AWS Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) enables you to securely control access to AWS services and resources. It allows you to manage users, groups, roles, and permissions for fine-grained access control.

Key Components of IAM

1. Users

IAM Users are individual entities created to represent people or applications requiring access to AWS. Users can have:

- A username and password for access to the AWS Management Console.
- Access keys for programmatic access via the AWS CLI, SDKs, or APIs.

Example: Creating an IAM User

```
resource "aws_iam_user" "example_user" {
  name = "example-user"
}
```

2. Groups

IAM Groups are collections of IAM Users that share the same permissions. You can assign policies to groups to simplify permission management.

Example: Creating an IAM Group

```
resource "aws_iam_group" "example_group" {
  name = "developers"
}
```

Adding a User to a Group

```
resource "aws_iam_group_membership" "example_group_membership" {
  group = aws_iam_group.example_group.name
  users = [aws_iam_user.example_user.name]
}
```

3. Roles

IAM Roles are entities assumed by trusted users or services. They grant temporary security credentials and are commonly used for cross-account access or to allow AWS services to perform tasks.

Example: Creating an IAM Role

```
resource "aws_iam_role" "example_role" {
  name = "example-role"

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

Example: Trusted Policy for Cross-Account Access

A trusted policy defines which entities can assume a role.

```
resource "aws_iam_role" "cross_account_role" {
  name = "cross-account-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          AWS = "arn:aws:iam::123456789012:root"
        },
        Action = "sts:AssumeRole"
      }
    ]
  })
}
```

4. Policies

IAM Policies define permissions that specify what actions are allowed or denied. Policies can be attached to users, groups, or roles.

Types of Policies:

1. **Customer Managed Policies**
    - Policies created and managed by you in your AWS account.
    - Offer greater flexibility and control.

    Example: Creating a Customer Managed Policy

```
resource "aws_iam_policy" "customer_managed_policy" {
  name        = "customer-managed-policy"
  description = "Custom policy for specific access"

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect   = "Allow",
        Action   = "dynamodb:PutItem",
        Resource = "arn:aws:dynamodb:us-east-1:123456789012:table/ExampleTable"
      }
    ]
  })
}
```

2. **Inline Policies**
   - Policies embedded directly into a single user, group, or role.
   - Used for one-off custom permissions.

   Example: Adding an Inline Policy to a Role

```
resource "aws_iam_role_policy" "inline_policy" {
  name = "inline-policy"
  role = aws_iam_role.example_role.name

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect   = "Allow",
        Action   = "s3:ListBucket",
        Resource = "arn:aws:s3:::example-bucket"
      }
    ]
  })
}
```

3. **Amazon Managed Policies**
   - Predefined policies created and maintained by AWS.
   - Provide common permissions for general use cases.

   Example: Attaching an Amazon Managed Policy

```
resource "aws_iam_role_policy_attachment" "managed_policy_attachment" {
  role       = aws_iam_role.example_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
}
```

Attaching a Policy to a Role

```
resource "aws_iam_role_policy_attachment" "example_role_attachment" {
  role       = aws_iam_role.example_role.name
  policy_arn = aws_iam_policy.example_policy.arn
}
```

5. Access Keys

Access keys provide programmatic access to AWS resources.

Example: Creating Access Keys for a User

```
resource "aws_iam_access_key" "example_access_key" {
  user = aws_iam_user.example_user.name
}
```