Readings 1.7: Introduction to AWS Identity and Access Management

What Is IAM?

IAM is a web service that enables you to manage access to your AWS account and resources. It also provides a centralized view of who and what are allowed inside your AWS account (authentication), and who and what have permissions to use and work with your AWS resources (authorization).

With IAM, you can share access to an AWS account and resources without having to share your set of access keys or password. You can also provide granular access to those working in your account, so that people and services only have permissions to the resources they need. For example, to provide a user of your AWS account with read-only access to a particular AWS service, you can granularly select which actions and which resources in that service they can access.

Get to Know the IAM Features

To help control access and manage identities within your AWS account, IAM offers many features to ensure security.

- IAM is global and not specific to any one Region. This means you can see and use your IAM configurations from any Region in the AWS Management Console.
- IAM is integrated with many AWS services by default.
- You can establish password policies in IAM to specify complexity requirements and mandatory rotation periods for users.
- IAM supports MFA.
- IAM supports identity federation, which allows users who already have passwords elsewhere—for example, in your corporate network or with an internet identity provider—to get temporary access to your AWS account.
- Any AWS customer can use IAM; the service is offered at no additional charge.

What Is an IAM User?

An IAM user represents a person or service that interacts with AWS. You define the user within your AWS account. And any activity done by that user is billed to your account. Once you create a user, that user can sign in to gain access to the AWS resources inside your account.

You can also add more users to your account as needed. For example, for your cat photo application, you could create individual users in your AWS account that correspond to the people who are working on your application. Each person should have their own login credentials. Providing users with their own login credentials prevents sharing of credentials.

IAM User Credentials

An IAM user consists of a name and a set of credentials. When creating a user, you can choose to provide the user:

- Access to the AWS Management Console
- Programmatic access to the AWS Command Line Interface (AWS CLI) and AWS Application Programming Interface (AWS API)

To access the AWS Management Console, provide the users with a user name and password. For programmatic access, AWS generates a set of access keys that can be used with the AWS CLI and AWS API. IAM user credentials are considered permanent, in that they stay with the user until there's a forced rotation by admins.

When you create an IAM user, you have the option to grant permissions directly at the user level.

This can seem like a good idea if you have only one or a few users. However, as the number of users helping you build your solutions on AWS increases, it becomes more complicated to keep up with permissions. For example, if you have 3,000 users in your AWS account, administering access becomes challenging, and it's impossible to get a top-level view of who can perform what actions on which resources.

If only there were a way to group IAM users and attach permissions at the group level instead. Guess what: There is!

What Is an IAM Group?

An IAM group is a collection of users. All users in the group inherit the permissions assigned to the group. This makes it easy to give permissions to multiple users at once. It's a more convenient and scalable way of managing permissions for users in your AWS account. This is why using IAM groups is a best practice.

If you have a an application that you're trying to build and have multiple users in one account working on the application, you might decide to organize these users by job function. You might want IAM groups organized by developers, security, and admins. You would then place all of your IAM users in the respective group for their job function.

This provides a better view to see who has what permissions within your organization and an easier way to scale as new people join, leave, and change roles in your organization.

Consider the following examples.

- A new developer joins your AWS account to help with your application. You simply create a new user and add them to the developer group, without having to think about which permissions they need.
- A developer changes jobs and becomes a security engineer. Instead of editing the user's permissions directly, you can instead remove them from the old group and add them to the new group that already has the correct level of access.

Keep in mind the following features of groups.

- Groups can have many users.
- Users can belong to many groups.
- Groups cannot belong to groups.

The root user can perform all actions on all resources inside an AWS account by default. This is in contrast to creating new IAM users, new groups, or new roles. New IAM identities can perform no actions inside your AWS account by default until you explicitly grant them permission.

The way you grant permissions in IAM is by using IAM policies.

What Is an IAM Policy?

To manage access and provide permissions to AWS services and resources, you create IAM policies and attach them to IAM users, groups, and roles. Whenever a user or role makes a request, AWS evaluates the policies associated with them. For example, if you have a developer inside the developers group who makes a request to an AWS service, AWS evaluates any policies attached to the developers group and any policies attached to the developer user to determine if the request should be allowed or denied.

IAM Policy Examples

Most policies are stored in AWS as JSON documents with several policy elements. Take a look at the following example of what providing admin access through an IAM identity-based policy looks like.

```
{
"Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "*",
        "Resource": "*"
     }]
}
```

In this policy, there are four major JSON elements: Version, Effect, Action, and Resource.

- The **Version** element defines the version of the policy language. It specifies the language syntax rules that are needed by AWS to process a policy. To use all the available policy features, include `"Version": "2012-10-17"` before the `"Statement"` element in all your policies.
- The **Effect** element specifies whether the statement will allow or deny access. In this policy, the Effect is `"Allow"`, which means you're providing access to a particular resource.
- The **Action** element describes the type of action that should be allowed or denied. In the above policy, the action is `"*"`. This is called a wildcard, and it is used to symbolize every action inside your AWS account.
- The **Resource** element specifies the object or objects that the policy statement covers. In the policy example above, the resource is also the wildcard `"*"`. This represents all resources inside your AWS console.

Putting all this information together, you have a policy that **allows** you to perform **all actions** on **all resources** inside your AWS account. This is what we refer to as an *administrator policy*.

Let's look at another example of a more granular IAM policy.

```
{
"Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
```

```
            "iam: ChangePassword",
            "iam: GetUser"
            ]
        "Resource": "arn:aws:iam::123456789012:user/${aws:username}"
    }]
```

}After looking at the JSON, you can see that this policy **allows** the IAM user to **change their own IAM password** (`iam:ChangePassword`) and **get information about their own user** (`iam:GetUser`). It only permits them to access their own credentials because the resource restricts access with the variable substitution `${aws:username}`.

Understand Policy Structure

When creating a policy, it is required to have each of the following elements inside a policy statement.

Element	Description	Required	Example
Effect	Specifies whether the statement results in an allow or an explicit deny	✓	`"Effect": "Deny"`
Action	Describes the specific actions that will be allowed or denied	✓	`"Action": "iam:CreateUser"`
Resource	Specifies the object or objects that the statement covers	✓	`"Resource": "arn:aws:iam::account-ID-without-hyphens:user/Bob"`

Resources

- *External Site: What is IAM?*
- *External Site: AWS IAM Identities*
- *External Site: Access Management with AWS IAM*