AWS IAM

Verified 2026-11-02

How AWS decides allow or deny — principals, identity and resource policies, roles, boundaries, SCPs, conditions, and debugging AccessDenied.

Contents

Mental model 2

Principals and policies 2

Policy evaluation 3

Identity vs resource policies 4

Roles and AssumeRole 5

Execution roles for compute 6

Boundaries and SCPs 7

Conditions and tags 8

Least privilege and debugging denials 9

Further reading 10

Mental model

Every AWS request starts as denied. An explicit `Allow` in an identity or resource policy carves an exception into that implicit deny, and an explicit `Deny` anywhere cancels every allow. Guardrails such as permissions boundaries, SCPs and session policies never grant anything: they cap the maximum, so the effective permission is the intersection of all of them.

Principals and policies

Principal	What it is
IAM user	Long-lived identity with keys or a password
IAM role	Identity with no credentials; assumed for a session
Role session	Temporary credentials from STS
Service principal	An AWS service, e.g. `lambda.amazonaws.com`

A policy is a JSON document of statements. Each statement has an `Effect`, a list of `Action`s, the `Resource` ARNs it applies to, and optional `Condition`s. Attach managed policies for reuse; inline policies live and die with one principal.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["s3:GetObject"],
```

```
    "Resource": "arn:aws:s3:::media/uploads/*"
  }]
}
```

> **Gotcha:** `"Version": "2012-10-17"` is the policy-language version, not a date to update. Omit it and policy variables such as `${aws:username}` are treated as literal strings.

Policy evaluation

Step	Result if it matches
Explicit `Deny` in any policy	Deny, final
SCP or RCP lacks an allow	Deny
Resource policy allows (same account)	Usually allow
Boundary or session policy lacks an allow	Deny
Identity policy allows	Allow
Nothing allows	Implicit deny

AWS gathers every applicable policy and evaluates them together; statement order and policy order do not matter. Within one account, an allow in either the identity policy or the resource policy is enough for most services. Role trust policies and KMS key policies are exceptions: they must allow the principal explicitly.

```
final = no explicit Deny
    AND allowed by SCPs and RCPs
```

```
    AND (identity policy allows
         OR resource policy allows)
    AND allowed by boundary
    AND allowed by session policy
```

> **Gotcha:** A `Deny` cannot be overridden by any `Allow`, even `AdministratorAccess`. When an admin is denied, search for a `Deny` statement before adding more allows.

Identity vs resource policies

Policy	Attached to	Has `Principal`?
Identity-based	User, group or role	No
Resource-based	Bucket, queue, key, function	Yes
Trust policy	A role	Yes

A resource policy names who may act on the resource. Cross-account access needs both sides: the resource policy in account A must allow the principal from B, and the identity policy in B must allow the action on A's resource.

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

```
      "AWS": "arn:aws:iam::111122223333:role/app"
    },
    "Action": "sqs:SendMessage",
    "Resource":
      "arn:aws:sqs:eu-west-1:444455556666:jobs"
  }]
}
```

Roles and AssumeRole

Setting	Value
Max session duration	1–12 hours, default 1 hour
`DurationSeconds`	900 s up to the role's maximum
Role chaining	Capped at 1 hour

A role has two policies: the trust policy says who may call `sts:AssumeRole`, and the permission policies say what the session may do. For cross-account access the caller also needs `sts:AssumeRole` on the role in its own identity policy.

```
aws sts assume-role \
  --role-arn arn:aws:iam::444455556666:role/deploy \
  --role-session-name ci-run-81 \
  --duration-seconds 3600
aws sts get-caller-identity  # who am I now?
```

> **Gotcha:** Using one role's session to assume another role is role chaining, limited to one hour. Asking for a longer `--duration-seconds` fails even if the target role allows 12 hours.

Execution roles for compute

Compute	Trusted service principal
Lambda	`lambda.amazonaws.com`
EC2 (via instance profile)	`ec2.amazonaws.com`
ECS task role	`ecs-tasks.amazonaws.com`

Your code never holds long-lived keys: the platform assumes the role and the SDK picks up rotating temporary credentials automatically. Give each function or service its own role so one bug cannot reach another workload's data.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "lambda.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }]
}
```

> **Warning:** Access keys in environment variables or code outlive the person who created them. If something runs on AWS, give it a role.

Boundaries and SCPs

Guardrail	Scope	Grants?
Permissions boundary	One user or role	Never
Service control policy (SCP)	Accounts in an organization	Never
Resource control policy (RCP)	Resources in member accounts	Never
Session policy	One assumed-role session	Never

A boundary is a managed policy that sets the most a principal can ever have; effective permissions are the intersection of the boundary and the identity policies. Use it to let developers create roles without letting them create a role more powerful than themselves.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["s3:*", "dynamodb:*", "logs:*"],
    "Resource": "*"
  }]
}
```

> **Gotcha:** Attaching this boundary grants nothing. A role with this boundary and no identity policy can do nothing at all.

Conditions and tags

Condition key	Checks
`aws:PrincipalTag/team`	Tag on the calling principal
`aws:ResourceTag/team`	Tag on the target resource
`aws:SourceArn`	Which resource made a service call
`aws:PrincipalOrgID`	Caller's organization
`aws:SecureTransport`	Request used TLS

Conditions narrow a statement to requests whose context matches. Matching principal tags to resource tags is attribute-based access control (ABAC): one policy covers every team, and access follows tags instead of growing lists of ARNs.

```
{
  "Effect": "Allow",
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/team":
        "${aws:PrincipalTag/team}"
    }
```

```
  }
}
```

> **Tip:** When a service principal such as `sns.amazonaws.com` writes to your resource, add an `aws:SourceArn` or `aws:SourceAccount` condition. Without it, any account's topic can use that grant.

Least privilege and debugging denials

Message fragment	Meaning
`no identity-based policy allows`	Implicit deny: add an allow
`explicit deny in an identity-based policy`	Remove or scope the `Deny`
`explicit deny in a service control policy`	An SCP blocks it
`no permissions boundary allows`	Boundary lacks the action

The `AccessDenied` message names the principal, action and resource, then the policy type that decided; explicit denials often include that policy's ARN. Some services return an encoded message instead, which `decode-authorization-message` turns into JSON. IAM Access Analyzer generates a policy from up to 90 days of CloudTrail activity, validates policy grammar, and reports unused roles, keys and permissions.

```
aws sts get-caller-identity
aws iam simulate-principal-policy \
```

```
  --policy-source-arn \
    arn:aws:iam::111122223333:role/app \
  --action-names s3:GetObject \
  --resource-arns arn:aws:s3:::media/a.png
aws sts decode-authorization-message \
  --encoded-message "$MSG"
```

> **Gotcha:** Many denials are the wrong principal, not the wrong policy. Run `get-caller-identity` first: a stale profile or an unexpected assumed role explains the error faster than reading policies.

Further reading

- IAM User Guide
- Policy evaluation logic
- Troubleshoot access denied errors
- IAM Access Analyzer