**AWS Identity and Access Management (IAM) - Detailed Notes**

**Table of Contents**

1. **Introduction to AWS IAM**
   * Purpose of IAM
   * Why IAM is Required
2. **Key Components of IAM**
   * Users
   * Roles
   * Policies
   * User Groups
3. **IAM Hierarchy & Workflow**
4. **Practical Implementation**
   * Creating an IAM User
   * Creating and Attaching Policies
   * Testing Permissions
   * Inline Policies
   * Permission Boundaries
5. **Real-World Use Case**
6. **Conclusion & Best Practices**

**1. Introduction to AWS IAM**

**Purpose of IAM**

AWS Identity and Access Management (IAM) is a service that helps securely control access to AWS resources. It allows administrators to:

* **Manage users and their permissions** (who can access what).
* **Define roles** for AWS services or external identities.
* **Set policies** to restrict or grant access to specific AWS services.

**Why IAM is Required?**

* **Security:** Prevents unauthorized access to AWS resources.
* **Granular Control:** Allows assigning specific permissions (e.g., read-only, full access).
* **Audit & Compliance:** Tracks user activities via AWS CloudTrail.

**Example Scenario:**  
A startup CEO (RMA) hires:

* A **developer** (needs access to EC2, Lambda).
* A **QA engineer** (needs access to testing tools).
* A **salesperson** (needs billing & analytics access).

Instead of giving full AWS access, IAM ensures each role gets only the necessary permissions.

**2. Key Components of IAM**

**A. Users**

* Represents an individual (e.g., developer, QA engineer).
* Each user has a unique **username & password** (or access keys).
* **Best Practice:** Avoid using the root account; create IAM users instead.

**B. Roles**

* Defines **what actions are allowed** on which AWS resources.
* Used by:
  + **AWS services** (e.g., Lambda assuming a role).
  + **Federated users** (e.g., Single Sign-On from Google).
* **Example:** A role allowing EC2 instances to access S3.

**C. Policies**

* JSON documents that define **permissions**.
* Two types:
  1. **Managed Policies** (AWS pre-defined or custom).
  2. **Inline Policies** (Directly attached to a user/role).
* **Example Policy:** Allows DescribeInstances on EC2.

**D. User Groups**

* A collection of IAM users (e.g., "Developers," "QA Team").
* Simplifies permission management (attach policies to groups instead of individual users).

**3. IAM Hierarchy & Workflow**

1. **Create a User** (e.g., test-user-1).
2. **Assign a Role** (defines allowed actions).
3. **Attach Policies** (grant permissions to specific services).
4. **Add User to a Group** (for bulk permissions).

**Example:**

* **User:** developer-1
* **Role:** EC2-Developer
* **Policy:** AmazonEC2ReadOnlyAccess
* **Group:** Dev-Team

**4. Practical Implementation**

**Step 1: Creating an IAM User**

1. Go to **IAM Console** → **Users** → **Create User**.
2. Enter **Username** (test-user-1).
3. Select **"Custom Password"** and uncheck **"Require password reset"** (optional).
4. Click **Next** → **Create User**.

**Step 2: Creating and Attaching Policies**

**A. Policy for EC2 DescribeInstances**

1. Go to **Policies** → **Create Policy**.
2. Use **JSON Editor**:

json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "ec2:DescribeInstances",

"Resource": "\*"

}

]

}

1. Name: test-user-policy → **Create Policy**.
2. Attach to test-user-1.

**B. Policy for EC2 Start/Stop with Tag-Based Restriction**

json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ec2:StartInstances",

"ec2:StopInstances"

],

"Resource": "arn:aws:ec2:REGION:ACCOUNT-ID:instance/\*",

"Condition": {

"StringEquals": {

"aws:ResourceTag/owner": "test-user-1"

}

}

}

]

}

* Replace REGION & ACCOUNT-ID.

**C. Policy for Route 53 Access**

json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "route53:ListHostedZones",

"Resource": "\*"

}

]

}

**Step 3: Testing Permissions**

* Log in as test-user-1.
* Verify:
  + Can **view EC2 instances** but not modify (unless policy allows).
  + Can **list Route 53 hosted zones** (if policy attached).

**Step 4: Inline Policies**

* Instead of creating separate policies, attach directly to a user:
  1. Go to **User** → **Add Inline Policy**.
  2. Paste JSON policy → **Review & Apply**.

**Step 5: Permission Boundaries**

* Restricts **maximum permissions** a user can have.
* **Example:**
  + User has **AdministratorAccess** but is restricted via boundary.
  + Boundary Policy:

json

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ec2:Describe\*",

"s3:List\*"

],

"Resource": "\*"

}

]

}

* **Result:** User cannot delete resources despite having admin rights.

**5. Real-World Use Case**

**Scenario:** A company has:

* **Developers** → Need EC2 & Lambda access.
* **Finance Team** → Needs billing reports (Cost Explorer).
* **DevOps Team** → Needs full AWS access but restricted to production.

**IAM Solution:**

1. **Groups:** Dev-Group, Finance-Group, DevOps-Group.
2. **Policies:**
   * Dev-Group → AmazonEC2FullAccess, AWSLambda\_FullAccess.
   * Finance-Group → AWSBillingReadOnlyAccess.
   * DevOps-Group → AdministratorAccess with **permission boundary** restricting production changes.

**6. Conclusion & Best Practices**

* **Least Privilege Principle:** Grant only necessary permissions.
* **Use Groups:** Easier to manage than individual users.
* **Enable MFA:** Adds an extra security layer.
* **Audit with CloudTrail:** Monitor IAM activities.
* **Avoid Root Account Usage:** Use IAM users instead.

**Final Note:** Always test policies in a non-production environment before applying them.

**End of Notes** 🚀