
AWS
IAM

❖ IDENTITY ACCESS MANAGEMENT (IAM)

- AWS IAM enables you to securely control access to AWS services and resources for your users and groups.
- Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.
- IAM is a global to all AWS Regions, creating a user account will apply to all the regions.
- Any new IAM user you create in an AWS account is created with NO access to any AWS services. This is non-explicit deny rule set on all new IAM users.
- All users beside the root user. users must be created with proper permissions
- Permissions are governed by Policies.

IAM FEATURES:

- Shared access to your AWS account
- Granular permissions
- Secure access to AWS resources for applications that run on Amazon EC2
- Multi-factor authentication (MFA)
- Identity federation
- Identity information for assurance
- Integrated with many AWS services

➢ MULTI FACTOR AUTHENTICATION (MFA):

- MFA adds an extra layer of protection on top of your user's name and password.
- When a user signs in to an AWS website, they will be prompted for their user's name & password.

NOTE:

- Each identity has its own MFA configuration.

➢ IAM PERMISSIONS & POLICIES:

- A policy is a document that formally states one or more permissions.
- If you manage a single account in AWS, then you define the permissions within that account using policies.
- If you manage permissions across multiple accounts, it is more difficult to manage permissions for your users.

- You can use IAM roles, resource-based policies, or access control lists (ACLs) for cross-account permissions.
- By default, an Explicit deny always overrides and an Explicit allow.
- IAM provides pre-built policy templates to assign to users and groups.
- You can also create custom IAM permissions policies using the policy generator or written from scratch.
- More than one policy can be attached to a user or group at the same time.
- Policies cannot be directly attached to AWS resources (such as EC2 instances)

POLICY TYPES:

- The following policy types, listed in order from most frequently used to less frequently used, are available for use in AWS.
- Policy Types are:
  - Identity-based policies
  - Resource-based policies
  - Permissions boundaries
  - Organizations SCPs
  - Access control lists (ACLs)
  - Session policies

IDENTITY-BASED POLICIES:

- These are JSON permissions policy documents that control what actions an identity (users, groups of users, and roles) can perform, on which resources, and under what conditions.
- Identity-based policies can be further categorized:
  - Managed policies
  - Inline policies

MANAGED POLICIES:

- Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account.
- There are two types of managed policies:

AWS MANAGED POLICIES:

- Managed policies that are created and managed by AWS.

CUSTOMER MANAGED POLICIES:

- These are providing more precise control over your policies than AWS managed policies.

INLINE POLICIES:

- Policies that you add directly to a single user, group, or role.
- Inline policies maintain a strict one-to-one relationship between a policy and an identity.
- They are deleted when you delete the identity.
- In most cases, we don't recommend using inline policies.

RESOURCE-BASED POLICIES:

- Resource-based policies control what actions a specified principal can perform on that resource and under what conditions.
- Resource-based policies are inline policies, and there are no managed resource-based policies.
- To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy.
- The IAM service supports only one type of resource-based policy called a role trust policy, which is attached to an IAM role.

PERMISSIONS BOUNDARIES:

- It is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity.
- When you set a permissions boundary for an 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. An explicit deny in any of these policies overrides the allow.

SERVICE CONTROL POLICIES (SCPS):

- AWS Organizations is a service for grouping and centrally managing the AWS accounts that your business owns.
- If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts.
- SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU).
- An explicit deny in any of these policies overrides the allow.

ACCESS CONTROL LISTS (ACLS):

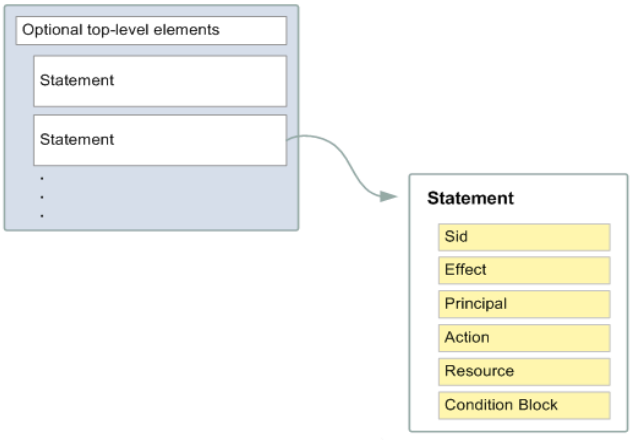
- ACLs are service policies that allow you to control which principals in another account can access a resource.
- ACLs cannot be used to control access for a principal within the same account.
- ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format.
- Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs.

SESSION POLICIES:

- Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user.
- 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. An explicit deny in any of these policies overrides the allow.

➢ OVERVIEW OF JSON POLICIES:

- Most policies are stored in AWS as JSON documents.
- It is not necessary for you to understand the JSON syntax.
- You can use the visual editor in the AWS Management Console to create and edit customer managed policies without ever using JSON.
- JSON policy document structure:



EXAMPLE OF JSON POLICY SYNTAX:

- **Identity-based policy** allows the implied principal to list a single Amazon S3 bucket named suncloud_bucket:

```
{
 "Version": "2012-10-17",
 "Statement": {
  "Effect": "Allow",
  "Action": "s3:ListBucket",
  "Resource": "arn:aws:s3:::suncloud_bucket"
 }
}
```

- **Resource-based policy** can be attached to an Amazon S3 bucket.

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Sid": "1",
  "Effect": "Allow",
  "Principal": {"AWS": ["arn:aws:iam::account-id:root"]},
  "Action": "s3:*",
  "Resource": [
   "arn:aws:s3:::mybucket",
   "arn:aws:s3:::mybucket/*"
  ]
 }]
}
```

MULTIPLE STATEMENTS AND MULTIPLE POLICIES:

- If you want to define more than one permission for an entity (user or role), you can use multiple statements in a single policy. You can also attach multiple policies.

```
    {
     "Version": "2012-10-17",
     "Statement": [
      {
       "Sid": "FirstStatement",
       "Effect": "Allow",
       "Action": ["iam:ChangePassword"],
       "Resource": "*"
```

```
    },
    {
      "Sid": "SecondStatement",
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    },
    {
      "Sid": "ThirdStatement",
      "Effect": "Allow",
      "Action": [
        "s3:List*",
        "s3:Get*"
      ],
      "Resource": [
        "arn:aws:s3:::confidential-data",
        "arn:aws:s3:::confidential-data/*"
      ],
      "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}}
    }
  ]
}
```

➢ ELEMENTS IN A STATEMENT:

- The information in a statement is contained within a series of elements.

VERSION:
Specify the version of the policy language that you want to use. As a best practice, use the latest 2012-10-17 version.

STATEMENT:
Use this main policy element as a container for the following elements. You can include more than one statement in a policy.

SID (OPTIONAL):
Include an optional statement ID to differentiate between your statements.

EFFECT:
Use Allow or Deny to indicate whether the policy allows or denies access.

PRINCIPAL (Required in only some circumstances):
If you create a resource-based policy, you must indicate the account, user, role, or federated user to which you would like to allow or deny access. If you are creating an IAM permissions policy to attach to a user or role, you cannot include this element. The principal is implied as that user or role.

ACTION:
Include a list of actions that the policy allows or denies.

RESOURCE (Required in only some circumstances):
If you create an IAM permissions policy, you must specify a list of resources to which the actions apply. If you create a resource-based policy, this element is optional. If you do not include this element, then the resource to which the action applies is the resource to which the policy is attached.

CONDITION (Optional):
Specify the circumstances under which the policy grants permission.

➢ IAM USERS & GROUPS:

IAM USERS:
- IAM users are identities in the service.
- IAM user is an entity that uses it to interact with AWS. A user consists of a name and credentials.
- IAM user with administrator permissions is not the same thing as the AWS account root user.
- If you create an IAM user, they can't access anything in your account until you give them permission. You give permissions to a user by creating an identity-based policy.

IAM GROUPS:
- An IAM group is a collection of IAM users.
- You can organize IAM users into IAM groups and attach a policy to a group.
- In that case, individual users still have their own credentials, but all the users in a group have the permissions that are attached to the group.

➢ SECURITY TOKEN SERVICE (STS):

- STS allows you to create temporary security credentials that grant trusted users access to your AWS resources.
- These temporary credentials are for short-term use as the name implies and can be active for a few minutes to several hours.
- Temporary security credentials are not stored with the user but are generated dynamically and provided to the user when requested.
- A couple of benefits when using STS:
  - Applications do not have to embed long-term security credentials in their code to have the access they need to AWS resources. (This is more secure).
  - Also, access to AWS resources can be provided to uses without having to define an AWS identity for them.

➢ IAM ROLES:

- An IAM role is an IAM entity that defines a set of permissions for making AWS service requests.
- IAM roles are not associated with a specific users or groups.
- IAM roles for EC2 instances. An EC2 instance can only have ONE role attached at a time.
- You can use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources.
- IAM roles use temporary credentials that are generated by STS which usually have a duration of up to few hours.
- When a user assumes a role, temporary security credentials are requested from STS dynamically and provided to the user.
- A role can be assumed by any of the following:
  - An IAM user in the same account as the IAM role, or in a different AWS account.
  - A web service offered by AWS such as EC2.
  - An external user (federated user) authenticated by an external identity provider service.
- IAM Roles for:
  - Amazon EC2.
  - Cross-account access

ROLES FOR AMAZON EC2:

- If you run applications on Amazon EC2 instances and those applications need access to AWS resources, you can provide temporary security credentials to your instances when you launch them.
- These temporary security credentials are available to all applications that run on the instance, so you don't need to store any long-term credentials on the instance.

ROLES FOR CROSS-ACCOUNT ACCESS:

- Many organizations maintain more than one AWS account. Using roles and cross-account access, you can define user identities in one account, and use those identities to access AWS resources in other accounts that belong to your organization.
- This is known as the **delegation approach** to temporary access.