**IAM – Identity and Access Management**

AWS IAM is a web service that helps you to **securely control access to AWS resources.** With IAM, you can **centrally manage permissions that control which AWS resources users can access.**

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

\* IAM is used for **authentication** and **Authorization** purpose. IAM makes it easy to provide multiple users secure access to AWS resources. Provides centralized control of your AWS account. Enables shared access to your AWS 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)**

**IMP:** 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**

1. **Username**
2. **Password**
3. **Permissions to access various resources (Policy)**

**IAM Features**

1. **Shared access to your AWS account:**

You can grant other people permission to administer and use resources in your AWS account without having to share your password or access key.

1. **Granular permissions**

You can apply granular permissions with IAM. You can grant different permissions to different people for different resources. For example, you might allow some users complete access to EC2, S3, RDS and EFS. For other users, you can allow read-only access to just some S3 buckets, or permission to administer just some EC2 instances, or to access your billing information but nothing else.

1. **Secure access to AWS resources for applications that run on Amazon EC2**

You can use IAM features to securely provide credentials for applications that run on EC2 instances. These credentials provide permissions for your application to access other AWS resources. Examples include S3 buckets and DynamoDB tables.

**Note-** It doesn’t mean that IAM is not used for application-level authentication.

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

1. **Identity federation**

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

1. **Multi-factor authentication (MFA)**

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.

1. **Integrated with many AWS services**
2. **IAM is eventually consistent.**

IAM achieves high availability by replicating data across multiple servers within Amazon's data centres around the world.

1. **Free to use**

AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) are features of your AWS account offered at no additional charge. You are charged only when you access other AWS services using your IAM users or AWS STS temporary security credentials.

**Some Important Point related to IAM**

* IAM is universal (global) and does not apply to regions.
* IAM replicates data across multiple data centres around the world.
* The “root account” is the account created when you 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 (One type of IAM 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.

**How to access IAM**

**AWS Management Console:**

The console is a browser-based interface to manage IAM and AWS resources. To sign-in you must provide your **account ID** or **account alias** in addition to a **user name** and **password**.

**To sign in on AWS management Console – Two ways:**

The sign-in URL includes the account ID or account alias, e.g.: https://*Account\_ID*.signin.aws.amazon.com/console/

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

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

**AWS Command Line Tools**

You can use the AWS command line tools to issue commands at your system's command line to perform IAM and AWS tasks. Using the command line can be faster and more convenient than the console. The command line tools are also useful if you want to build scripts that perform AWS tasks.

**AWS SDKs**

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, etc.).

**IAM Query API**

You can access IAM and AWS programmatically by using the IAM Query API, which lets you issue HTTPS requests directly to the service.

**IAM Elements**

**Principals:**

* A user or application that uses the AWS account- eg - **Root User**, **IAM user**, **IAM role** and **IAM Role for Application** to sign in and make requests to AWS Resources.
* An entity that can take an action on an AWS resource.
* Your administrative IAM user is your first principal.
* You can allow users and services to assume a role.
* We have already seen that IAM supports federated users. IAM supports programmatic access to allow an application to access your AWS account. **IAM users, roles, federated users, and applications** are all AWS principals.
* Hence, we can say that, principal is a person or an application that can make a **request** for an **action** or an operation on aws resources.

**Requests:**

* Principals send requests via the Console, CLI, SDKs, or APIs.
* **Requests are:**

Actions (or operations) that the principal wants to perform on resources.

**Authentication:**

* A principal sending a request must be authenticated to send a request to AWS.
* To authenticate from the console, you must sign in with your user’s name and password. To authenticate from the API or CLI, you must provide your access key and secret access key.

**Authorization:**

* IAM uses values from the request context to check for matching policies and determines whether to allow or deny the request.
* IAM policies are stored in IAM as JSON documents and specify the permissions that are allowed or denied.
* **IAM policies can be:** (Based on where is attached)
  + Identity based policies.
  + Resource-based policies.
* IAM checks each policy that matches the context of your request.
* If a single policy has a deny action IAM denies the request and stops evaluating (explicit deny).
* Evaluation logic:
  + By default, all requests are denied (implicit deny).
  + An explicit allow overrides the implicit deny.
  + An explicit deny overrides any explicit allows.
* Only the root user has access to all resources in the account by default.

**Actions:**

* Actions are defined by a service.
* Actions are the things you can do to a resource such as viewing, creating, editing, deleting.
* Any actions on resources that are not explicitly allowed are denied.
* To allow a principal to perform an action you must include the necessary actions in a policy that applies to the principal or the affected resource.

**IAM Resources:**

The user, group, role, policy, and identity provider objects that are stored in IAM. As with other AWS services, you can add, edit, and remove resources from IAM.

**Users-** An entity that you create to interact with AWS

**Groups-** A collection of users with common set of permissions.

**Roles-** You can create roles and can then assign to AWS resources.

**Policies-** The document that defined one or more permissions.

**IAM Identities**

* The IAM resource objects that are used to identify resource- these include **users, groups,** and **roles**
* You can attach a policy to an IAM identity.

**Authentication Methods**

**Console password:**

* A password that the user can enter to sign into user interface such as the AWS Management Console.
* You can allow users to change their own passwords.
* You 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:**

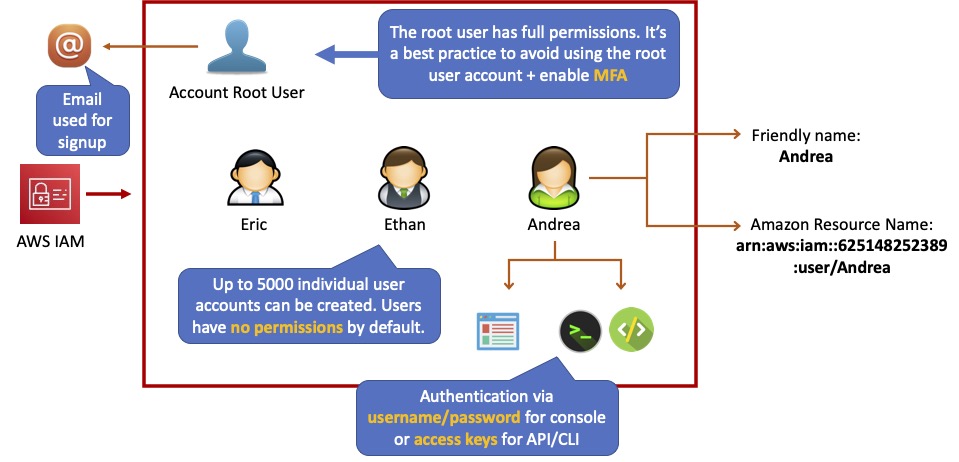
* A combination of an **access key** and a **secret access key.**
* You can assign two active access keys to a user at a time, access key once lost can not be recovered. You have to create new access key.
* These can be used to make programmatic calls to AWS when using the API in program code or at a command prompt when using the AWS CLI or the AWS PowerShell tools.
* You 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).
* You can disable a user’s access key which prevents it from being used for API calls.

**IAM User**

* 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, we have to provide access with the help of policies.
* You can allow or disallow the ability to change passwords using an IAM policy.
* Access keys and passwords should be changed regularly.
* IAM user has its permeant credentials.

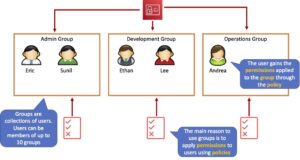
**Best practice for root user:**

* The root user has full administrative permissions, and these cannot be restricted or removed.
* 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 (Multi Factor Authentication)
* IAM users can be created to represent applications, and these are known as “service accounts”.
* You 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.
* You should create individual IAM accounts for individual 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 generated once and must be regenerated if lost.
* A password policy can be defined for enforcing password length, complexity etc. (applies to all users).



**Groups**

* Groups are collections of users and have policies attached to them.
* A group is not full identity and cannot be identified as a principal in an IAM policy.
* Use groups to assign permissions to users.
* Use the principal of least privilege when assigning permissions.
* You cannot nest groups (groups within groups), nesting of groups is not possible.



**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.
* You 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.

|  |
| --- |
| **Interview Question**  What is IAM Role?  What is difference between IAM user and IAM Role? |

**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.

**Level of Accesses**

**General accesses:**

1. Real access
2. Write access
3. Full access

**AWS Services Accesses:**

1. List
2. Read
3. Write

As per the IAM console any user can have either one, two or below three accesses:

1. **User Management Access**
2. **Services Access**
3. **Billing Dashboard Access**

While Creating IAM/Power user two more login accesses are there, as below:

**Select AWS access type:**

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from accessing the console using an assumed role. Access keys and auto generated passwords are provided in the last step.

**Select AWS credential type**

* **Access key - Programmatic access**

Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.

* **Password - AWS Management Console access**

Enables a password that allows users to sign-in with the AWS Management Console.

**Three types of users-** Root User, IAM user, Power user.

**Root User access’s:**

**All Services (Resources) access.**

**User management access.**

**Billing dashboard access.**

**Root user only access (special access)**

**IAM User:**

**All/Selected Services (Resources) access.**

**User management access.**

**Billing dashboard access. – can be given or removed**

**Power user/Super user:**

**Has by default access to all services but 'has no user management access.'** (Can’t add, remove or give permissions to any user). Can’t grant or remove any privileges. No IAM permission.

**All Services (Resources) access.**

**~~User management access.~~**

**~~Billing dashboard access.~~**

**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).

You manage access in AWS by creating policies and attaching them to IAM identities (users, groups, and roles) or AWS resources.

All permissions are implicitly denied by default and can be allowed by policies.

**There are 3 types of policies:**

1. **AWS Managed Policies.**

**AWS Managed - Job Functions.**

1. **Customer managed policies.**
2. **Inline policies.**

**Identity-based policies** – Attach [**managed**](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#managedpolicy)(AWS managed & Customer managed) and [**inline**](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#inline) policies to IAM identities (users, groups, or roles). Identity-based policies **grant permissions to an identity.** Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles).

[**Resource-based policies**](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#policies_resource-based) – **Attach inline policies to 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. Principals can be in the same account as the resource or in other accounts.

1. **AWS Managed Policy:**

* Created and administered by AWS.
* Used for common use cases based on job function.
* You don’t have to create policies yourself.
* Can be attached to multiple users, groups, or roles within and across AWS accounts.
* You cannot change the permissions defined in AWS managed policies.

**AWS Managed- Job Functions:**

* Some AWS managed policies are designed for specific job functions.
* **Administrator.**
* **Billing.**
* **Database Administrator.**
* **Data Scientist.**
* **Developer Power User.**
* **Network Administrator.**
* **Security Auditor.**
* **Support User.**
* **System Administrator.**
* **View-Only User.**

1. **Customer Managed Policy:**

* Standalone policy that you create and manage in your aws account.
* Can be attached to multiple users, groups, and roles – but only within your own account.
* Can be created by copying an existing AWS managed policy and then customizing it.
* Recommended for use cases where the existing AWS Managed Policies don’t meet the needs of your environment.
* AWS recommends that provide the least access to your Identities.

1. **Inline Policy:**

* Inline policies are embedded within the user, group, or role to which it is applied.
* Strict 1:1 relationship between the entity and the policy.
* When you delete the user, group, or role in which the inline policy is embedded, the policy will also be deleted.
* In most cases, AWS recommends using Managed Policies instead of inline policies.
* Inline policies are useful when you want to be sure that the permissions in a policy are not inadvertently assigned to any other user, group, or role.

|  |
| --- |
| **Interview Question:** what is difference between Custom Policy and inline policy? |

**IAM Policy Evaluation Logic**

* By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access).
* An explicit allow in an identity-based or resource-based policy overrides this default.
* If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
* An explicit deny in any policy overrides any allows.

**IAM permissions boundaries** – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role).

**AWS Organizations service control policies (SCPs)** – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU). Session policies – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user.

**AWS Security Token Service (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](https://sts.amazonaws.com/)
* You can optionally send your AWS STS requests to endpoints in any region (can reduce latency).
* 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, if the user requesting them still has permission to do so.
* Advantages of STS are:
* You do not have to distribute or embed long-term AWS security credentials with an application.
* You can provide access to your 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 you 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 (you 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 you 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).

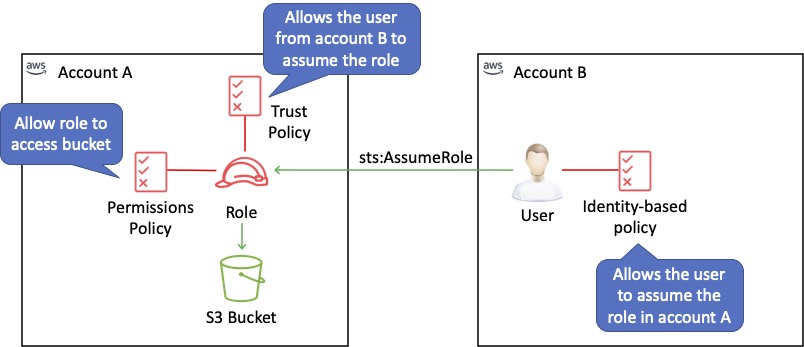
**GetFederationToken** – can be used by an IAM user or AWS account root user.

Does not need to be a user in IAM.

Single sign-on allows users to login to the AWS console without assigning IAM credentials.

**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.
* Useful for situations where an AWS customer has separate AWS account – for example for development and production resources.
* Cross Account Access makes is easier to work productively within a multi-account (or multi-role) AWS environment by making is easy to switch roles within the AWS Management Console.
* Can sign-in to the console using your IAM user name and then switch the console to manage another account without having to enter another user name and password.
* Let’s 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.



**IAM Best Practices**

* To secure AWS resources it is recommended that you follow these best practices:
* Lock away your AWS account root user access keys.
* Use roles to delegate permissions.
* Grant least privilege.
* Get started using permissions with AWS managed policies.
* Validate your policies.
* Use customer managed policies instead of inline policies.
* Use access levels to review IAM permissions.
* Configure a strong password policy for your users.
* Enable MFA.
* Use roles for applications that run on Amazon EC2 instances.
* Do not share access keys.
* Rotate credentials regularly.
* Remove unnecessary credentials.
* Use policy conditions for extra security.
* Monitor activity in your AWS account.