AWS Identity and Access Management

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. With IAM, you can manage permissions that control which AWS resources users can access. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. IAM provides the infrastructure necessary to control authentication and authorization for your AWS accounts.

How IAM Works?

IAM verifies that a user or service has the necessary authorization to access a particular service in the AWS cloud. We can also use IAM to grant the right level of access to specific users, groups, or services. For example, we can use IAM to enable an EC2 instance to access S3 buckets by requesting fine-grained permissions.

Sumeha Jakate

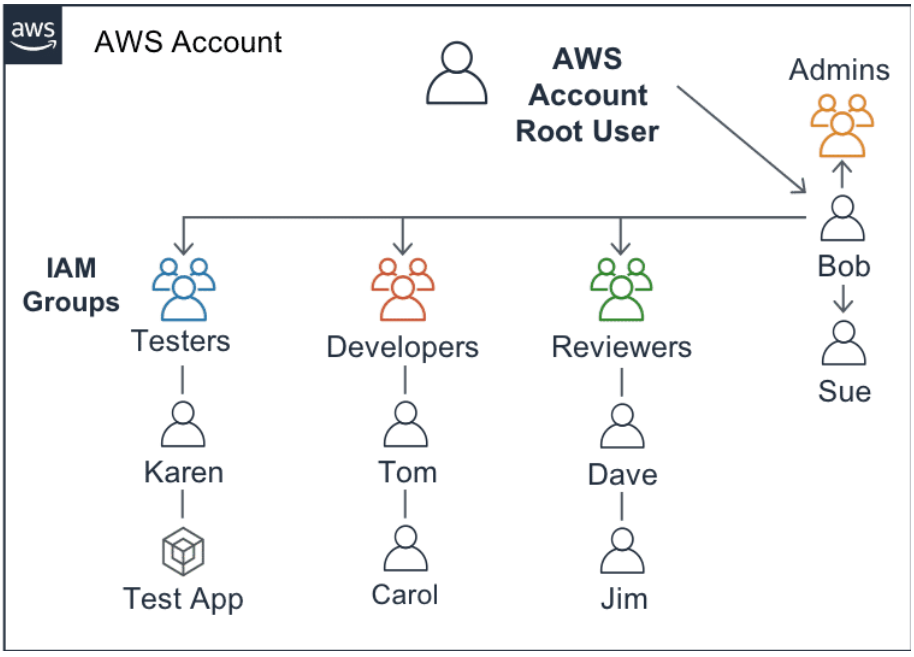
IAM Fundamentals

1) Users and groups

Careful management of access credentials is the foundation of how you will secure your resources in the cloud. When you open an AWS account, the identity you begin with has access to all AWS services and resources in that account. You use this identity to establish less-privileged users and role-based access in IAM. IAM is a centralized mechanism for creating and managing individual users and their permissions with your AWS account. Every interaction you make with AWS is authenticated.

Users and groups should be used for managing access to resources *within* your account. When you have an organization that spans multiple AWS accounts, you need to manage access to all the AWS accounts centrally via identity federation because users and groups are not scalable.

An IAM group is a collection of users. Groups allow you to specify permissions for similar types of users. For example, if you have a group named *Developers,* you can give that group the types of permissions that developers typically need. Create groups that reflect organization roles, not technical commonality.



2) Roles

IAM roles allow you to delegate access to users, applications, or services that normally don't have access to your organization's AWS resources. You can assume a role to obtain temporary security credentials that you can use to make AWS API calls, just like Suri was added to the Internal Candidate List and able to use the company boat. Consequently, you don't have to share long-term credentials or define permissions for each entity that requires access to a resource. When you assume a role, it's like putting on a different hat. Each role, or hat, comes with specific access permissions, and assuming each role allows you take on the access permissions of that role.

Roles can be used in the following scenarios:

- An IAM user or role in the same or different AWS account that needs the access that the role provides
- Applications on an Amazon Elastic Compute Cloud (Amazon EC2) instance that need access to AWS resources
- An AWS service that needs to call other services on your behalf or create and manage resources in your account.
- An external user authenticated by an identity provider service that is compatible with SAML 2.0 or OpenID Connect, or a custom-built identity broker

If your organization uses multiple accounts, roles will be a key part of your strategy of centrally managing users' access across multiple accounts

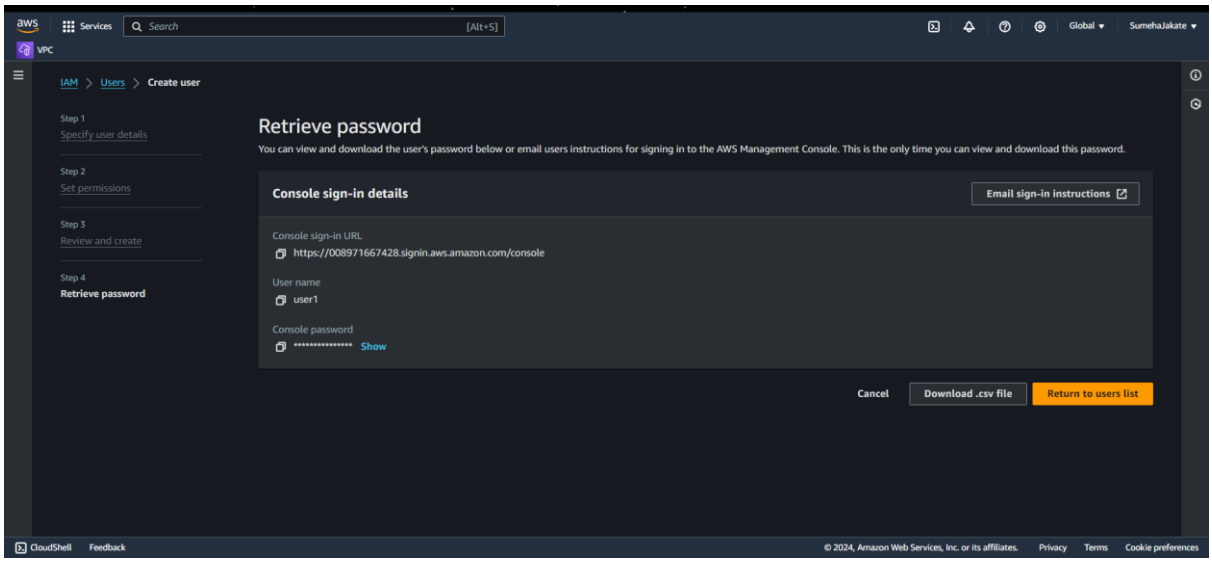
Types of AWS credentials

A user in AWS consists of a name, a password to sign in to the AWS Management Console, and up to two access keys, which can be used with the API or CLI.

1) Username and Password
2) Multi-Factor Authentication
3) User Access Keys

1) Username and Password

   A password policy is a set of rules that define the type of password an IAM user can set. You should define a password policy for all your IAM users to enforce strong passwords and to require your users to regularly change their passwords. Password requirements are similar to those found in most secure online environments.

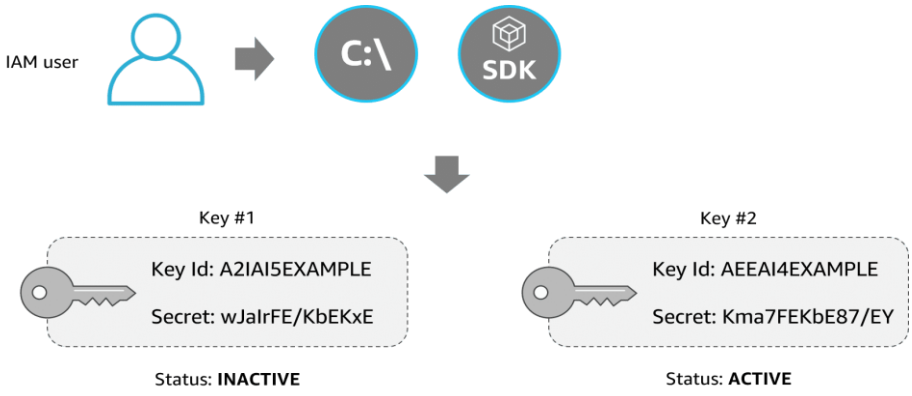


2) Multi-Factor Authentication

Multi-factor authentication (MFA) is an additional layer of security for accessing AWS services. With this authentication method, more than one authentication factor is checked before access is granted, which consists of a user name, a password, and the single-use code from the MFA device. AWS CLI also supports MFA.

3) User Access Keys

Users need their own access keys to make programmatic calls to AWS using the AWS CLI or the AWS SDKs, or direct HTTPS calls using the APIs for individual AWS services. Access keys are used to digitally sign API calls made to AWS services. Each access key credential consists of an access key ID and a secret key. Each user can have two active access keys, which is useful when you need to rotate the user's access keys or revoke permissions.



IAM Policy

In order to talk about IAM policies, you first need to cover the three main pieces of logic that define what is in the policy and how the policy actually works. These pieces make up the request context that is authenticated by IAM and authorized accordingly. You can think of the principal, action, and resource as the subject, verb, and object of a sentence, respectively.

Principal

- User, role, external user, or application that sent the request and the policies associated with that principal

Action

- What the principal is attempting to do

Resource

- AWS resource object upon which the actions or operations are performed

Access through identity-based policies

You manage access in AWS by creating policies and attaching them to IAM identities or AWS resources. An identity-based policy is an object in AWS that, when associated with an IAM identity, defines their permissions. AWS evaluates these policies when a principal entity (IAM user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents.

There are three types of identity-based policies:

IAM policy example

Below is a simple example of an IAM identity-based policy granting access to a certain Amazon Simple Storage Service (Amazon S3) bucket.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "ListObjectsInBucket",
   "Effect": "Allow",
   "Action": "s3:ListBucket",
   "Resource": ["arn:aws:s3:::myBucket" ]
  },
  {
   "Sid": "AllObjectsActions",
   "Effect": "Allow",
   "Action": "s3:*Object",
   "Resource": ["arn:aws:s3:::myBucket/*"]
  }]
}
```

- **Statement**

  The Statement element is the main element for a policy. It can contain a single statement or an array of individual statements. Each individual statement block must be enclosed in curly braces { }.

- **Sid**

  The Sid element is optional and provides a brief description of the policy statement.

- **Effect**

  The Effect element specifies whether the statement will explicitly allow or deny access.

- **Action**

  The Action element describes the type of access that should be allowed or denied. The "*" shown here is a wildcard, which means that any Amazon S3 action ending in "Object" will be allowed. Some examples include GetObject and PutObject.

- **Resource**

  The Resource element specifies the object or objects that the policy statement covers.

Yes, but I have conditions

IAM allows you to add conditions to your policy statements. The Condition element is optional and lets you specify conditions for when a policy is in effect. In the condition element, you build expressions in which you use condition operators (equal, less than, etc.) to match the condition keys and values in the policy against keys and values in the request.

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

For example, the following condition can be added to an Amazon S3 bucket policy to further restrict access to the bucket. In this case, the condition includes the StringEquals operator to ensure that only requests made by JohnDoe will be allowed.

```
"Condition" : { "StringEquals" : { "aws:username" : "JohnDoe" }}
```

Policy types

AWS supports six types of policies: identity-based policies, resource-based policies, IAM permissions boundaries, AWS Organizations service control policies (SCPs), access control lists (ACLs), and session policies. All of these polices are evaluated before a request is either allowed or denied.

1) **Identity-based**

   Also known as IAM policies, identity-based policies are managed and inline policies attached to IAM identities (users, groups to which users belong, or roles).

   Impacts IAM principal permissions

2) **Resource-based**

   These are inline policies that are attached to AWS resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies. Resource-based policies grant permissions to the principal that is specified in the policy; hence, the principal policy element is required.

   Grants permission to principals or accounts (same or different accounts)

3) **Permissions boundaries**

   A permissions boundary sets the maximum permissions that an identity-based policy can grant to an IAM entity. The entity can perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role as the principal are not limited by the permissions boundary.

   Restricts permissions for the IAM entity attached to it.

4) **AWS Organizations SCPs**

AWS Organizations is a service for grouping and centrally managing AWS accounts. If you enable all features in an organization, then you can apply SCPs to any or all of your accounts. SCPs specify the maximum permissions for an account, or a group of accounts, called an organizational unit (OU).
Restricts permissions for entities in an AWS account, including AWS account root users

5) **ACLs**

Use ACLs to control which principals in other accounts can access the resource to which the ACL is attached. ACLs are supported by Amazon S3 buckets and objects. They are similar to resource-based policies although they are the only policy type that does not use the JSON policy document structure. ACLs are cross-account permissions policies that grant permissions to the specified principal. ACLs cannot grant permissions to entities within the same account

6) **Session policies**

A session policy is an inline permissions policy that users pass in the session when they assume the role. The permissions for a session are the intersection of the identity-based policies for the IAM entity (user or role) used to create the session and the session policies. Permissions can also come from a resource-based policy. Session policies limit the permissions that the role or user's identity-based policies grant to the session. An upcoming section will cover session policies in more detail.
Restricts permissions for assumed roles and federated users

Feature	Authentication	Authorization
Definition	The process of verifying the identity of a user or service.	The process of determining what an authenticated identity can do.
Purpose	To ensure that the entity trying to access the system is who it claims to be.	To determine the level of access and actions permitted for the authenticated entity.
Mechanism	Involves the presentation and validation of credentials.	Involves the evaluation of policies attached to users, groups, or roles.
When it Occurs	Occurs at the initial stage of access.	Occurs after successful authentication, during resource access attempts.
Scope	Establishes the identity of the user or service.	Defines what actions the authenticated user or service can perform on AWS resources.
Examples	Logging into AWS Management Console Using AWS CLI with access keys Setting up MFA for an account	Allowing read access to an S3 bucket Denying access to delete EC2 instances Granting role permissions to access specific services
Verification Method	Credentials (passwords, access keys, MFA tokens)	Policy evaluation against the action, resource, and conditions.
Outcome	Identity is confirmed.	Access permissions are determined and enforced.