Enterprise Identity and Access Management (IAM) Implementation on AWS

Author: Adedayo
Specialization: Cloud Security & Identity Management
Platform: Amazon Web Services (AWS)

A healthcare technology organization managing sensitive patient data identified inconsistent identity and access controls across its AWS environment. Multiple teams had administrative privileges, IAM policies were not standardized, and access reviews were performed manually without central visibility. Due to regulatory requirements such as data privacy and healthcare compliance standards, the organization required a structured IAM governance framework to enforce least privilege and centralized access control. I designed and implemented this framework to strengthen access controls, enforce security standards, automate identity workflows, and improve operational visibility. The solution supports secure operations, regulatory compliance, and long-term scalability.

---

1. Objectives

The primary objectives of this implementation were to:

- Enforce least privilege access across all AWS accounts
- Require multi-factor authentication (MFA) for all users
- Automate IAM access key rotation
- Centralize logging and monitoring
- Improve audit readiness
- Reduce identity-related security risks
- Establish consistent identity governance

---

2. User and Access Management

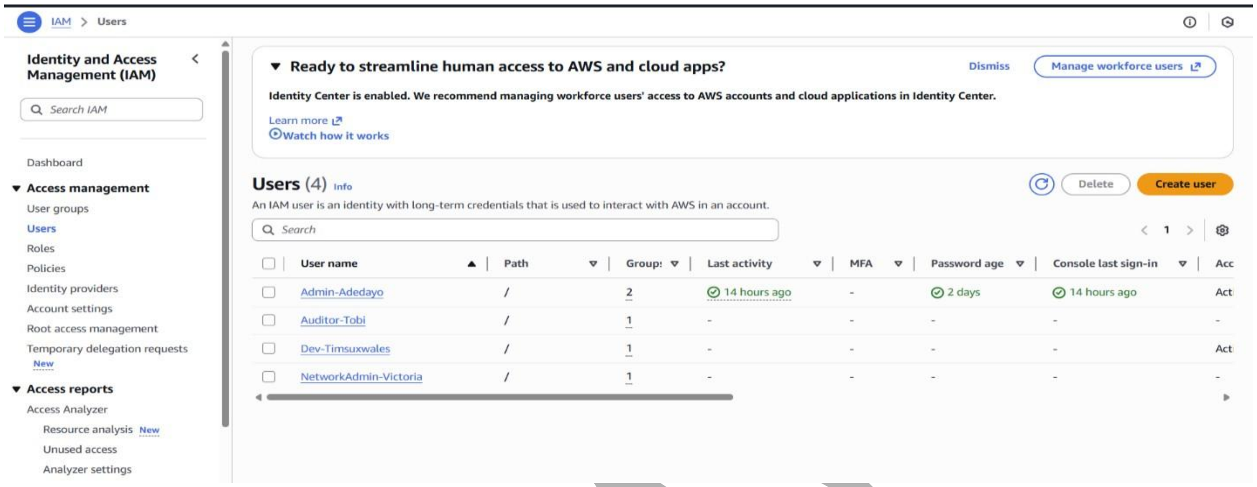
2.1 User Provisioning

I provisioned IAM users based on operational roles within the organization:

- Administrator
- Developer

- Auditor
- Network Administrator

Each user was assigned permissions aligned with job responsibilities and business requirements.


IAM > Users

Identity and Access Management (IAM)

Search IAM

Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management
Temporary delegation requests New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings

Ready to streamline human access to AWS and cloud apps? Dismiss Manage workforce users

Identity Center is enabled. We recommend managing workforce users' access to AWS accounts and cloud applications in Identity Center.

Learn more
Watch how it works

Users (4) Info

Delete Create user

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Path	Group:	Last activity	MFA	Password age	Console last sign-in	Acc
Admin-Adedayo	/	2	✓ 14 hours ago	-	✓ 2 days	✓ 14 hours ago	Act
Auditor-Tobi	/	1	-	-	-	-	-
Dev-Timsuxwales	/	1	-	-	-	-	Act
NetworkAdmin-Victoria	/	1	-	-	-	-	-

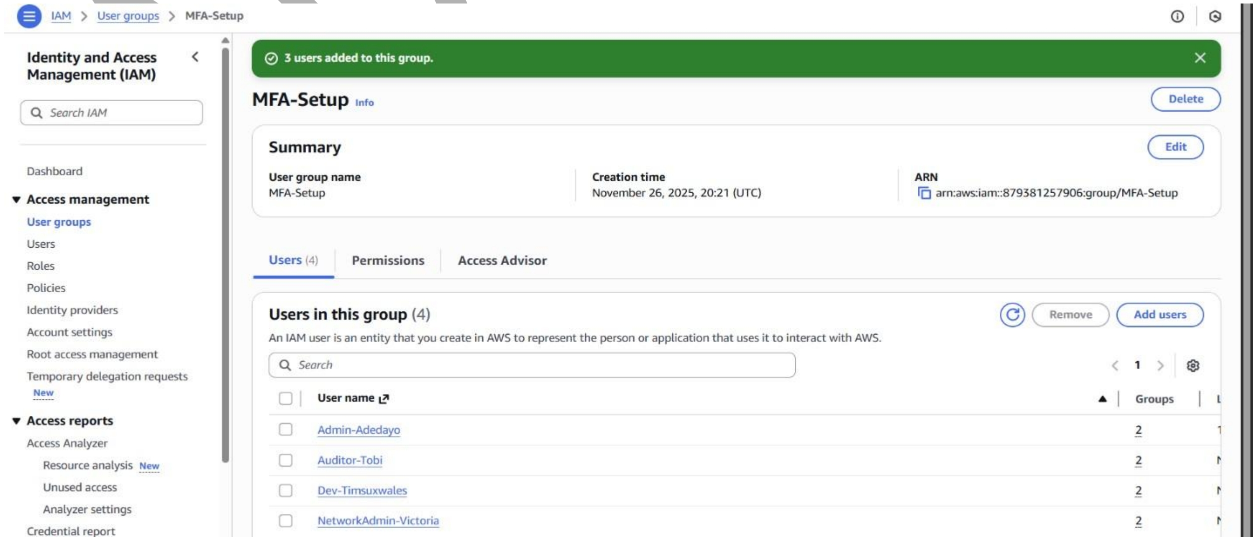
[Figure 1: IAM Users]

2.2 MFA Enforcement

I implemented a mandatory MFA onboarding process.

All new users were placed in a restricted onboarding group and required to complete MFA enrollment before accessing any AWS resources. Until MFA was enabled, access remained limited.

This approach ensured consistent enforcement of strong authentication controls.


IAM > User groups > MFA-Setup

Identity and Access Management (IAM)

Search IAM

Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management
Temporary delegation requests New
Access reports
Access Analyzer
Resource analysis New
Unused access
Analyzer settings
Credential report

3 users added to this group.

MFA-Setup Info

Delete

Summary

Edit

User group name: MFA-Setup
Creation time: November 26, 2025, 20:21 (UTC)
ARN: arn:aws:iam::879381257906:group/MFA-Setup

Users (4) | Permissions | Access Advisor

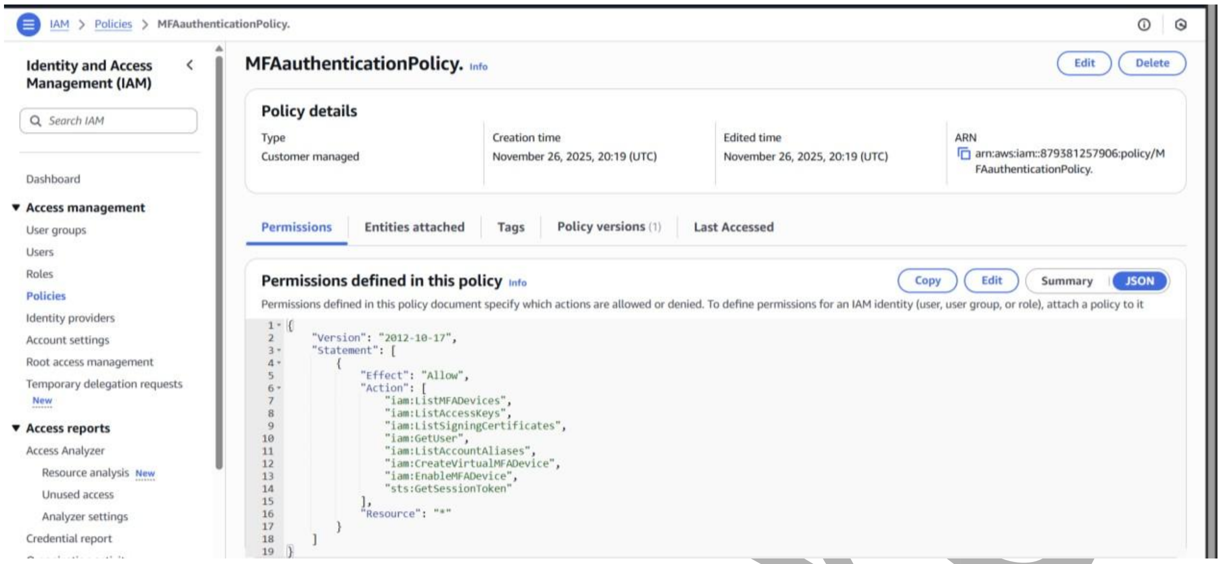
Users in this group (4)

Remove Add users

An IAM user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS.

User name	Groups
Admin-Adedayo	2
Auditor-Tobi	2
Dev-Timsuxwales	2
NetworkAdmin-Victoria	2

[Figure 2: MFA Group Configuration]



MFAauthenticationPolicy.

Policy details

Type	Creation time	Edited time	ARN
Customer managed	November 26, 2025, 20:19 (UTC)	November 26, 2025, 20:19 (UTC)	arn:aws:iam::879381257906:policy/MFAauthenticationPolicy.

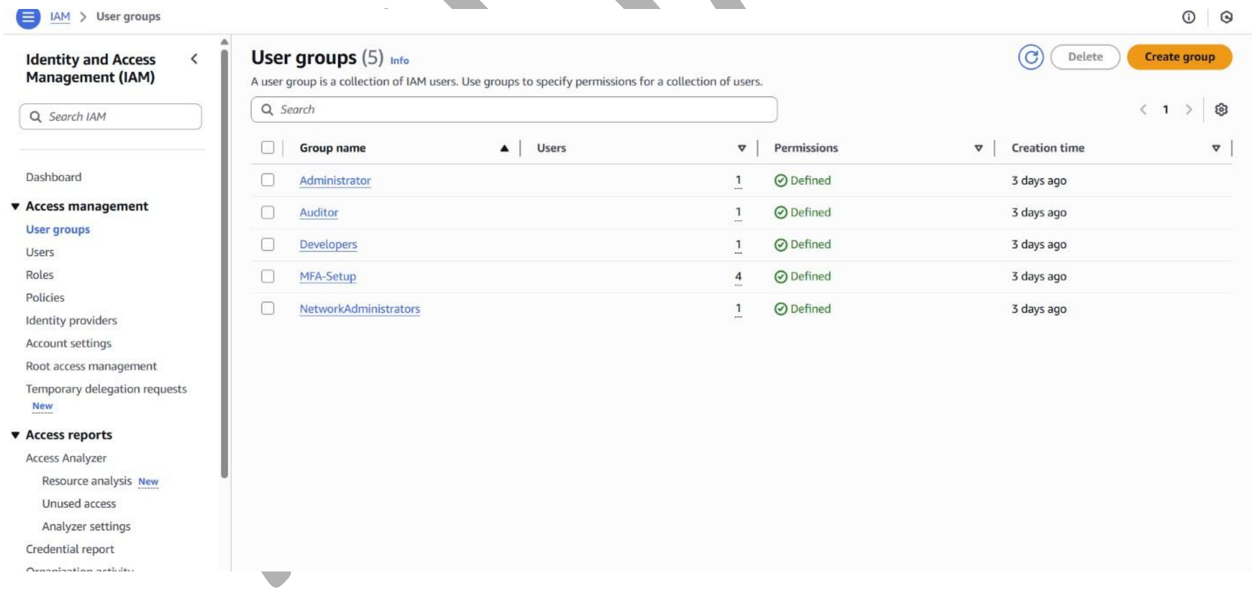
Permissions defined in this policy

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListMFADevices",
                "iam:ListAccessKeys",
                "iam:ListSigningCertificates",
                "iam:GetUser",
                "iam:ListAccountAliases",
                "iam:CreateVirtualMFADevice",
                "iam:EnableMFADevice",
                "sts:GetSessionToken"
            ],
            "Resource": "*"
        }
    ]
}
```

3.3 Role-Based Group Structure and Access Lifecycle

Permanent access groups were created for each role.



User groups (5)

A user group is a collection of IAM users. Use groups to specify permissions for a collection of users.

Group name	Users	Permissions	Creation time
Administrator	1	✓ Defined	3 days ago
Auditor	1	✓ Defined	3 days ago
Developers	1	✓ Defined	3 days ago
MFA-Setup	4	✓ Defined	3 days ago
NetworkAdministrators	1	✓ Defined	3 days ago

During initial setup and troubleshooting, temporary administrative permissions were granted to allow efficient configuration. Once the environment stabilized, I removed permanent administrative access and replaced it with controlled role assumption.

This reduced long-term exposure to privileged access.

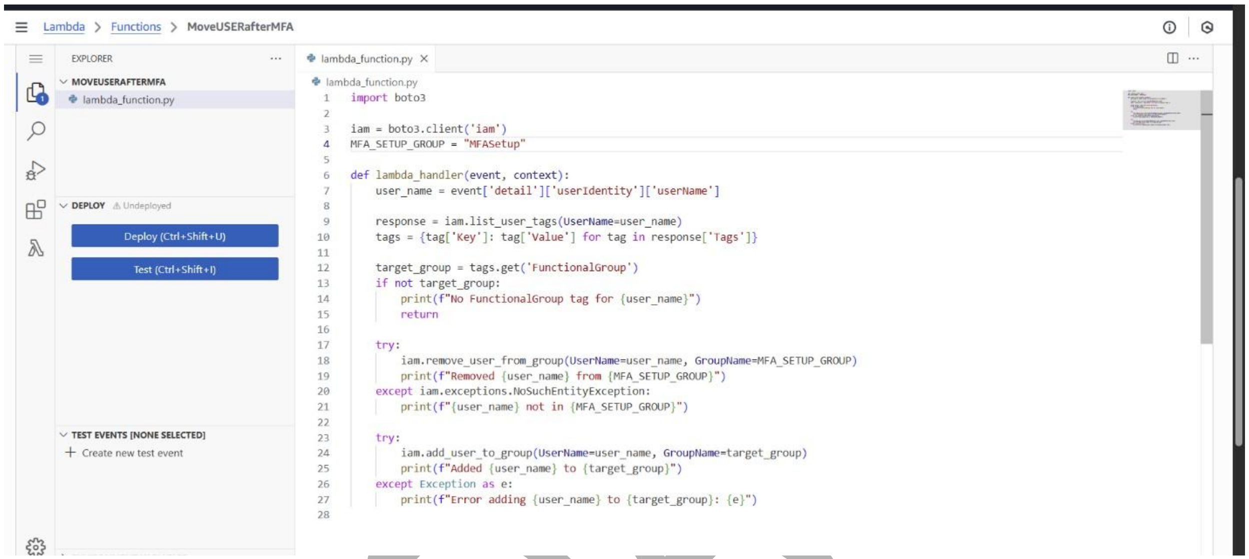
All access changes were documented to support audit and compliance reviews.

3. MFA Automation and User Lifecycle Management

3.1 Automation Architecture

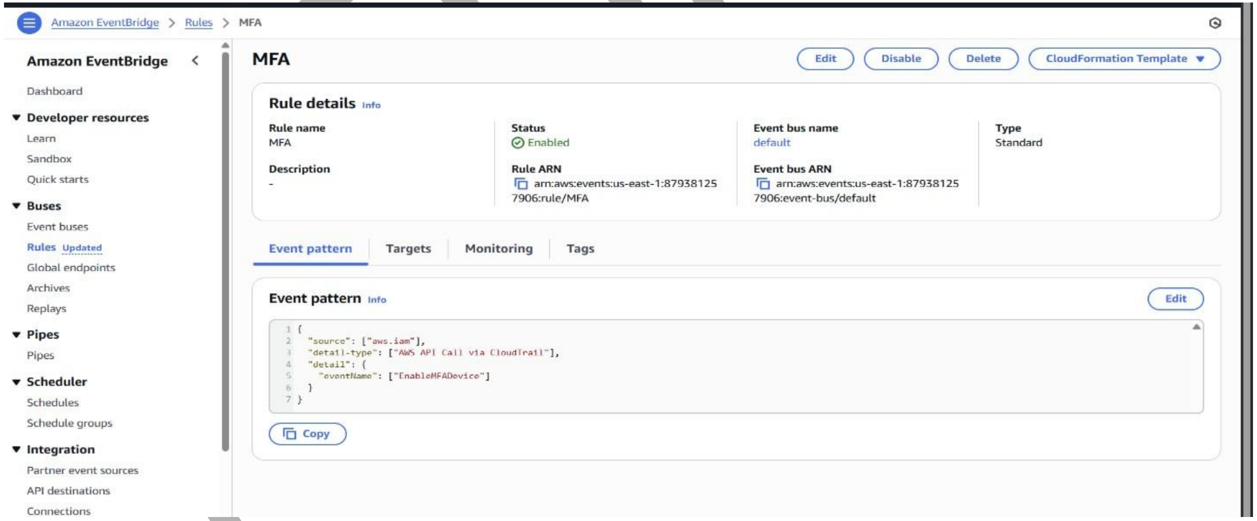
I designed and deployed an automated user lifecycle workflow using:

- AWS Lambda



```python
import boto3

iam = boto3.client('iam')
MFA_SETUP_GROUP = "MFASetup"

def lambda_handler(event, context):
    user_name = event['detail']['userIdentity']['userName']

    response = iam.list_user_tags(UserName=user_name)
    tags = {tag['Key']: tag['Value'] for tag in response['Tags']}

    target_group = tags.get('FunctionalGroup')
    if not target_group:
        print(f"No FunctionalGroup tag for {user_name}")
        return

    try:
        iam.remove_user_from_group(UserName=user_name, GroupName=MFA_SETUP_GROUP)
        print(f"Removed {user_name} from {MFA_SETUP_GROUP}")
    except iam.exceptions.NoSuchEntityException:
        print(f"{user_name} not in {MFA_SETUP_GROUP}")

    try:
        iam.add_user_to_group(UserName=user_name, GroupName=target_group)
        print(f"Added {user_name} to {target_group}")
    except Exception as e:
        print(f"Error adding {user_name} to {target_group}: {e}")
```

- Amazon Event Bridge.



Rule details Info

Rule name	Status	Event bus name	Type
MFA	Enabled	default	Standard

Description	Rule ARN	Event bus ARN
-	arn:aws:events:us-east-1:879381257906:rule/MFA	arn:aws:events:us-east-1:879381257906:event-bus/default

Event pattern Info

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

- IAM Roles and Policies
- User Tagging

A dedicated automation role and Lambda function managed user transitions between groups after MFA activation.

3.2 Tag-Based Access Assignment

I implemented standardized tagging for all IAM users.

Tags defined each user's operational role and department. The automation workflow relied on these tags to determine final group placement.

Tag compliance became a mandatory part of the user provisioning process.

4.3 Operational Challenges and Resolution

Several issues were identified and resolved during implementation.

Missing User Tags

Some users did not migrate after MFA activation due to missing metadata.

Resolution:
I enforced mandatory tagging standards and updated onboarding procedures.

Incorrect Group Reference

An incorrect group name in the Lambda function prevented proper removal from the onboarding group.

Resolution:
I corrected the group reference in the automation logic and revalidated the workflow.

Validation and Cleanup

After remediation, I re-tested user migrations using multiple roles and manually cleaned up legacy group assignments.

This ensured consistency across all accounts.

---

4. IAM Access Key Lifecycle Management

4.1 Credential Auditing

I regularly reviewed IAM Credential Reports to evaluate:

- Access key age

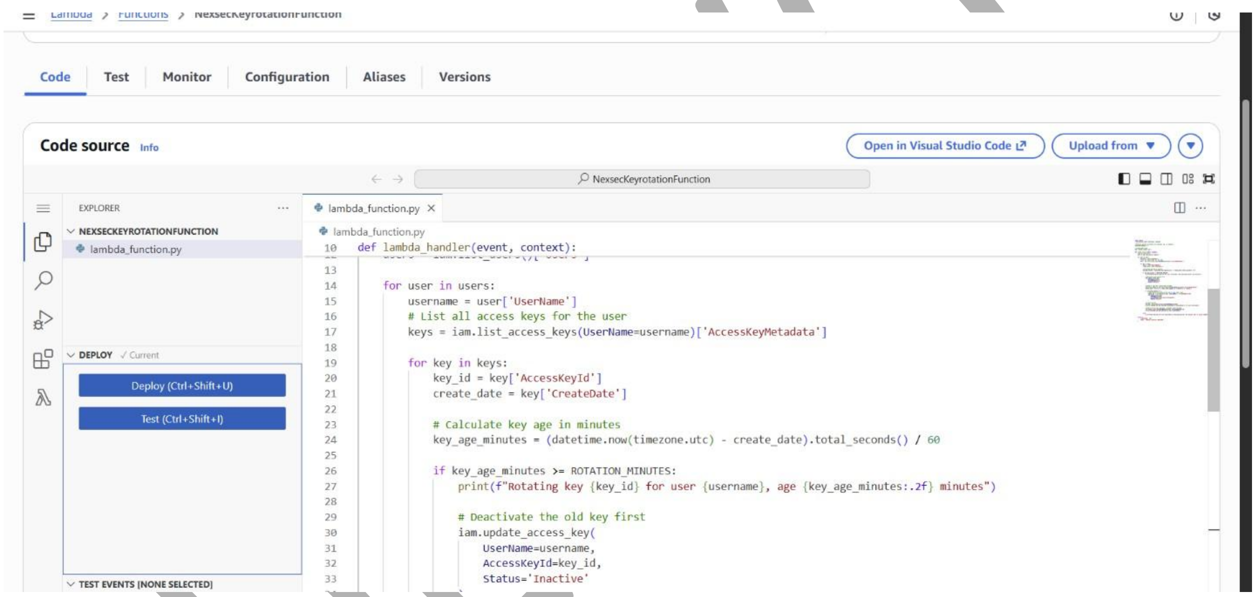
- MFA status
- Password compliance
- Credential usage

This supported proactive identification of weak or unused credentials.

4.2 Automated Key Rotation

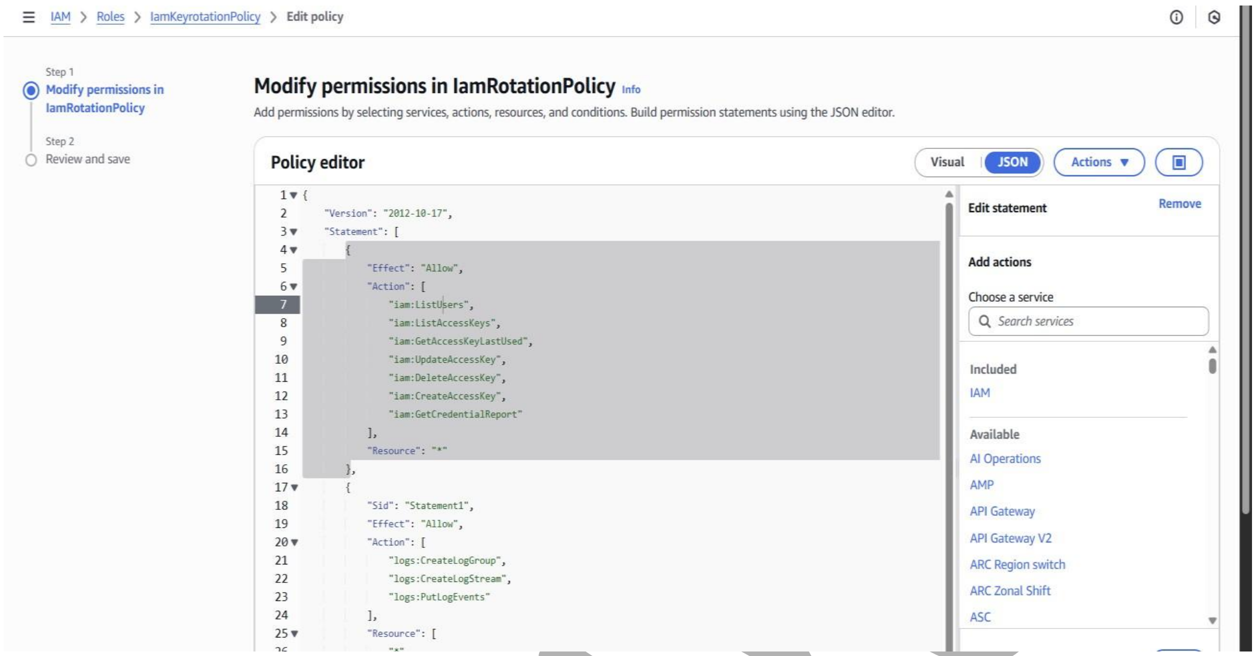
I implemented an automated access key rotation framework using:

- AWS Lambda



```python
def lambda_handler(event, context):

    for user in users:
        username = user['UserName']
        # List all access keys for the user
        keys = iam.list_access_keys(UserName=username)['AccessKeyMetadata']

        for key in keys:
            key_id = key['AccessKeyId']
            create_date = key['CreateDate']

            # Calculate key age in minutes
            key_age_minutes = (datetime.now(timezone.utc) - create_date).total_seconds() / 60

            if key_age_minutes >= ROTATION_MINUTES:
                print(f"Rotating key {key_id} for user {username}, age {key_age_minutes:.2f} minutes")

                # Deactivate the old key first
                iam.update_access_key(
                    UserName=username,
                    AccessKeyId=key_id,
                    Status='Inactive'
```

- IAM custom policies


IAM > Roles > IamKeyrotationPolicy > Edit policy
Step 1
Modify permissions in IamRotationPolicy
Step 2
Review and save

Modify permissions in IamRotationPolicy

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListUsers",
                "iam:ListAccessKeys",
                "iam:GetAccessKeyLastUsed",
                "iam:UpdateAccessKey",
                "iam:DeleteAccessKey",
                "iam:CreateAccessKey",
                "iam:GetCredentialReport"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
```

- Amazon EventBridge scheduling

Rotation logic was validated using short execution intervals before being deployed with standard production schedules.

This reduced the risk associated with long-lived credentials.

---

5. Role-Based Access Control (RBAC)

5.1 Administrative Access Model

I replaced permanent administrative privileges with controlled role assumption using AWS Security Token Service (STS).

A dedicated administrative role was created for elevated access.

Users received temporary privileges only when required.

5.2 Trust Policy Models

Two trust models were implemented:

- Direct user-based trust policies
- Group-based trust policies

This supported both centralized and distributed administrative workflows.

6.3 Least Privilege Enforcement and Access Analysis I

created custom roles for:

- Development
- Audit
- Network Operations

Each role followed least-privilege principles and enforced MFA requirements.

I enabled IAM Access Analyzer to continuously evaluate policies for:

- Excessive permissions
- Wildcard usage
- Unused access
- Public exposure

Analyzer findings were reviewed regularly and remediated when necessary.

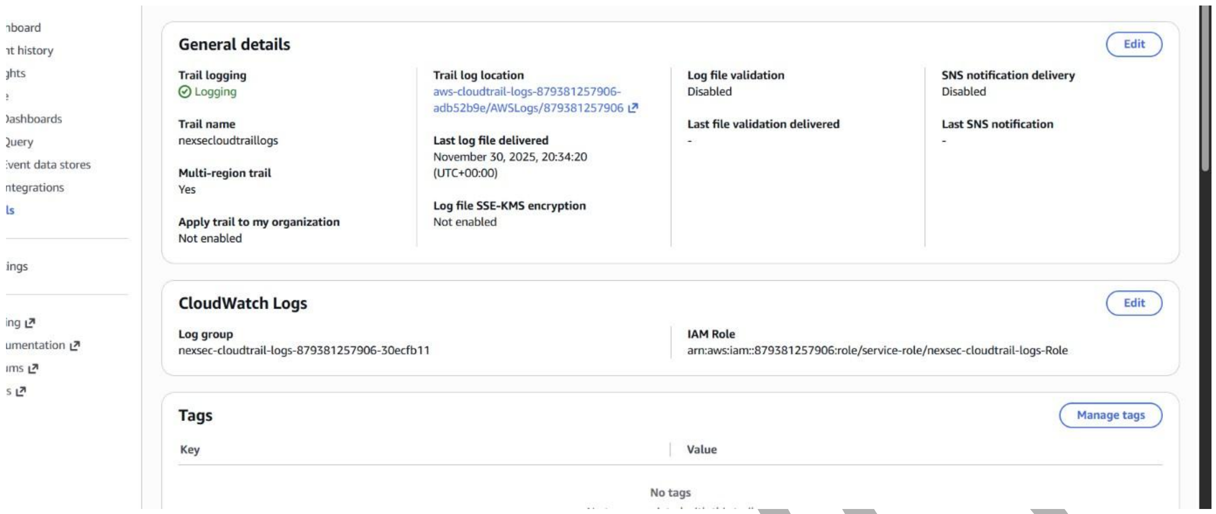
6. Logging, Monitoring, and Operational Visibility

6.1 CloudTrail Implementation

I configured a centralized CloudTrail trail to capture all API activity.

CloudTrail Insights and Data Events were enabled to detect abnormal behavior, privilege escalation attempts, and automation-related changes.

This provided a complete audit trail.



nboard
nt history
ghts
Dashboards
Query
Event data stores
ntegrations
ls
ings
ing
umentation
ıms
s

General details

Trail logging
Logging

Trail name
nexsecloudtraillogs

Multi-region trail
Yes

Apply trail to my organization
Not enabled

Trail log location
aws-cloudtrail-logs-879381257906-adb52b9e/AWSLogs/879381257906

Last log file delivered
November 30, 2025, 20:34:20 (UTC+00:00)

Log file SSE-KMS encryption
Not enabled

Log file validation
Disabled

Last file validation delivered
-

SNS notification delivery
Disabled

Last SNS notification
-

CloudWatch Logs

Log group
nexsec-cloudtrail-logs-879381257906-30ecfb11

IAM Role
arn:aws:iam::879381257906:role/service-role/nexsec-cloudtrail-logs-Role

Tags

Key	Value

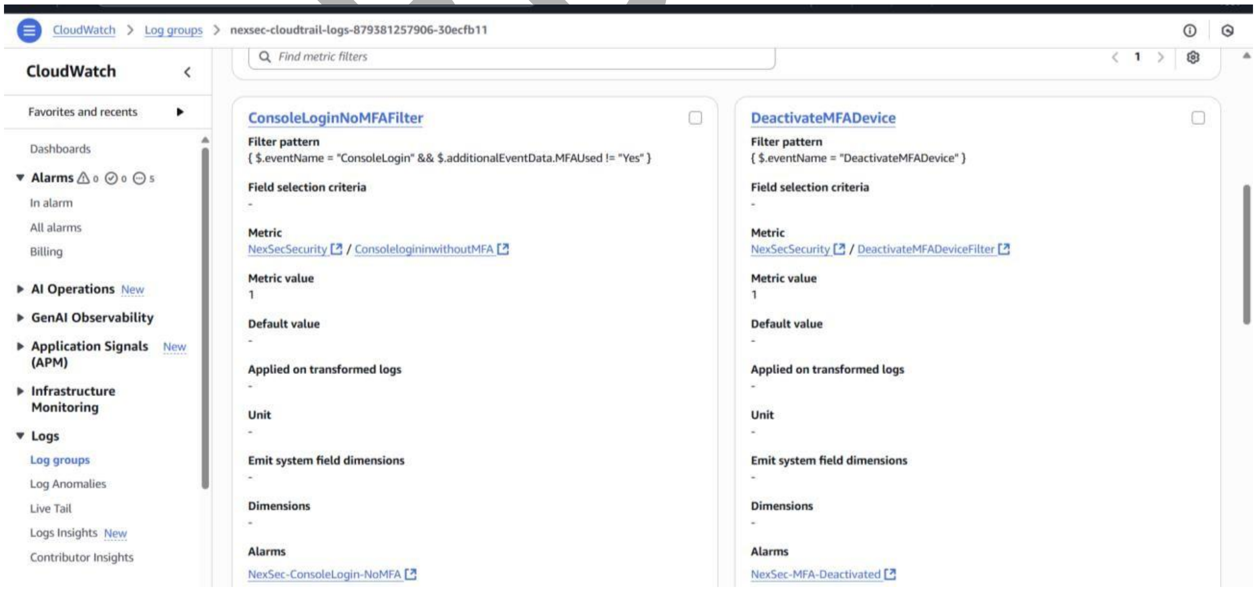
No tags

7.2 CloudWatch Integration and Metric Validation CloudTrail logs

were streamed into CloudWatch Log Groups.

I created metric filters for critical IAM actions, including:

- Unauthorized console access
- Access key creation and modification
- MFA deactivation



ConsoleLoginNoMFAFilter

Filter pattern
{ $.eventName = "ConsoleLogin" && $.additionalEventData.MFAUsed != "Yes" }

Field selection criteria
-

Metric
NexSecSecurity / ConsoleloginwithoutMFA

Metric value
1

Default value
-

Applied on transformed logs
-

Unit
-

Emit system field dimensions
-

Dimensions
-

Alarms
NexSec-ConsoleLogin-NoMFA

DeactivateMFADevice

Filter pattern
{ $.eventName = "DeactivateMFADevice" }

Field selection criteria
-

Metric
NexSecSecurity / DeactivateMFADeviceFilter

Metric value
1

Default value
-

Applied on transformed logs
-

Unit
-

Emit system field dimensions
-

Dimensions
-

Alarms
NexSec-MFA-Deactivated

- Role assumption

Initially, metrics were not visible because no matching events had occurred.

To resolve this, I manually generated test IAM actions to trigger CloudTrail logs. Once events were recorded, metrics became available and alarms could be created.

This validation ensured monitoring worked as intended before production use.

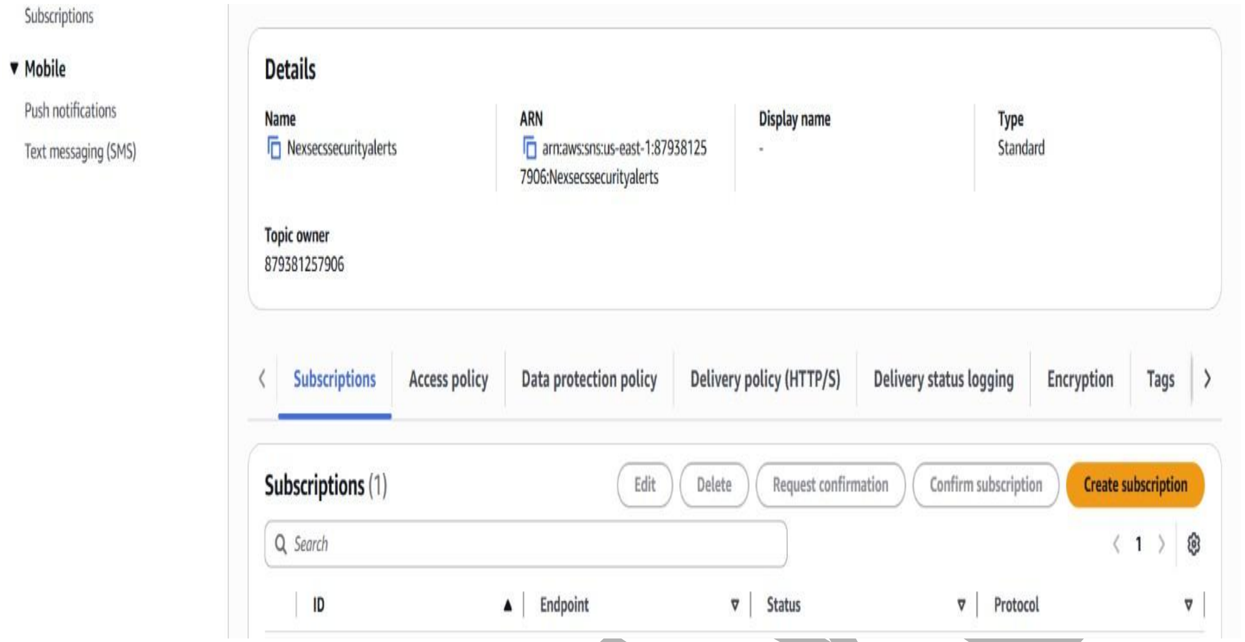
---

7. Alerting and Encryption Management

7.1 Alert Configuration

I configured CloudWatch alarms for each high-risk IAM event.

SNS was used to distribute security alerts to the operations team.



Details

Name: Nexsecssecurityalerts

ARN: arn:aws:sns:us-east-1:879381257906:Nexsecssecurityalerts

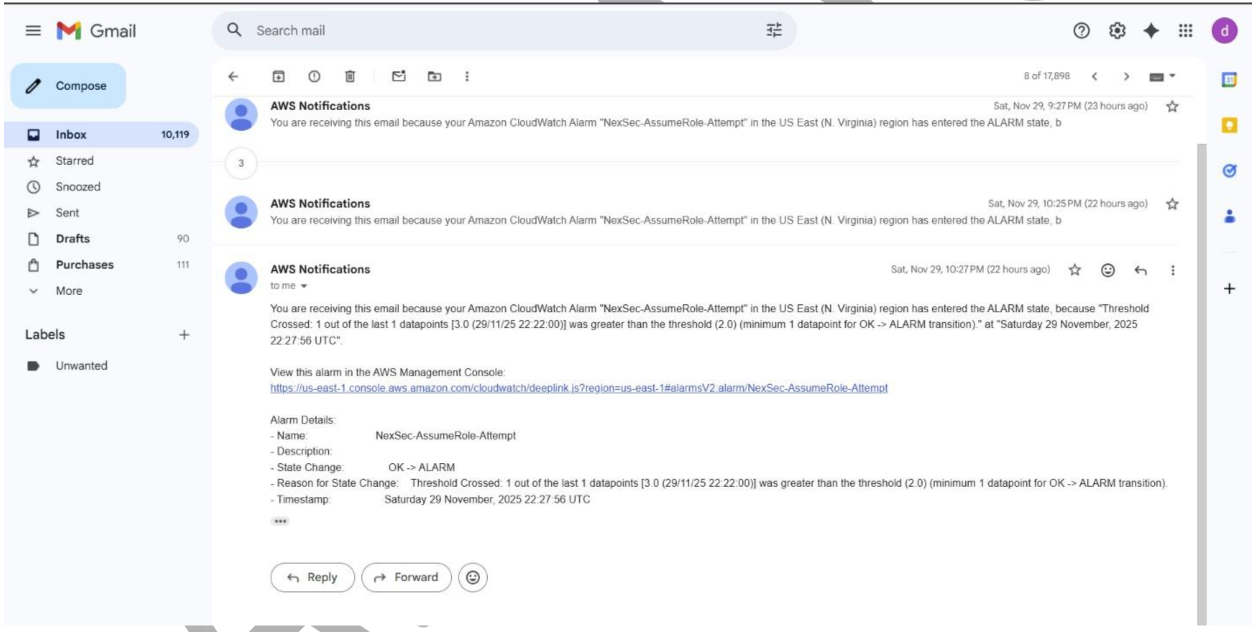
Display name: -

Type: Standard

Topic owner: 879381257906

Subscriptions (1)

ID	Endpoint	Status	Protocol



AWS Notifications — Sat, Nov 29, 9:27 PM (23 hours ago)
You are receiving this email because your Amazon CloudWatch Alarm "NexSec-AssumeRole-Attempt" in the US East (N. Virginia) region has entered the ALARM state, b

AWS Notifications — Sat, Nov 29, 10:25 PM (22 hours ago)
You are receiving this email because your Amazon CloudWatch Alarm "NexSec-AssumeRole-Attempt" in the US East (N. Virginia) region has entered the ALARM state, b

AWS Notifications — Sat, Nov 29, 10:27 PM (22 hours ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "NexSec-AssumeRole-Attempt" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [3.0 (29/11/25 22:22:00)] was greater than the threshold (2.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Saturday 29 November, 2025 22:27:56 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/NexSec-AssumeRole-Attempt

```
Alarm Details:
- Name:                      NexSec-AssumeRole-Attempt
- Description:
- State Change:              OK -> ALARM
- Reason for State Change:   Threshold Crossed: 1 out of the last 1 datapoints [3.0 (29/11/25 22:22:00)] was greater than the threshold (2.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp:                 Saturday 29 November, 2025 22:27:56 UTC
```

7.2 KMS Encryption Resolution

During testing, alerts were triggered but not delivered.

Root cause analysis showed that encrypted SNS topics lacked proper KMS permissions for CloudWatch.

I updated the KMS key policy to grant publishing rights.

After remediation, alert delivery was restored and verified.

8. Advanced Security and Compliance Controls

8.1 Threat Detection

I enabled AWS GuardDuty to provide continuous monitoring for:

- Credential compromise
- Suspicious API activity
- Network anomalies

Findings were integrated into incident response workflows.

9.2 Configuration and Compliance Monitoring I

implemented AWS Config to track:

- Root account MFA status
- IAM password policies
- Public access settings
- Unused roles
- Over-permissioned policies

Configuration changes were logged and reviewed for compliance.

9. Documentation, Governance, and Audit Support

All IAM roles, policies, workflows, and architectural decisions were documented.

This documentation supported:

- Internal reviews
- Security audits
- Incident investigations
- Knowledge transfer

Audit logs, credential reports, and access reviews were retained according to governance requirements.

10. Outcomes and Impact

This implementation delivered measurable improvements:

- Established enterprise-grade IAM governance
- Enforced strong authentication standards
- Reduced credential-related risk
- Improved visibility into identity activity
- Centralized monitoring and alerting
- Enhanced audit readiness
- Strengthened operational resilience

---

11. Conclusion

I designed and implemented a secure, scalable, and auditable IAM framework that supports enterprise security and compliance requirements.

Through automation, least-privilege enforcement, continuous monitoring, and strong governance, this environment enables secure operations and long-term growth.

This implementation reflects hands-on experience in building and operating cloud identity systems in production-like enterprise environments.