ALL major components of AWS IAM Policies include:

✔ Identity Policies
✔ Resource Policies
✔ Permission Boundaries
✔ Organization SCPs
✔ Session Policies
✔ Inline vs Managed Policies
✔ Conditions (with real examples)
✔ Policy Evaluation Logic
✔ Deny > Allow
✔ Wildcards
✔ Advanced examples

This is the **most complete explanation** you will find — suitable for learning, interviews, and real architecture work.

AWS IAM Policies — Complete, Detailed Tutorial

AWS IAM Policies control **who can do what on which resource under what conditions**.

At the core, every policy answers:

> **"Which principal is allowed/denied which actions on which resources, under which conditions?"**

----

1. Types of IAM Policies

There are **5 major types**, and interviews often ask the difference.

Policy Type	Attached To	Purpose
Identity Policy (IAM Policy)	User/Group/Role	Grants permissions to identities
Resource Policy	On resources (S3, KMS, Lambda, SNS, SQS)	Controls who can access the resource
Permission Boundary	IAM Role/User	Maximum allowed permissions
Service Control Policy (SCP)	AWS Organization OU/Account	Limit permissions account-wide
Session Policy	Temporary sessions (STS, AssumeRole)	Additional temporary restrictions

---

==========

2. Identity Policies (IAM Policies)

These are the most common policies.

Attached to:

- IAM User
- IAM Group
- IAM Role

They contain:

```
{
  "Effect": "Allow",
  "Action": ["s3:PutObject"],
  "Resource": ["arn:aws:s3:::mybucket/*"]
}
```

They *grant* permissions but do NOT override:

- SCPs
- Permissions boundaries

- Resource policies
- Explicit Deny

Identity Policies alone cannot give access to:

- S3 bucket if bucket policy denies
- KMS key if key policy denies
- Account if SCP denies

==========

3. Resource-Based Policies

These are attached directly to the resource.

Examples:

- S3 Bucket Policy
- SQS Policy
- SNS Topic Policy
- Lambda Resource Policy
- KMS Key Policy
- API Gateway Resource Policy
- Secrets Manager Resource Policy

Example: S3 bucket policy allowing another account

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::396982710430:role/test-role"},
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::mybucket",
      "arn:aws:s3:::mybucket/*"
    ]
  }]
}
```

Special thing:

✔ Supports **Principal** field (identity policies do not).
✔ Can allow cross-account access.

==========

4. Permission Boundaries

Permission Boundary = Maximum permission a role/user can have.

Even if identity policy allows something → **boundary can restrict it**.

Use case:

- Delegate admin tasks to developers
- Restrict what roles they can create or modify
- Restrict temporary roles created by Lambda

Example boundary:

Delegated Admin / Developer Self-Service

Scenario

Your organization wants developers to:

- Create IAM roles and Create Lambda functions
- Manage CloudWatch logs
- Deploy to S3 buckets for their team only

But you must **prevent them from:**

- Deleting production data
- Creating IAM users or roles with admin access
- Accessing KMS keys
- Managing EC2 / VPC resources

Problem Without Permission Boundaries

If developers have iam:CreateRole, they could create a role with **AdministratorAccess**, and escalate their privilege.

→ Very dangerous!

Solution

Apply a **Permission Boundary** so that **even if** they create roles or apply policies, they remain restricted by the boundary.

Full Working Example

Permission Boundary Policy (Maximum Allowed Capabilities)

This policy defines **what the user/role is allowed to ever do**, even in the best case.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDevServices",
      "Effect": "Allow",
      "Action": [
        "lambda:*",
        "logs:*",
        "s3:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyAdminDangerousActions",
      "Effect": "Deny",
      "Action": [
        "iam:*",
        "kms:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Meaning:

- Developers can *only* manage Lambda, S3, CloudWatch Logs
- They are **blocked from IAM, KMS, EC2**
- This is the **maximum** they can ever do
- Even if they attach a policy granting more access, it’s ignored

Think of it as:

> **IAM Policy gives permissions**
> **Boundary limits permissions**

==========

5. Service Control Policies (SCPs)

Part of AWS Organizations.

Applied to:

- OU (Organizational Unit)
- Specific AWS accounts

SCPs set the maximum permissions allowed for the entire account.

Even the root user is blocked.

Example: Block usage of non-approved regions

```
{
  "Effect": "Deny",
  "Action": "*",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {"aws:RequestedRegion": ["us-east-1","us-west-2"]}
  }
}
```

Difference from Permission Boundary:

SCP	Permission Boundary
Applies to whole AWS account	Applies only to user/role
No direct permission granted	Does not grant permissions
Only restricts	Only restricts
Org-level	IAM-level

==========

6. Session Policies

These are temporary restrictions when you **assume a role**.

Example:

```
aws sts assume-role \
  --role-arn arn:aws:iam::123456789012:role/DevRole \
  --policy '{"Version":"2012-10-17","Statement":[{"Effect":"Deny","Action":"s3:*","Resource":"*"}]}'
```

This session CANNOT use S3, even if the role normally can.

==========

7. Inline vs Managed Policies

Inline Policies

- Embedded directly inside user/role
- Deleted if attached resource is deleted

- Harder to reuse
- Used for **one-off, tightly coupled permissions**

Managed Policies

- Reusable
- Can attach to multiple identities
- Two types:
  - AWS managed (e.g., `AdministratorAccess`)
  - Customer managed

==========

8. IAM Policy Structure (with explanation)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1",
      "Effect": "Allow",         // Allow or Deny
      "Action": "s3:PutObject",  // API that can be performed
      "Resource": "arn:aws:s3:::mybucket/*", // Which resources
      "Condition": {             // Optional logic
        "StringEquals": {
          "s3:x-amz-acl": "public-read"
        }
      }
    }
  ]
}
```

==========

9. Conditions in IAM Policies (Most Important Part)

Conditions determine *when* a policy applies.

9.1 String Conditions

```
"Condition": {
  "StringEquals": {"s3:x-amz-acl": "private"}
}
```

9.2 IP Address Conditions

```
"Condition": {
  "IpAddress": {"aws:SourceIp": "10.0.0.0/16"}
}
```

9.3 Date Conditions

```
"Condition": {
  "DateGreaterThan": {"aws:CurrentTime": "2025-01-15T00:00:00Z"}
}
```

9.4 MFA Conditions

```
"Condition": {
  "Bool": {"aws:MultiFactorAuthPresent": "true"}
}
```

9.5 Source VPC Endpoint

```
"Condition": {
  "StringEquals": {"aws:sourceVpce": "vpce-12345"}
}
```

9.6 Tags in Policies (Tag-based access control)

User must have tag:

```
"Condition": {
  "StringEquals": {"aws:PrincipalTag/department": "Finance"}
}
```

Resource must have tag:

```
"Condition": {
  "StringEquals": {"aws:ResourceTag/owner": "team1"}
}
```

==========

10. Explicit Deny Always Wins

Example:

Identity policy:

`"Effect": "Allow", "Action": "s3:*"`

SCP:

`"Effect": "Deny", "Action": "s3:DeleteObject"`

Result:
➡ Can perform everything EXCEPT DeleteObject.

==========

11. Policy Evaluation Logic — AWS Algorithm

AWS evaluates in this order:

1. **Explicit Deny** → overrides everything
2. **Allow** → only if no Deny
3. **Default = Implicit Deny**

==========

12. Real-World Examples

Example 1: Developer role allowed ONLY S3 operations

Boundary:

```
{
  "Effect": "Allow",
  "Action": ["s3:*"],
  "Resource": "*"
}
```

Identity Policy:

```
{
  "Effect": "Allow",
  "Action": ["ec2:*"],
  "Resource": "*"
```

```
}
```

Final:

```
  – EC2
✔ Only S3 (because boundary limits it)
```

Example 2: Deny delete in S3

```
{
  "Effect": "Deny",
  "Action": ["s3:DeleteObject"],
  "Resource": "arn:aws:s3:::mybucket/*"
}
```

Even admin can't delete.

Example 3: Allow role access to bucket only from VPC endpoint

```
"Condition": {
  "StringEquals": {"aws:SourceVpce": "vpce-12345"}
}
```

Example 4: SCP blocking expensive resources

```
{
  "Effect": "Deny",
  "Action": [
    "ec2:RunInstances",
    "rds:CreateDBInstance",
    "ec2:CreateVolume"
  ],
  "Resource": "*"
}
```

==========

13. When to Use Which Policy (Cheat Sheet)

Use Case	Policy Type
Give user permission →	Identity Policy
Control resource access from other accounts →	Resource Policy
Restrict user's maximum permissions →	Permission Boundary
Restrict the entire account →	SCP
Temporary restrictions during AssumeRole →	Session Policy
Must allow cross-account access →	Resource Policy
Must enforce tagging →	Conditions
Must block actions in certain regions →	SCP