IAM in DevOps Workflows

How do you use IAM roles with EC2 or Lambda?

🖥️ With EC2:

- Create an **IAM role** with required permissions (e.g., S3 access).
- Attach the role to the **EC2 instance** (during or after launch).
- EC2 uses the role to get **temporary credentials** automatically.
- No need to hardcode AWS access keys.

🌀 With Lambda:

- Create an **IAM role** with permissions for services Lambda needs (e.g., DynamoDB, S3).
- Assign the role when you **create the Lambda function**.
- Lambda uses the role to access AWS services securely.
- Role is automatically assumed during function execution.

✅ IAM roles help EC2 and Lambda access AWS securely **without storing keys**.

How do you manage permissions for Jenkins deployed on EC2 to access S3?

Create a role with required S3 permissions and attach it to the EC2 instance.

How do you secure access to AWS services in CI/CD pipelines?

Use IAM roles or temporary credentials instead of long-term access keys.

- **Enable IAM roles for service accounts (IRSA)** if using EKS/Kubernetes.
- Use **environment variables or secrets manager** to inject credentials securely.
- **Rotate credentials regularly** and avoid hardcoding them in code or scripts.
- Apply **least privilege principle** — only allow necessary actions.
- Use **OIDC integration** for tools like GitHub Actions to assume roles dynamically.
- Enable **logging (CloudTrail)** and **monitoring** to audit pipeline actions.

✅ Always aim for **temporary, scoped, and auditable access**.

How do you manage IAM permissions for containers running in ECS or EKS?

For ECS (Elastic Container Service):

- Use **IAM Roles for Tasks** (task roles) to assign permissions to containers.
- Create an **IAM role** with required permissions.
- Attach the role to the **ECS task definition**.
- Containers get temporary credentials via the task role.
- Avoid using EC2 instance roles for container permissions.

For EKS (Elastic Kubernetes Service):

- Use **IAM Roles for Service Accounts (IRSA)**.
- Create an **IAM role** and link it to a Kubernetes service account.
- Annotate the service account with the IAM role ARN.
- Pods assume the role when running, getting temporary credentials.
- Provides fine-grained, pod-level permissions.

✅ Both approaches provide **secure, least-privilege, and temporary AWS access** to containers.

What is the AWS IAM Roles for Service Accounts (IRSA) feature in EKS?

✅ **AWS IAM Roles for Service Accounts (IRSA) in EKS:**

- Lets **Kubernetes pods assume IAM roles** securely.
- Uses **service accounts** in Kubernetes linked to **IAM roles**.
- Pods using the service account **get temporary AWS credentials**.
- Uses **OIDC (OpenID Connect)** for authentication between EKS and IAM.
- Allows **fine-grained, pod-level permissions** (not node-wide).
- Avoids using **instance roles**, which affect all pods on a node.
- Improves **security** and **access control** in multi-tenant clusters.

✅ IRSA = Secure, scalable way for EKS pods to access AWS services.

What is the use case of Web Identity Federation in IAM?

Web Identity Federation lets you **trust external identity providers** (like Facebook, Google, Amazon, or OIDC-compliant providers) to authenticate users, and then let those users **assume an IAM role** to access AWS resources **without needing AWS credentials**.

Use Case Example:

You have a mobile or web app that allows users to sign in with **Facebook Login**. Once signed in, the app needs to **upload a photo to S3**. Instead of embedding AWS credentials in the app, you use **Web Identity Federation** to allow the Facebook user to assume a role **securely and temporarily**.

Step-by-Step: Setting Up Facebook Web Identity Federation in AWS

Step 1: Create a Facebook App

1. Go to Facebook Developer Console.
2. Create a new app (type: Consumer).
3. Note down the **App ID**.
4. Add `https://www.amazon.com` or your app's domain in **OAuth redirect URIs** (under App Settings > Facebook Login > Settings).

Step 2: Add Facebook as an Identity Provider in AWS

1. Go to the **AWS IAM Console**.

2. In the left sidebar, click **Identity providers**.
3. Click **Add provider**.
4. Select **Web identity** as provider type.
5. Under **Provider**, choose `Facebook`.
6. Enter the **Facebook App ID**.
7. Click **Add provider**.

📌 This step tells AWS: "Trust users authenticated by this Facebook App ID."

Step 3: Create an IAM Role for Federated Users

1. Go to **IAM > Roles > Create role**.
2. Choose **Web Identity** as trusted entity type.
3. Select **Facebook** as the identity provider.
4. Choose the provider and enter audience (typically: `aws`).
5. Attach **permissions** — e.g., `AmazonS3PutObject`, `DynamoDBReadOnlyAccess`, or custom policy.

✅ Example permission policy:

```
json
-----------
{
  "Effect": "Allow",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::my-photo-bucket/*"
}
```

6. Review and **create the role**.

Step 4: Your App Requests Temporary Credentials

Your **mobile/web app** performs the following flow:

1. **User logs in via Facebook SDK**, gets a **Facebook access token**.
2. The app calls **AWS STS** to assume the role using that token:

📌 Using AWS SDK (e.g., in JavaScript):

```
js
-----------
AWS.config.credentials = new AWS.WebIdentityCredentials({
  RoleArn: "arn:aws:iam::ACCOUNT_ID:role/YourFacebookFederatedRole",
  ProviderId: "graph.facebook.com",
  WebIdentityToken: "<FACEBOOK_ACCESS_TOKEN>"
});
```

3. AWS STS verifies the token with Facebook.

4. If valid, it issues **temporary AWS credentials** valid for up to 1 hour.
5. App uses the credentials to access S3, DynamoDB, etc.

Trust Policy of the Role

The IAM role's **trust policy** looks like this:

```
json
-----------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<ACCOUNT_ID>:oidc-
provider/graph.facebook.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "graph.facebook.com:app_id": "YOUR_FACEBOOK_APP_ID"
        }
      }
    }
  ]
}
```

✅ Security Best Practices

- Do **not** hardcode credentials in your app.
- Always use **temporary credentials** via `AssumeRoleWithWebIdentity`.
- Limit the IAM role permissions to **least privilege** (just what the app needs).
- Enable **CloudTrail** to monitor who assumed the role.

Summary

Step	Description
1	Create a Facebook app
2	Add Facebook as an Identity Provider in AWS
3	Create IAM Role with a trust policy for Facebook
4	Your app gets a Facebook token and uses it to assume the role
5	AWS STS returns temporary credentials

Q: What is the main purpose of assigning IAM roles to EC2 instances or Lambda functions?

A. To manage billing access

B. To allow the service to interact securely with other AWS resources

C. To enable multi-region deployments

D. To enforce strong password policies

✅ **Correct Answer:** B

How do you secure access to AWS services in CI/CD pipelines?

Q: What are best practices for securing AWS access in a CI/CD pipeline?

A. Use root credentials in your CI pipeline

B. Create one IAM user with full admin access for all stages

C. Use IAM roles with limited scopes and temporary credentials

D. Store AWS access keys in public GitHub repos

✅ **Correct Answer:** C

How do you manage IAM permissions for containers running in ECS or EKS?

Q: How should IAM permissions be assigned to containers in ECS and EKS?

A. Embed IAM credentials in container images

B. Use task roles in ECS and IAM roles for EC2 in EKS

C. Use task roles in ECS and IAM roles for service accounts in EKS

D. Grant full access to all containers by default

✅ **Correct Answer:** C