Identity and Access Management: Ankit Mohari

General IAM Concepts

AWS Identity and Access Management (IAM) is a web service that helps us securely control access to AWS resources.

We use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

IAM makes it easy to provide multiple users secure access to AWS resources.

When we first create an AWS account, we begin with a single sign-in identity that has complete access to all AWS services and resources in the account.

This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that we used to create the account.

IAM can be used to manage:

- Users.
- Groups.
- Access policies.
- Roles.
- User credentials.
- User password policies.
- Multi-factor authentication (MFA).
- API keys for programmatic access (CLI).

IAM provides the following features:

- Shared access to your AWS account.
- Granular permissions.
- Secure access to AWS resources for application that run on Amazon EC2.
- Multi-Factor authentication.
- Identity federation.
- Identity information for assurance.
- PCI DSS compliance.
- Integrated with may AWS services.
- Eventually consistent.
- Free to use.

We can work with AWS Identity and Access Management in any of the following ways:

- AWS Management Console.
- AWS Command Line Tools.
- AWS SDKs.
- IAM HTTPS API.

By default new users are created with NO access to any AWS services – they can only login to the AWS console.

Permission must be explicitly granted to allow a user to access an AWS service.

IAM users are individuals who have been granted access to an AWS account.

Each IAM user has three main components:

- A user-name.
- A password.
- Permissions to access various resources.

We can apply granular permissions with IAM.

We can assign users individual security credentials such as access keys, passwords, and multi-factor authentication devices.

IAM is not used for application-level authentication.

Identity Federation (including AD, Facebook etc.) can be configured allowing secure access to resources in an AWS account without creating an IAM user account.

Multi-factor authentication (MFA) can be enabled/enforced for the AWS account and for individual users under the account.

MFA uses an authentication device that continually generates random, six-digit, single-use authentication codes.

We can authenticate using an MFA device in the following two ways:

- Through the **AWS Management Console** – the user is prompted for a user name, password and authentication code.
- Using the **AWS API** – restrictions are added to IAM policies and developers can request temporary security credentials and pass MFA parameters in their AWS STS API requests.

- Using the **AWS CLI** by obtaining temporary security credentials from STS (aws sts get-session-token).

It is a best practice to always setup multi-factor authentication on the root account.

IAM is universal (global) and does not apply to regions.IAM is eventually consistent.IAM replicates data across multiple data centres around the world.

The "root account" is the account created when we setup the AWS account. It has complete Admin access and is the only account that has this access by default.

It is a best practice to not use the root account for anything other than billing.

Power user access allows all permissions except the management of groups and users in IAM.

Temporary security credentials consist of the AWS access key ID, secret access key, and security token.

IAM can assign temporary security credentials to provide users with temporary access to services/resources.

To sign-in we must provide our account ID or account alias in addition to a user name and password.

The sign-in URL includes the account ID or account alias, e.g:

https://***My_AWS_Account_ID***.signin.aws.amazon.com/console/.

Alternatively, we can sign-in at the following URL and enter our account ID or alias manually:

https://console.aws.amazon.com/.

IAM integrates with many different AWS services.

IAM supports PCI DSS compliance.(The Payment Card Industry Security Standards Council (**PCI** SSC), the **compliance** scheme aims to secure credit and debit card transactions against data theft and fraud).

AWS recommend that you use the AWS SDKs to make programmatic API calls to IAM.

However, we can also use the IAM Query API to make direct calls to the IAM web service.

Authentication Methods:

Console password:

- A password that the user can enter to sign in to interactive sessions such as the AWS Management Console.
- We can allow users to change their own passwords.
- We can allow selected IAM users to change their passwords by disabling the option for all users and using an IAM policy to grant permissions for the selected users.

Access Keys:

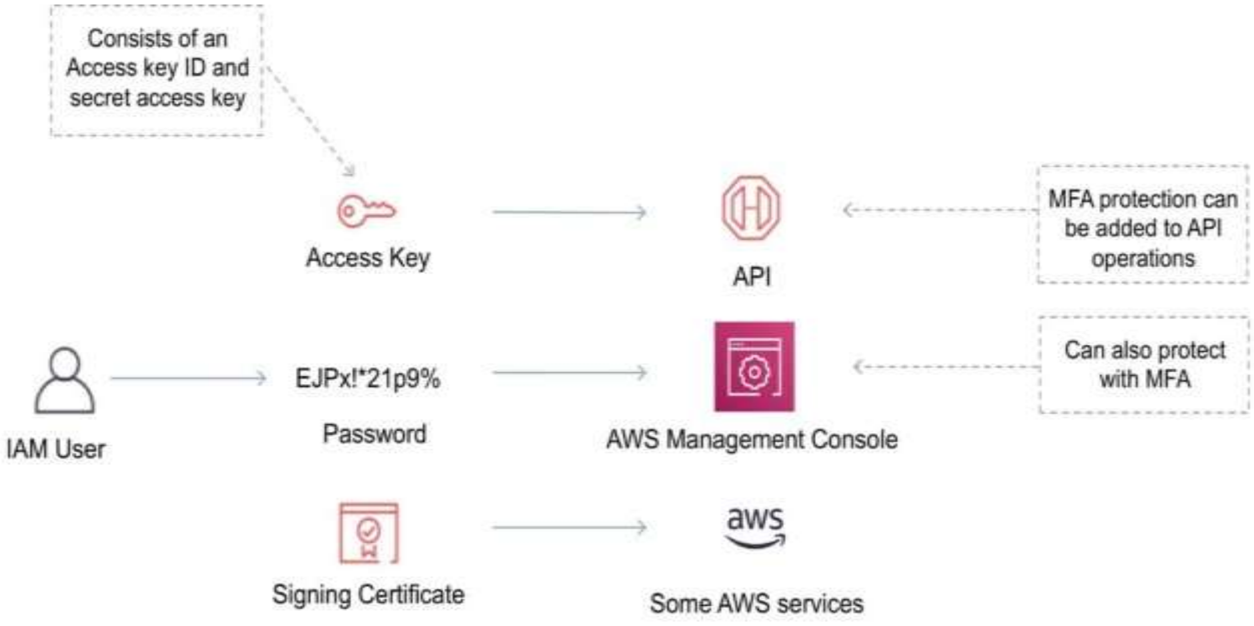
- A combination of an **access key ID** and a **secret access key.**
- We can assign two active access keys to a user at a time.
- These can be used to make programmatic calls to AWS when using the **API** in program code or at a command prompt when using the **AWS CLI** or the **AWS PowerShell** tools.
- We can create, modify, view or rotate access keys.

- When created IAM returns the access key ID and secret access key.
- The secret access is returned only at creation time and if lost a new key must be created.
- Ensure access keys and secret access keys are stored securely.
- Users can be given access to change their own keys through IAM policy (not from the console).
- We can disable a user’s access key which prevents it from being used for API calls.

Server certificates:

- SSL/TLS certificates that we can use to authenticate with some AWS services.
- AWS recommends that we use the AWS Certificate Manager (ACM) to provision, manage and deploy your server certificates.
- Use IAM only when we must support HTTPS connections in a region that is not supported by ACM.

The following diagram shows the different methods of authentication available with IAM:


Consists of an Access key ID and secret access key
Access Key
API
MFA protection can be added to API operations
IAM User
EJPx!*21p9%
Password
AWS Management Console
Can also protect with MFA
Signing Certificate
aws
Some AWS services

IAM Users

An IAM user is an entity that represents a person or service.

Can be assigned:

- An access key ID and secret access key for programmatic access to the AWS API, CLI, SDK, and other development tools.
- A password for access to the management console.

By default, users cannot access anything in your account.

The account root user credentials are the email address used to create the account and a password.

The root account has full administrative permissions and these cannot be restricted.

Best practice for root accounts:

- Don’t use the root user credentials.
- Don’t share the root user credentials.
- Create an IAM user and assign administrative permissions as required.
- Enable MFA.

IAM users can be created to represent applications and these are known as “service accounts”.

We can have up to 5000 users per AWS account.

Each user account has a friendly name and an ARN which uniquely identifies the user across AWS.

A unique ID is also created which is returned only when you create the user using the API, Tools for Windows PowerShell or the AWS CLI.

We should create individual IAM accounts for users (best practice not to share accounts).

The Access Key ID and Secret Access Key are not the same as a password and cannot be used to login to the AWS console.

The Access Key ID and Secret Access Key can only be used once and must be regenerated if lost.

A password policy can be defined for enforcing password length, complexity etc. (applies to all users).

You can allow or disallow the ability to change passwords using an IAM policy.

Access keys and passwords should be changed regularly.

Groups

Groups are collections of users and have policies attached to them.

A group is not an identity and cannot be identified as a principal in an IAM policy.

Use groups to assign permissions to users.

Use the principle of least privilege when assigning permissions.

We cannot nest groups (groups within groups).

Roles

Roles are created and then “assumed” by trusted entities and define a set of permissions for making AWS service requests.

With IAM Roles you can delegate permissions to resources for users and services without using permanent credentials (e.g. user name and password).

IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls.

We can delegate using roles.

There are no credentials associated with a role (password or access keys).

IAM users can temporarily assume a role to take on permissions for a specific task.

A role can be assigned to a federated user who signs in using an external identity provider.

Temporary credentials are primarily used with IAM roles and automatically expire.

Roles can be assumed temporarily through the console or programmatically with the **AWS CLI**, **Tools for Windows PowerShell** or **API.**

IAM roles with EC2 instances:

- IAM roles can be used for granting applications running on EC2 instances permissions to AWS API requests using instance profiles.
- Only one role can be assigned to an EC2 instance at a time.
- A role can be assigned at the **EC2 instance creation time or at any time afterwards.**
- When using the AWS CLI or API instance profiles must be created manually (it's automatic and transparent through the console).
- Applications retrieve temporary security credentials from the instance metadata.

Role Delegation:

- Create an IAM role with two policies:
  - Permissions policy – grants the user of the role the required permissions on a resource.
  - Trust policy – specifies the trusted accounts that are allowed to assume the role.
- Wildcards (*) cannot be specified as a principal.
- A permissions policy must also be attached to the user in the trusted account.

Policies

Policies are documents that define permissions and can be applied to users, groups and roles.

Policy documents are written in JSON (key value pair that consists of an attribute and a value).

All permissions are implicitly denied by default.

The most restrictive policy is applied.

The IAM policy simulator is a tool to help you understand, test, and validate the effects of access control policies.

The Condition element can be used to apply further conditional logic.

STS

The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for IAM users or for users that you authenticate (federated users).

By default, AWS STS is available as a global service, and all AWS STS requests go to a single endpoint at **https://sts.amazonaws.com**.

We can optionally send our AWS STS requests to endpoints in any region (can reduce latency).

All regions are enabled for STS by default but can be disabled.

The region in which temporary credentials are requested must be enabled.

Credentials will always work globally.

STS supports AWS CloudTrail, which records AWS calls for your AWS account and delivers log files to an S3 bucket.

Temporary security credentials work almost identically to long-term access key credentials that IAM users can use, with the following differences:

- Temporary security credentials are short-term.
- They can be configured to last anywhere from a few minutes to several hours.
- After the credentials expire, AWS no longer recognizes them or allows any kind of access to API requests made with them.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested.
- When (or even before) the temporary security credentials expire, the user can request new credentials, as long as the user requesting them still has permission to do so.

Advantages of STS are:

- We do not have to distribute or embed long-term AWS security credentials with an application.
- We can provide access to our AWS resources to users without having to define an AWS identity for them (temporary security credentials are the basis for IAM Roles and ID Federation).
- The temporary security credentials have a limited lifetime, so we do not have to rotate them or explicitly revoke them when they're no longer needed.
- After temporary security credentials expire, they cannot be reused (we can specify how long the credentials are valid for, up to a maximum limit).

The AWS STS API action returns temporary security credentials that consist of:

- An access key which consists of an access key ID and a secret ID.
- A session token.
- Expiration or duration of validity.
- Users (or an application that the user runs) can use these credentials to access your resources.

With STS we can request a session token using one of the following APIs:

- AssumeRole – can only be used by IAM users (can be used for MFA).
- AssumeRoleWithSAML – can be used by any user who passes a SAML authentication response that indicates authentication from a known (trusted) identity provider.
- AssumeRoleWithWebIdentity – can be used by an user who passes a web identity token that indicates authentication from a known (trusted) identity provider.
- GetSessionToken – can be used by an IAM user or AWS account root user (can be used for MFA).
- GetFederationToken – can be used by an IAM user or AWS account root user.

AWS recommends using Cognito for identity federation with Internet identity providers.

Users can come from three sources.

Federation (typically AD):

- Uses SAML 2.0.
- Grants temporary access based on the users AD credentials.
- Does not need to be a user in IAM.
- Single sign-on allows users to login to the AWS console without assigning IAM credentials.

Federation with Mobile Apps:

- Use Facebook/Amazon/Google or other OpenID providers to login.

Cross Account Access:

- Lets users from one AWS account access resources in another.
- To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.
- Or you must assume a role (identity-based policy) within that account with the permissions you need.

There are a couple of ways STS can be used.

Scenario 1:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker always authenticates with LDAP first, then with AWS STS.
3. Application then gets temporary access to AWS resources.

Scenario 2:

1. Develop an Identity Broker to communicate with LDAP and AWS STS.
2. Identity Broker authenticates with LDAP first, then gets an IAM role associated with the user.
3. Application then authenticates with STS and assumes that IAM role.
4. Application uses that IAM role to interact with the service.

IAM Best Practices:

Lock away the AWS root user access keys.

Create individual IAM users.

Use AWS defined policies to assign permissions whenever possible.

Use groups to assign permissions to IAM users.

Grant least privilege.

Use access levels to review IAM permissions.

Configure a strong password policy for users.

Enable MFA for all users.

Use roles for applications that run on AWS EC2 instances.

Delegate by using roles instead of sharing credentials.

Rotate credentials regularly.

Remove unnecessary credentials.

Use policy conditions for extra security.

Monitor activity in your AWS account.