

# Enterprise Identity and Access Management (IAM) Implementation on AWS

**Author:** Adedayo

**Specialization:** Cloud Security & Identity Management

**Platform:** Amazon Web Services (AWS)

---

## 1. Introduction

This document describes the design, implementation, and ongoing management of an enterprise-grade Identity and Access Management (IAM) framework in an AWS environment.

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.

---

## 2. 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
- 

## 3. User and Access Management

### 3.1 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.

The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, there's a navigation sidebar with options like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management' (with 'Users' selected), 'Roles', 'Policies', 'Identity providers', 'Account settings', 'Root access management', and 'Temporary delegation requests'. Below that is 'Access reports' with 'Access Analyzer' and 'Unused access' sections. At the top right, there are buttons for 'Dismiss' and 'Manage workforce users'. The main area is titled 'Users (4)' and contains a table with columns: User name, Path, Group, Last activity, MFA, Password age, Console last sign-in, and Account status. The users listed are Admin-Adedayo, Auditor-Tobi, Dev-Timsuxwales, and NetworkAdmin-Victoria. Each user has a checkbox next to their name and a 'Details' button.

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]

### 3.2 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.

**MFA-Setup**

**Summary**

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)**

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

**Permissions** | Entities attached | Tags | Policy versions (1) | Last Accessed

**Permissions defined in this policy**

```

1+ [
2   "Version": "2012-10-17",
3+   "Statement": [
4+     {
5       "Effect": "Allow",
6       "Action": [
7         "iam:ListMFADevices",
8         "iam:ListAccessKeys",
9         "iam:ListSigningCertificates",
10        "iam:GetUser",
11        "iam:ListAccountAliases",
12        "iam:CreateVirtualMFADevice",
13        "iam:EnableMFADevice",
14        "sts:GetSessionToken"
15      ],
16      "Resource": "*"
17    }
18  ]
19 ]
  
```

### 3.3 Role-Based Group Structure and Access Lifecycle

Permanent access groups were created for each role.

The screenshot shows the AWS IAM User groups page. On the left, there's a navigation sidebar with 'Identity and Access Management (IAM)' selected. Under 'Access management', 'User groups' is also selected. The main area displays a table titled 'User groups (5)'. The table has columns for 'Group name', 'Users', 'Permissions', and 'Creation time'. The data is as follows:

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.

## 4. MFA Automation and User Lifecycle Management

### 4.1 Automation Architecture

I designed and deployed an automated user lifecycle workflow using:

- AWS Lambda

```

lambda_function.py

1 import boto3
2
3 iam = boto3.client('iam')
4 MFA_SETUP_GROUP = "MFASetup"
5
6 def lambda_handler(event, context):
7     user_name = event['detail']['userIdentity']['userName']
8
9     response = iam.list_user_tags(UserName=user_name)
10    tags = {tag['Key']: tag['Value'] for tag in response['Tags']}
11
12    target_group = tags.get('FunctionalGroup')
13    if not target_group:
14        print(f"No FunctionalGroup tag for {user_name}")
15        return
16
17    try:
18        iam.remove_user_from_group(UserName=user_name, GroupName=MFA_SETUP_GROUP)
19        print(f"Removed {user_name} from {MFA_SETUP_GROUP}")
20    except iam.exceptions.NoSuchEntityException:
21        print(f"{user_name} not in {MFA_SETUP_GROUP}")
22
23    try:
24        iam.add_user_to_group(UserName=user_name, GroupName=target_group)
25        print(f"Added {user_name} to {target_group}")
26    except Exception as e:
27        print(f"Error adding {user_name} to {target_group}: {e}")

```

- Amazon Event Bridge.

**MFA**

**Rule details**

Rule name MFA	Status Enabled	Event bus name default	Type Standard
Description	Rule ARN arn:aws:events:us-east-1:87938125 7906:rule/MFA	Event bus ARN arn:aws:events:us-east-1:87938125 7906:event-bus/default	

**Event pattern**

```

1 {
2     "source": ["aws.iam"],
3     "detail-type": ["AWS API Call via CloudWatch Events"],
4     "detail": {
5         "eventName": ["EnableMFADevice"]
6     }
7 }

```

- IAM Roles and Policies
- User Tagging

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

## 4.2 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.

---

### **5. IAM Access Key Lifecycle Management**

#### **5.1 Credential Auditing**

I regularly reviewed IAM Credential Reports to evaluate:

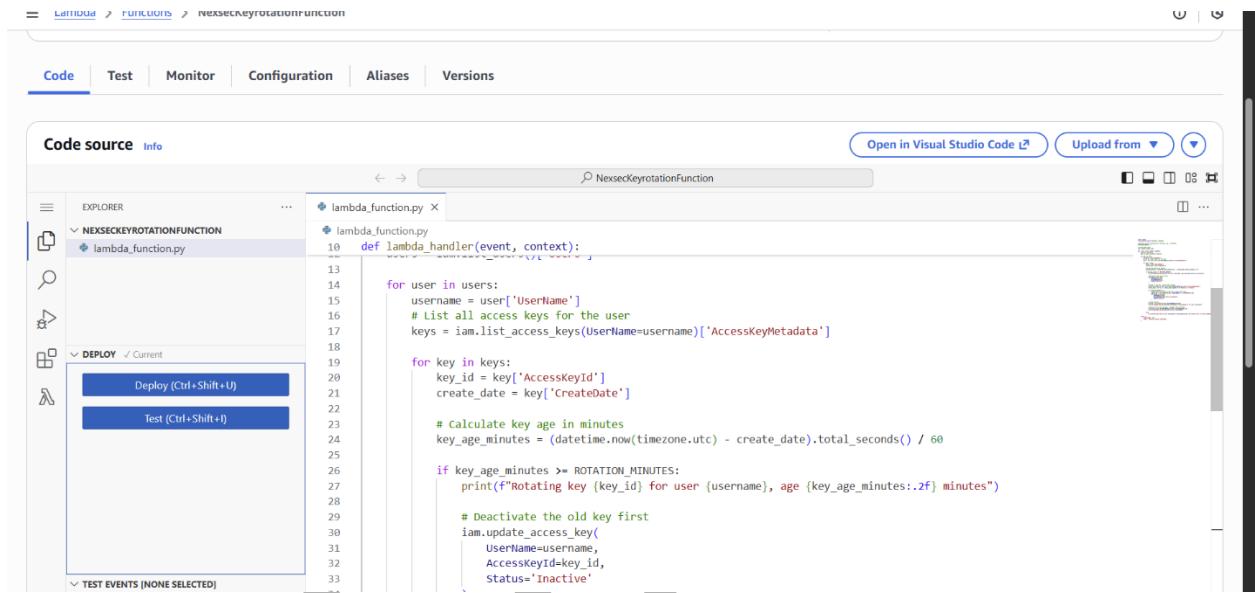
- Access key age
- MFA status
- Password compliance
- Credential usage

This supported proactive identification of weak or unused credentials.

## 5.2 Automated Key Rotation

I implemented an automated access key rotation framework using:

- AWS Lambda



```
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']

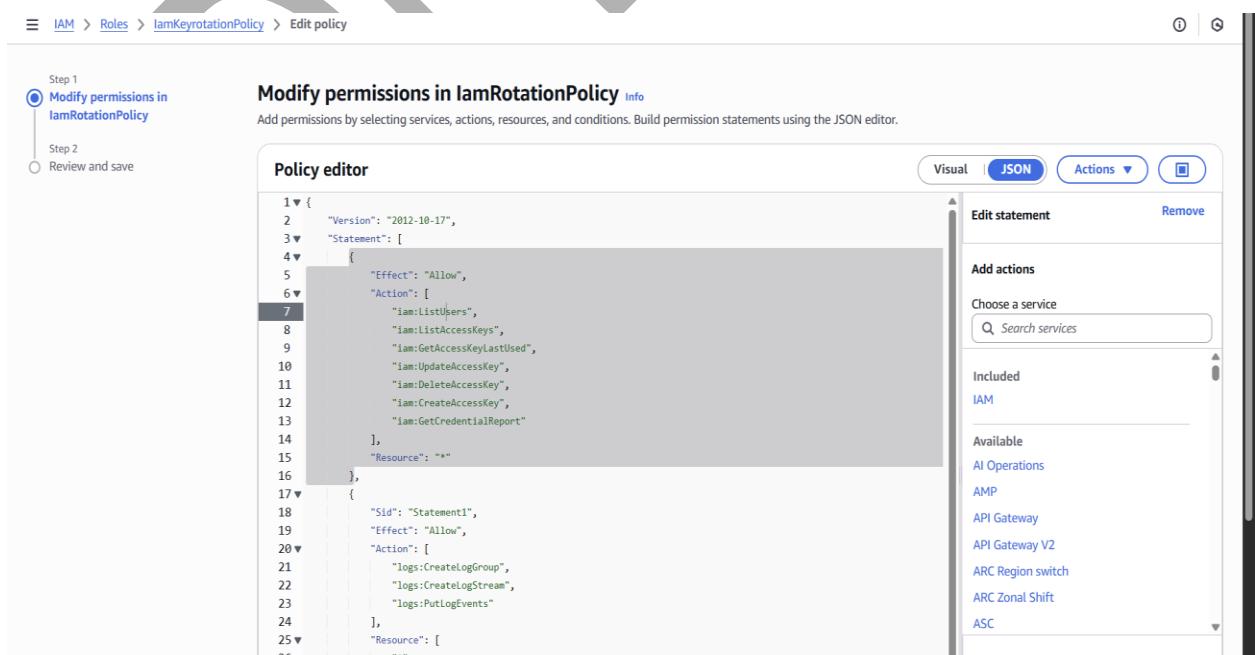
        for key in keys:
            key_id = key['AccessKeyId']
            create_date = key['CreateDate']

            # 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



Step 1  
 Modify permissions in IamRotationPolicy  
 Step 2  
 Review and save

Modify permissions in IamRotationPolicy [Info](#)  
Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```
1 ▼ {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "iam:ListUsers",
8                 "iam:ListAccessKeys",
9                 "iam:GetAccessKeyLastUsed",
10                "iam:UpdateAccessKey",
11                "iam:DeleteAccessKey",
12                "iam>CreateAccessKey",
13                "iam:GetCredentialReport"
14            ],
15            "Resource": "*"
16        },
17        {
18            "Sid": "Statement1",
19            "Effect": "Allow",
20            "Action": [
21                "logs:CreateLogGroup",
22                "logs:CreateLogStream",
23                "logs:PutLogEvents"
24            ],
25            "Resource": [
26                "*"
27            ]
28        }
29    ]
30}
```

Visual [JSON](#) [Actions](#) [Edit statement](#) [Remove](#)  
Add actions  
Choose a service   
Included IAM  
Available AI Operations AMP API Gateway API Gateway V2 ARC Region switch ARC Zonal Shift ASC

- 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.

---

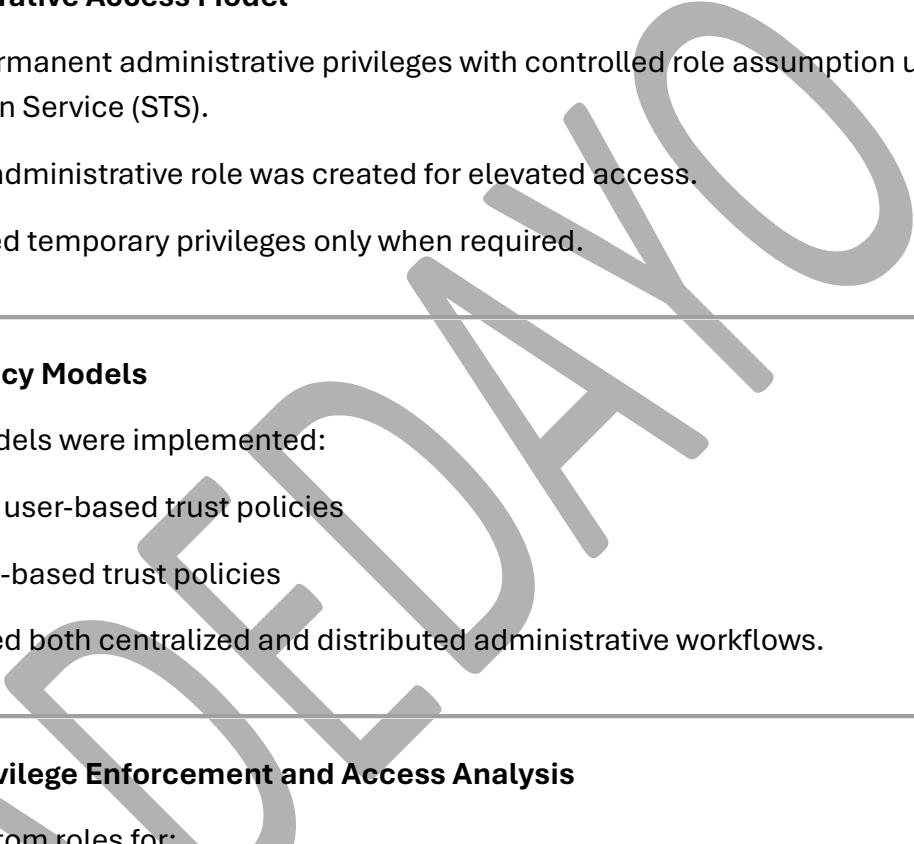
## 6. Role-Based Access Control (RBAC)

### 6.1 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.



---

### 6.2 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.

---

## 7. Logging, Monitoring, and Operational Visibility

### 7.1 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.

The screenshot shows the AWS CloudTrail Trail configuration page. On the left, there's a sidebar with navigation links like Dashboard, History, Metrics, Dashboards, Query, Event Data Stores, Integrations, and Help. The main content area has three tabs: General details, CloudWatch Logs, and Tags. The General details tab is active, showing information such as Trail logging (Enabled), Trail name (nexsec-cloudtrail-logs), Multi-region trail (Yes), Log file SSE-KMS encryption (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 validation (Disabled), Last file validation delivered (None), and SNS notification delivery (Disabled). The CloudWatch Logs tab shows a Log group (nexsec-cloudtrail-logs-879381257906-30ecfb11) and an IAM Role (arn:aws:iam:879381257906:role/service-role/nexsec-cloudtrail-logs-Role). The Tags tab shows a table with columns 'Key' and 'Value', currently empty. There are 'Edit' and 'Manage tags' buttons at the top of each tab.

### 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

The screenshot shows the AWS CloudWatch Metrics interface. On the left, there's a sidebar with navigation links like CloudWatch, Dashboards, Alarms, AI Operations, GenAI Observability, Application Signals (APM), Infrastructure Monitoring, and Logs. Under Logs, Log groups is selected. The main area displays two metric filters:

- 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.

---

## 8. Alerting and Encryption Management

### 8.1 Alert Configuration

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

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

The screenshot displays two main sections: the AWS SNS Subscriptions interface and the Gmail inbox.

**AWS SNS Subscriptions:**

- Details:** Shows a subscription for "Nexsecsecurityalerts" with ARN arn:aws:sns:us-east-1:879381257906:Nexsecsecurityalerts, Type Standard, and Topic owner 879381257906.
- Subscriptions (1):** A table with one row, showing the subscription details.
- Actions:** Buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription.

**Gmail inbox:**

- Compose:** Compose button.
- Inbox:** 10,119 messages.
- Messages:**
  - AWS Notifications:** 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. Sat, Nov 29, 9:27 PM (23 hours ago)
  - AWS Notifications:** 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. Sat, Nov 29, 10:25 PM (22 hours ago)
  - AWS Notifications:** 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". Sat, Nov 29, 10:27 PM (22 hours ago)

## 8.2 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.

---

## 9. Advanced Security and Compliance Controls

### 9.1 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.

---

## 10. 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.

---

## 11. 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
- 

## 12. 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.