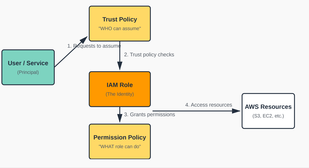
AWS IAM Trust Policies - Complete Guide

What is a Trust Policy?

A trust policy is a JSON document attached to an IAM role that defines which **principals** (users, services, accounts) are allowed to assume that role.

Key Point: Every IAM role must have a trust policy. Without it, no one can use the role!



Trust Policy Structure

A trust policy uses the same JSON structure as other IAM policies, but with specific elements:

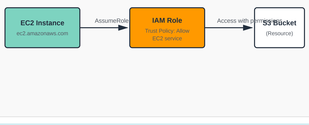
```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", // Allow or Deny "Principal": { ... }, // WHO can assume "Action": "sts:AssumeRole", // The assume action "Condition": { ... } // Optional conditions } ] }
```

Element	Description
Version	Policy language version (always use "2012-10-17")
Statement	Array of permission statements
Effect	Allow or Deny the action
Principal	WHO can assume the role (AWS account, IAM user, service, etc.)
Action	Usually "sts:AssumeRole" or "sts:AssumeRoleWithWebIdentity"
Condition	Optional constraints (MFA, IP address, time, etc.)

Example 1: Simple EC2 Service Role

This trust policy allows EC2 instances to assume the role:

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



Use Case: An EC2 instance needs to access S3 buckets. Instead of storing AWS credentials on the instance, you attach this role to the instance, allowing it to automatically assume the role and get temporary credentials.

Example 2: Cross-Account Access

Allow another AWS account to assume this role:

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS": "arn:aws:iam::123456789012:root" }, "Action": "sts:AssumeRole", "Condition": { "StringEquals": { "sts:ExternalId": "unique-external-id-12345" } } } ] }
```

Use Case: A third-party service provider needs access to your AWS resources. The External ID prevents the "confused deputy" problem - it ensures that only the authorized third party can assume the role, even if they know your account ID.

Example 3: Lambda Execution Role

Multiple AWS services can assume the same role:

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

Example 4: IAM User with MFA Requirement

Require multi-factor authentication for assuming the role:

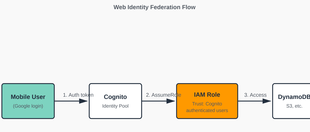
```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS": "arn:aws:iam::123456789012:user/Admin-user" }, "Action": "sts:AssumeRole", "Condition": { "Bool": { "aws:MultiFactorAuthPresent": "true" }, "NumericLessThan": { "aws:MultiFactorAuthAge": "3600" // 1 hour } } } ] }
```

Security Tip: This ensures that even if credentials are compromised, an attacker cannot assume the role without the MFA device. The MFA must have been used within the last hour.

Example 5: Web Identity Federation (OIDC)

Allow users authenticated through Google, Facebook, or Amazon Cognito:

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Federated": "cognito-identity.amazonaws.com" }, "Action": "sts:AssumeRoleWithWebIdentity", "Condition": { "StringEquals": { "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-1234-1234-1234-123456789012" }, "ForAnyValue:StringLike": { "cognito-identity.amazonaws.com:amr": "authenticated" } } } ] }
```



Use Case: Mobile or web applications where users log in with Google/Facebook. The app gets temporary AWS credentials to directly access AWS resources like S3 or DynamoDB without exposing long-term credentials in the app.

Example 6: Specific IAM User Access

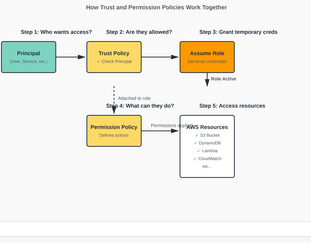
Grant access to specific IAM users within your account:

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS": [ "arn:aws:iam::123456789012:user/developer1", "arn:aws:iam::123456789012:user/developer2" ] }, "Action": "sts:AssumeRole" } ] }
```

Common Trust Policy Patterns

Pattern	Principal Type	Common Use Case
Service Role	`"Service": "ec2.amazonaws.com"`	EC2, Lambda, ECS tasks needing AWS permissions
Cross-Account	`"AWS": "arn:aws:iam::ACCOUNT:root"`	Sharing resources between AWS accounts
Federated (SAML)	`"Federated": "arn:aws:iam::ACCOUNT:saml-provider/NAME"`	Corporate SSO integration
Federated (Web Identity)	`"Federated": "cognito-identity.amazonaws.com"`	Mobile/web apps with social login
Specific IAM User	`"AWS": "arn:aws:iam::ACCOUNT:user/NAME"`	Elevated privileges for admin tasks

The Complete Picture



Key Takeaways

Remember:

- **Trust Policy:** Controls WHO can assume the role (authentication)
- **Permission Policy:** Controls WHAT the role can do (authorization)
- **Both are required:** A role needs both policies to function
- **Temporary credentials:** Assuming a role gives you temporary security credentials (default 1 hour, max 12 hours)
- **Conditions add security:** Use MFA, IP restrictions, time-based access, and external IDs
- **Least privilege:** Only grant the minimum permissions necessary

Best Practices

1. **Use External IDs for cross-account access** - Prevents confused deputy attacks
2. **Require MFA for sensitive roles** - Add an extra layer of security
3. **Use conditions to limit access** - Restrict by IP, time, or other factors
4. **Prefer roles over users** - For services and applications
5. **Regular audits** - Review who can assume your roles periodically
6. **Use specific principals** - Avoid wildcard (*) in trust policies when possible