DevOps

[IAM (Identity and Access Management) 2](#_Toc193568808)

[IAM Overview 2](#_Toc193568809)

[IAM Groups 3](#_Toc193568810)

[IAM Roles 6](#_Toc193568811)

[Identity Provider / Federation / Identity Center Overview 8](#_Toc193568812)

[SAML 10](#_Toc193568813)

[Identity Center 12](#_Toc193568814)

[IAM Tags 14](#_Toc193568815)

[Cross-Account Access in AWS Using IAM Roles 17](#_Toc193568816)

[IAM GUI Checklist 19](#_Toc193568817)

# IAM (Identity and Access Management)

## IAM Overview

**AWS IAM & Security Notes**

**1. IAM (Identity and Access Management)**

* IAM is **Global** and **Free**.
* Used for **Security** purposes.
* Controls access to **AWS resources** centrally by defining **permissions**.
* **Root User** has **Full Permissions**.
* **IAM Users** have **Limited Permissions** (based on assigned **Policies**).

**Best Practices for IAM Users:**

* **Do not use the Root account** for daily tasks.
* Use **IAM Users** for performing operations.
* **Enable MFA (Multi-Factor Authentication)** for extra security.
* Do not share your **email/password** with others.
* **Root Account should create IAM users** and provide necessary permissions.

**2. Authentication Methods**

**Ways to Access AWS:**

1. **Console Access** (AWS GUI)
   * Login using **Email/Password or Username/Password**.
   * Supports **MFA for extra security**.
2. **Programmatic Access**
   * Used for **CLI, SDKs, and Developer Tools**.
   * Requires **Access Key and Secret Key**.
   * Keys are **user-specific** and should not be shared.
   * Configure AWS CLI using aws configure.

**3. Security & Access Keys**

* **Access & Secret Keys** are generated only **once** during creation.
* Lost keys **cannot be retrieved**, but new ones can be generated.
* IAM users can have a **maximum of 2 sets of keys**.
* **Do not create or use Root account keys**.
* Best practice is to **rotate passwords and keys periodically**.

**4. IAM User Assignments & Roles**

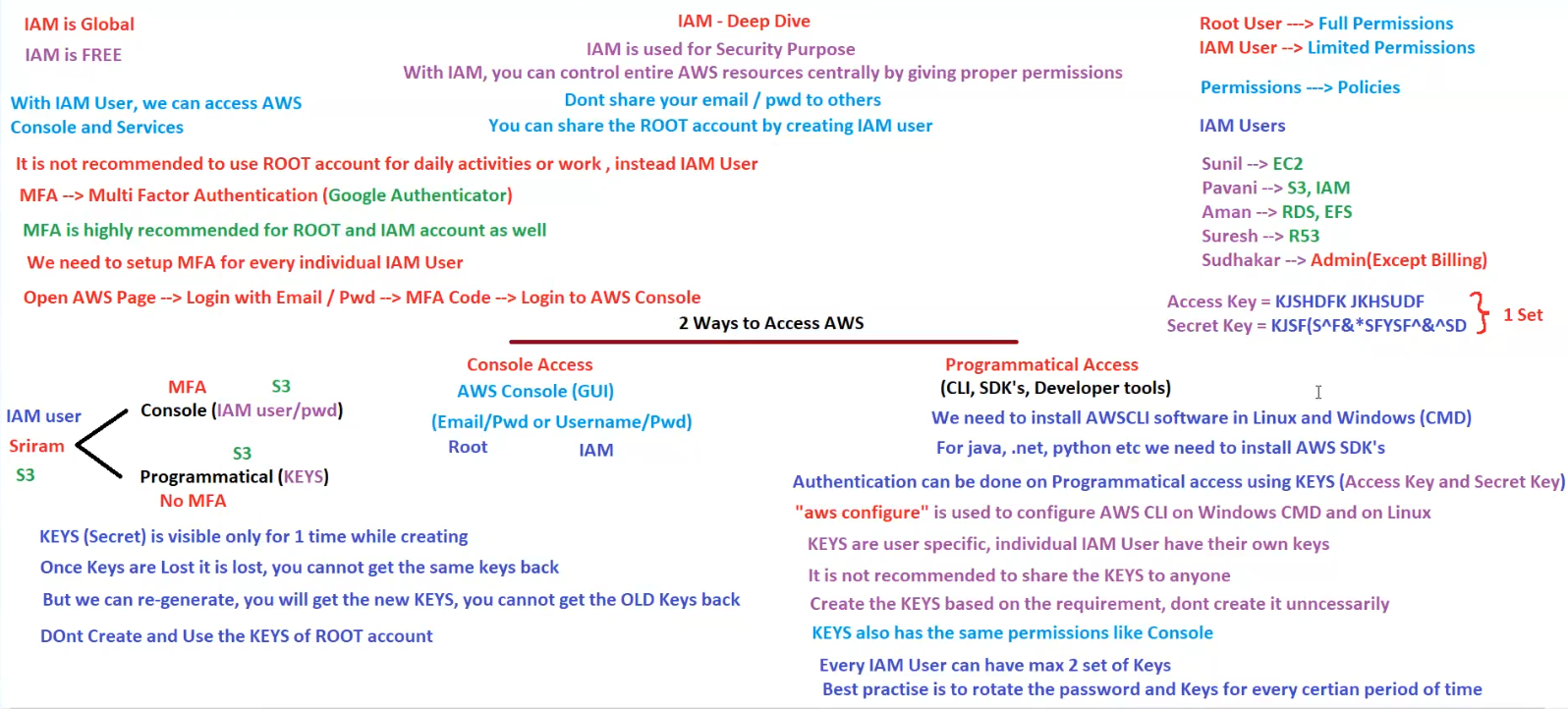
* **Sunil** → EC2
* **Pavani** → S3, IAM
* **Aman** → RDS, EFS
* **Suresh** → R53
* **Sudhakar** → Admin (Except Billing)

**5. Security Measures**

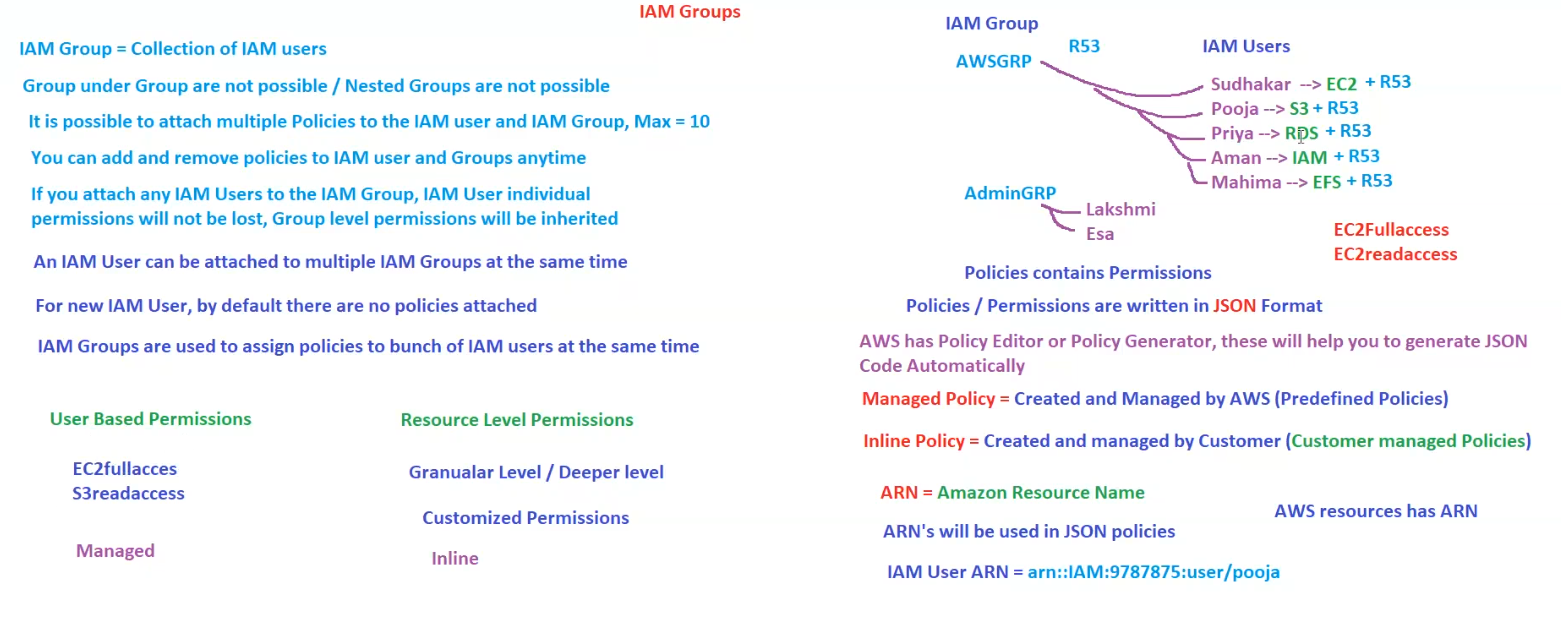
* **IAM Policies** → Define permissions for users.
* **AWS MFA** → Recommended for all IAM users.
* **AWS Config** → Monitors changes in AWS resources.
* **AWS CloudTrail** → Tracks, audits, and records account activities.

**6. General Best Practices**

* **Never share credentials** (Access Key, Secret Key, Passwords).
* Use **IAM roles** instead of Access Keys when possible.
* **Limit permissions** based on the **principle of least privilege**.
* **Rotate security credentials periodically** for better security.



## IAM Groups



**AWS IAM Groups and Policies**

**1. Introduction to IAM Groups**

* **IAM Group** = A collection of IAM users.
* **Nested Groups are not allowed** – A group cannot contain another group.
* **Purpose**: Used to assign policies to multiple IAM users at the same time.
* **IAM users inherit permissions from IAM groups**, but their individual permissions remain unchanged.
* **A single IAM User can be part of multiple IAM Groups** at the same time.
* **New IAM users have no policies attached by default.**

**2. IAM Policies and Permissions**

* **IAM Policies define permissions** for users, groups, or roles.
* **Policies are written in JSON format** and specify the allowed or denied actions on AWS resources.
* **AWS provides tools like**:
  + **Policy Editor**
  + **Policy Generator**  
    These tools help generate JSON policies automatically.

**Types of IAM Policies**

1. **Managed Policies**
   * Created and managed by AWS.
   * Predefined policies (e.g., AmazonS3ReadOnlyAccess, EC2FullAccess).
2. **Inline Policies**
   * Created and managed by customers.
   * Customized for specific users, groups, or roles.

**3. IAM Groups and Users Example**

**IAM Groups**

* **AWSGRP** (Has permission for Route 53 - R53)
* **AdminGRP** (Administrators, contains users with high-level access)

**IAM Users and Their Permissions**

| **IAM User** | **Permissions** |
| --- | --- |
| **Sudhakar** | EC2 + Route 53 |
| **Pooja** | S3 + Route 53 |
| **Priya** | RDS + Route 53 |
| **Aman** | IAM + Route 53 |
| **Mahima** | EFS + Route 53 |
| **Lakshmi** | (Admin Group) |
| **Esa** | (Admin Group) |

**4. User-Based vs. Resource-Level Permissions**

**User-Based Permissions**

* Applied to individual users.
* Examples:
  + **EC2FullAccess**
  + **S3ReadAccess**
* Typically **Managed Policies**.

**Resource-Level Permissions**

* Applied at a **granular level** (specific resources).
* Examples:
  + **Custom permissions for EC2 instances, S3 buckets, etc.**
* Typically **Inline Policies**.

**5. Amazon Resource Name (ARN)**

* **ARN (Amazon Resource Name)** uniquely identifies AWS resources.
* **ARNs are used in JSON policies** to specify resources.
* **Example ARN for an IAM user**:
* arn::IAM:9787875:user/pooja

**6. Key Takeaways**

✅ **IAM Groups simplify permissions management** by allowing policies to be assigned to multiple users.  
✅ **Users can belong to multiple IAM Groups** and inherit permissions from them.  
✅ **Managed Policies are AWS-defined, whereas Inline Policies are customer-defined.**  
✅ **Policies are written in JSON and use ARNs to specify resources.**

## IAM Roles

**AWS IAM Roles and EC2 Access Management**

**1. Introduction to IAM Roles**

* IAM Roles provide **temporary access** without requiring long-term credentials (keys).
* If IAM Roles are used, **no need to configure access keys** on machines.
* Based on the permissions attached to the IAM Role, the instance gains access to specific AWS services.
* **IAM Roles can be attached to any AWS service**, not just EC2.

**2. IAM Role Attachments and Restrictions**

* **One EC2 instance can have only one IAM Role attached** at a time.
* **One IAM Role can be attached to multiple EC2 instances.**
* AWS services **cannot communicate with each other by default**—IAM Roles must be used to allow such communication.

**3. IAM User vs IAM Role**

| **Feature** | **IAM User** | **IAM Role** |
| --- | --- | --- |
| **Purpose** | Used for accessing AWS Console and services. | Used for granting temporary access to AWS services. |
| **Access Keys** | Required for authentication. | Not required; provides temporary credentials. |
| **Login to EC2** | Can access AWS console but **cannot login to EC2**. | Used for service-to-service communication. |

**4. EC2 Instance Access and Security Considerations**

**Scenario 1: Accessing AWS Services (e.g., S3) from an EC2 Instance**

* If you **do not configure access keys** on an EC2 instance, you **cannot access AWS services like S3**.
* If you **configure access keys on the EC2 instance**, the keys are stored **locally** on the instance, which is:
  + **Not safe**
  + **Not secure**
  + A security risk if compromised.

**Solution: Use IAM Roles**

* Instead of manually adding access keys, attach an **IAM Role** to the EC2 instance.
* The **IAM Role grants permissions dynamically**, reducing security risks.
* Example:
  + **Trusted Entity** = EC2
  + **Permissions** = S3, RDS (or any required service)

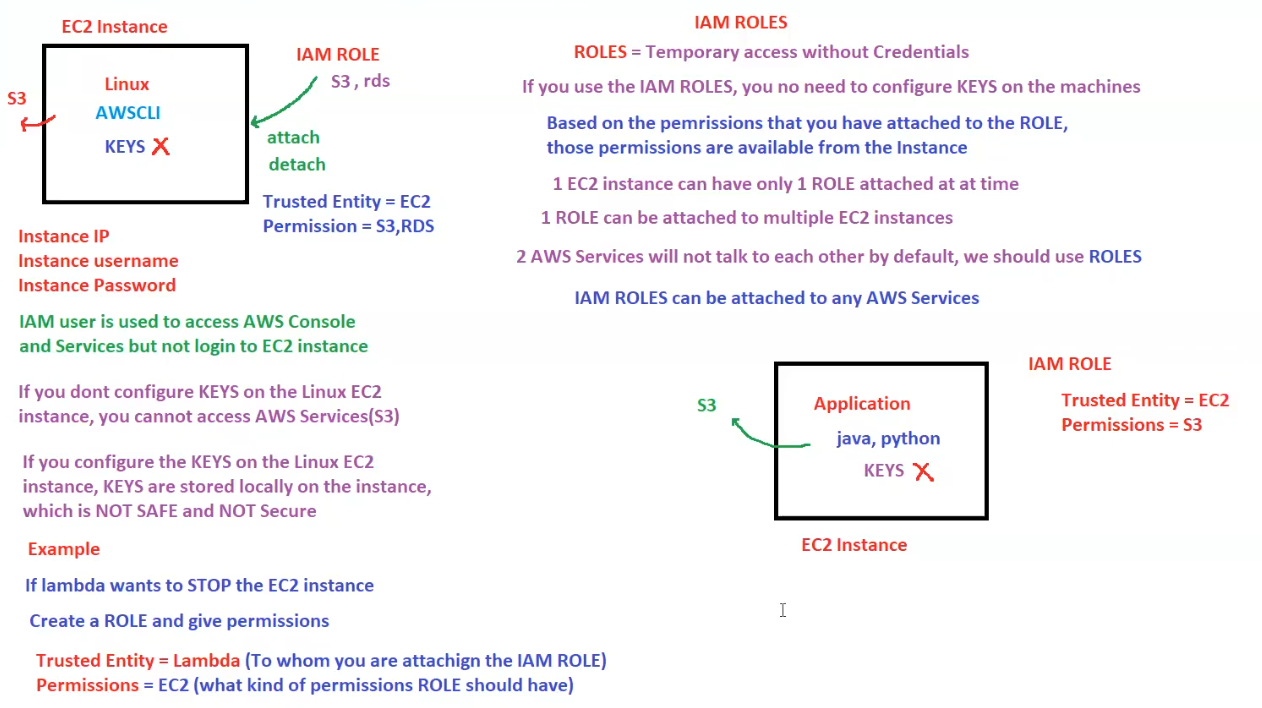
**5. Practical Example: AWS Lambda Managing EC2 Instances**

**Scenario: Lambda Function Needs to Stop an EC2 Instance**

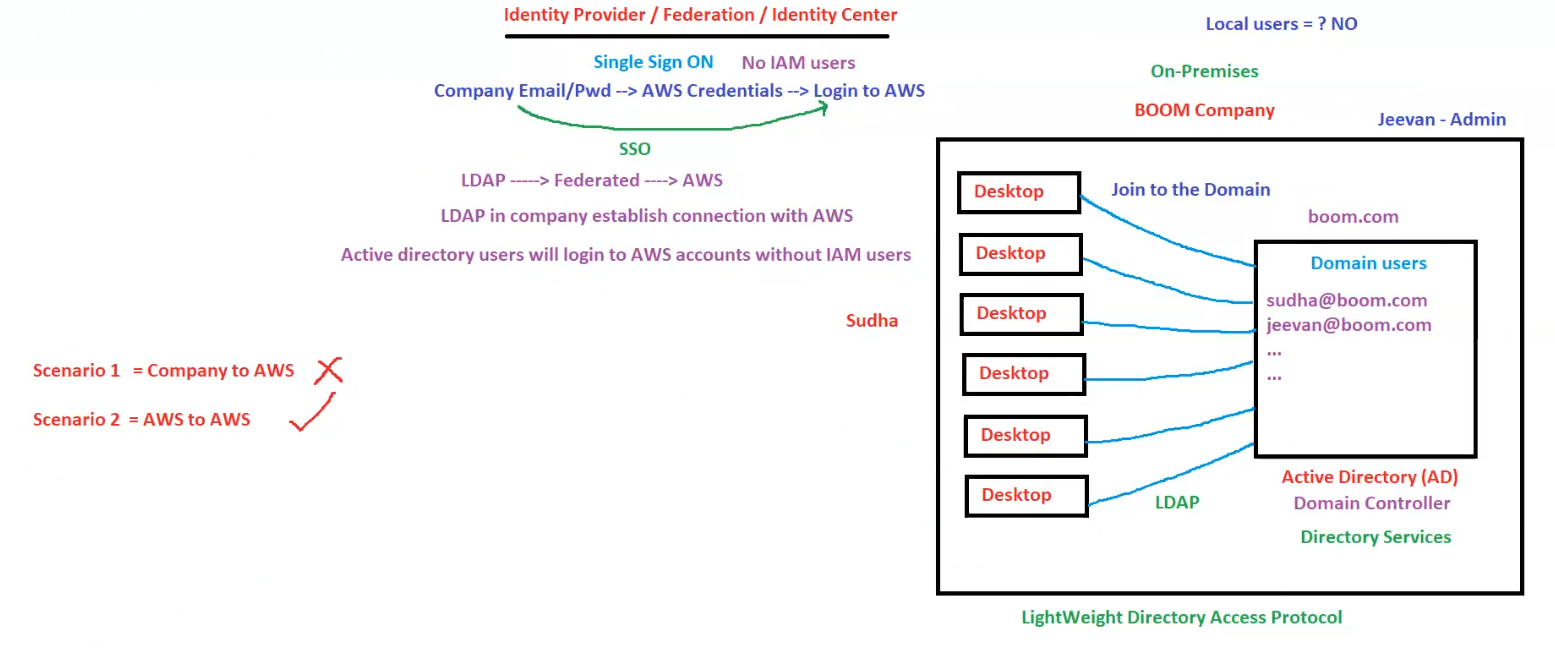
1. Create an **IAM Role**.
2. Assign permissions to allow Lambda to manage EC2 instances.
3. Role details:
   * **Trusted Entity** = Lambda (Lambda will assume this role).
   * **Permissions** = EC2 (Defines actions Lambda can perform on EC2, e.g., stopping an instance).

**6. Key Takeaways**

✅ **Use IAM Roles for secure access** instead of manually configuring keys on EC2 instances.  
✅ **IAM Roles provide temporary credentials**, reducing security risks.  
✅ **AWS services cannot communicate with each other by default**; use IAM Roles to establish secure interactions.  
✅ **One IAM Role per EC2 instance, but one role can be attached to multiple instances.**



## Identity Provider / Federation / Identity Center Overview



**Single Sign-On (SSO) with AWS**

Single Sign-On (SSO) allows users to log in to AWS without requiring separate IAM user accounts. Instead, authentication is handled through the company's identity provider (IdP) and federation mechanisms.

**Key Features:**

* **No IAM Users Needed**: Users authenticate using their company credentials instead of AWS IAM credentials.
* **Authentication Flow**:
  1. **User Login**: Employees use their **Company Email/Password** to log in.
  2. **Federation & Credential Exchange**: The IdP validates credentials and provides **temporary AWS credentials**.
  3. **AWS Access**: Users log in to AWS with these temporary credentials.

**LDAP & Federation Integration**

The company's existing **Lightweight Directory Access Protocol (LDAP)** system is used to authenticate users, which is then federated with AWS.

**Authentication Process:**

* **LDAP** acts as an intermediary between the company’s **Active Directory (AD)** and AWS.
* The company establishes a **federated connection** between LDAP and AWS.
* AWS trusts LDAP authentication, allowing **Active Directory users** to log in without requiring IAM users.
* The process ensures that users can securely access AWS resources while maintaining centralized identity management within the company.

**Company On-Premises Infrastructure**

The company, **BOOM Company**, operates an **on-premises Active Directory (AD)** to manage user authentication and access control.

**Key Infrastructure Components:**

1. **Domain Controller**: Manages authentication requests and domain-related services.
2. **Directory Services**: Provides identity management and access control.
3. **Workstations (Desktops)**: Joined to the company’s domain for authentication.
4. **Users & Credentials**: Employees use **domain credentials** to access company resources.

**Domain Details:**

* **Domain Name**: boom.com
* **Administrator**: Jeevan
* **Users**:
  + [sudha@boom.com](mailto:sudha@boom.com)
  + [jeevan@boom.com](mailto:jeevan@boom.com)
  + Additional employees with similar domain-based email accounts.

**Workstation Setup & Authentication**

Each desktop within the organization is part of the **Active Directory domain** and is authenticated through **LDAP**.

**Process:**

1. **User logs in to a desktop** using their **Active Directory credentials**.
2. **The desktop communicates with the Domain Controller (AD)** for verification.
3. **LDAP establishes a secure connection** with AWS for federated authentication.
4. **User gains access to AWS resources** without requiring separate AWS credentials.

**Scenarios for AWS Integration**

AWS authentication can be implemented in different ways. Below are two possible scenarios:

**Scenario 1: Company to AWS (Rejected ❌)**

* This scenario involves direct authentication from the company's on-premises infrastructure to AWS.
* The company’s Active Directory would authenticate users directly in AWS without any federation layer.
* **Drawbacks**:
  + Complex configuration and maintenance.
  + Security concerns with direct authentication.
  + Does not leverage AWS's built-in identity management features.

**Scenario 2: AWS to AWS (Accepted ✅)**

* Authentication is fully **managed within AWS** using a federated identity provider (IdP).
* Users log in using **SSO**, which leverages AWS Identity Center (formerly AWS SSO).
* **Advantages**:
  + **Seamless authentication**: Users authenticate once and access multiple AWS accounts.
  + **Enhanced security**: No need to manage IAM users separately.
  + **Centralized identity management**: Integrated with the company's directory service.

**Key Takeaways**

* **Local user authentication is NOT used**; all authentication happens via the company's domain.
* **SSO enables seamless AWS access** without creating individual IAM users.
* **LDAP federated authentication ensures secure AWS login** while maintaining centralized user management.
* **AWS-to-AWS authentication is the preferred approach**, leveraging AWS Identity Center for managing user access efficiently.

## SAML

**Security Assertion Markup Language (SAML) 2.0 - Overview**

**1. What is SAML?**

* SAML (**Security Assertion Markup Language**) is an open federation standard.
* It enables an **Identity Provider (IdP)** to authenticate users and securely pass identity and security information to a **Service Provider (SP)**.
* This information is transmitted in **XML format**.

**2. Key Components of SAML Authentication Flow**

**Identity Provider (IdP)**

* The entity responsible for **authenticating users**.
* Examples: **Active Directory, Okta, etc.**
* The IdP verifies a user's identity and generates an **SAML assertion** in XML format.

**Service Provider (SP)**

* The entity that **provides access to a service** based on the authentication and authorization received from the IdP.
* Example: **AWS (Amazon Web Services)**.
* AWS uses **IAM Roles** to grant permissions based on SAML authentication.

**3. How SAML Works in an AWS Environment**

1. **User Authentication at Identity Provider (IdP)**:
   * Users authenticate with the **company's Active Directory** or other IdP.
   * The IdP generates an **SAML assertion** (XML document) containing authentication details.
2. **SAML Assertion Sent to AWS (Service Provider)**:
   * The SAML assertion is transmitted securely from the IdP to AWS.
3. **AWS Uses SAML Assertion to Grant Access**:
   * AWS receives the **SAML assertion** and validates it.
   * Based on the assertion, AWS assigns the user to a specific **IAM Role** in an AWS account.
4. **User Gains Access to AWS Resources**:
   * The assigned **IAM Role** determines what actions the user can perform in AWS.

**4. AWS Multi-Account SAML Authentication**

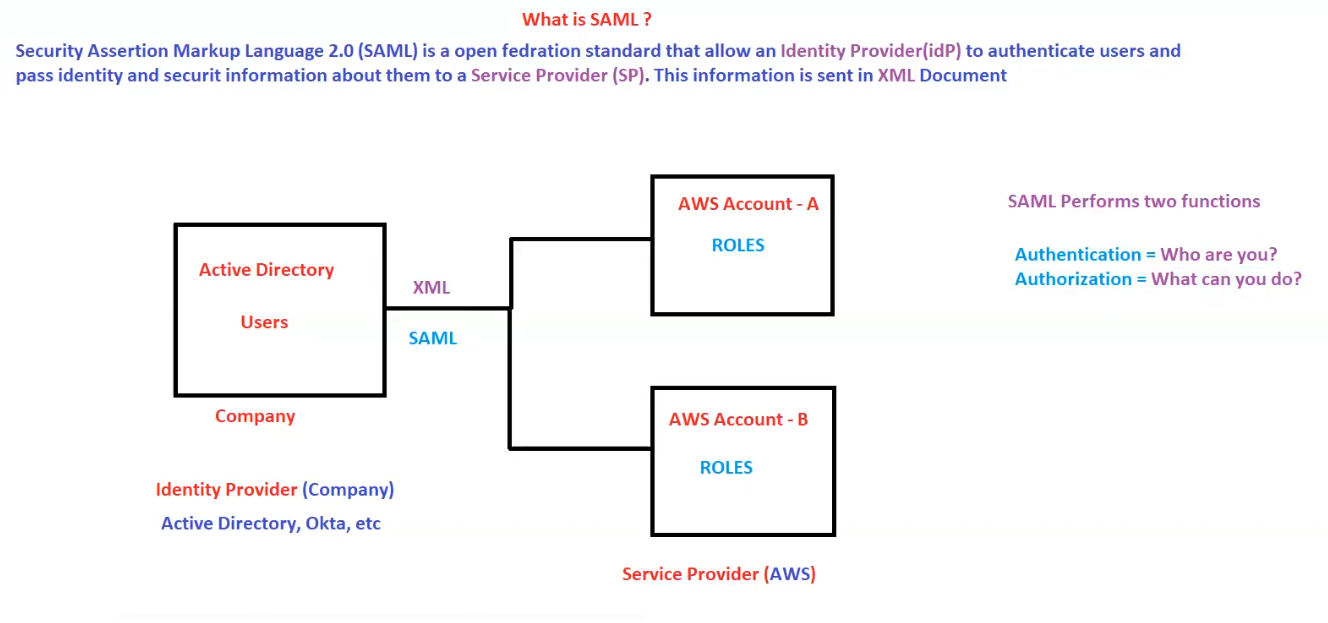
* A single **IdP** can authenticate users and provide access to **multiple AWS accounts**.
* Each AWS account has **IAM Roles** mapped to users via SAML.
* Example:
  + **AWS Account A** → IAM Roles for specific access.
  + **AWS Account B** → IAM Roles for different access.

**5. SAML Performs Two Key Functions**

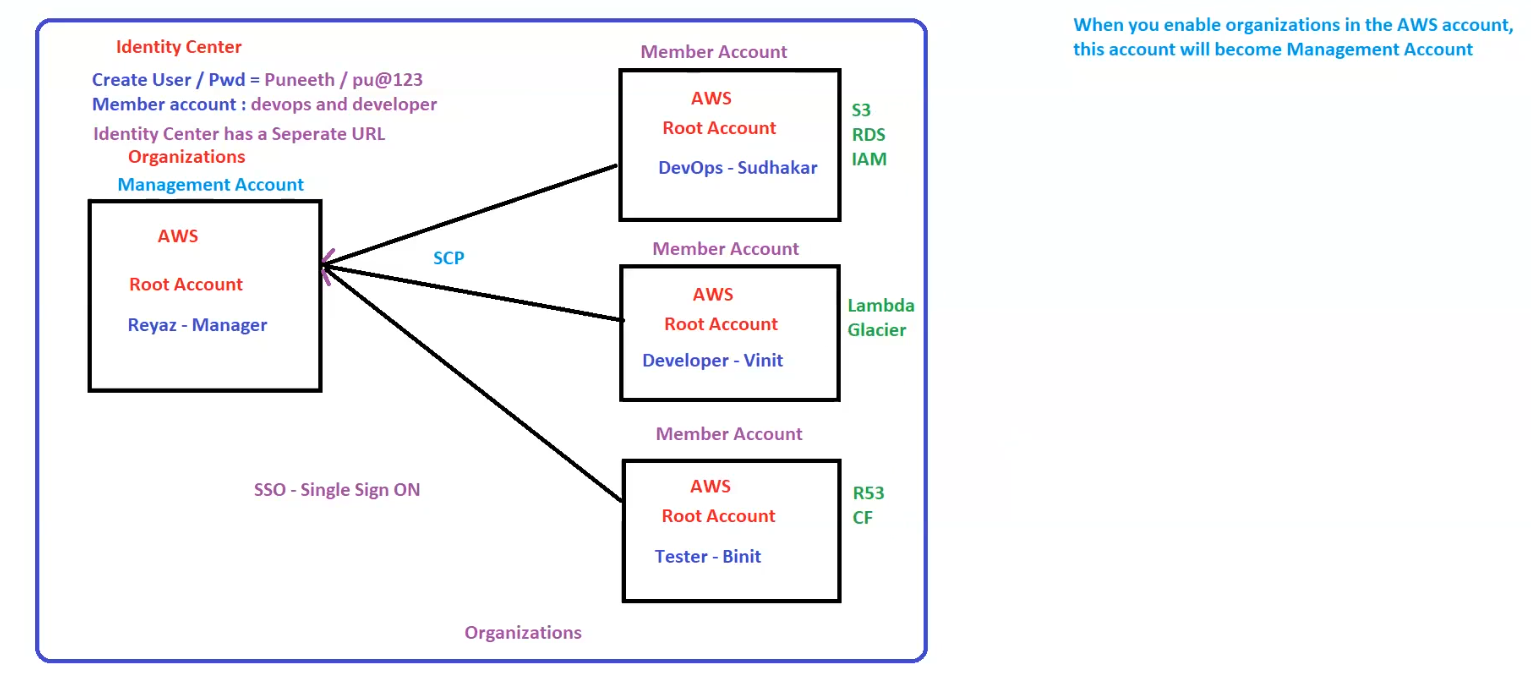
1. **Authentication** – Confirms **who** the user is.
2. **Authorization** – Defines **what** the user is allowed to do.

**6. Benefits of Using SAML in AWS**

* **Centralized authentication**: Users sign in once with IdP credentials.
* **No need to store AWS credentials**: Secure access without managing AWS IAM user passwords.
* **Supports multiple AWS accounts**: Easily switch between different AWS environments.
* **Enhanced security**: Uses SAML assertions to verify identity without exposing credentials.



## Identity Center



**AWS Organizations & Identity Center Overview**

**1. AWS Identity Center**

* **Purpose**: Manages user identities and access control across multiple AWS accounts.
* **User Creation**:
  + **Example User**: Puneeth
  + **Password**: pu@123
  + **Member Accounts Assigned**: **DevOps** and **Developer**
* **Identity Center Access**:
  + It has a **separate URL** for user authentication and management.

**2. AWS Organizations**

* **Organizations** enable centralized management of multiple AWS accounts.
* **When AWS Organizations is enabled**, the **AWS Management Account** is automatically assigned.

**Management Account (Root)**

* **Account Type**: **Root AWS Account**
* **Manager**: Reyaz
* **Role**: Administers and controls the organization.
* **Policies Applied**: **SCP (Service Control Policies)**  
  *(SCPs help enforce permission boundaries on member accounts.)*
* **Supports**: **SSO (Single Sign-On)** for easier access to AWS resources.

**3. Member Accounts**

AWS Organizations contains multiple **Member Accounts**, each with separate users and resources.

**Member Account 1 (DevOps)**

* **Account Type**: **Root AWS Account**
* **User**: Sudhakar (DevOps Engineer)
* **AWS Services Used**:
  + **S3** (Simple Storage Service)
  + **RDS** (Relational Database Service)
  + **IAM** (Identity and Access Management)

**Member Account 2 (Developer)**

* **Account Type**: **Root AWS Account**
* **User**: Vinit (Developer)
* **AWS Services Used**:
  + **Lambda** (Serverless Computing)
  + **Glacier** (Data Archiving and Backup)

**Member Account 3 (Tester)**

* **Account Type**: **Root AWS Account**
* **User**: Binit (Tester)
* **AWS Services Used**:
  + **Route 53** (DNS and Domain Management)
  + **CloudFormation (CF)** (Infrastructure as Code)

**4. Key Concepts**

* **Management Account**: The primary account that controls and manages AWS Organizations.
* **Member Accounts**: Individual AWS accounts managed under the organization.
* **SCP (Service Control Policies)**: Policies applied to restrict or allow specific actions across accounts.
* **SSO (Single Sign-On)**: A method to log in once and access multiple AWS accounts seamlessly.

**5. Important Notes**

* When **AWS Organizations is enabled**, the account becomes the **Management Account**.
* The **Identity Center** provides a centralized way to manage user access.
* **Each member account** has distinct roles, services, and responsibilities.
* **SCPs are enforced from the Management Account** to control access across the organization.

## IAM Tags

**AWS IAM, EC2, and Tags Overview**

**1. EC2 Instance Information**

* Example EC2 instance:
  + **Name** = WebServer
  + **Tags**: Key-Value pairs (Max 50 tags per resource)
* EC2 instances can be managed using **Lambda functions** and **Tags** to automate actions (e.g., stopping all instances).

**2. IAM Users Management**

**Hard Worker Approach**

* Manually checks all **500 IAM users** and deletes them **one by one**.
* Includes users from different categories:
  + **DevOps**
  + **Admins**
  + **Security**
  + **Testers**
  + **Ali (Example user)**

**Smart Worker Approach**

* Uses **automation** with **Tags**.
* Example:
* Delete IAM Users where Key = Name, Value = Testers
* Efficient way to manage IAM users **based on tags**.

**3. IAM - Tags**

* **Tags are Key-Value Pairs** used for:
  + **Identification** of AWS resources.
  + **Billing purposes** (cost tracking).
  + **Automation** (e.g., stopping instances with Lambda).
  + **Cost Optimization**.
* **Tags are AWS-wide**, not IAM-specific.
* **Maximum 50 Tags per resource**.

**Example of Using Tags for Automation**

* **Stopping EC2 Instances Using Lambda & Tags**:
  + Example:
  + Using Lambda and tags, we can automate stopping all EC2 instances.

**4. IAM Reports and Analysis Tools**

**IAM Credentials Report**

* A report listing all **IAM users** and the status of their **credentials**.

**IAM Access Advisor**

* Shows **service permissions** granted to a user.
* Displays **last used time** for permissions.

**IAM Access Analyzer**

* **Analyzes IAM user access**.
* Identifies **unused** or **external** access permissions.

**5. IAM Policy Structure**

IAM policies consist of **rules** that define **permissions**.

**IAM Policy Components**

1. **Version** – Policy language version.
2. **ID** – Unique identifier for the policy (Optional).
3. **Statement** – Defines one or more rules.

**IAM Policy Statement Components**

* **Sid** (Optional) – Identifier for the statement.
* **Effect** – Allow or Deny.
* **Principal** – Account, User, or Role to which the policy applies.
* **Action** – List of actions the policy allows or denies.
* **Resource** – Specifies the AWS **ARN** (Amazon Resource Name) of applicable resources.
* **Conditions** (Optional) – Defines conditions when the policy is in effect.

**6. Example of a Tagging System (Non-AWS Example)**

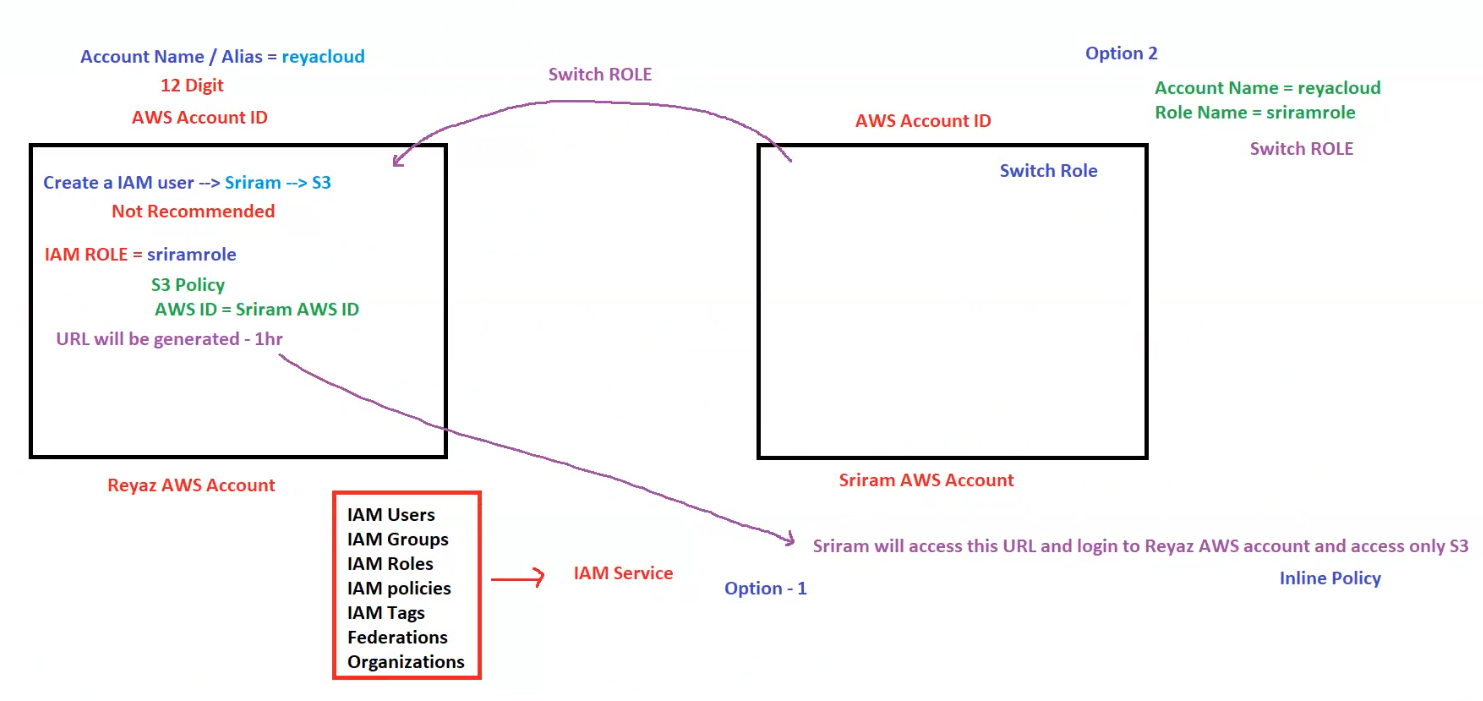
* **Example**: Using Tags for a product like **Jeans**.
  + **Price** = 15k
  + **Size** = (Not mentioned)
  + **Brand** = Levi
  + Follows a **Key-Value Pair** system.

**7. Summary**

* **IAM Users** can be managed efficiently using **Tags** and **automation**.
* **IAM Policies** define access permissions using structured rules.
* **Tags** play a crucial role in **AWS resource management**, **cost optimization**, and **automation**.
* **IAM Analysis Tools** help monitor and manage user access.
* **Smart Work** = Automating IAM user and EC2 instance management using **Lambda & Tags**.



## Cross-Account Access in AWS Using IAM Roles



**Cross-Account Access in AWS Using IAM Roles**

**1. AWS Account Details**

* **Account Name / Alias**: reyacloud
* **AWS Account ID**: 12-digit unique identifier

**2. IAM User vs IAM Role for Cross-Account Access**

**Option 1: IAM User (Not Recommended)**

* **Create an IAM user**: Sriram
* Assign **S3 access** directly to the IAM user.
* **Drawback**: Not recommended as managing individual IAM users across multiple accounts can be complex.

**Option 2: IAM Role (Recommended)**

* **IAM Role Name**: sriramrole
* **Attached Policies**:
  + **S3 Policy** (Grants permission to access specific S3 resources)
  + **AWS ID**: Associated with **Sriram’s AWS ID**
* **Generated URL**:
  + Temporary URL provided for **1 hour** to assume the role.
  + Sriram can use this URL to access **Reyaz AWS Account** and use **S3**.

**3. Switching Roles for Cross-Account Access**

* **AWS Account A (Reyaz AWS Account)**
  + IAM Role: sriramrole
  + Configured with an **S3 Policy**.
  + Allows trusted users from **Sriram AWS Account** to assume this role.
* **AWS Account B (Sriram AWS Account)**
  + Instead of creating a user, Sriram **switches roles** to sriramrole in reyacloud AWS Account.
  + Provides **temporary access** to **S3** without creating direct IAM users.

**4. IAM Services Involved**

* **IAM Users** – Not recommended for cross-account access.
* **IAM Groups** – Can be used for permission grouping.
* **IAM Roles** – The preferred method for cross-account access.
* **IAM Policies** – Define permissions for IAM users and roles.
* **IAM Tags** – Used for automation and cost management.
* **Federations** – Allow integration with external identity providers.
* **Organizations** – Manage multiple AWS accounts centrally.

**5. Steps for Cross-Account Access (Using IAM Role)**

**Step 1: Create an IAM Role in reyacloud AWS Account**

1. Go to **IAM Console** → **Roles**.
2. Click **Create Role** → Select **Another AWS Account**.
3. Enter **Sriram’s AWS Account ID**.
4. Attach **S3 Policy** to restrict access to only S3.
5. Create the role sriramrole.

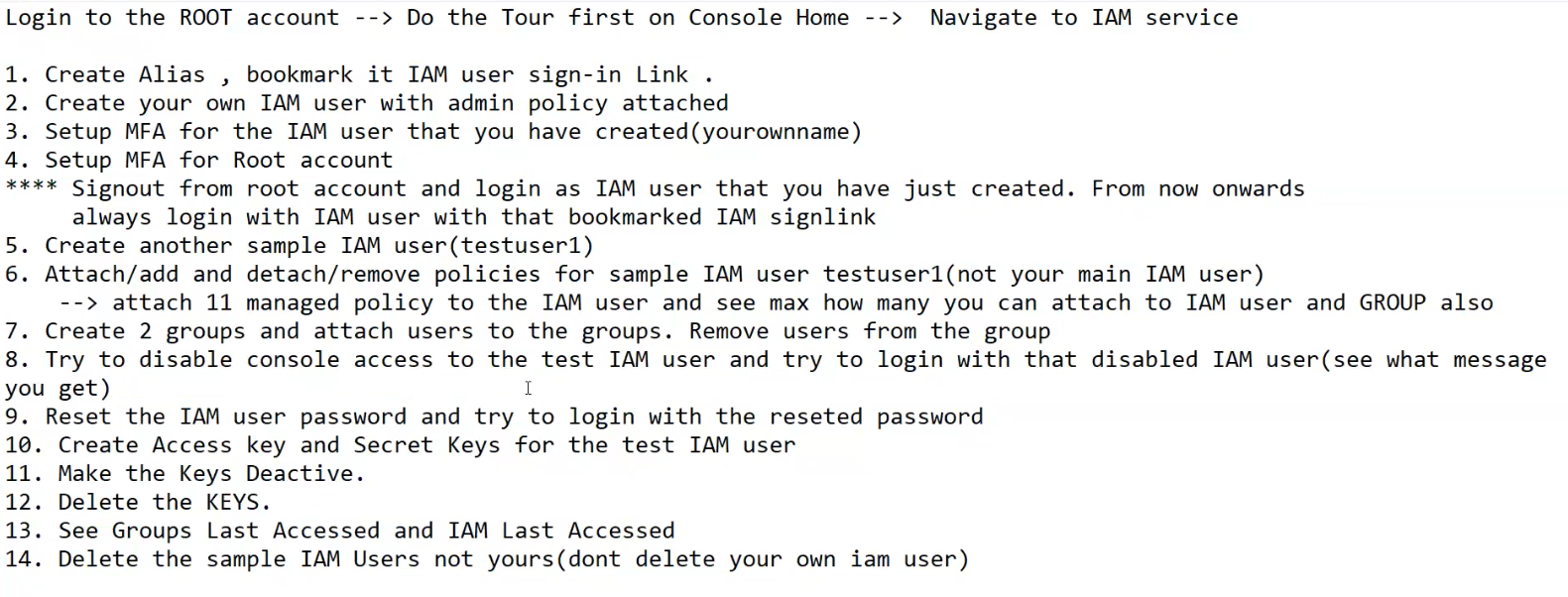
**Step 2: Sriram Switches Role in AWS Console**

1. Go to **AWS Console** → Click on **Switch Role**.
2. Enter:
   * **Account Name**: reyacloud
   * **Role Name**: sriramrole
3. Assume the role and get temporary access.

**6. Summary**

* **Creating IAM users for cross-account access is not recommended**.
* **IAM roles with trusted policies** provide secure and flexible cross-account access.
* **Sriram can switch roles to sriramrole** in the reyacloud AWS Account.
* **Temporary URLs allow one-time access for up to 1 hour**.
* **Inline Policies and S3 Role Policies** restrict permissions to only S3.

## IAM GUI Checklist



Login to the ROOT account -- > Do the Tour first on Console Home -- > Navigate to IAM service

1. Create Alias , bookmark it IAM user sign-in Link .

2. Create your own IAM user with admin policy attached

3. Setup MFA for the IAM user that you have created(yourownname)

4. Setup MFA for Root account

\*\*\*\* Signout from root account and login as IAM user that you have just created. From now onwards

always login with IAM user with that bookmarked IAM signlink

5. Create another sample IAM user(testuser1)

6. Attach/add and detach/remove policies for sample IAM user testuser1(not your main IAM user)

-- > attach 11 managed policy to the IAM user and see max how many you can attach to IAM user and GROUP also

7. Create 2 groups and attach users to the groups. Remove users from the group

8. Try to disable console access to the test IAM user and try to login with that disabled IAM user(see what message

you get)

9. Reset the IAM user password and try to login with the reseted password

10. Create Access key and Secret Keys for the test IAM user

11. Make the Keys Deactive.

12. Delete the KEYS.

13. See Groups Last Accessed and IAM Last Accessed

14. Delete the sample IAM Users not yours(dont delete your own iam user)