AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS services and resources for your users. Using IAM, you can:

- **Manage Users and Their Permissions:** Create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.
- **Define Fine-Grained Access Controls:** Specify who can access which resources under what conditions.
- **Secure Your AWS Environment:** Implement security best practices like multi-factor authentication (MFA).
- **Global Service** (IAM entities like roles can be used in any region without recreation)
- **IAM Query API** can be used to make direct calls to the IAM web service (using access key ID and secret access key for authentication)
- By default, IAM users **do not have access to the AWS Billing and Cost Management console**.

AWS Identity and Access Management (IAM)

Core IAM Components

Understanding the core components of IAM is essential. Here's a breakdown:

a. Users

- **Definition:** An IAM user represents a person or service that interacts with AWS resources.
- **Features:**
    - Each user has unique credentials (password, access keys).
    - Users can be assigned permissions directly or via groups.

b. Groups

- **Definition:** A collection of IAM users.
- **Features:**
    - Simplifies permission management by assigning permissions to groups instead of individual users.
    - Users inherit permissions assigned to their groups.

c. Roles

- **Definition:** An IAM role is an entity with permissions that determine what actions can be performed and what resources can be accessed.
- **Features:**
    - **Assumed by Trusted Entities:** Can be assumed by users, applications, or services (like EC2 instances).
    - **Temporary Credentials:** Roles provide temporary security credentials, enhancing security.

AWS Identity and Access Management (IAM)

d. Policies

- **Definition**: Policies are JSON documents that define permissions.
- **Types**:
    - **Managed Policies**: Predefined by AWS or created by you. They can be attached to multiple users, groups, or roles.
    - **Inline Policies**: Policies embedded directly into a single user, group, or role.

e. Permissions

- **Definition**: Permissions determine what actions are allowed or denied on specific AWS resources.
- **Components**:
    - **Actions**: Operations like s3:PutObject, ec2:StartInstances.
    - **Resources**: AWS resources like S3 buckets, EC2 instances.
    - **Effect**: Whether to allow or deny the specified actions.

Policies contain permissions — not the other way around. A **policy** is essentially a document that defines the **permissions** (allowed or denied actions) for a user, group, or role.

Example of this relationship:

- **Policy**: A JSON document that specifies:
    - **Actions** (e.g., s3:PutObject, ec2:StartInstances)
    - **Resources** (e.g., a specific S3 bucket or EC2 instance)
    - **Effect** (Allow or Deny)
- **Permissions**: The actual actions specified inside the policy that define what the user, group, or role can do or cannot do.

AWS Identity and Access Management (IAM)

These are 06 main core components of the IAM, there are other few key terms, you might need to understand & keep in mind

Access Key

- Access keys are used for programmatic access to AWS resources via the AWS CLI, SDKs, or APIs. They consist of an **access key ID** and a **secret access key**.

Multi-Factor Authentication (MFA)

- MFA adds an additional layer of security by requiring not just a password, but also a second authentication method, such as a code sent to your phone.

Root User

- The **root user** is the account owner with complete access to all AWS services and resources. It's recommended to use the root user sparingly and enable MFA for added security.

Principal

- A **principal** is an entity (such as an IAM user, role, or service) that is making a request in AWS.

Service-Linked Role

- A **service-linked role** is a type of IAM role that is predefined by AWS and is linked to an AWS service to enable the service to access other AWS resources on your behalf.

Trust Policy

- A **trust policy** defines which entities can assume a role. It's used to specify who can "trust" a role to assume it.

Identity Federation

- **Federation** allows you to use external identities (like Google or Facebook) to access AWS resources via temporary credentials.

AssumeRole

- The **AssumeRole** API operation allows you to assume a role and receive temporary security credentials with the permissions of the role.

Please read: Criss Cross relationship between different entities of IAM

https://lisireddy.medium.com/aws-iam-criss-cross-relationships-71abc6686376

AWS Identity and Access Management (IAM)

Now we will dig into the Core Components more & more until we reach the last mile of understanding.

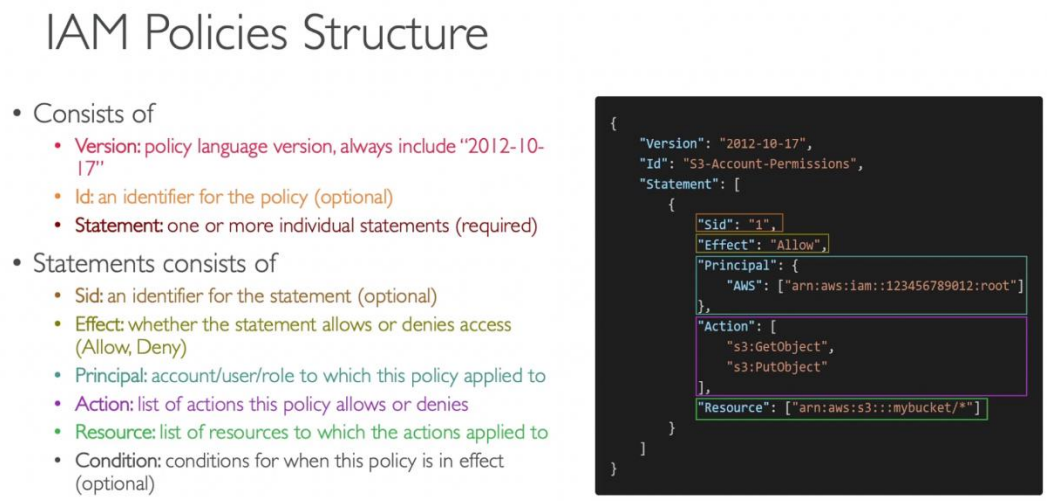
Policy & Permissions relationship, you have learnt in the previous page -- Now what are the types we will see

Types of Policies in AWS IAM:

1. **Managed Policies:**
    - **AWS Managed Policies:**
        - These are pre-built policies created and maintained by AWS. They are designed to provide common sets of permissions (e.g., AmazonS3ReadOnlyAccess).
        - Maintained by AWS (updated as new APIs are added)
        - Good for administrators
    - **Customer Managed Policies:**
        - Policies created and managed by you (the AWS account owner). These policies can be attached to multiple users, groups, or roles.
        - Created and managed by us
        - Best practice (allow us to have fine-grained access control)
        - Version controlled (can rollback)
2. **Inline Policies:**
    - Inline policies are policies embedded directly into a specific IAM user, group, or role. These are tightly coupled with the entity they are attached to and are not reusable.
    - Policy directly applied to an IAM principal (tied to a single principal)
    - 1 - 1 relationship, cannot be reused
    - Max 2KB in size (cannot specify a lot of permissions)
    - No versioning
    - Policy is deleted if the IAM principal is deleted

3. **Service-Linked Policies:**
    - These are automatically created by AWS services when you use them (e.g., Amazon RDS, Lambda). They are attached to IAM roles to grant the service permissions to access other AWS resources on your behalf.
4. **Resource-Based Policies:**
    - These policies are attached directly to AWS resources like S3 buckets, or KMS keys, defining who can access the resource and how.
    - Example: An S3 bucket policy that allows a specific user or AWS account to access the bucket.
5. **Permissions Boundaries:**
    - These are advanced policies that set a limit on the maximum permissions a user or role can have. Even if an entity has more permissions attached via other policies, it cannot exceed the boundary set by the permissions boundary.
6. **Organization Service Control Policies (SCPs):**
    - Used within AWS Organizations to control what actions and services accounts within an organization can use. SCPs act as a filter, allowing or denying actions across all accounts in the organization.
7. **Access Control Lists (ACLs):**
    - ACLs are used to manage access to certain resources like S3 buckets and objects. They define simple access permissions (read/write) for other AWS accounts or users.

Structure



AWS Identity and Access Management (IAM)

To be Crystal Clear – Main 3 points about Policies to keep in mind

Access Policy:

Purpose: Defines what actions an IAM entity (like a user, group, or role) can perform on specific AWS resources.
Attached To: Usually attached to IAM entities such as users, groups, or roles.
Controls: Determines the permissions granted to the IAM entity. For example, a policy attached to a role might allow or deny access to certain AWS services or actions.

Resource-Based Policy:

Purpose: Defines what actions can be performed on a specific AWS resource and by whom. It controls access to the resource itself.
Attached To: Attached directly to AWS resources such as S3 buckets, Lambda functions, SQS queues, etc.
Controls: Specifies who (principals) can access the resource and what actions they can perform. This is useful for granting permissions to entities outside of your AWS account.

Trust Policy:

Purpose: Defines **who** (entities) can assume an IAM role. It specifies which AWS services, users, or external identities are trusted to assume the role.
Attached To: Attached directly to the IAM role.
Controls: Determines who is allowed to assume the role and under what conditions. This is used to establish trust relationships between the role and the entities or services that need to assume it.

Types of Permissions in AWS IAM:

Allow Permissions:

These grant users, groups, or roles the ability to perform specific actions on AWS resources. Most policies use **Allow** to specify what actions are permitted.

Deny Permissions:

These explicitly deny access to certain actions or resources, regardless of any **Allow** permissions. If there's both an **Allow** and a **Deny**, the **Deny** takes precedence.

Implicit Deny:

By default, all actions are denied until explicitly allowed. If a user is not given permission for a specific action, they are implicitly denied access.

Explicit Allow:

An action that is explicitly allowed by a policy attached to a user, group, or role.

Explicit Deny:

An explicit deny will always override any other allow permissions. This is useful when you want to ensure that certain actions or access are strictly blocked.

AWS Identity and Access Management (IAM)

What is the difference between IAM User & Roles

Quick Analogy:

- **User:** Think of it as a person with an ID badge that lets them into specific rooms all the time.
- **Role:** Think of it as a guest pass that someone borrows temporarily to access certain rooms, but only for a limited time.

IAM User:

- **Permanent Identity**: A user is a permanent identity created for a specific individual or service.
- **Credentials**: Users have **long-term credentials** (username, password, access keys) to access AWS.
- **Direct Permissions**: Permissions are assigned to the user or through groups, and they remain consistent unless manually changed.
- **Use Case**: Typically used for people (like employees or developers) who need ongoing access to AWS.

IAM Role:

- **Temporary Identity**: A role is a temporary identity that AWS services or users can assume to perform specific tasks.
- **No Credentials**: Roles don't have long-term credentials like users. Instead, they provide **temporary security credentials** when assumed.
- **Assume for Specific Tasks**: Users, services, or other AWS accounts can temporarily assume a role to gain certain permissions.
- **Use Case**: Used for granting temporary access to resources or when AWS services (like EC2, Lambda) need to interact with other AWS services.

Keep in mind

Groups are formed by a set of users, not by a set of roles. An **IAM Group** is simply a collection of IAM users, and it makes it easier to manage permissions for multiple users at once.

- You can attach **policies** to a group, and all users within the group inherit those permissions.
- Roles, on the other hand, cannot be members of groups.

AWS Identity and Access Management (IAM)

Before we move on – we need to understand one more angle into the AWS IAM Role,

Most important aspect of the Role is **Trust Relationship** or **Assume Role** Functionality

Trust relationships in IAM are crucial for defining **who** or **what** can assume an IAM role. **They essentially specify the trusted entities that are allowed to use the role**.

Why We Need Trust Relationships:

- **Delegation**: They allow you to delegate permissions to entities outside your AWS account or to AWS services within your account.
- **Security**: By explicitly defining who can assume a role, you prevent unauthorized entities from accessing resources.
- **Temporary Access**: They provide a way to grant temporary permissions without having to share long-term credentials.

This is how a trust policy looks like

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
        "Effect": "Allow",
        "Principal": {
            "Service": "ec2.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
}
```

In AWS IAM, roles are not directly attached to IAM users or IAM groups. Instead, roles are assumed by users or services, and this process is different from how permissions are granted through direct attachment of policies to users or groups.

Only users and services can assume a role (not groups).

A new IAM user created using the AWS CLI or AWS API has no AWS credentials.

IAM Groups cannot be identified as principal in an IAM policy.

AWS Identity and Access Management (IAM)

You did understand the main core components of the AWS IAM and How they related to each other …!! Now we need to understand few more things

Policy Simulator

- Online tool that allows us to check what API calls an IAM User, Group or Role is allowed to perform based on the permissions they have.

Permission Boundaries

- Set the maximum permissions an IAM entity can get
- Can be applied to users and roles (not groups)
- Used to ensure some users can't escalate their privileges (make themselves admin)

Policy Evaluation

Read : https://lisireddy.medium.com/union-of-iam-policies-and-resource-based-policies-policy-evaluation-logic-90b1c6b1465c

- If there's an explicit deny on an action, the final decision after evaluating the Policy will be deny even if another policy allows it.
- The union of IAM Policies and Resource-based Policies make up the Total Policy. Example: even if we remove the IAM policy of a user that was allowing it access to an S3 bucket but the bucket policy allows it, the final decision will be allow.

Dynamic Policies

- Leverage variables in the Policy document
- Example: an IAM policy to grant users access to their own home folders in S3

```
{
   "Sid": "AllowAllS3ActionsInUserFolder",
   "Action":["s3:*"],
   "Effect":"Allow",
   "Resource": ["arn:aws:s3:::my-company/home/${aws:username}/*"]
}
```

Passing a Role to a Service

To pass a Role to an AWS service, to assume, requires **iam:PassRole** permission for the user. The user should also have **iam:GetRole** permission to view the role being passed.

Example: Passing a role to EC2 instances to allow access to S3. The EC2 instance can assume that role and perform the required action. The below policy allows the principal to pass S3Access role to any service that can assume that role.

```
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::123456789012:role/S3Access"
}
```

Roles can only be passed to those services that are allowed to assume that role (specified in the role's Trust Policy)

IAM Principals

IAM Principals

- Any principal

  `"Principal": "*"` or `"Principal": { "AWS": "*" }`

- Any user or role in the account

  `"Principal": { "AWS": "123456789012" }` or
  `"Principal": { "AWS": "arn:aws:iam::123456789012:root" }`

- An IAM User

  `"Principal": { "AWS": "arn:aws:iam::123456789012:user/user-name" }`

- An IAM Role

  `"Principal": { "AWS": "arn:aws:iam::123456789012:role/role-name" }`

- An assumed role

  `"Principal": { "AWS": "arn:aws:sts::123456789012:assumed-role/role-name/role-session-name" }`

- Federated identity

  `"Principal": { "Federated": "cognito-identity.amazonaws.com" }`
  `"Principal": { "Federated": "arn:aws:iam::123456789012:saml-provider/provider-name" }`

- AWS Services

```
"Principal": {
  "Service": [
    "ecs.amazonaws.com",
    "elasticloadbalancing.amazonaws.com"
  ]
}
```

Cross-account Access

To grant access to a resource in account A to users in account B:

1. The account A administrator creates an IAM role and attaches a permissions policy that grants permissions on resources in account A to the role.
2. The account A administrator attaches a trust policy to the role that identifies account B as the principal who can assume the role.
3. The account B administrator delegates the permission to assume the role to any users in account B.

AWS Identity and Access Management (IAM)

Protect IAM Accounts

- **Password Policy**
  - Used to enforce standards for password
    - password rotation
    - password reuse
  - Prevents **brute force** attack
- **Multi Factor Authentication (MFA)**
  - Both root user and IAM users should use MFA

Access Keys

- Need to use access keys for AWS CLI and SDK
- Don't share access keys with anyone (every user can generate their own access keys)
- Access keys are only shown once and if you lose them you need to generate a new access key
- Access Key ID ~ username
- Secret Access Key ~ password
- If access keys are compromised, invalidate the access keys by deleting them.

Reporting Tools

Credentials Report

- lists all the users and the status of their credentials (MFA, password rotation, etc.)
- **account level** - used to audit security for all the users

Access Advisor

- shows the service permissions granted to a user and when those services were last accessed
- **user-level**
- used to revise policies for a specific user

AWS Identity and Access Management (IAM)

Best Practices of IAM

- Use root account only for account setup
- 1 physical user = 1 IAM user
- Enforce MFA for both root and IAM users
- Never share IAM credentials & Access Keys
- Use groups to assign permissions to users
- Use standalone policies instead of inline policies
- Delete (don’t generate) access keys for the root user
- Rotate access keys periodically
- Use Temporary Security Credentials (IAM Roles) instead of long-term access keys
- Don't embed access keys directly into code
- Use different access keys for different applications
- Delete unused or compromised access keys
- The root account should only be accessible by one admin user with MFA

AWS IAM Scenario based Questions

https://lisireddy.medium.com/aws-iam-scenario-based-questions-ef01ad3ff896

Happy Learning ..!!

Love Reddy