AWS IAM Concepts

who (identity) has *what access* (role) for *which resource*

Created by Julian Wiegmann - November 2022 V1

Notes:
Terminology used is official AWS terminology.
For IAM concept details see official AWS documentation
This is not an exhaustive list of security tips/risks around IAM!
Nor covers all Identity features/services in AWS

Can use AWS Organizations service control policy (SCP) to limit the permissions of root user

SCP

Organization
Optional

AWS Organizations helps you centrally govern your environment as you scale your workloads in AWS

Manage & Use

SCP - Service Control Policy

Organization policy to manage permissions in your organization. SCPs offer central control over the maximum available permissions for all accounts in your organization.

Contains & Manage

Security credentials are account-specific

AWS Account

AWS account is a container for your AWS resources. You create / manage your AWS resources in an AWS account, & provides administrative capabilities for access and billing

Full Control over
Root can close account

Contains

Manage

Resources (Service)
EC2, S3 etc.

Can use

Access control lists (ACLs)

Cannot be used to control access for a principal within the same account.

Authorization 2
Similar to resource-based policies, although they do not use the JSON format. Amazon S3, AWS WAF, and Amazon VPC key services that support ACLs.

Session Policies (for Roles)

No more space but AWS best practice when using federation

Create distinctive role session permissions or to further restrict session permissions, users or systems can set a session policy when assuming a role on the fly.
Session policy are inline permissions policy which users pass, or your identity-broker, when they assume the role.
The effective permissions of the session are the intersection of the role's identity-based policies and the session policy. You can pass a single inline session policy programmatically by using the policy parameter with the AssumeRole, AssumeRoleWithSAML, AssumeRoleWithWebIdentity, and GetFederationToken API operations.

Do not use for day2day tasks

Don't create long-term access keys for your root user. If malicious user gains access to your root user access keys = game over

Root User
Email & Password

Cannot use

Manages

AWS IAM
Not a regional service

Contains

IAM Resources

Root is also a Principal

Signed (authenticated) or unsigned (anonymous) you can configure and not just read. Full control & write to a storage bucket for example are possible. depends on service

Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship.

Some service. You can specify ANY user or even whole other AWS accounts to access /edit/ use resources here! This is not a Role!

Not the same

Type

Authorization

Policies (Permission Policy)
All JSON policy

Resource Based Policy
For Services/Resources

You specify who has access to the resource and what actions they can perform on the resource

Manage

Roles are both a Principals & Resources

Policies are attached to Principlas

Not Policies managed in Policies but under Roles but "same"

Not recommended by AWS for Users

Are always of type

Principals
Sometimes called Identities

Person or application
Principals include federated users and assumed roles

Type

Identity Based Policies
Attached to an identity

Control what actions the identity can perform, on which resources, and under what conditions

Type

Inline Policies

Policies that are embedded directly into a single user, group, or role OR Resource

Permission Policy
What can role do

Trust Policy
Principal
(who can assume role!)

You can specify ANY user or even whole other AWS accounts to access resources here!

Type

Type

Type

Authentication happens here
IAM Entities / Identities (User or Role)

Managed Policies

You can specify who has access to the resource and what actions they can perform on it

Policies that are attached to a single user, group, or role

Attached here for Roles

Special Type of Policies

Support

Permission Boundaries

A policy that sets the maximum permissions an IAM entities can have

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

Requires Policy

Role
A role is intended to be assumable by anyone who needs it

Subtypes

Unless Inline

Types

Group
User Groups

Contains

User
Account ID / alias
user name

Use Groups to assign permissions not individual users

AssumeRole

Cross Account and Cross Service is possible and Confused Deputy Problem

You can allow Roles to assume other Roles. Role Chaining

AWS Service Role
For AWS services to use to do their function

EC2 Service Role
For apps running on EC2

AWS Service Linked Role
predefined by the service include all the permissions

Federation
Oauth, SAML 2.0
(external IDP)

AWS Managed
~998 Policies Exist

Careful with "FullAccess" Policies

AWS managed policies don't grant least privilege permissions. (this is what AWS says in their documentation. Take note other CSPs

Customer Managed
You define

For Users or Roles. NOT Groups!

Have their own ARN e.g. arn:aws:iam::aws:policy/IAMReadOnlyAccess

Humans can authenticate with these keys

API based access using access keys

AKIA
Access Keys
Access key ID +
Secret Access Key

ASIA
Access Keys
Access key ID +
Secret Access Key
+ expire time

These temporary credentials are called via AWS STS (right?) AssumeRole

Often abused with IMDS v1 SSRF. E.g. vuln webapp can call role creds and attacker can use

Key known to the creator of the key and consumer (if used by 3rd party). Do not expire

Max two access keys per user (for rotation)

Humans can authenticate with these keys

IAM JSON Policy

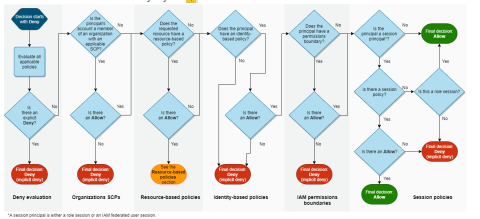
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*"
    },
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": [
        "iam:DeleteInstanceProfile",
        "iam:UpdateAssumeRolePolicy",
        "iam:SetInstanceProfile",
        "iam:ListRoleTags",
        "iam:PutRolePermissionsBoundary",
        "iam:ListPoliciesGrantingServiceAccess",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:ListInstanceProfileTags",
        "iam:DetachRolePolicy",
        "iam:CreatePolicyVersion",
        "iam:SetDefaultPolicyVersion"
      ],
      "Resource": [
        "arn:aws:iam::959321979655:group/*",
        "arn:aws:iam::959321979655:user/*",
        "arn:aws:iam::959321979655:instance-profile/myserver_*",
        "arn:aws:iam::959321979655:policy/*",
        "arn:aws:iam::959321979655:role/*"
      ],
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "210.75.1.1"
        },
        "DateLessThan": {
          "aus:CurrentTime": "2025-06-30T23:59:59Z"
        }
      }
    }
```

Used throughout AWS IAM policies.
Over ~13 000 permissions (actions) exist for the services. Action is allow statement but NotAction (deny) also exist. This can be combined with 'Effect' Deny. E.g. Deny IAM* except Multifactor was done.
Scope can be defined on resources.
Wildcards are possible throughout.
Conditions can use Boolean expressions.
Can include things like "must MFA before allow"

Policy evaluation logic

This is key

If a single permissions policy includes a denied action, AWS denies the entire request and stops evaluating. This is called an *explicit deny*. Because requests *are denied by default*



*A session principal is either a role session or an IAM federated user session.