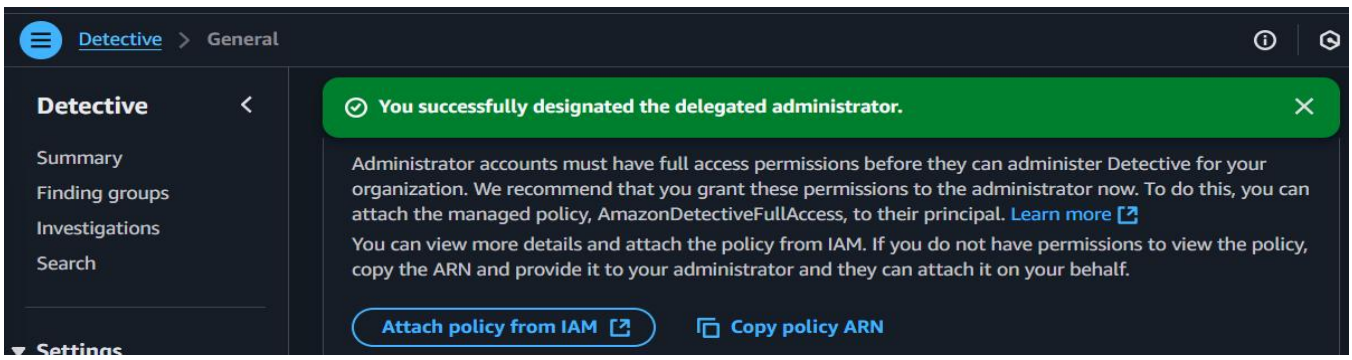
4. **Integrate with GuardDuty and other tools:**
   - Security Hub automatically integrates with GuardDuty.

   Enable integration for:

   - IAM Access Analyzer
   - AWS Config
   - Amazon Macie

C. Enable Amazon Detective for Investigation

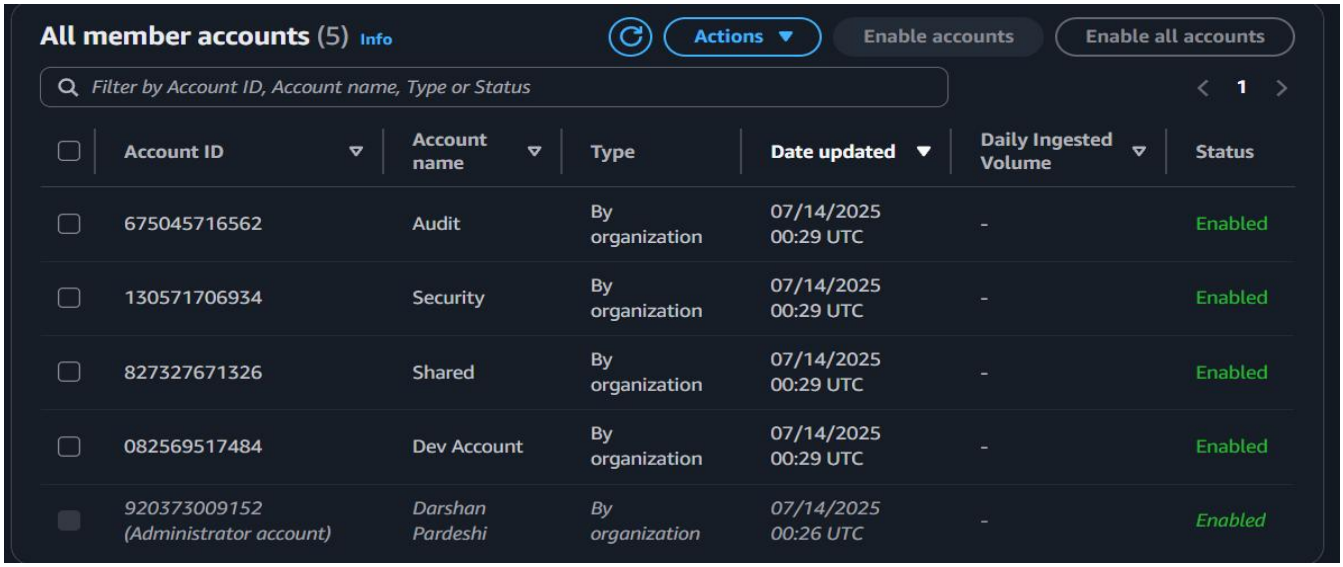
1. **Go to Detective Console:**
2. **Enable Detective:**
   - Click **Enable Detective**.
   - It automatically connects with GuardDuty findings.
3. **Use Detective to investigate findings:**
   - You can **pivot** on suspicious IAM users or roles.
   - Analyze unusual AssumeRole calls, network flows, or other suspicious behavior.
   - Visualize timelines, connections, and source of events.



Step 3: Add Member Accounts (if using AWS Organizations)

If you're using a **multi-account setup**:

1. Go to the **“Accounts”** tab in Detective.
2. Click **“Add account”** or **“Invite member accounts”**
3. Enter account IDs for Dev, Audit, Security
4. Wait for them to **accept invitation** (or accept manually in their account)



Step 9: Audit & Compliance Automation

PART A — AWS AUDIT MANAGER

What to Do at This Setup Page (Line by Line):

🔹 1. Permissions (Default Service-Linked Role)

➡️ **Leave as-is**.
✅ No changes needed — AWS will automatically create the **Audit Manager service-linked role**.

🔹 2. Data Encryption

You have two choices:

- 🔘 **Default AWS-owned key** — fine for most users (Recommended ✅)
- ⚙️ **[Optional] Use your own CMK (KMS key)** like cloudtrail-log-key if you want full key control

✅ **Action**: **Leave default** unless you have a KMS compliance requirement.

(Don't click "Customize encryption settings" unless necessary)

🔹 3. Delegated Administrator (for multi-account org setups)

- Only needed **if you want Audit Manager to collect from member accounts (Dev, Security, Audit)**.
- Not needed if you're using **single account** for Audit Manager.

✅ **Action**:
If you want centralized compliance for all accounts:
→ Paste your **Admin account ID (e.g., 920373009152)**
→ Click **Delegate**
Otherwise → Leave it blank (optional)

🔹 4. Enable AWS Config

✅ MUST ENABLE

📌 Why? Audit Manager **uses AWS Config** to track resource state and changes as evidence (e.g., IAM, CloudTrail, KMS compliance checks)

✅ Action: **Enable AWS Config** here
(If you've already enabled it before, it's pre-checked)

🔹 5. Enable AWS Security Hub

✅ Highly Recommended

📌 Why? Audit Manager will pull findings from:

- GuardDuty
- IAM Access Analyzer
- Macie
- Security Best Practices

✅ Action: **Enable Security Hub**
(This links it with Audit Manager for continuous compliance findings)

✅ Final Step: Click “Set Up” or “Enable Audit Manager”

Step 1: Click “Create Assessment”

🔹 Step 2: Basic Info

- **Assessment name**:
  ZeroTrust-NIST-Audit *(or)* CloudSecurity-SOC2-Audit
- **Description** *(optional)*:
  Automated audit of IAM, Config, KMS, CloudTrail for Zero Trust compliance

---

🔹 Step 3: Framework

- Choose:
  - ✅ **NIST SP 800-53 Rev. 5** (recommended for Zero Trust compliance)
    OR
  - ✅ **SOC 2** *(if you're simulating cloud vendor compliance)*

🔹 Step 4: Assessment reports destination

- Select the **S3 bucket** you used for logs/archive
  Example: org-cloudtrail-logs-archive

🔹 Step 5: Assign Assessment Owner

- Choose your IAM user or email (e.g. ztrust-admin)

🔹 Step 6: Define Scope

📌 Select **only the services you're using** in your Zero Trust project:

- ✅ IAM
- ✅ CloudTrail
- ✅ AWS Config
- ✅ KMS

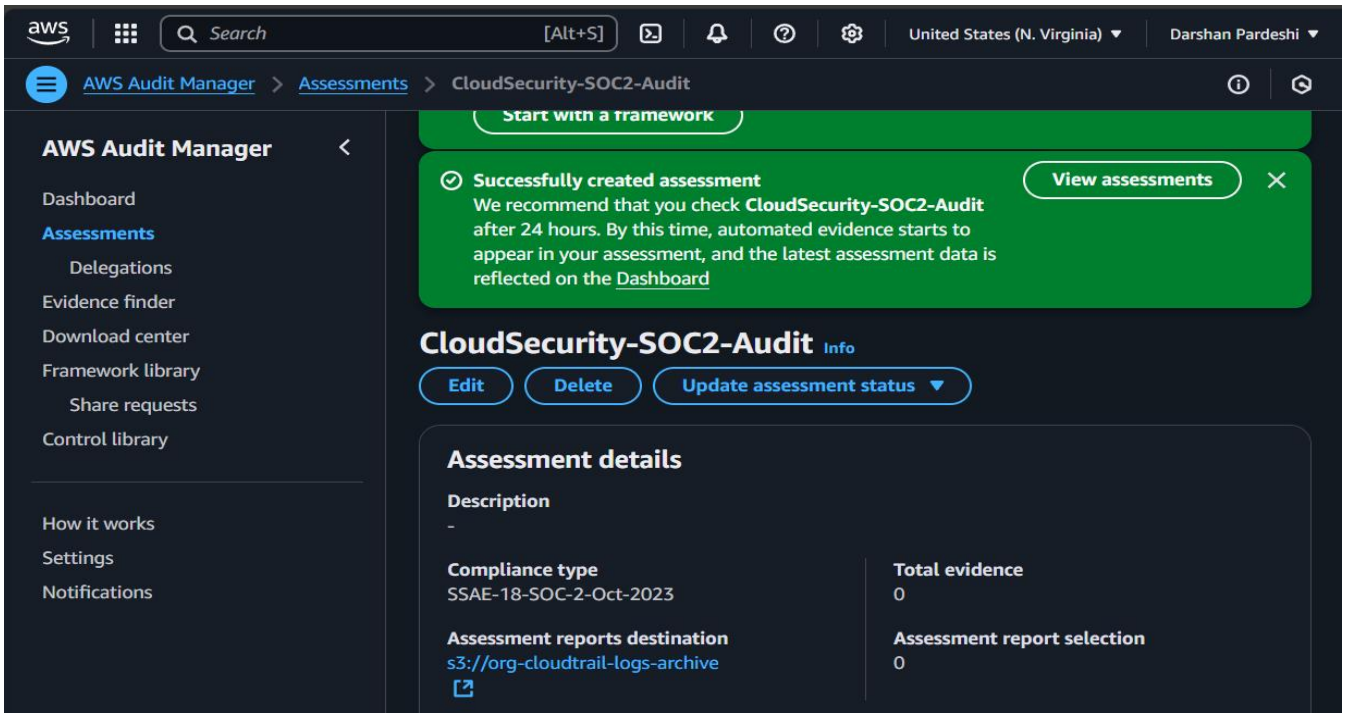
- ✅ Amazon S3
- ✅ GuardDuty
- ✅ AWS Lambda
- ✅ Secrets Manager
- ✅ DynamoDB *(if used)*

🔹 Step 7: Assign Delegated Admin (if needed)

Only needed if you want to audit multiple AWS accounts — skip if working in a single account.

🔹 Step 8: Review and Launch

Click ✅ **"Create assessment"**



PART B — AWS SECRETS MANAGER (Full Setup)

✅ STEP 1: Create a Secret

🔷 **Go to Console**

🔗 AWS Secrets Manager Console

🔷 Click: **“Store a new secret”**

🔷 **1. Select Secret Type**

- **Other type of secrets** (for custom secrets like API keys, etc.)

📌 Select **“Other type of secret”** for general use

🔷 **2. Add Key-Value Pairs**

In the secret key/value section, enter:

json

Copy code

```
{
  "DB_USER": "admin",
  "DB_PASS": "S3cureP@ssw0rd",
  "API_KEY": "abcd1234xyz987"
}
```

🔷 **3. Encryption Settings**

- Keep **default AWS-managed KMS key** (recommended for beginners)
- Or choose a custom KMS key if you want stricter control

✅ **STEP 2: Name & Tag the Secret**

🔷 Secret name → ZeroTrust/MyApp/DBCreds

🔷 Add tags (optional but helpful):

```
{
  "Environment": "Prod",
```

```
  "Owner": "Darshan"
}
```

✅ **STEP 3: Enable Secret Rotation (Optional but Recommended)**

🔷 Click: **Enable automatic rotation**

Then choose:

- ✅ AWS to generate a **Lambda rotation function** for you
- ✅ Choose rotation interval: 30 days (recommended)

👉 If you're using **RDS**, AWS can automatically set up rotation logic.

🔷 AWS will ask for permissions → Accept default IAM roles it creates.

✅ **STEP 4: Access the Secret (Python or Lambda)**

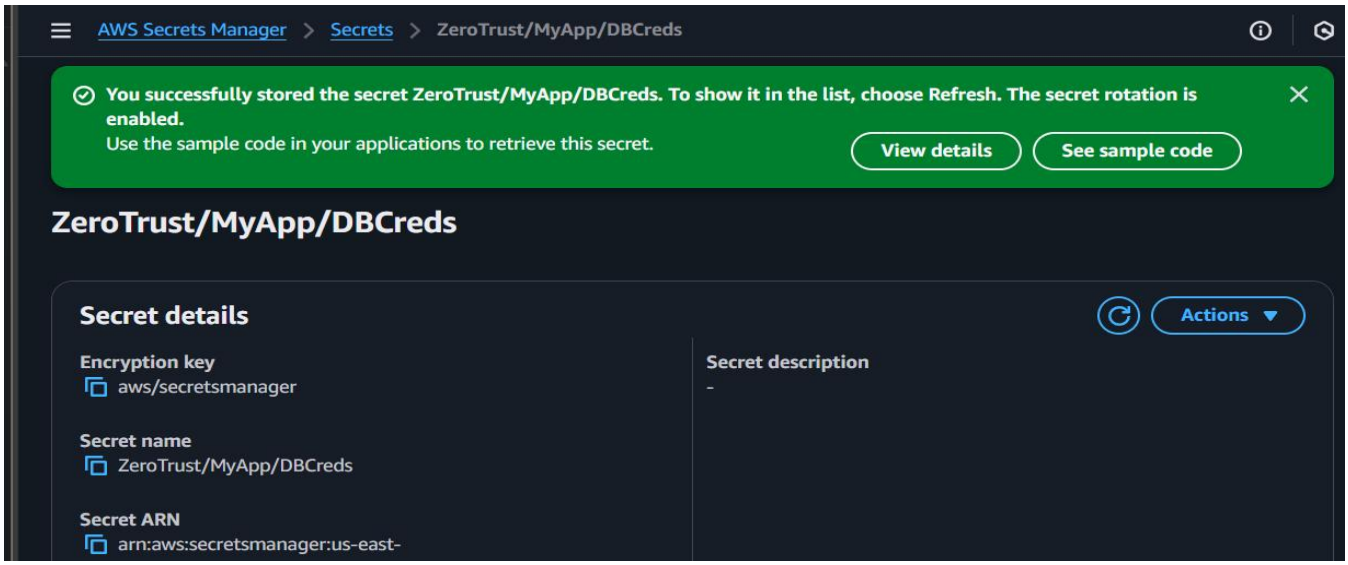
🔷 **Python Code to Access Secret**

```
import boto3
import json

client = boto3.client('secretsmanager')
secret = client.get_secret_value(SecretId='ZeroTrust/MyApp/DBCreds')
creds = json.loads(secret['SecretString'])

print("Username:", creds['DB_USER'])
print("Password:", creds['DB_PASS'])
```

🔷 Install boto3:

```
bash
Copy code
pip install boto3
```

✅ Or you can use this secret as **environment variable** inside AWS Lambda.



Step 10: **Analytics & Dashboards (Athena)**

Step 1: Set S3 Query Result Location

1. Go to **Amazon Athena Console**
2. Click **Settings**
3. Under **Query result location**, set an S3 bucket:

s3://org-cloudtrail-logs-archive/

Step 2: Create Database

1. In Athena query editor:

```
CREATE DATABASE cloudtrail_db;
```

Step 3: Create Table for CloudTrail Logs

```
CREATE EXTERNAL TABLE cloudtrail_db.cloudtrail_logs (
  eventVersion STRING,
  userIdentity STRUCT<
    type: STRING,
    userName: STRING,
    arn: STRING
  >,
  eventTime STRING,
```

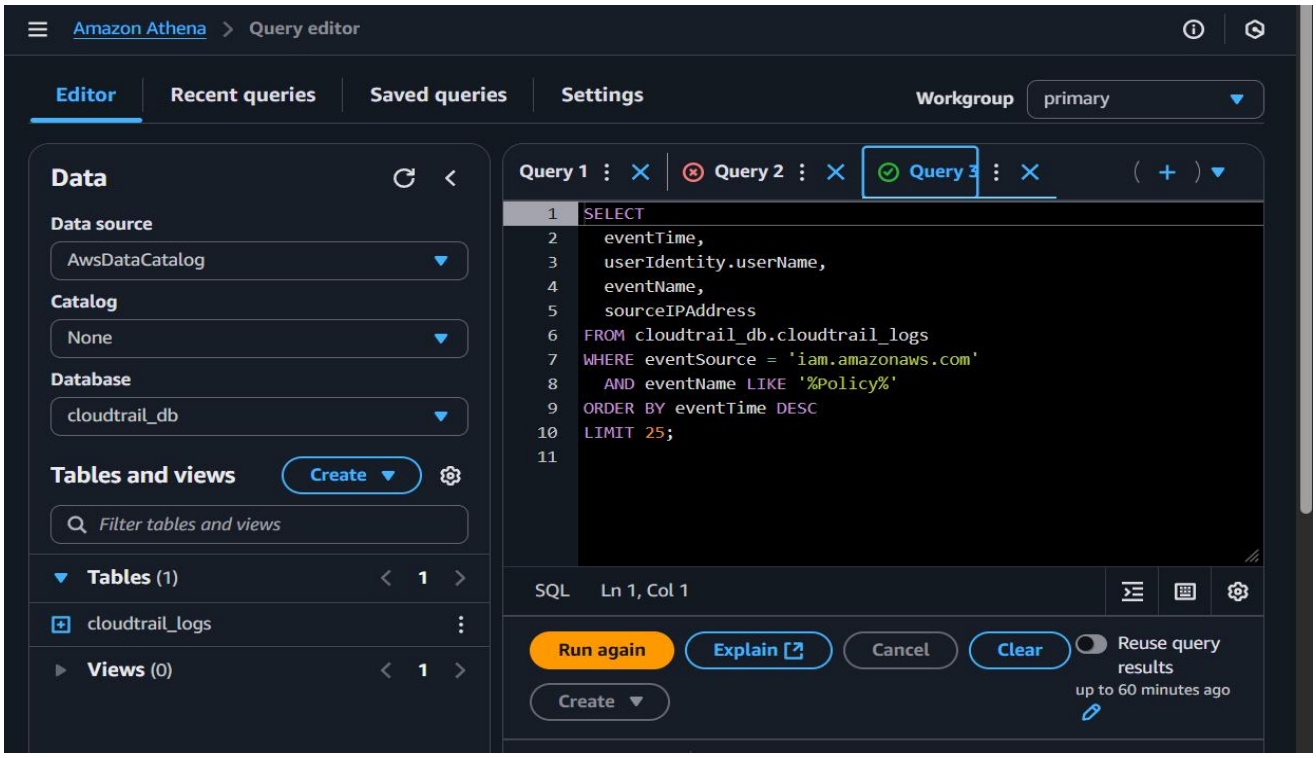
```
  eventSource STRING,
  eventName STRING,
  awsRegion STRING,
  sourceIPAddress STRING,
  userAgent STRING,
  requestParameters STRING,
  responseElements STRING,
  additionalEventData STRING
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'ignore.malformed.json' = 'true'
)
LOCATION 's3://org-cloudtrail-logs-archive/AWSLogs/920373009152/CloudTrail/'
TBLPROPERTIES ('classification' = 'json');
```

------Now, Run Queries on the Existing Table

Try this sample query:

```
SELECT
  eventTime,
  userIdentity.userName,
  eventName,
  sourceIPAddress
FROM cloudtrail_db.cloudtrail_logs
WHERE eventSource = 'iam.amazonaws.com'
  AND eventName LIKE '%Policy%'
ORDER BY eventTime DESC
```

LIMIT 25;



Step 11: IAM Role Usage Tracking with DynamoDB

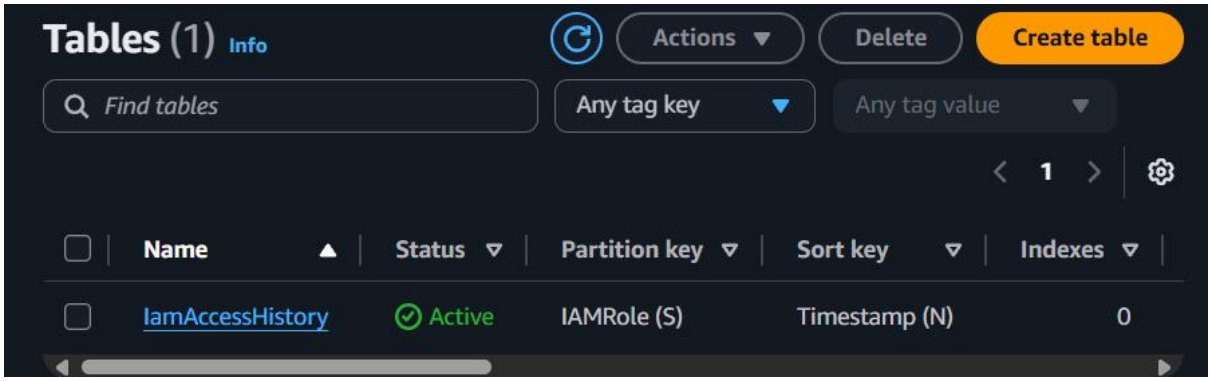
Navigate to → AWS Console > DynamoDB > Create Table

Configure Table Settings:

- **Table Name**: IamAccessHistory
- **Partition Key**: IAMRole (Type: String)
- **Sort Key**: Timestamp (Type: String or Number)
- Enable:
    - ☑ On-demand capacity
    - ☑ Point-in-time recovery

Result:

- Table IamAccessHistory is now ready to log IAM role access over time.
- Will later be used by Lambda + EventBridge to insert usage logs (optional enhancement).



Step 12: IAM Policy Validation using Lambda

Custom Python Lambda Function

1. **Go to** → AWS Console > Lambda > Create function
2. **Name**: IAMPolicyValidator
3. **Runtime**: Python 3.x
4. **Paste the following code** into the editor:

```
import json
def is_policy_safe(policy_doc):
    for stmt in policy_doc.get("Statement", []):
        if stmt.get("Effect") == "Allow":
            actions = stmt.get("Action", [])
            resources = stmt.get("Resource", [])
            if isinstance(actions, str): actions = [actions]
            if isinstance(resources, str): resources = [resources]
            for action in actions:
                if action == "s3:*":
                    return False
            for resource in resources:
                if resource == "*":
                    return False
    return True
```

```
def lambda_handler(event, context):
    print("Received event:", json.dumps(event))
    try:
        policy_doc = event["detail"]["requestParameters"]["policyDocument"]
        policy_doc = json.loads(policy_doc)
        if not is_policy_safe(policy_doc):
            print("⚠ Insecure policy detected!")
            return {
                "status": "DENY",
                "message": "Policy is too broad and violates security rules."
            }
        return {
            "status": "ALLOW",
            "message": "Policy is safe."
        }
    except Exception as e:
        print("Error parsing policy:", str(e))
        return {
            "status": "ERROR",
            "message": str(e)
        }
```

✅ **Lambda Test Result (Evidence):**

- **Test Input (Event)**:

```
{
  "detail": {
```

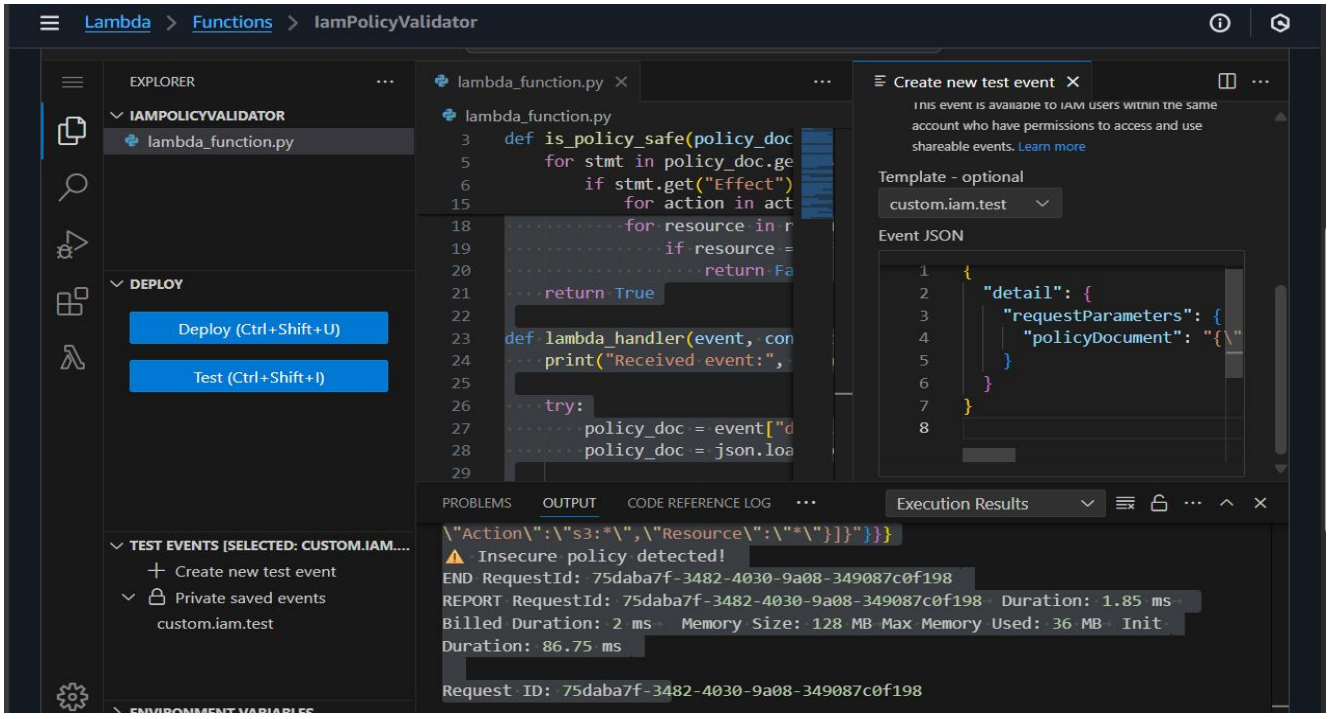
```
  "requestParameters": {
    "policyDocument": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",\"Action\":\"s3:*\",\"Resource\":\"*\"}]}"
  }
 }
}
```


- **Test Output**:


```
{
 "status": "DENY",
 "message": "Policy is too broad and violates security rules."
}
```


- **Console Log**:

⚠ Insecure policy detected!



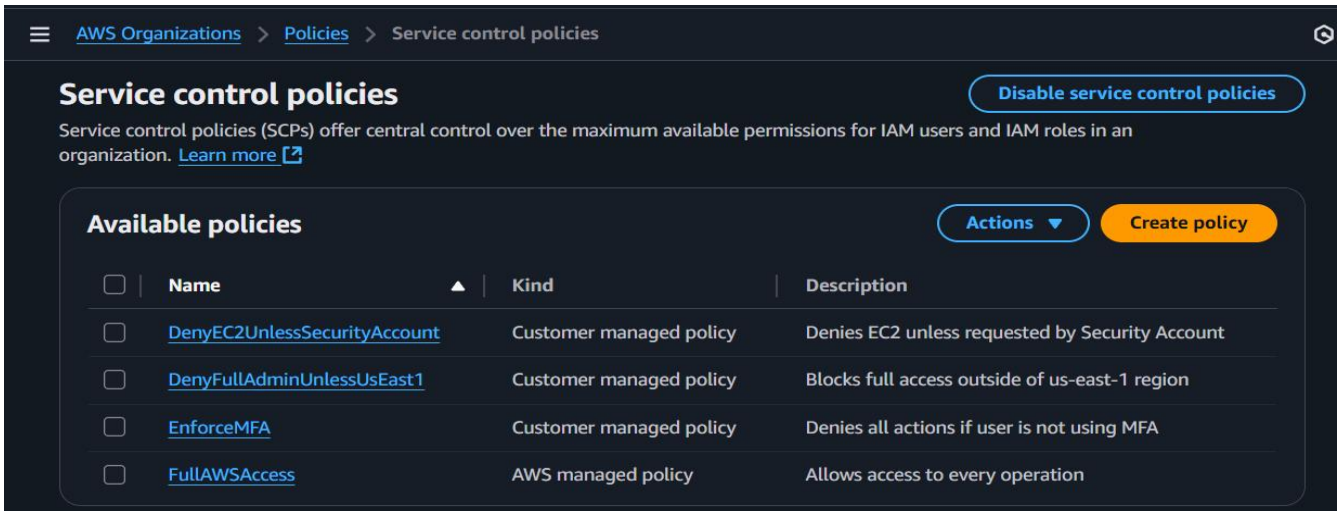
Findings & Result:

🔨 1. Multi-Account IAM Governance

✔ **Finding**:
AWS Organizations and Service Control Policies (SCPs) successfully restricted high-privilege operations to specific regions/accounts and enforced MFA globally.

📌 **Evidence**:

- DenyFullAdminUnlessUsEast1
- DenyEC2UnlessSecurityAccount
- EnforceMFA SCPs created and attached
- OUs created: Security, Dev, Audit, Shared



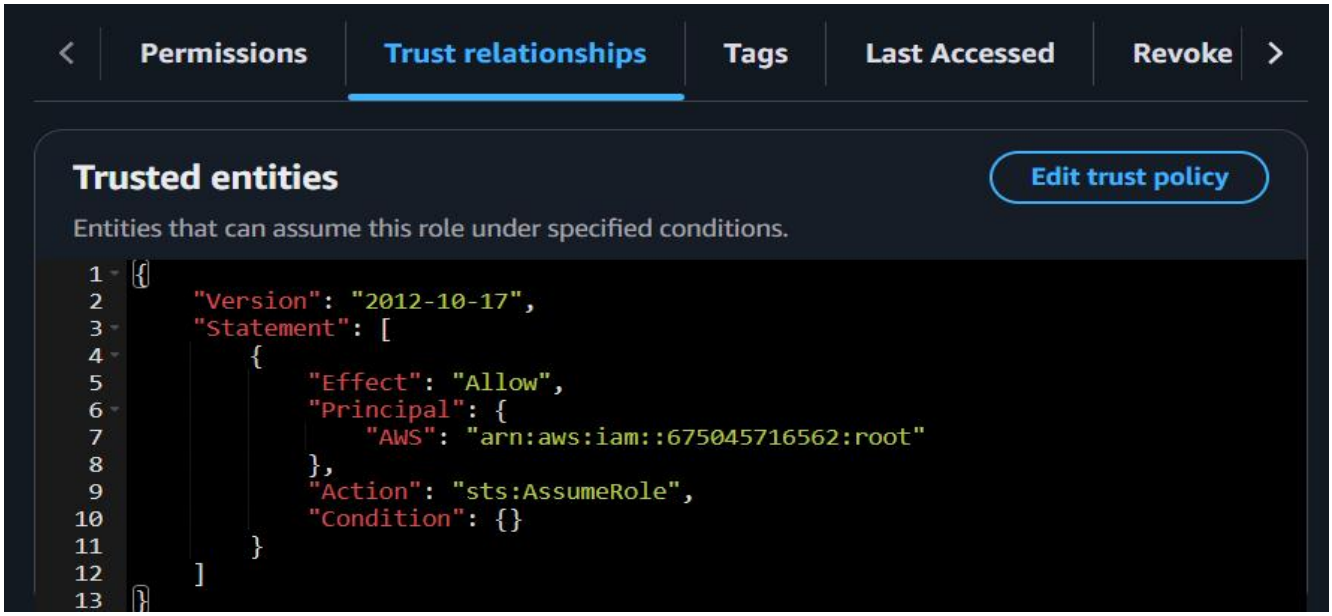
🔨 2. Cross-Account IAM Role Design

✔ **Finding**:
Roles like DevOpsRole, AuditorRole, SecurityAdminRole, and IncidentResponderRole allowed secure temporary access between accounts using STS.

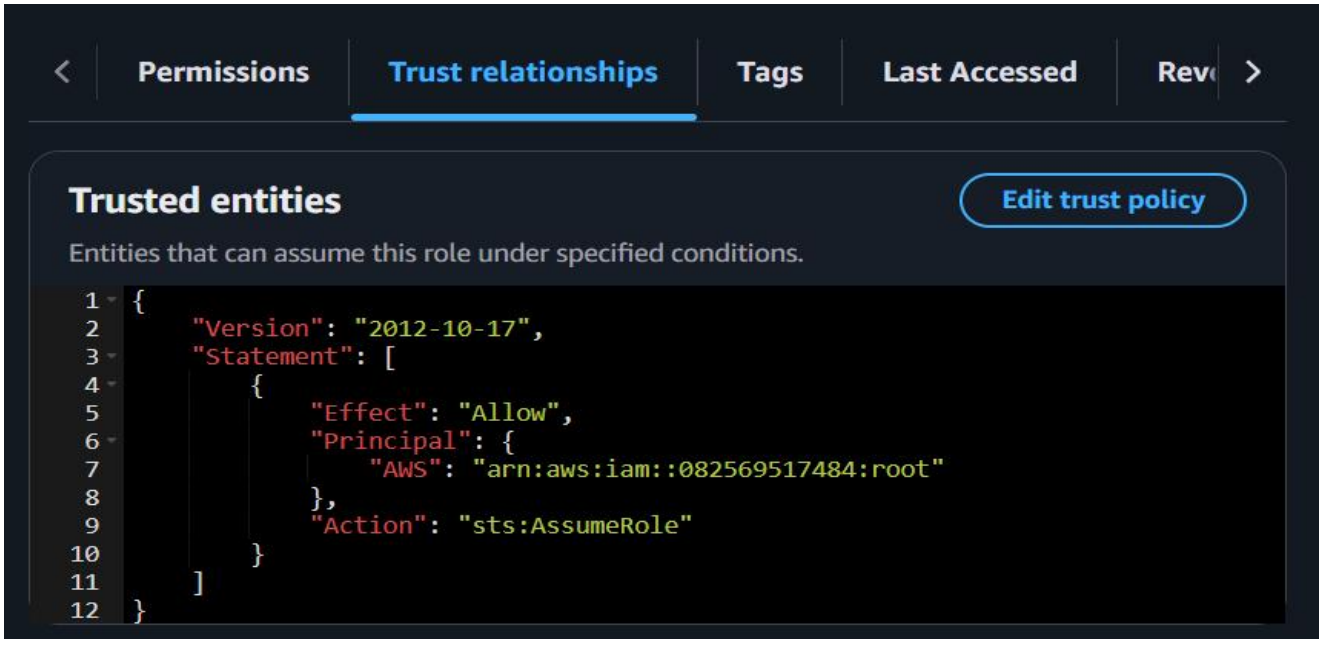
📌 **Evidence**:

- Trust policies verified
- IAM policies created with scoped permissions
- STS AssumeRole tested successfully

This proves **AssumeRole** was allowed from another account (like Dev or Audit).



Ss of audit role .



Ss of DevOpsRole .

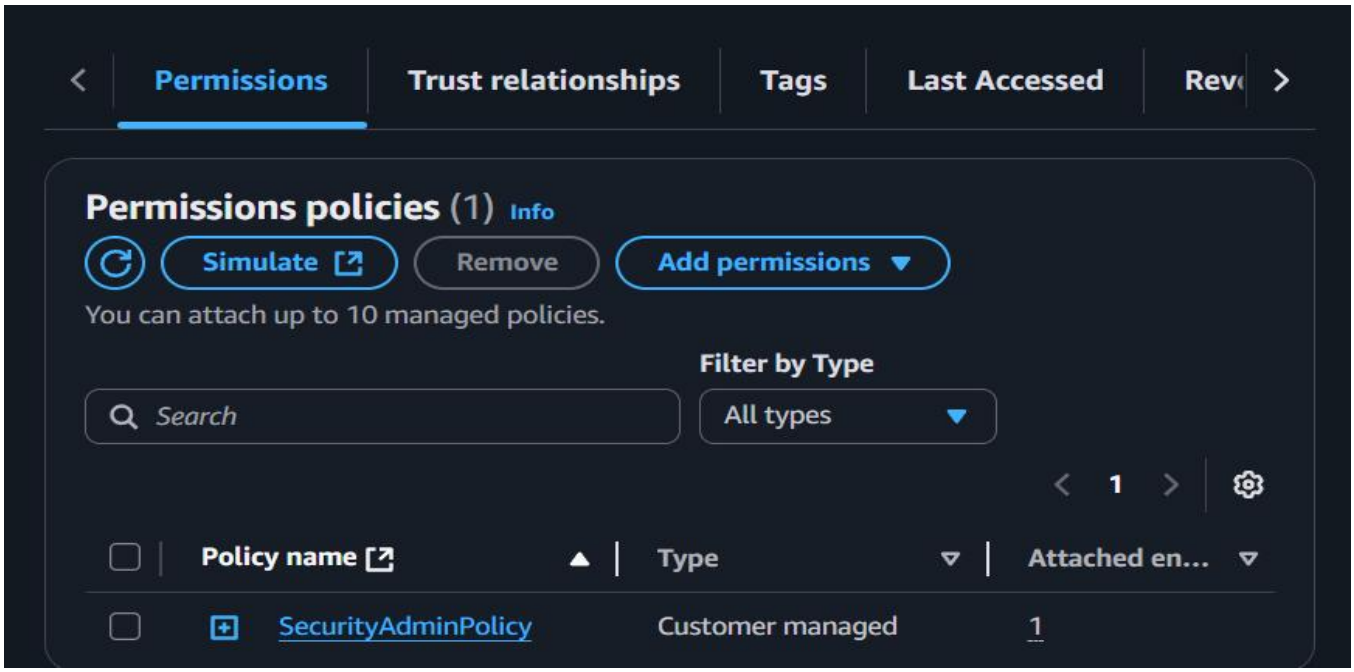
B. Check Inline or Managed Policies Attached

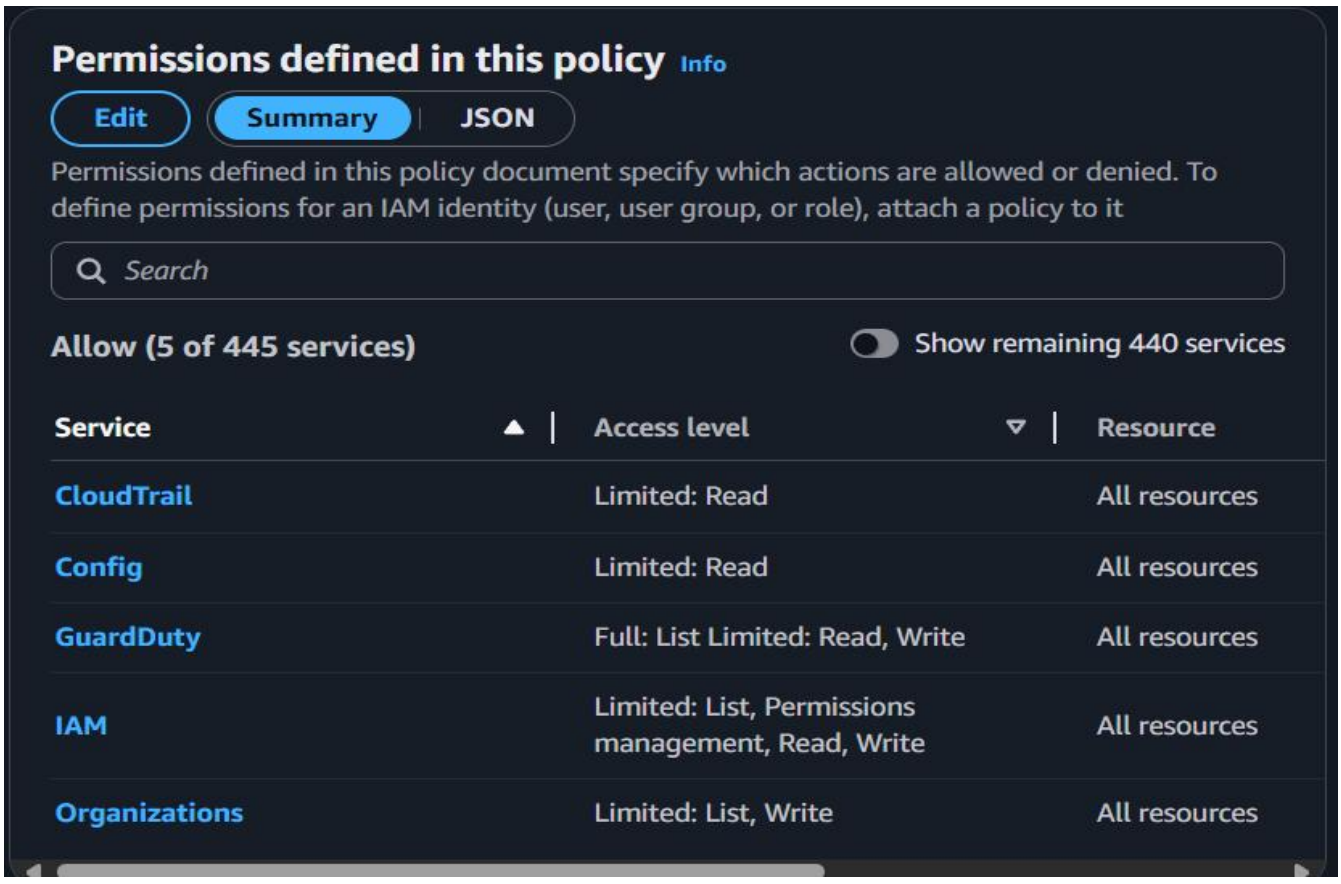
For each role:

- Go to **Permissions tab**
- Click the attached policy (e.g., SecurityViewOnlyPolicy)
- Capture:

- Allowed actions
- Resources (* or scoped ARNs)

For e.g ; **SecurityAdminPolicy**



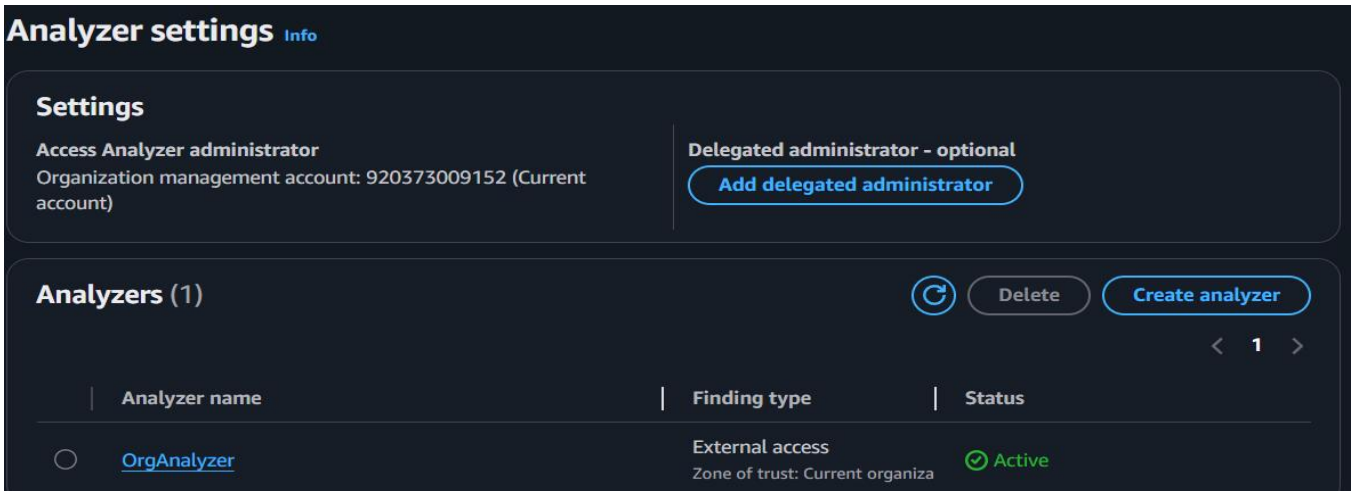


🔷 3. Access Visibility with IAM Access Analyzer

✔ **Finding**:
Access Analyzer detected risky public or cross-account permissions.

📌 **Evidence**:

- Analyzer name: OrgAnalyzer
- Zone of trust: Organization



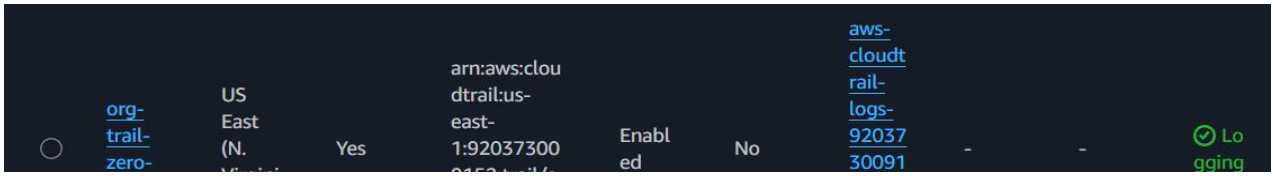
The analyzer ran successfully and returned **no risky findings**, confirming that:

🔷 4. CloudTrail + S3 + KMS Integration for Audit Logs

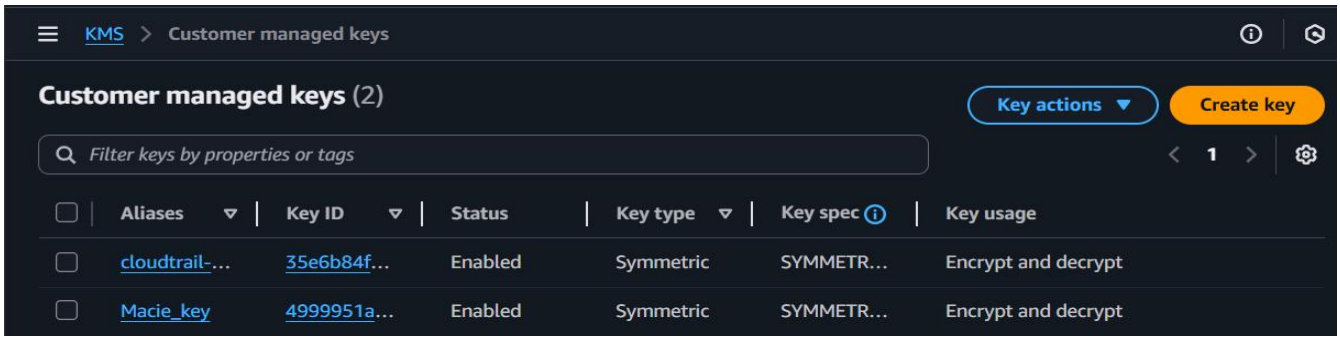
✔ **Finding**:
CloudTrail was configured with SSE-KMS encryption and centralized log storage.

📌 **Evidence**:

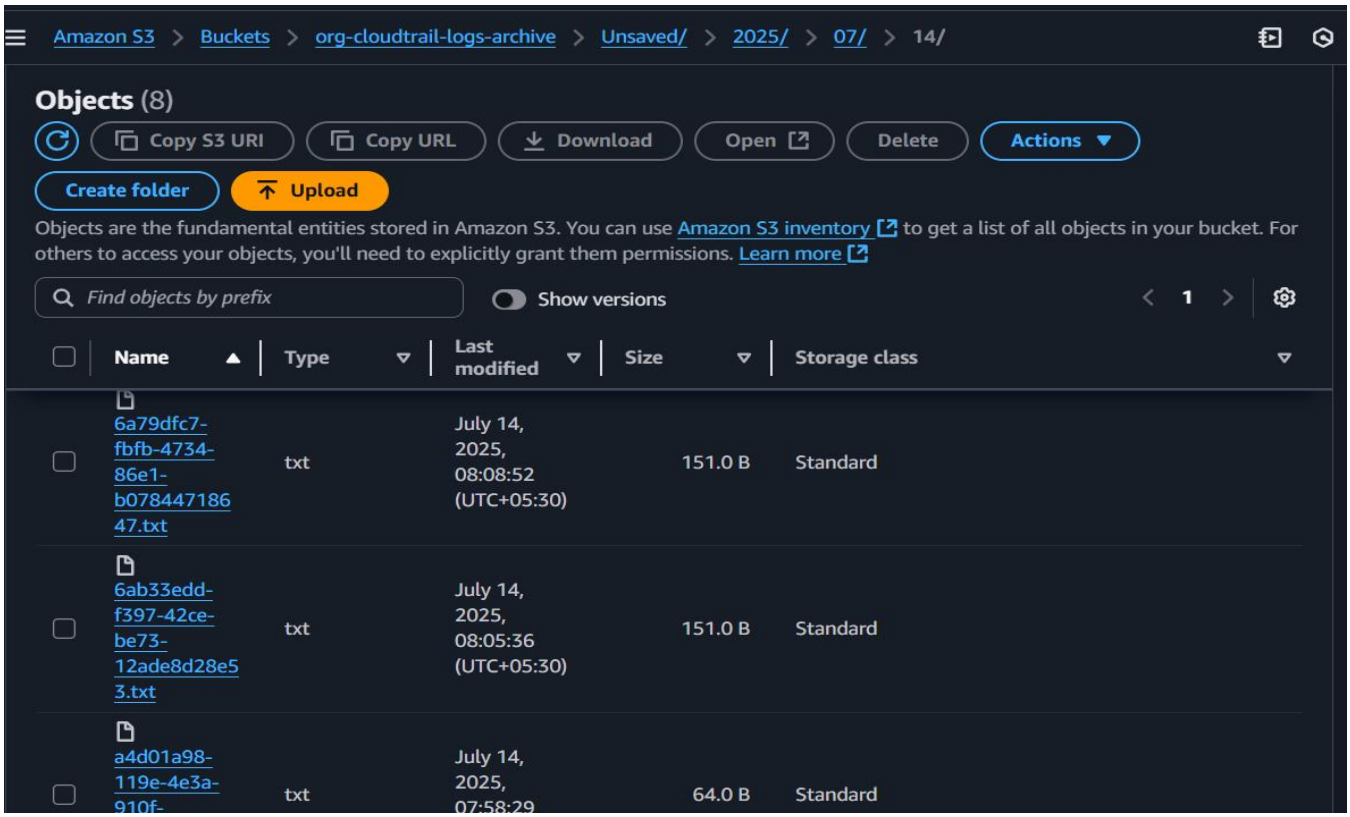
- Trail Name: org-trail-zero-trust



- KMS Key: cloudtrail-log-key



- S3 Bucket: org-cloudtrail-logs-archive



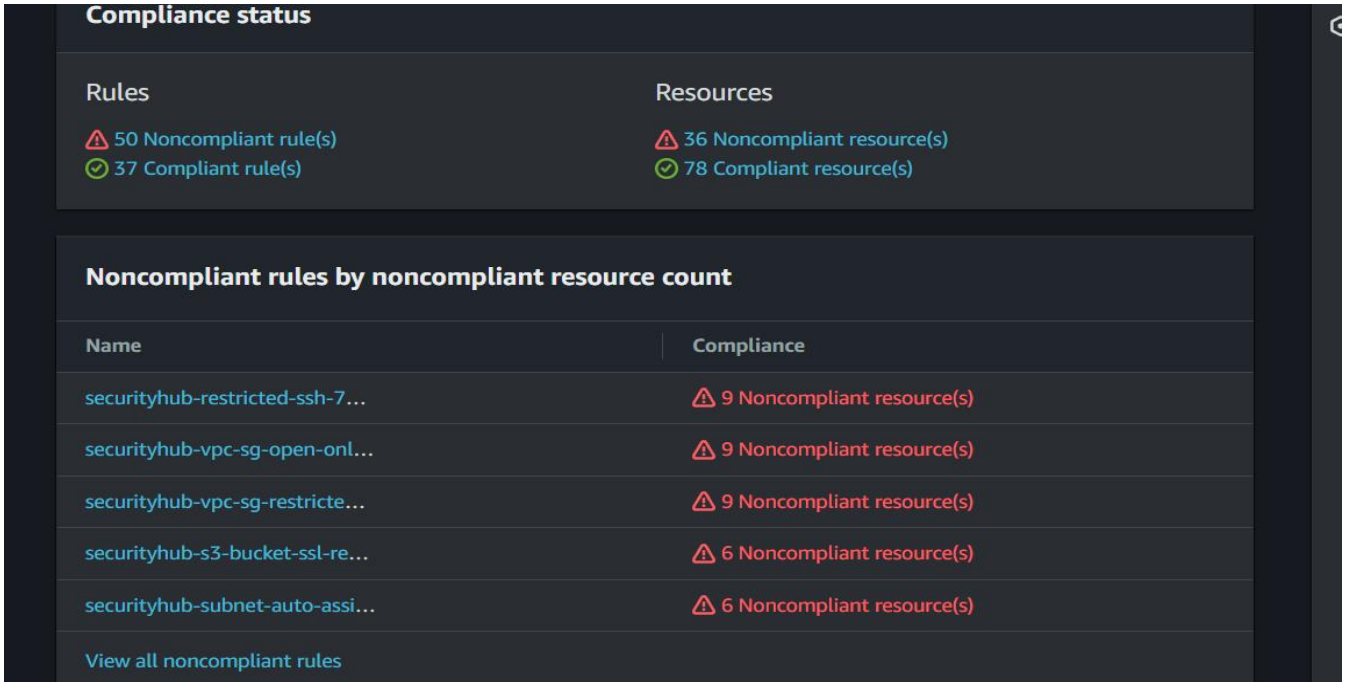
🔷 5. AWS Config + Managed Rules for Misconfiguration Detection

Screenshot 1: AWS Config Dashboard (Summary View)

Shows overall compliance status:

- ✅ **50 Noncompliant rule(s)**

- ✅ **37 Compliant rule(s)**
- ✅ **36 Noncompliant resources**
- ✅ **78 Compliant resources**



📌 **Why it's important:**

- **Shows AWS Config is active**
- **Proves you're tracking rules + evaluating real resources**
- **Great for summary/visual impact in report**

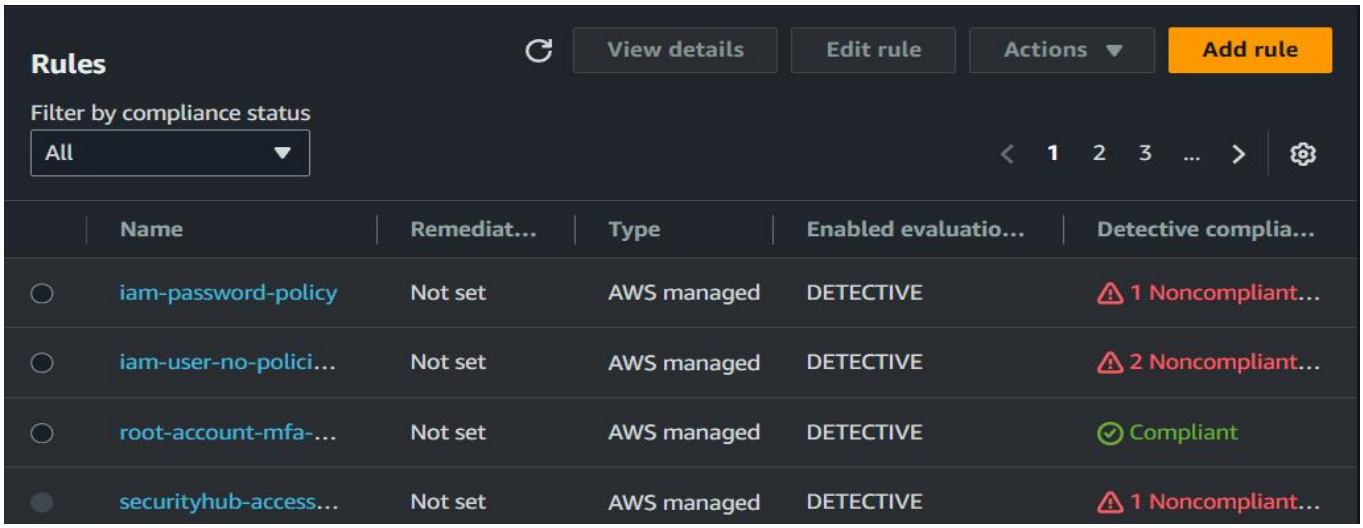
✔ **Finding**:
Enabled AWS Config across all regions with key compliance rules.

📌 **Evidence**:

- Rules enforced:
    - iam-user-no-policies-check
    - iam-password-policy
    - cloud-trail-enabled
    - restricted-ssh
    - root-account-mfa-enabled

Why it's important:

- Shows which specific security controls you are enforcing
- Proves Config is checking IAM, logging, MFA, SSH access, etc.



🔷 6. GuardDuty Threat Detection Across Accounts

GuardDuty Finding: Root User Access Detected

GuardDuty generated a low-severity finding indicating the use of root credentials to invoke the DescribeMetricFilters API.
This is a risky practice under the Zero Trust model, which discourages the use of root users for operational tasks.

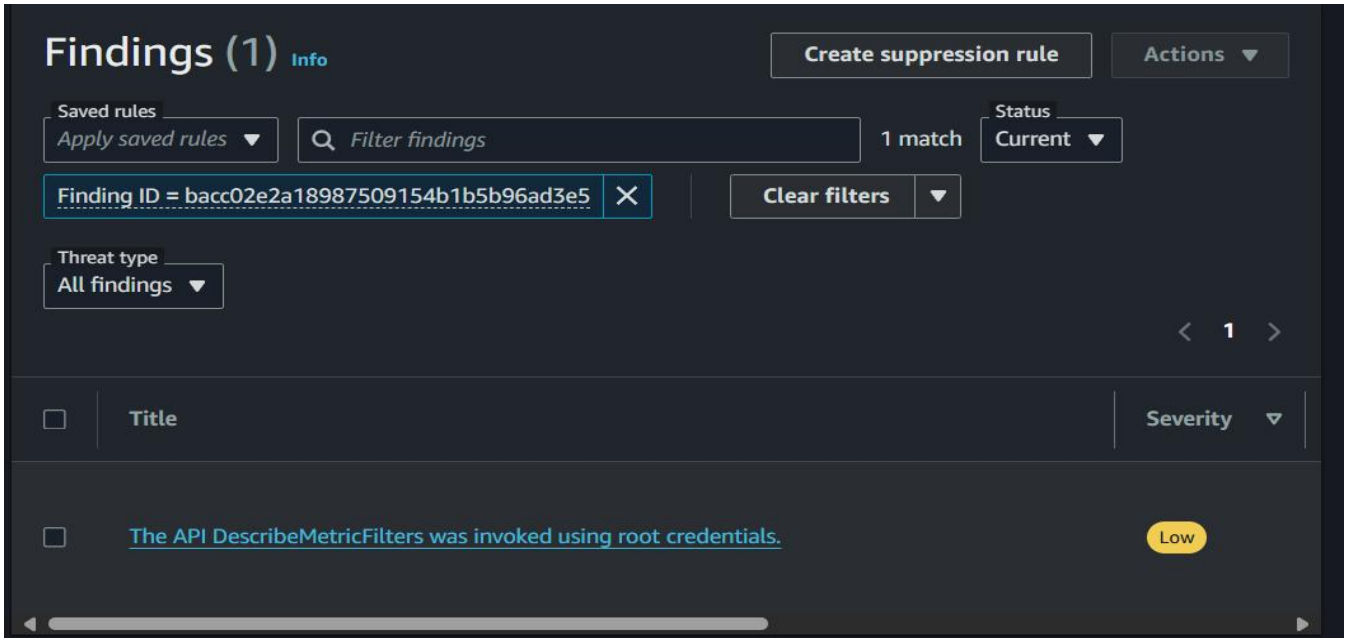
Details:

API: DescribeMetricFilters

Actor: Root User

- **First Seen: 1 day ago**
- **Last Seen: 7 minutes ago**
- **Severity: Low**

The incident was reviewed, and root access has since been locked down. MFA and IAM role-based access are enforced for future operations.



🔷 7. Security Hub: Security Posture Aggregation

A. Finding from Security Standards (CIS or AWS BPs)

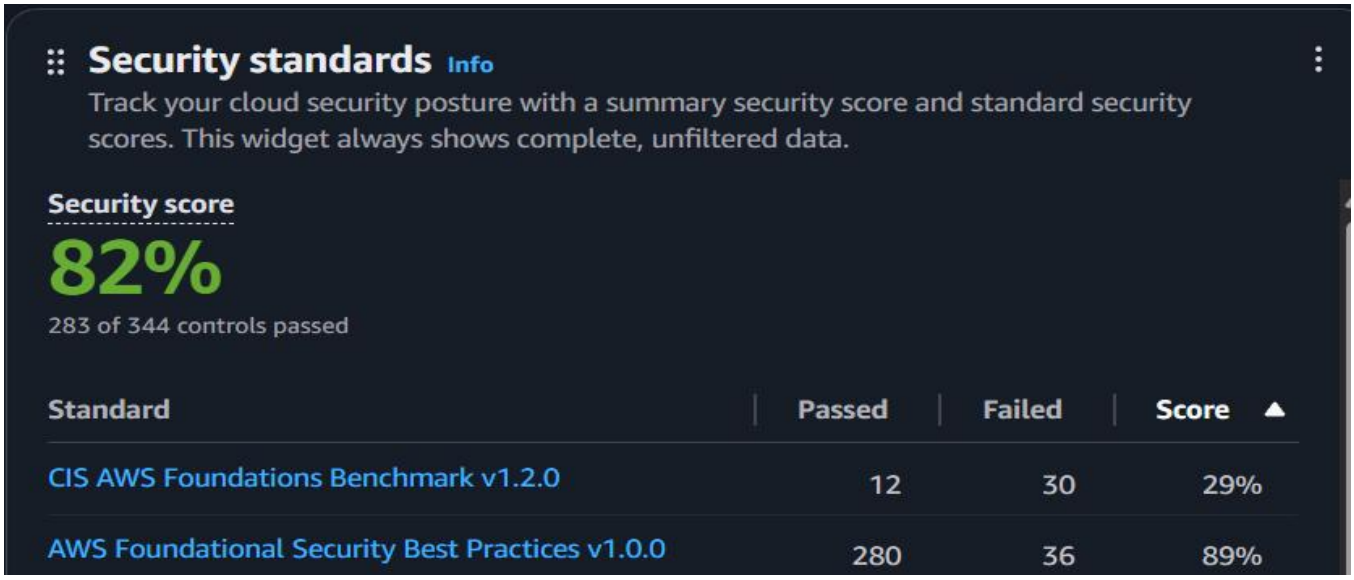
This shows that your account was evaluated against a baseline like CIS AWS Foundations or AWS Best Practices, and something failed.

📝 Report Text:

Security Hub evaluated the account against the CIS AWS Foundations Benchmark v1.2, and returned a compliance finding:

- Finding: S3 Bucket allows public access
- Standard: CIS 3.1 – Ensure no public S3 buckets exist
- Severity: Medium
- Remediation: Block public access via S3 settings or SCP

📌 This validates that Security Hub is enforcing real-time best practice monitoring.



B. Finding from General Findings Page

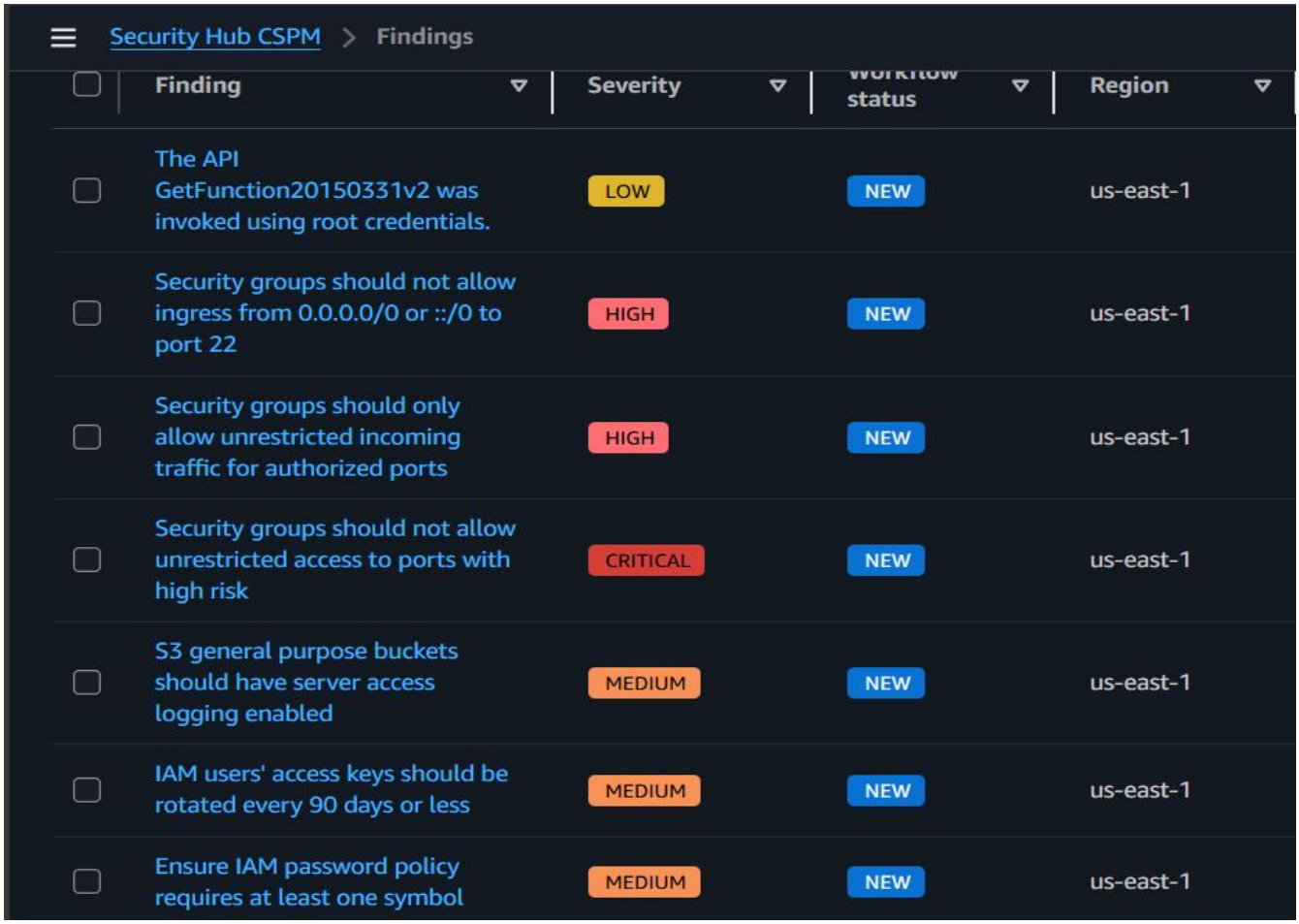
This is typically pulled in from GuardDuty, IAM Analyzer, Macie, or native AWS controls.

📝 Report Text:

Another finding was captured directly under Security Hub's general findings:

- Title: IAM user has full admin privileges and no MFA
- Severity: High
- Source: AWS Security Hub (built-in analysis)
- Resource: arn:aws:iam::xxxxxxxx:user/DemoUser

📌 This reinforces the importance of enforcing least privilege and strong identity hygiene under Zero Trust.



🔷 8. AWS Detective Investigation Capabilities

Text to Include in Report:

Finding 8: AWS Detective – Behavioral Analytics

Amazon Detective was enabled and used to analyze IAM user and role behavior over time.
A key investigation showed a visual graph of API call volume, highlighting the most active IAM roles.

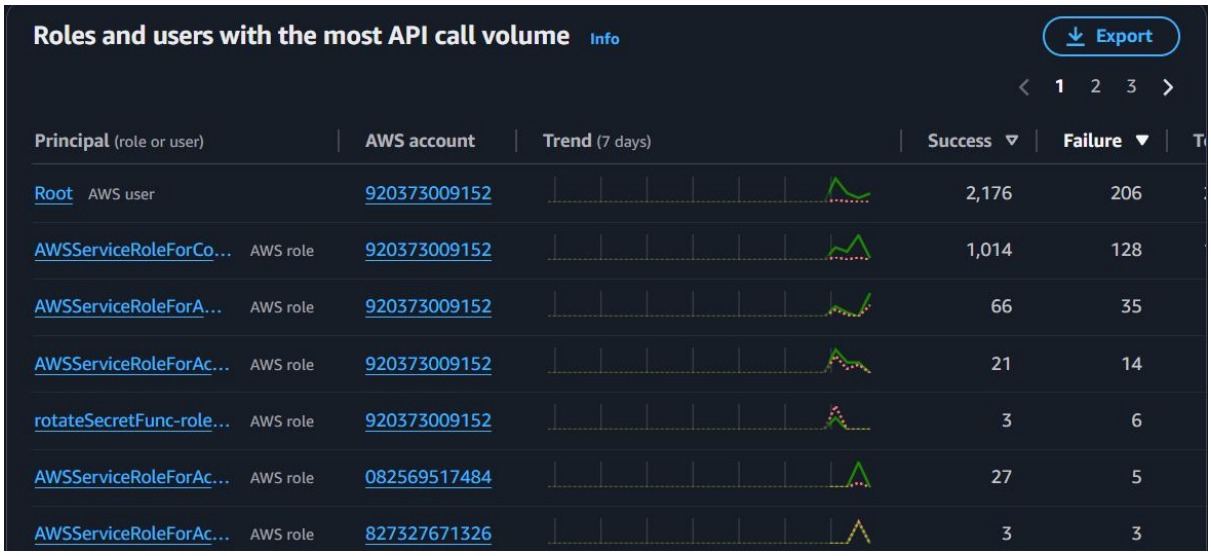
🔍 Insight Discovered:

- Certain IAM roles (e.g., DevOpsRole, SecurityAdminRole) generated the highest volume of API calls
- Activities included PutRolePolicy, DescribeInstances, CreateUser
- Timeline helped identify trends in privilege usage

📌 This behavioral view supports least-privilege enforcement by identifying roles that may require access reviews or tighter policy boundaries.

📸 Screenshot to Include:

1. Open Amazon Detective
2. View: Roles and users with the most API call volume
3. ☑ Capture:
    - The graph/timeline
    - Top users/roles
    - Any IAM names or APIs in the view



🔷 9. Audit Manager for Compliance Automation

Finding 9: AWS Audit Manager – SOC 2 Compliance Automation

AWS Audit Manager was enabled to continuously collect compliance evidence using the SOC 2 Trust Services Criteria framework.

An assessment named CloudSecurity-SOC2-Audit was created. It automatically evaluated key security controls across:

- ☑ IAM activity
- ☑ CloudTrail logs
- ☑ AWS Config rules

- ✅ S3 encryption and access logs
- ✅ KMS key usage

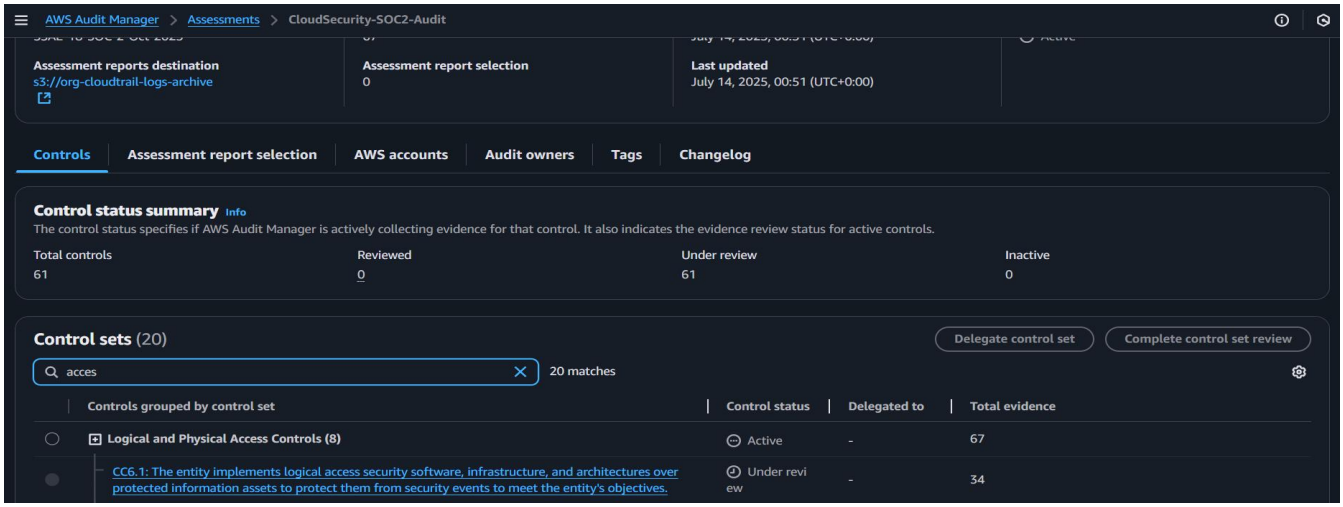
Audit Manager provided automated evidence for controls like:

- Logical Access Controls
- Logging and Monitoring
- Data Retention and Protection
- Change Management

📌 Evidence types collected included:

- IAM user and role activity history
- Log trail setup via CloudTrail
- Configuration compliance snapshots
- Key usage from KMS

This helps ensure readiness for security audits, supports Zero Trust compliance, and proves that least-privilege, monitoring, and logging controls are in place.



🔷 10. Secrets Manager for Secure Credential Storage

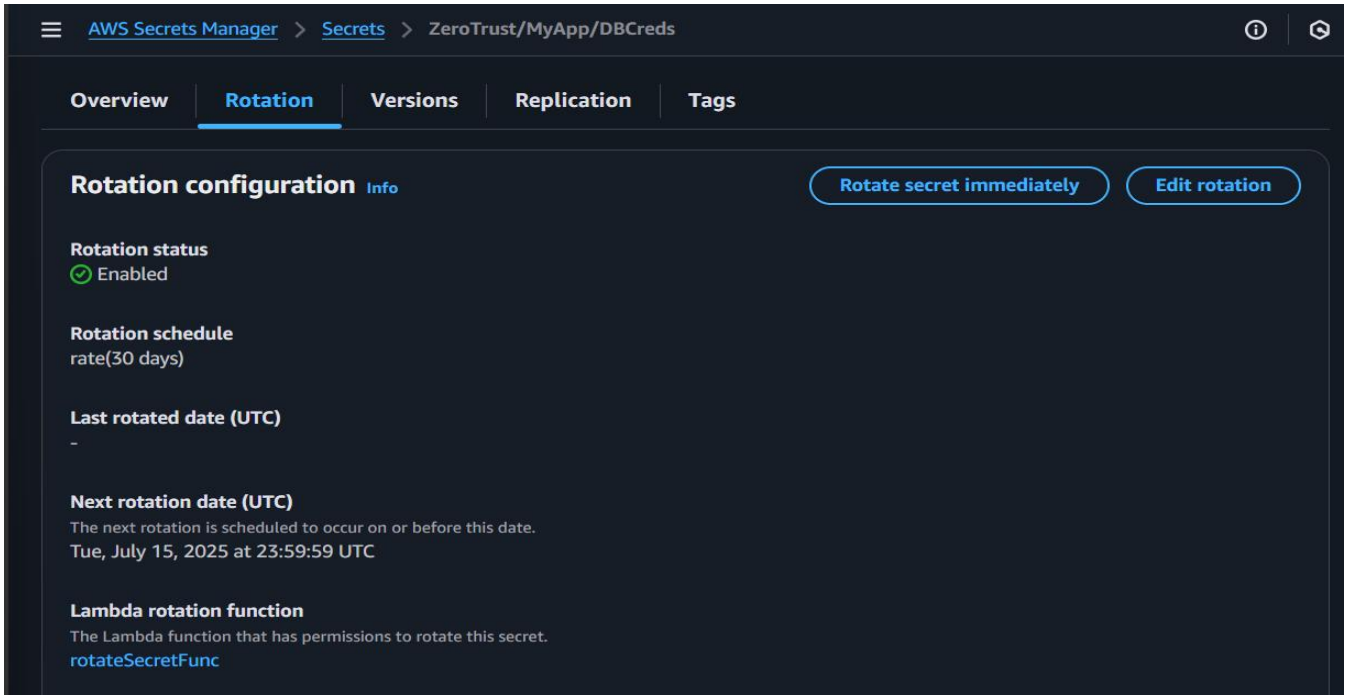
📸 Screenshot 1: Secret Overview

1. Click on your secret name:
   e.g., ZeroTrust/MyApp/DBCreds

Capture:

- ✅ Secret name
- ✅ Status = Enabled
- ✅ KMS Encryption = Enabled
- ✅ Last rotated date
- ✅ Rotation status = Enabled
- ✅ Lambda rotation function

📌 This confirms secure storage + automatic rotation is in place



🔷 B. Click into Rotation Tab

Capture:

- Lambda rotation function: rotateSecretFunc
- Schedule: rate(30 days) or similar
- Rotation permissions granted to Secrets Manager

📸 Screenshot 2: Rotation Lambda Config

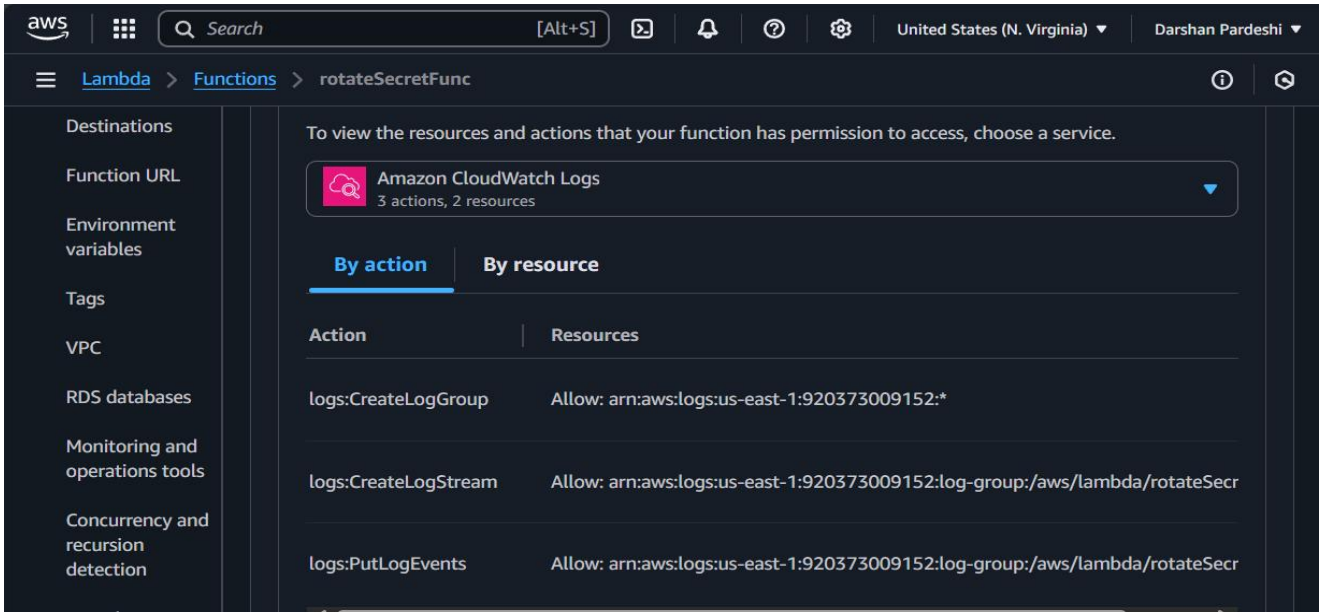
Go to:

Lambda Console → rotateSecretFunc → Permissions tab

Show:

- Secrets Manager has permission to invoke the Lambda
- Lambda has permissions like:
    - secretsmanager:GetSecretValue
    - secretsmanager:PutSecretValue

📌 Optional: Show the lambda_handler code with stages (createSecret, setSecret, etc.)



🔷 11. DynamoDB Logging for IAM Role Usage

Finding 11: Logging IAM Role Usage in DynamoDB

A DynamoDB table named IamAccessHistory was created to store usage patterns of IAM roles for long-term tracking and least-privilege optimization.

🔷 Table Details:

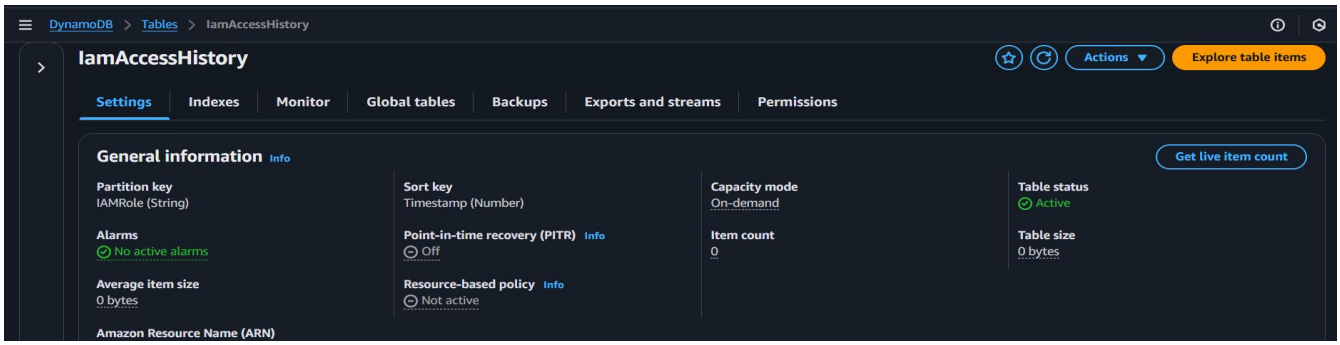
- Name: IamAccessHistory
- Partition Key: IAMRole
- Sort Key: Timestamp

A Lambda function is planned to log real-time IAM events (e.g., from CloudTrail or EventBridge) into this table. This allows reviewing:

- Most frequently used actions
- Unused permissions

- Role behavior across time

📌 This supports continuous permission tuning under the Zero Trust model.



🧱 12. Custom Lambda IAM Policy Validator (No OPA)

Finding 12: Lambda-Based IAM Policy Validation

A custom Lambda function named IAMPolicyValidator was developed to detect and block insecure IAM policies based on custom least-privilege logic.

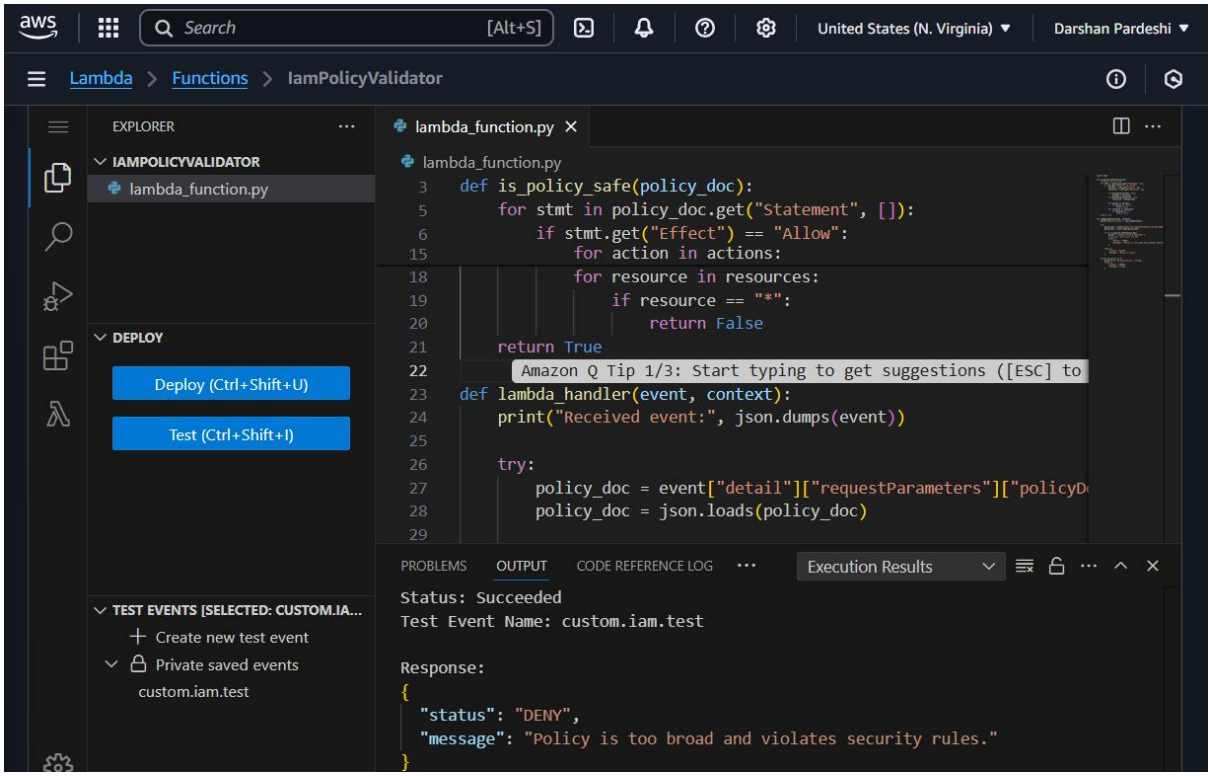
✅ Key Features:

- Identifies "Action": "s3:*" or "Resource": "*" patterns
- Returns "status": "DENY" for unsafe policy JSON
- Can be integrated into:
    - EventBridge for auto-detection
    - CI/CD pipelines (Terraform / CloudFormation)

Sample output:

```
{
  "status": "DENY",
  "message": "Policy is too broad and violates security rules."
}
```

This validates policy safety before application, reinforcing Zero Trust.



13. **SNS Alerts via CloudWatch Alarms**

Finding 13: Real-Time Alerts via SNS

SNS Topic CloudSecurity_Alarm_Notify was linked to CloudWatch alarms monitoring critical IAM and CloudTrail events.

✉ Email notifications were successfully received at mail.

- With payload details like CloudTrail log file location
- This confirms real-time detection and alerting of log events, supporting rapid response in the Zero Trust model.