Author: Madhurima Rawat

Cloud Security: Identity and Access Management (IAM)

This experiment focuses on managing users, permissions, and security policies in the cloud using AWS Identity and Access Management (IAM). It provides hands-on experience in configuring access controls, defining security policies, and ensuring proper authorization within cloud environments.

The setup utilizes Docker, LocalStack, and AWS CLI to simulate IAM functionalities, enabling the creation and management of users, roles, and policies in a local cloud-like environment.

Identity and Access Management (IAM)

Identity and Access Management (IAM) is a framework that ensures the right individuals and entities have appropriate access to technology resources. In AWS IAM, users can securely control access to AWS services and resources.

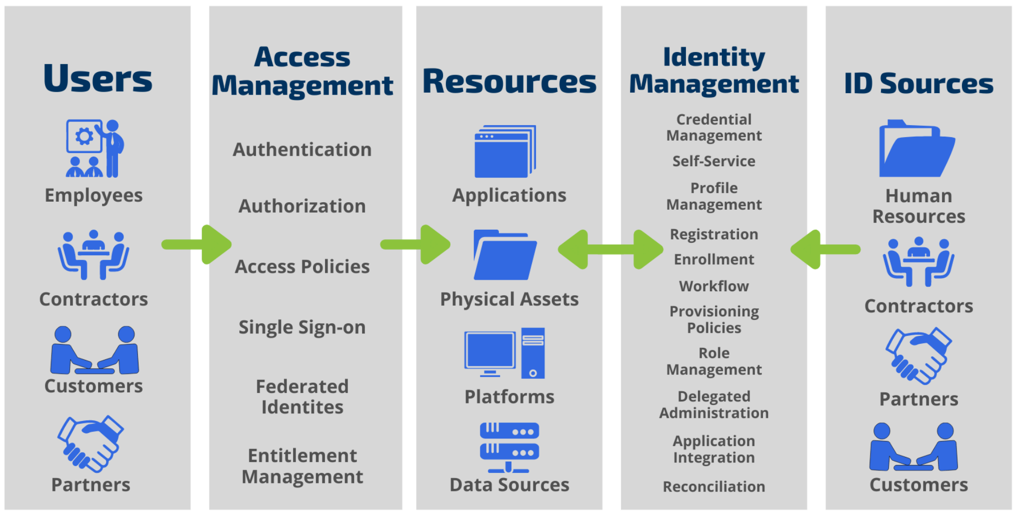
Key Concepts of IAM

1. Identities in IAM

IAM identities define who can access AWS resources and what actions they can perform. These include:

- **Users** – Individual accounts with login credentials.
- **Groups** – Collections of users sharing permissions.
- **Roles** – Assigned temporary permissions for AWS services or external users.

Example: A developer needs access to an S3 bucket. Instead of assigning permissions directly, they are added to the "Developers" IAM group, which has the required permissions.



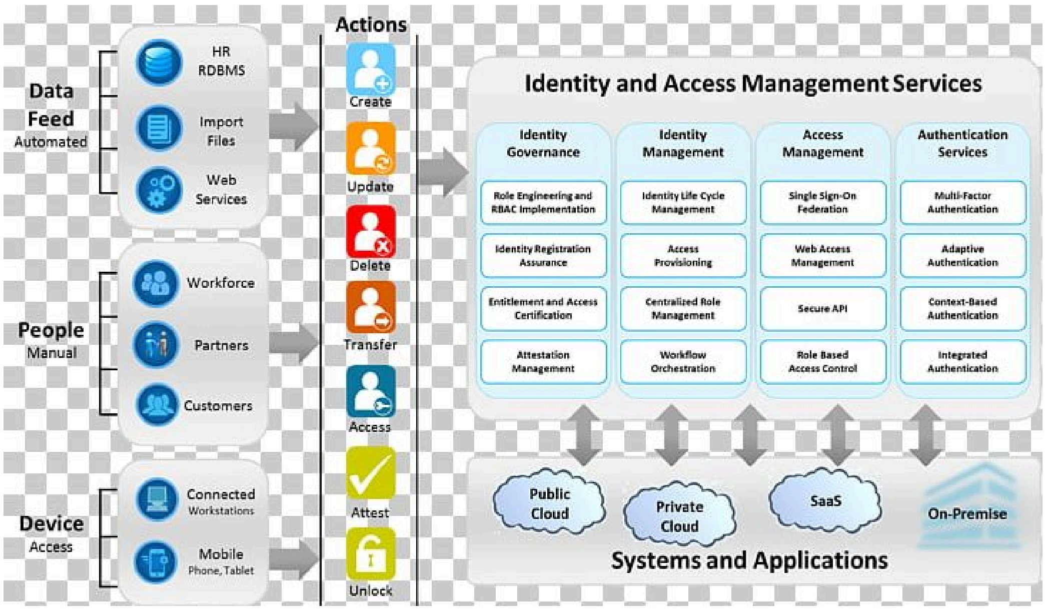
2. IAM Policies

IAM policies define permissions using JSON-based policy documents specifying what actions are allowed or denied on AWS resources.

Example Policy (Allow full S3 access to a user):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*"
    }
  ]
}
```

This grants full S3 permissions to the user.



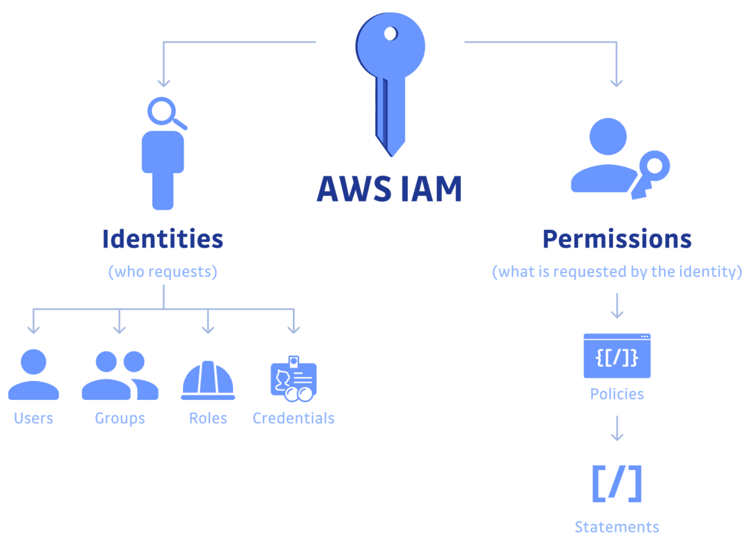
3. IAM Roles and Temporary Credentials

IAM roles allow users or AWS services to assume temporary permissions to interact with resources securely.

Example Use Cases:

- **EC2 Instance Roles** – Instead of embedding credentials in an EC2 instance, an IAM role is assigned with the required permissions.

- **Cross-Account Access** – A company allows external contractors to access AWS services via IAM roles without sharing credentials.



4. Multi-Factor Authentication (MFA)

IAM supports MFA, requiring an additional authentication step beyond a username and password.

Example Use Case:

A finance team member accessing sensitive billing data in AWS must verify using Google Authenticator before accessing the console.

5. IAM Best Practices

- **Follow Least Privilege Principle** – Users and roles should only have the minimum permissions needed.
- **Enable MFA for All Users** – Adds an extra security layer.
- **Use IAM Roles for AWS Services** – Avoid storing credentials in applications.
- **Rotate Access Keys Regularly** – If access keys are compromised, rotation prevents misuse.

Real-World IAM Use Cases

1. Enterprise-Level Role-Based Access Control (RBAC)

A large organization has departments like HR, Development, and Finance. Each department has specific access needs:

- **HR Team:** Can access employee databases but not development resources.
- **Developers:** Can deploy applications but cannot modify billing information.
- **Finance:** Can view billing dashboards but cannot access production environments.

IAM ensures proper segregation of duties.

2. IAM in Cloud Security & Compliance

A financial institution must comply with **PCI-DSS (Payment Card Industry Data Security Standard)**:

- IAM enforces policies ensuring only authorized personnel access cardholder data.
- MFA is required for accessing critical systems.
- Audit Logs via AWS CloudTrail help track unauthorized access attempts.

Case Study: IAM in a Tech Startup

Company: CloudTech Solutions

Problem: Developers needed access to AWS services, but granting full access posed security risks.

Solution: The company implemented IAM roles and groups:

- **Developers**: Given read-only access to production and write access to development environments.
- **Admins**: Given full AWS access with MFA enabled.
- **Automated IAM Policy Auditing** detected and removed unused permissions.

Outcome:

- Reduced security risks.
- Improved compliance with security best practices.
- Simplified access management.

Conclusion

IAM plays a critical role in cloud security by managing users, permissions, and authentication. By leveraging IAM policies, roles, MFA, and best practices, organizations can maintain **secure, scalable, and efficient access control** for their cloud environments.

Identity and Access Management (IAM) with Localstack

Explanation of AWS IAM Commands with Output (Using LocalStack)

These commands use the **AWS CLI** to manage IAM users, roles, and policies. The `--endpoint-url=%AWS_ENDPOINT_URL%` parameter is used to run the commands on **LocalStack**, which emulates AWS services locally.

1. Create a User

```
aws iam create-user --user-name test-user
--endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Creates a new IAM user named `test-user`.
- This user can be assigned roles and policies later.

Expected Output:

```
{
  "User": {
    "Path": "/",
    "UserName": "test-user",
    "UserId": "AIDAEXAMPLEID",
    "Arn": "arn:aws:iam::123456789012:user/test-user",
    "CreateDate": "2024-03-06T12:00:00Z"
  }
}
```

2. Create a Role

```
aws iam create-role --role-name ec2-role --assume-role-policy-
document file://trust-policy.json --endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Creates an IAM role named `ec2-role`.
- The role uses a **trust policy** (stored in `trust-policy.json`) to define which AWS services or users can assume this role.

Explanation of the Trust Policy Document

This **Trust Policy** is a JSON document that defines **who** (which AWS service, user, or account) is allowed to **assume** an IAM role. This is particularly useful when granting permissions to AWS services like EC2, Lambda, or other entities to access AWS resources.

Breakdown of the JSON Policy

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

1. **`"Version": "2012-10-17"`**

- Specifies the version of the IAM policy language being used.
- `"2012-10-17"` is the latest and recommended version for AWS policies.

2. **`"Statement": [...]`**

- Contains one or more **statements** defining the permission rules.

Inside the `Statement` Block:

`"Effect": "Allow"`

- Defines whether the policy **allows** or **denies** the action.
- `"Allow"` means this policy **grants** permissions.

`"Principal": { "Service": "ec2.amazonaws.com" }`

- Specifies **who** is allowed to assume this role.
- `"Service": "ec2.amazonaws.com"` means that the **EC2 service** (Amazon Elastic Compute Cloud) is allowed to assume this role.

`"Action": "sts:AssumeRole"`

- Specifies the action that is permitted.
- `"sts:AssumeRole"` allows the **EC2 service** to assume this role using the AWS **Security Token Service (STS)**.
- This is required when an EC2 instance needs temporary credentials to interact with other AWS services **on behalf of the IAM role**.

Use Case Example

- When you create an **IAM Role** (e.g., `ec2-role`) with this trust policy, **EC2 instances** can assume this role.
- This is useful when an EC2 instance needs access to **S3, DynamoDB, Lambda, etc.**, without using long-term credentials.
- AWS **securely provides temporary credentials** to the EC2 instance through this role.

How This Works in Practice

1. Create an **IAM Role** with this trust policy.
2. Attach **permissions** to the role (e.g., S3 Read Access).
3. Assign the **IAM Role** to an **EC2 instance** when launching it.
4. The EC2 instance automatically receives **temporary credentials** and can access AWS resources as defined in the role's permissions.

Expected Output:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "ec2-role",
    "RoleId": "AROAXAMPLEID",
    "Arn": "arn:aws:iam::123456789012:role/ec2-role",
    "CreateDate": "2024-03-06T12:05:00Z",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "ec2.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

3. Attach a Policy to the User

```
aws iam attach-user-policy --user-name test-user
--policy-arn arn:aws:iam::aws:policy/AdministratorAccess
--endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Attaches the `AdministratorAccess` policy to `test-user`, giving full access to AWS resources.

Expected Output:
No direct output. The policy is attached successfully.

To verify, use the `list-attached-user-policies` command.

4. List Users

```
aws iam list-users --endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Lists all IAM users in the account.

Expected Output:

```
{
  "Users": [
    {
      "Path": "/",
      "UserName": "test-user",
      "UserId": "AIDAEXAMPLEID",
      "Arn": "arn:aws:iam::123456789012:user/test-user",
      "CreateDate": "2024-03-06T12:00:00Z"
    }
  ]
}
```

5. List Roles

```
aws iam list-roles --endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Lists all IAM roles in the account.

Expected Output:

```
{
  "Roles": [
    {
      "Path": "/",
      "RoleName": "ec2-role",
      "RoleId": "AROAXAMPLEID",
      "Arn": "arn:aws:iam::123456789012:role/ec2-role",
      "CreateDate": "2024-03-06T12:05:00Z"
    }
  ]
}
```

6. List Attached Policies for a User

```
aws iam list-attached-user-policies --user-name test-user
--endpoint-url=%AWS_ENDPOINT_URL%
```

What it does:

- Displays the policies attached to `test-user`.

Expected Output:

```
{
  "AttachedPolicies": [
    {
      "PolicyName": "AdministratorAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
    }
  ]
}
```

These commands help manage IAM users, roles, and permissions efficiently. Using **LocalStack**, this can be tested in a local environment before deploying in AWS.