🔰 Part 1: IAM Basic Hands-on Implementation

✅ 1. Create an IAM User with Console Access

🔹 **Steps:**

1. Sign in to AWS Console as **root** or an **admin IAM user**.
2. Go to **IAM > Users > Add users**.
3. Enter username: devuser.
4. **Select AWS Management Console access**.
5. Choose **Custom password** and uncheck "Require password reset".
6. Click **Next**.

🔹 **Permissions:**

1. Select **"Attach policies directly"**.
2. Search and attach **AmazonS3ReadOnlyAccess**.
3. Click **Next > Create user**.

🔹 **Test:**

1. Sign in using the IAM user URL (provided on creation).
2. Try to access **S3** and verify **read-only access**.

✅ 2. Create a Group and Assign Policies

🔹 **Steps:**

1. IAM > **User groups > Create group**.
2. Name: DevGroup.
3. Attach policy: AmazonEC2ReadOnlyAccess.
4. Add user devuser to the group.

🔹 **Test:**

- Now devuser has both S3 read-only (direct) and EC2 read-only (via group).

✅ 3. Create an IAM Policy (Custom)

🔹 Steps:

1. IAM > **Policies > Create policy**.
2. Choose **JSON**, paste:

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

3. Click **Next > Name**: DynamoDBListTablesPolicy.
4. Create the policy.
5. Attach it to devuser.

🔹 Test:

- Login as devuser and use **DynamoDB Console** to verify ListTables works, but table creation fails.

✅ 4. Enable MFA (Multi-Factor Authentication)

🔹 Steps:

1. IAM > **Users > devuser > Security credentials**.
2. Click **Assign MFA device**.
3. Choose **Virtual MFA device** (e.g., Google Authenticator).

4. Scan QR code and enter 2 consecutive codes.
5. Complete setup.

🚀 Part 2: IAM Advanced Hands-on Implementation

✅ 5. Create an IAM Role for EC2 Access to S3

🔹 **Steps:**

1. IAM > **Roles > Create role**.
2. Select **AWS service > EC2**.
3. Permissions: AmazonS3FullAccess.
4. Role name: EC2S3AccessRole.

🔹 **Attach Role to EC2:**

1. Launch or modify an EC2 instance.
2. Under **IAM Role**, select EC2S3AccessRole.

🔹 **Test:**

- SSH into EC2 instance.
- Run:

aws s3 ls

You'll be able to access S3 from EC2 using the IAM role.

✅ 6. IAM Policy with Conditions (Time-based access)

🔹 **Create Custom Policy:**

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AccessBetween9to5",
```

```
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": "*",
    "Condition": {
     "DateGreaterThan": {
      "aws:CurrentTime": "2025-04-21T09:00:00Z"
     },
     "DateLessThan": {
      "aws:CurrentTime": "2025-04-21T17:00:00Z"
     }
    }
   }
 ]
}
```

Allows access only between 9 AM and 5 PM UTC on a specific day.

🔹 **Test:**

- Attach to a test user and try accessing S3 at different times.

✅ **7. IAM Role for Lambda Execution**

🔹 **Steps:**

1. IAM > **Roles > Create role**.
2. Select **AWS Service > Lambda**.
3. Permissions: AWSLambdaBasicExecutionRole.
4. Role name: LambdaBasicRole.

🔹 **Attach Role to Lambda:**

- Create a Lambda function and select LambdaBasicRole.

✅ 8. IAM Permissions Boundary

🔹 Use Case:

Limit maximum permissions that a user or role can have, even if a broader policy is attached.

🔹 Steps:

1. IAM > Policies > Create policy (as a boundary):

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "ec2:*",
   "Resource": "*"
  }
 ]
}
```

2. Attach this boundary to a user/role.
3. Even if a user is given more permissions (e.g., S3), they will not be effective.

✅ 9. IAM Access Analyzer (Security Tool)

🔹 Steps:

1. IAM > **Access Analyzer > Create Analyzer**.
2. Name: MyAnalyzer, Type: Account.
3. It analyzes all **external access** to your resources.
4. Review findings and secure over-permissive resources.

✅ 10. IAM Policy Simulator

🔹 Steps:

1. IAM > **Policy Simulator**.
2. Select user/role.
3. Simulate actions (e.g., s3:PutObject, ec2:StartInstances).
4. Review **Allow/Deny** results.
5. Helps debug permission issues.

Scenario : Create IAM User for Developer with Limited Access

📝 **Objective:** Create a devuser who can only read EC2 and S3, and access AWS Console with MFA.

🔧 **Step-by-step:**

1. **Login to AWS Console** as root or admin.
2. Go to IAM > Users > Add Users.
3. **User name**: devuser.
4. **Access type**:
    - ✅ Check: “AWS Management Console access”
    - ✅ Set: Custom password (e.g., Dev@12345)
    - ❌ Uncheck: "Require password reset"
5. Click **Next: Permissions**

---

6. **Permissions options**:
    - Select: Add user to group
    - Click: Create group
        - Group name: DevReadOnlyGroup
        - Policies to attach:
            - AmazonEC2ReadOnlyAccess
            - AmazonS3ReadOnlyAccess
    - Click **Create group**

7. Back on user page:
    - Select DevReadOnlyGroup
    - Click **Next**
8. **Tags** (optional):
    - Key: Department, Value: Development
    - Key: Project, Value: AppX
9. Click **Create user**

---

10. Copy the **Console login URL** and test access.

🔐 **Add MFA to devuser**

1. IAM > Users > devuser > Security credentials tab.
2. Click **Manage MFA device** > Choose **Virtual MFA**.
3. Open Google Authenticator, scan QR code.
4. Enter **2 consecutive codes**.
5. Save.

🧪 **Test**: Log in again – should now prompt for MFA.

EC2-Based IAM Policy Examples with Conditions

✅ **1. Allow EC2 Start/Stop Only for Instances Tagged with Owner=${aws:username}**

🧠 **Use case:** Users can manage **only their own EC2 instances**.

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

```
 "Statement": [
  {
   "Sid": "AllowStartStopOwnEC2",
   "Effect": "Allow",
   "Action": [
    "ec2:StartInstances",
    "ec2:StopInstances"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "ec2:ResourceTag/Owner": "${aws:username}"
    }
   }
  }
 ]
}
```

📌 **Tip**: When launching EC2 instances, always tag them with Owner = username.

✅ **2. Allow Describe EC2 Instances Only in a Specific Region (ap-south-1)**

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DescribeEC2OnlyMumbai",
   "Effect": "Allow",
   "Action": "ec2:DescribeInstances",
   "Resource": "*",
```

```
    "Condition": {
      "StringEquals": {
        "aws:RequestedRegion": "ap-south-1"
      }
    }
  }
 ]
}
```

✅ **3. Deny EC2 Launch If Instance Type Is t2.micro (Force Bigger Instances)**

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
    "Sid": "DenyT2MicroLaunch",
    "Effect": "Deny",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "ec2:InstanceType": "t2.micro"
      }
    }
  }
 ]
}
```

Allow EC2 Start/Stop Only – For Specific Tag

📘 **Use Case:**

Let developers control EC2 instances **only if tagged with Environment=Dev**.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "ec2:StartInstances",
    "ec2:StopInstances"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "ec2:ResourceTag/Environment": "Dev"
    }
   }
  }
 ]
}
```

IAM Advanced Identity-Based Policy Scenarios

✅ **1. Allow EC2 Termination Only From a Specific IP Address**

🔐 *Tight control — e.g., only admins from office IP can terminate instances.*

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

```
  {
    "Sid": "TerminateOnlyFromOffice",
    "Effect": "Allow",
    "Action": "ec2:TerminateInstances",
    "Resource": "*",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": "203.0.113.0/24"
      }
    }
  }
 ]
}
```

✅ **2. Allow EC2 Launch Only if AMI is from Specific Owner (Secure AMI)**

🛡️ Prevent users from launching from unapproved AMIs (e.g., only from official AMIs)

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
    "Sid": "LaunchFromTrustedAMI",
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "ec2:ImageOwner": "123456789012"
      }
```

```
    }
   }
 ]
}
```

✅ 3. Allow S3 Access ONLY if MFA is Used

🔐 Ensure sensitive S3 actions require MFA authentication.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowOnlyWithMFA",
   "Effect": "Deny",
   "Action": [
    "s3:PutObject",
    "s3:DeleteObject"
   ],
   "Resource": "arn:aws:s3:::secure-mfa-bucket/*",
   "Condition": {
    "BoolIfExists": {
     "aws:MultiFactorAuthPresent": "false"
    }
   }
  }
 ]
}
```

✅ 4. Allow Only a Specific IAM Role to Launch EC2 Instances

🔒 Use aws:PrincipalArn to restrict actions to specific IAM role.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowOnlyFromTrustedRole",
   "Effect": "Allow",
   "Action": "ec2:RunInstances",
   "Resource": "*",
   "Condition": {
    "ArnEquals": {
     "aws:PrincipalArn": "arn:aws:iam::123456789012:role/EC2LaunchRole"
    }
   }
  }
 ]
}
```

✅ **5. Allow S3 Uploads with Specific Object Prefix (User-Specific Folder Structure)**

🎯 Users can upload only into folders that match their IAM username.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "UserPrefixUpload",
   "Effect": "Allow",
   "Action": "s3:PutObject",
   "Resource": "arn:aws:s3:::training-uploads/${aws:username}/*"
```

```
  }
 ]
}
```

✅ 6. Allow EC2 Launch ONLY in a Specific Availability Zone

💡 Prevent launching EC2 in expensive or unauthorized AZs.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "RestrictToAZ",
   "Effect": "Allow",
   "Action": "ec2:RunInstances",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "ec2:AvailabilityZone": "ap-south-1a"
    }
   }
  }
 ]
}
```

✅ 7. Allow Actions Only If Request Comes Through AWS Console (Not CLI/SDK)

🎛️ Force users to use AWS Console only — not automate actions with CLI or SDK.

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

```
  {
    "Sid": "OnlyFromConsole",
    "Effect": "Allow",
    "Action": "ec2:StartInstances",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:ViaAWSService": "console.amazonaws.com"
      }
    }
  }
 ]
}
```

✅ **8. Allow Actions Only on Specific Resources Based on Tag Combination (AND logic)**

🎯 Users can perform actions only if BOTH tags are present.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
    "Sid": "AllowTaggedResources",
    "Effect": "Allow",
    "Action": "ec2:StartInstances",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "ec2:ResourceTag/Environment": "Dev",
```

```
        "ec2:ResourceTag/Project": "SmartCity"
      }
    }
  }
 ]
}
```

ROLES:

Scenario: EC2 Instance Accessing an S3 Bucket via IAM Role

🛠️ Step-by-Step in AWS Console

🔹 **Step 1: Go to IAM Console**

- Open AWS IAM Console
- Click **"Roles"** on the left
- Click **"Create Role"**

🔹 **Step 2: Select Trusted Entity**

You'll see options like:

- ✅ AWS Service *(Choose this for EC2/Lambda/ECS)*
- AWS Account
- Web Identity
- SAML 2.0
- Custom Trust Policy

✅ Select:

- **Trusted Entity type:** AWS service
- **Use case:** EC2 (click Next)

🔹 **Step 3: Attach Permissions**

Choose a policy (permission set) that allows access to a service. For example:

- Search: AmazonS3ReadOnlyAccess
- Select the checkbox
- Click **Next**

🔹 **Step 4: Add Tags (optional)**

You can skip this or add tags like:

- Key = Project, Value = WebApp

Click **Next**

🔹 **Step 5: Name, Review & Create**

- **Role name:** EC2S3ReadOnlyRole
- Review the trusted entity: should be ec2.amazonaws.com
- Review permissions: AmazonS3ReadOnlyAccess
- Click **Create Role**

✅ Role is now created.

🔹 **Step 6: Attach Role to an EC2 Instance**

- Go to **EC2 > Instances**
- Select your instance
- Click **Actions → Security → Modify IAM Role**
- Select the role: EC2S3ReadOnlyRole
- Click **Update IAM Role**

✅ Done! Your EC2 instance now has permission to access S3 using this role.

🧪 **Optional Test (on EC2)**

SSH into the EC2 instance and run:

aws s3 ls

You should see your S3 buckets listed — proving the IAM role works.

Cross-Account Role — AWS Account A (Dev) assumes a role in AWS Account B (Prod)

🛠 Step-by-Step Implementation (Console)

🟢 Step 1: Login to AWS Account B (Target Account – Role Will Be Created Here)

1. Go to **IAM Console** → **Roles** → Click **Create Role**

🟢 Step 2: Select Trusted Entity

- Choose: AWS account
- Choose: ✅ **Another AWS Account**
- Enter the **Account ID** of **Account A (Dev)** — this is the account that will assume the role
- (Optional) Enable **Require external ID** for added security

Click **Next**

🟢 Step 3: Attach Permissions

- Choose a permission policy that the role should have **once assumed**
    - Example: AmazonS3FullAccess (if you want access to S3)
    - You can also create a **custom policy**

Click **Next**

🟢 Step 4: Add Tags (Optional)

- Add key-value pairs (optional), e.g. Environment: CrossAccountAccess

Click **Next**

🟢 **Step 5: Name, Review, Create**

- **Role Name:** CrossAccountS3AccessRole
- Review:
    - Trusted account ID
    - Permission policy attached

Click **Create Role**

✅ Role is created in **Account B** (the account being accessed).

🔐 **Step 6: IAM Policy in Account A (Calling Account)**

Now switch to **Account A**, where you want to allow users (or services) to assume that role.

Attach this **inline or managed policy** to a user or role in **Account A**:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "sts:AssumeRole",
   "Resource": "arn:aws:iam::ACCOUNT_B_ID:role/CrossAccountS3AccessRole"
  }
 ]
}
```

Replace:

- ACCOUNT_B_ID → the 12-digit Account B ID
- CrossAccountS3AccessRole → the role name you created earlier

Step 7: Assume Role (CLI or Code)

From **Account A**, run:

```
aws sts assume-role \
 --role-arn arn:aws:iam::ACCOUNT_B_ID:role/CrossAccountS3AccessRole \
 --role-session-name demoSession
```

You'll get:

- AccessKeyId
- SecretAccessKey
- SessionToken

Use these credentials to perform actions on behalf of Account B.

✅ **Test It (Optional)**

Use the temporary credentials to list buckets or access S3 in Account B:

bash

CopyEdit

```
aws s3 ls --profile cross-account
```

(You must set a named profile using the temp credentials)

Step 7 Alternative: Assume Role via **AWS Console (Switch Role)**

Scenario: Assume a Cross-Account Role Using AWS Console (Switch Role)

🛠 **Use Case:**

You have **two AWS accounts**:

- **Account A (source account)**: The account you're logged into.
- **Account B (target account)**: The account that contains the IAM role you want to assume.

Steps to Assume a Role Using AWS Console

🔹 **Step 1: Log in to AWS Console (Account A - Source Account)**

1. **Go to** AWS Console.

2. **Log in** using your IAM credentials for **Account A** (the account you're currently using).

🔹 **Step 2: Access the "Switch Role" Option**

1. In the **top-right corner** of the AWS Console, click on your **Account Name** or **Account Number**.
    - This is where you see your account details and settings.
2. In the dropdown, select **"Switch Role"**.

🔹 **Step 3: Fill in the Role Information**

You will now be on the **"Switch Role"** page, where you'll need to enter some details to switch to the role in **Account B**.

1. **Account ID**:
    - This is the 12-digit AWS **Account ID** of **Account B** (where the role exists).
    - If you're unsure, you can find the Account ID in Account B by going to **IAM > Dashboard** in Account B. The Account ID is visible at the top.
2. **Role Name**:
    - This is the **name of the role** you created in **Account B** (e.g., CrossAccountS3AccessRole).
    - Make sure you have the correct role name from **Account B** that you want to assume.
3. **Display Name** *(optional)*:
    - This is the name that will be displayed in the top-right corner after you switch to the role. It's optional but can help you differentiate between roles if you have multiple.
4. **Color** *(optional)*:
    - You can choose a color to help visually distinguish the role in the top-right corner. It makes it easier to spot which role you're currently assuming.

Once you've filled in these details, click **"Switch Role"**.

🔹 **Step 4: You're Now Acting as the Role in Account B**

- After clicking **Switch Role**, you'll be logged into **Account B** but with the **permissions of the role** you just selected.
- The top-right corner will now show:
  - The **name of the role** you're assuming (e.g., CrossAccountS3AccessRole).
  - The **color** you selected (if you chose one).

You are now in **Account B** (through the role), with the permissions attached to that role. You can perform actions that the role allows you to do (e.g., access S3 buckets, etc.).

🔹 **Step 5: To Switch Back to Account A**

To go back to your original account (Account A), simply:

1. Click the **role name** in the top-right corner.
2. Click **"Back to [Your Account Name]"**.

This will bring you back to your original Account A, where you were logged in initially.

Select Trusted Entity — What It Means

A **trusted entity** is **who will use this role**. It can be:

- An **AWS service** (like EC2 or Lambda)
- A **user or role from another AWS account**
- A **web identity provider** (like Cognito, Google, Facebook)
- A **SAML-based identity provider**
- A **custom trust policy** (advanced, manual setup)

📜 Options Explained in Detail

🔹 1. AWS Service

Use this when an **AWS service** (like EC2, Lambda, ECS) needs to **assume the role to access AWS resources**.

🔸 **Example:**

You have an EC2 instance that needs to access an S3 bucket.
You create a role with `AmazonS3ReadOnlyAccess`, and choose:

Trusted entity: AWS service → EC2

✅ **Common for:** EC2, Lambda, ECS, SageMaker, etc.

🔹 2. AWS Account

Use this when you want to **let another AWS account assume this role** — good for **cross-account access**.

🔸 **Example:**

You have **two AWS accounts** (dev and prod).
You want to allow a user or service in **dev account** to access a role in **prod account**.

You'd enter the **Account ID** of the dev account.

✅ **Common for:** Cross-account IAM roles, shared services, MSP access.

🔹 3. Web Identity

Use this when you're authenticating users from **Cognito**, **Google**, **Facebook**, etc.

🔸 **Example:**

You have a **mobile app** and use Amazon Cognito to manage users.
Users log in via Facebook or Google.
Once authenticated, they assume this IAM role to access AWS resources (like S3 or DynamoDB).

✅ **Common for:** Mobile and web apps with social login.

🔹 4. SAML 2.0 Federation

Use this when you're using **enterprise Single Sign-On (SSO)** from your corporate directory (like Microsoft AD, Okta, etc.).

🔸 **Example:**

Your company uses Okta or Azure AD for internal logins.
You configure SAML SSO so your employees can sign in to the AWS Console using their corporate credentials.

✅ **Common for:** Corporate/enterprise environments

🔹 5. Custom Trust Policy

Use this for full manual control — you write the **trust policy JSON yourself**.

🔸 **Example:**

You want to allow multiple types of entities (like both an AWS account and a Lambda function) to assume the role.
Or you want very specific conditions, like IP address restrictions, external IDs, etc.

✅ **Common for:** Complex setups, advanced access control