Overview

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources.

You use IAM to control who is **authenticated** (signed in) and **authorized** (has permissions) to use resources.

It's a Global Service and Service usage is **FREE**

Basic Terminologies

1. Resources:

- The user, group, role, policy, and identity provider objects that are stored in IAM.
- As with other AWS services, you can add, edit, and remove resources from IAM.

2. Identities:

- The IAM resource objects are **used to identify and group**.
- You can attach a policy to an IAM identity.
- These include Users, Groups, and Roles.

3. Entities:

- The IAM resource objects that **AWS uses for authentication**.
- These include IAM users, federated users, and assumed IAM roles.

4. Principals:

- A person or application that uses the AWS account.
  Root user, an IAM user, or an IAM role to sign in and make requests to AWS.

IAM Users

1. For greater security and organization, you can give access to your AWS account to specific users—identities that you create with custom permissions

2. When you create an IAM user, they **CAN NOT** access anything in your account until you give them permission.
3. You give permissions to a user by creating an **identity-based policy,** which is a policy that is attached to the user or a group to which the user belongs

   Example:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "arn:aws:dynamodb:us-east-2:123456789012:table/Books"
  }
}
```

Types of users

i. **IAM Root user**
The user is created when the AWS Account is created.
User email and password is the credential for the account and he has full administrative rights

ii. **IAM User**
A new user has **implicit deny** for all aws resources when he/she is created.
One or more policies need to add to grant the permission to access AWS resources

iii. **Federated User**
If the users in your organization already have a way to be authenticated, such as by signing in to your corporate network, you don't have to create separate IAM users for them.
Instead, you can **federate** those user identities into AWS.

- Your users already have identities in a corporate directory:
  - Microsoft Active Directory or SAML identity federation
- Your users already have Internet identities:
  - Amazon, Facebook, Google, OpenID connect

iv. Power Users:
These are the users who have full access to AWS Services but management of IAM users, groups is **not allowed** to them.

Note: An IAM user doesn't have to represent an actual person; you can create an IAM user in order to generate an access key for an application that runs in your corporate network and needs AWS access.

IAM Groups

An IAM group is a collection of IAM users.

Groups let you specify permissions for multiple users, which can make it easier to manage the permissions for those users.

IAM Permissions

What is permission and why do we need it?

Permissions let you specify Access to AWS resources.

Permissions are granted to IAM identities (users, groups, and roles) and by default these entities start with **no permissions**.

In other words, IAM identities can do nothing in AWS until you grant them your desired permissions.

How to give permission?

To give entities permissions, you can attach a policy that specifies the type of access, the actions that can be performed, and the resources on which the actions can be performed.

IAM Policy

A policy is an object in AWS that, when associated with an **identity** or **resource**, defines their permissions.

Policy contains permissions, which determines whether the request is allowed or denied

IAM Policies specify what you are allowed to do with any AWS resources.

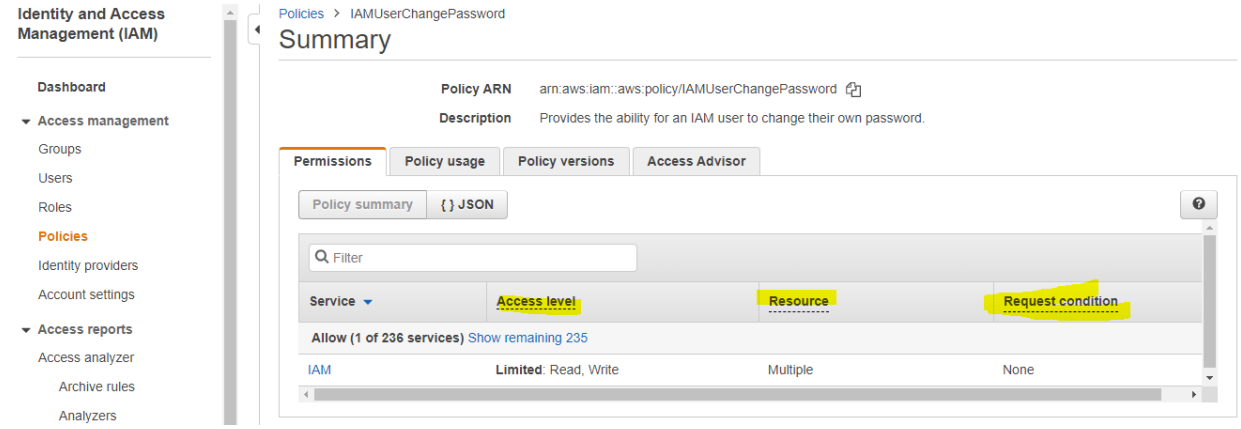
They are **Global** and apply to all areas of AWS.

Policies are attached to either **Identities** (Users, Group, or Roles) or **Resources** (ex. S3)

Types of Policies:

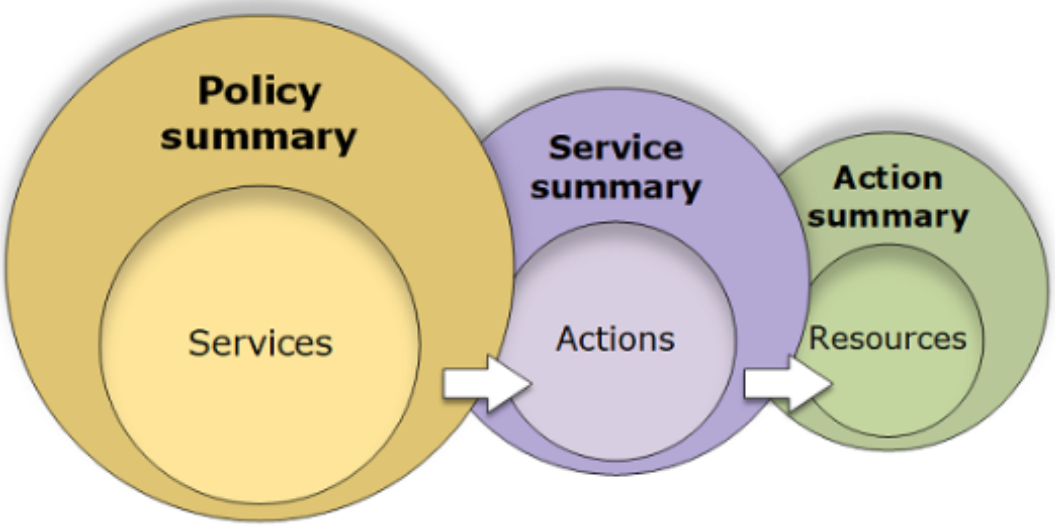
1. Identity based policies
2. Resource based policies

The IAM console includes ***policy summary tables*** that describe the access level, resources, and conditions that are allowed or denied for each service in a policy.



Policies are summarized in three tables:

1. the policy summary
2. the service summary
3. the action summary



1. Identity based policies

These are the policies that you **can attach to an Identity.**
Ex: Users, Groups, Roles

These can define what Identity can do.

***Identity based policies control* what actions the identity can perform, on which resources, and under what conditions**

These can be further categorized into the following 2 types

A. AWS Managed Policies
B. Inline Policies

A.Managed Policies:

These are standalone identity based policies that you can attach to:

- **Multiple** Users
- Groups
- Roles

These can be divided in two categories as below:

I. AWS Managed Policies:

These are created and managed by AWS. These are not modifiable.
These are **READ ONLY**

II. Customer Managed Policies:

These are created and managed by customers

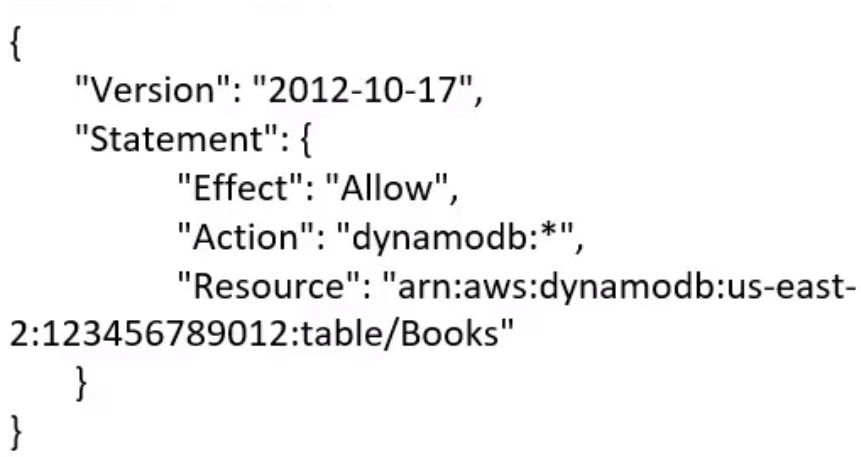
B.Inline Policies:

These are created and managed that are embedded directly to: A **Single User, Group or a Role**.

One to one relation between user/group/role and policy.

In most cases, AWS doesn't recommend using inline policies.

Example for Identity based policy:



```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "dynamodb:*",
        "Resource": "arn:aws:dynamodb:us-east-
2:123456789012:table/Books"
    }
}
```

EAR is Identity based policy document attributes**. E - Effect A - Action R - Resource**

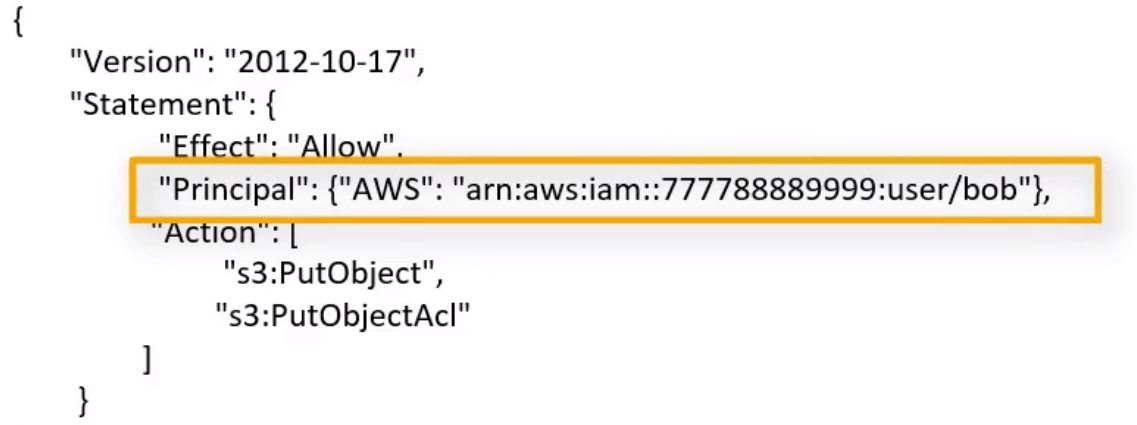
2. Resource based policy

These are the policies that **you can attached to a Resource**, such as S3 bucket

Resource-based policies **control** **what actions a specified principal can perform on that resource and under what conditions.**

To enable **cross-account access**, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy.

These are **inline policies** and there is no managed resource based policies



```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::777788889999:user/bob"},
        "Action": [
            "s3:PutObject",
            "s3:PutObjectAcl"
        ]
    }
```

EAP is Resource based policy document attributes**. E - Effect A - Action P - Principal**

Policy Evaluation Logic

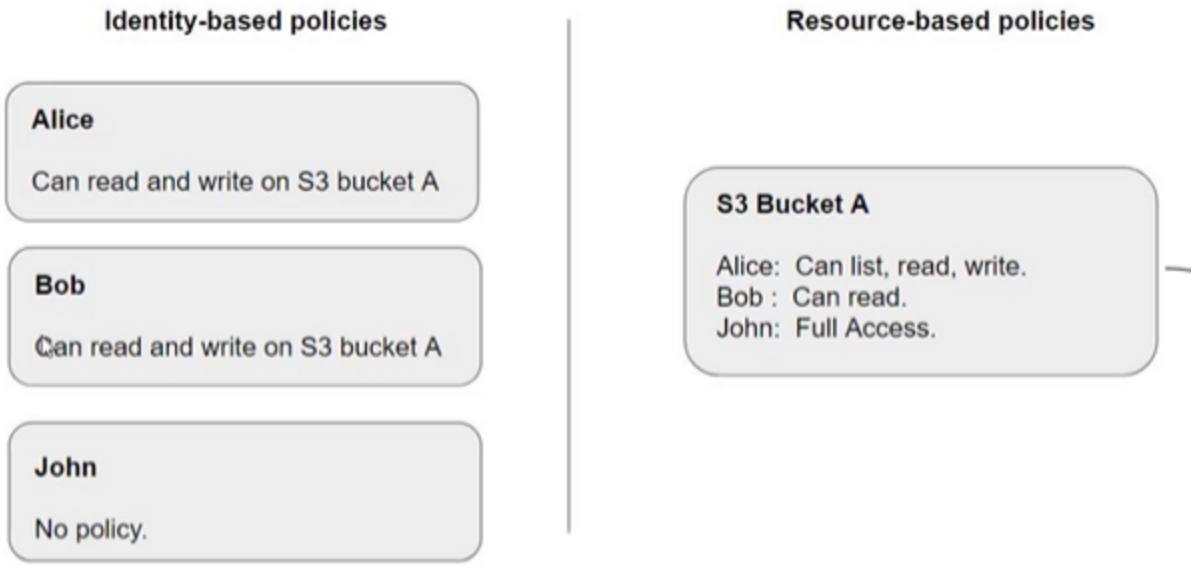
When you have Conflicting Permissions, final solution is done via Policy Evaluation Logic

- All requests are DENIED by default - “Default Deny”
- If there is an **explicit Deny,** then the **final decision is DENY**
- When there is no explicit Deny specified in the policy, it checks whether there is any explicit “Allow”
- If there is Allow then the final decision is Allow.
- However, there is no Deny or Allow specified then the final decision is “Deny”

1. First look for any Explicit Deny
2. Then look for any Explicit Allow
3. If none of them applied then Default Deny

IAM Password Policy allows to create strong password for IAM account

Explain the reason for the results in the following table based on the policy evaluation logic:



IAM User	Operations	Resource
Alice	List, Read, Write	S3 Bucket A
Bob	Read,Write	S3 Bucket A
John	Full Access	S3 Bucket A

What is ABAC for AWS?

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes.

In AWS, these attributes are called *tags*

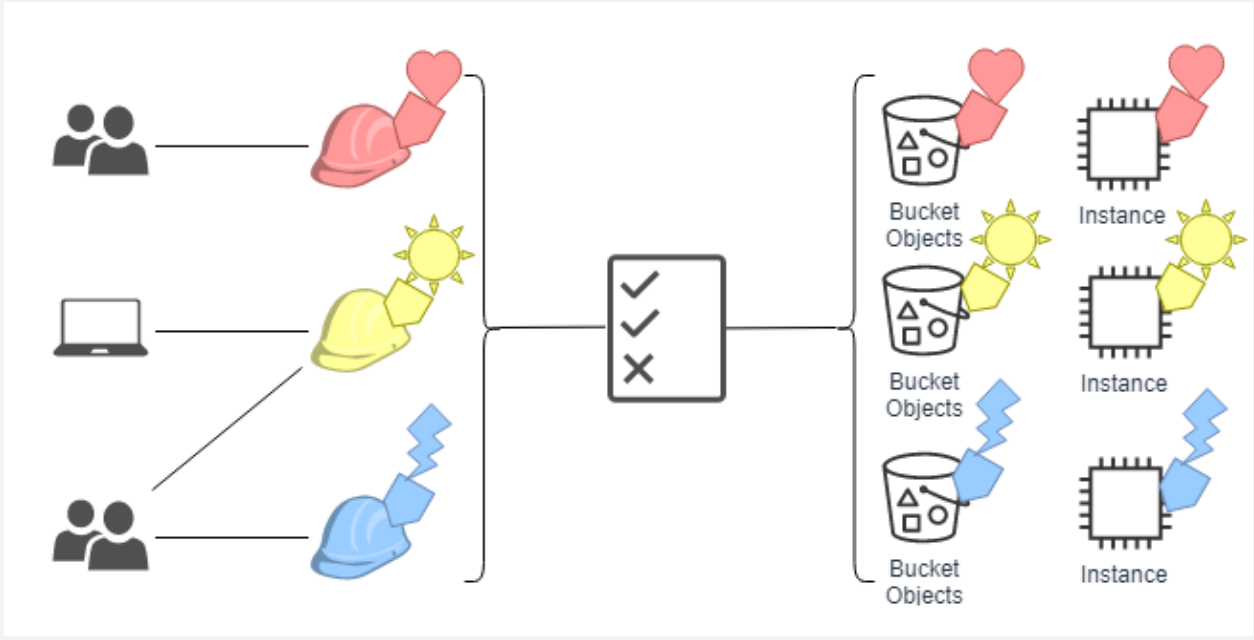
Tags can be attached to IAM principals (users or roles) and to AWS resources.

You can create a single ABAC policy or small set of policies for your IAM principals.

These ABAC policies can be designed to allow operations when the principal's tag matches the resource tag.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

For example:
You can create three roles with the `access-project` tag key. Set the tag value of the first role to `Heart`, the second to `Sun`, and the third to `Lightning`.
You can then use a single policy that allows access when the role and the resource are tagged with the same value for `access-project`



IAM Role

1. What is the IAM Role?

An IAM role is an **IAM Identity** that you can create in your account that has specific permissions.

An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS.

However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it.

Also, a role does not have standard long-term credentials such as a password or access keys associated with it.

Instead, when you assume a role, it provides you with **temporary security credentials for your role session**.

When a user assumes a role, **temporary security credentials are created dynamically and provided to that user - making it more secure way to grant access**

Roles are the primary way to grant cross-account access

IAM roles are **not associated** with a specific user or group. Instead, trusted entities assume roles, such as IAM users, applications, or AWS services such as EC2.

2. Who all can use the IAM Role?

Use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources

1. Users in your AWS account access to the resources they don't usually have.
2. Users in one AWS account access to resources in another account
3. A mobile app to use AWS resources, but not want to embed AWS key within the app.
4. Users who already have identities defined outside of AWS, such in your corporate directory
5. Their parties so that they can perform an audit on your resources

3. Why to use a Role?

It allows you to delegate access with defined permissions to trusted entities without having to **share long-term access keys**

Never share security credentials between accounts to allow users from another AWS Account to access resources in your AWS account. Instead always use IAM Roles.

4. When to use IAM Role?

1. Provide access for services offered by AWS to AWS resource
2. Provide access for an IAM user in one AWS account that you own to access resources in another account that you own
3. Provide access for external authenticated users (Identity federation)
4. Provide access to IAM users in AWS accounts owned by third parties (SAML 2.0 federation)

4. How to assume an IAM role?

You assume an IAM role by calling the **AWS Security Token Service (STS) AssumeRole APIs** (in other words AssumeRole, AssumeRoleWithWebIdentity, and AssumeRolewithSAML). These APIs return a set of temporary security credentials that applications can then use to sign requests to AWS service APIs.

You can associate one IAM Role with an EC2 instance at this time.

5. IAM Role Limitations

You can **NOT** associate **more than ONE IAM role with an EC2 instance**.

This limit of one role per instance can not be increased

You can NOT add an IAM Role to an IAM Group

You **CAN NOT** attach EC2 IAM Role to **on premise servers.**

Refer to the following question.

Question: I have an on-premises personal server that I would like to use to perform AWS API calls.

a. **I should run 'aws configure' and put my credentials there. Invalidate them when I am done - Correct**

b. **I should attach an EC2 IAM Role to a personal server - WRONG ANSWER. You can't attach EC2 IAM roles to on premise servers**

6. IAM PassRole Option

When you create an EC2 instance, you assign a role to it. In order to assign a role to an EC2 instance, you need **IAM:PassRole**

This is not just only for EC2 resources. If you want to assign any resource to any roles first that resource needs to get IAM:PassRole

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Admin rights on EC2 service",
            "Action": "ec2:*",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "Pass an IAM Role",
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::<account-id>:role/Sample-EC2-Role"
        }
    ]
}
```

Permissions Boundaries for IAM Entities

AWS supports *permissions boundaries* for IAM entities (users or roles)

A permissions boundary is an advanced feature for using a managed policy to set the **maximum permissions that an identity-based policy can grant to an IAM entity**.

An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

https://www.youtube.com/watch?v=_4Gw3T1otS4&t=826s&ab_channel=KnowledgeIndiaAWSAzureTutorials

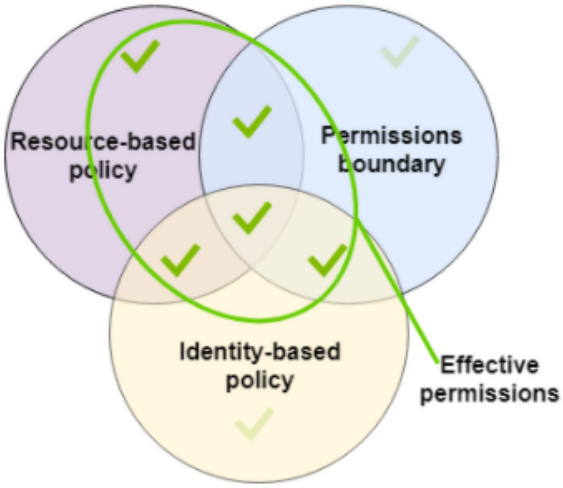
1. Identity-based policies with boundaries

Identity-based policies are inline or managed policies that are attached to a user, group of users, or role.
Identity-based policies grant permission to the entity, and permissions boundaries limit those permissions.

2. Resource-based policies

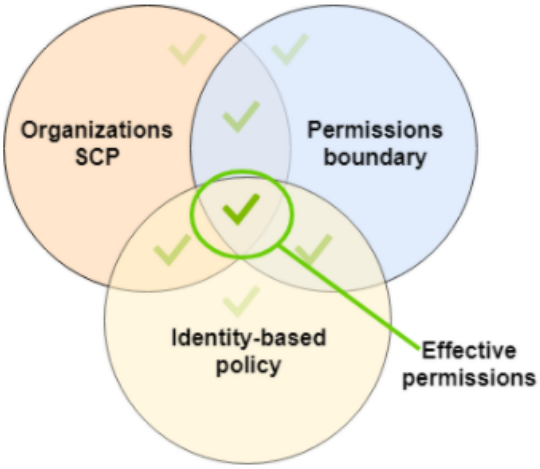
Resource-based policies control how the specified principal can access the resource to which the policy is attached.

In this case, the effective permissions are everything that is allowed by the resource-based policy *and* the intersection of the permissions boundary and the identity-based policy. An explicit deny in any of these policies overrides the allow.



3. Organizations SCPs

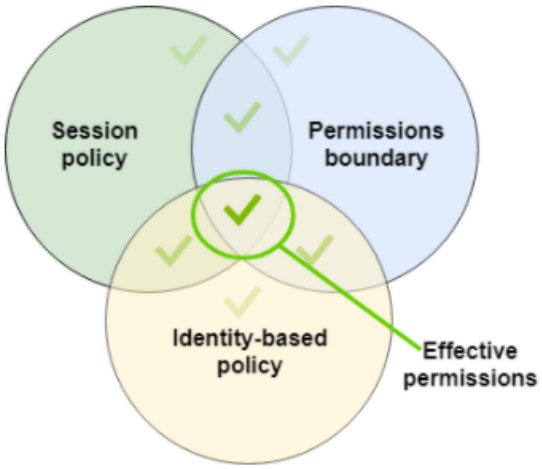
SCPs are applied to an entire AWS account.
They limit permissions for every request made by a principal within the account.
An IAM entity (user or role) can make a request that is affected by an SCP, a permissions boundary, and an identity-based policy.



In this case, the request is allowed only if all three policy types allow it. The effective permissions are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow.

4. Session Policies

Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user.



The effective permissions for this set of policy types are the intersection of all three policy types. An explicit deny in any of these policies overrides the allow. For more information about session policies

IAM Policy Simulator

Online tool to test your policy. It requires calling API operations to do two things:

1. Evaluate the policies and return the list of context keys that they reference.
2. Simulate the policies, providing a list of actions, resources, and context keys.

Use the aws iam simulate-custom-policy command

AWS CLI Dry Run

We can include a dry-run command in the CLI to ensure that policy has rights to perform operations before it actually does that operation.

Ex: Creating an EC2 instance using CLI. If we include --dry-run in the cli command we can ensure that the EC2 instance has valid policy to create instances without actually creating it.

AWS EC2 Instance Metadata

It allows AWS EC2 instances to learn about themselves without using an IAM Role for that purpose

The URL: http://169.254.169.254/latest/meta-data/

You can access that url inside the EC2 instance.

Exam point:

Point 1:
To configure many AWS services, you must pass IAM Role to service. This allows the service to assume the role and perform the action on your behalf. You only have to pass the role to the service during the setup.

For example, when you want to perform some authentication actions on your AWS resource, you need to create a role and pass it to the EC2 when you create it. Then every time, when EC2 assumes the role, it performs the actions we specified via policy on the role.

To pass role to service, you must have pass role permission. This helps the administrator to perform only the valid user can perform this action.

If you want to grant a user to pass role to AWS resources, you need to perform the following steps:

Step 1: an IAM **Permission policy** attached to the role that determines what role can do on the resource.

Step 2: A **Trust policy** for the role that is going to attach to the resource

Step 3: IAM **Permission policy** attached to the user that helps users to pass roles that are approved. IAM PassRole permission and IAM GetRole permission

Set up an IAM permissions policy attached to the IAM Role that determines the actions that it must perform. Afterwards, create a trust policy for the role that allows the service to assume the role and ***Set up an IAM permissions policy attached to the IAM user that allows the user to pass only those roles that are approved. Use the*** `iam:PassRole` ***and*** `iam:GetRole` ***permissions in order for the user to get the details of the role to be passed***

Point 2:
You're planning on allowing an administrator to set up an EC2 instance. The EC2 instance will host an application that would need access to a DynamoDB table. What are the steps required?

1. **A Trust Policy** for the role that allows the service to assume the role.
    a. Example: You could attach the following trust policy to the role with the **UpdateAssumeRolePolicy** action. This trust policy allows Amazon EC2 to use the role and the permission attached to the role.

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

2. An **IAM permission policy** attached to the IAM user that allows the user to pass only the roles that are approved. iam:PassRole usually is accompanied by iam:GetRole so that the user can get the details of the role to be passed.
    a. Example: User can pass only roles that exist in the specified account with names that begin with EC2-roles-for-XYZ

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::<account-id>:role/EC2-roles-for-XYZ-*"
    }]
}
```

3. Note in the above second point the only thing that I missing to show is an IAM Permission policy on the role to show what actions that role needs to perform.

Knowledge Check:

1. What are AWS IAM Identities / Principals?
2. What are AWS Resources?
3. What is IAM Policy?
4. What are the types of IAM Policies?
5. What is the difference between AWS Managed policy and Inline Policy?
6. What is the difference between Resource based Policy and Identity based Policy?
7. Why do we need an IAM Role?
8. Who all can use the IAM Role?
9. Why to use a Role?
10. How to assume an IAM role?
11. When would a Resource Based Policy be better than IAM Role?
12. What is IAM PassRole? And why do we need that?
13. Can an AWS user belong to multiple groups?

Demos

1. Create a User Group with Billing access information.
    a. Granting access to your billing information and tool
        i. Where do you find **IAM User and Role Access to Billing Information**
2. Define permissions to access AWS resources based on tags

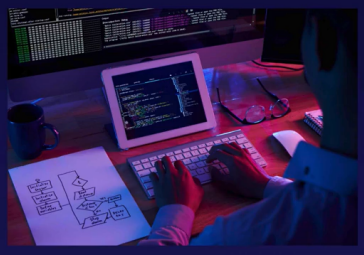
IAM Credential Report

Credential Report

Lists all users in your account and the status of their various credentials, including passwords, access keys, and MFA devices.

Console

Head to IAM section, find Credential report on the left and download a CSV containing the report.



CLI

```
aws iam generate-credential-report
aws iam get-credential-report
```