Interview questions on Advanced IAM Features

What are permission boundaries in IAM?

https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html

Permission boundaries in AWS Identity and Access Management (IAM) are **advanced features** that set the **maximum permissions** an IAM role or user can have. Even if the IAM user or role is granted more permissions through their policies, **they cannot exceed what's defined in the permission boundary**.

Think of it like this:

- **IAM Policy** = Grants permissions
- **Permission Boundary** = Restricts permissions to a maximum limit

So, **effective permissions = intersection of IAM policy and permission boundary**

✅ Use Case:

Imagine you're allowing developers to create their own IAM roles via automation. You want to ensure that they **can't give themselves full admin rights**, even if they try. You apply a **permission boundary** that says:

> "You can create roles, but those roles can only manage S3 and Lambda, not EC2 or IAM."

📌 Example:

Suppose a user has this **IAM policy**:

```
json
-----------
{
  "Effect": "Allow",
  "Action": "s3:*",
  "Resource": "*"
}
```

And the **permission boundary** is:

```
json
-----------
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:ListBucket"
  ],
  "Resource": "*"
}
```

Then, even though the IAM policy allows all S3 actions, **the user will only be allowed to perform** `GetObject` **and** `ListBucket` due to the permission boundary.

Summary:

- **Permission boundary** = **upper limit** of what a user/role can do.
- They are attached to **IAM roles or users**, not groups.
- They're mainly used to **control delegated access** and enforce **security guardrails**.

How do IAM roles work with cross-account access?

- **Account A** has an S3 bucket).
- **Account B** wants to **use** it safely.
- **IAM Role** is the **gatekeeper** that lets them in — **with rules**.

Examples

🔷 Step 1: Create Role in Account A

In Account A, create a role and define a **trust policy** to allow Account B to assume it.

✅ **Trust Policy (Account A's role):**

```
json
-----------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::ACCOUNT_B_ID:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

> This means: "I trust all users in Account B to assume this role."

STS (Security Token Service)

🔹 Step 2: Attach Permissions to the Role in Account A

Add a permissions policy to the role. For example, give access to an S3 bucket:

✅ **Permissions Policy:**

```
json
-----------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::my-bucket-name/*"
    }
  ]
}
```

> This means: "If someone assumes this role, they can read files from my S3 bucket."

🔹 Step 3: Assume the Role from Account B (Using AWS CLI)

From Account B, assume the role using `aws sts assume-role`. Replace ARNs with actual values.

✅ **CLI Command (from Account B):**

```
bash
-----------
aws sts assume-role \
  --role-arn arn:aws:iam::ACCOUNT_A_ID:role/RoleName \
  --role-session-name MySession
```

✅ **Response:**

You’ll get temporary credentials:

```
json
------------
{
  "Credentials": {
    "AccessKeyId": "ASIAxxxxxxxxxxxx",
    "SecretAccessKey": "xxxxxxxxxxxxxxxx",
    "SessionToken": "xxxxxxxxxxxxxx",
    "Expiration": "2025-05-22T08:20:00Z"
  }
}
```

🔹 Step 4: Use Temporary Credentials

Use those temporary keys to **access Account A's resources**.

✅ **Example: Use `aws s3` with temporary keys**

```
bash
------------
export AWS_ACCESS_KEY_ID=ASIAxxxxxxxxxxxx
export AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxx
export AWS_SESSION_TOKEN=xxxxxxxxxxxxxx

aws s3 ls s3://my-bucket-name
```

> This shows the files in the S3 bucket **from Account B using the role in Account A**.

Final Recap

Step	What You Do	Code
✅ Create Role in Account A	Trust Account B	JSON trust policy
✅ Add Permissions	Grant limited access	S3 read-only example
✅ Assume Role	Use `sts assume-role`	AWS CLI
✅ Use Role	Export temporary keys	Access S3

Explain how STS (Security Token Service) works for temporary credential management ? □ How does AssumeRole work and when would you use it in a DevOps pipeline? Answer in bullet points,short sentences.

✅ What is STS (Security Token Service)?

- STS gives **temporary credentials** (Access Key, Secret Key, Session Token).
- Used for **short-term, limited access** to AWS resources.

- Helps avoid hardcoding permanent credentials.
- Credentials usually last **15 mins to a few hours**.
- Used for **cross-account access**, CI/CD pipelines, federated users, etc.

✅ How `AssumeRole` Works

- A user or app **calls** `AssumeRole` using the AWS CLI or SDK.
- STS checks if the **trust policy allows** the caller.
- If allowed, STS returns **temporary credentials**.
- These credentials are then used to **access AWS resources**.
- Permissions are based on the **role's permission policy**.

✅ When to Use `AssumeRole` in DevOps Pipelines

- 🔁 To **switch roles** and deploy to another AWS account (e.g., staging → prod).
- 🔒 To ensure **least privilege** – only give access during build/deploy steps.
- 🔑 To avoid **storing long-term credentials** in CI/CD tools (like Jenkins, GitHub Actions).
- 📦 To allow pipelines to access **specific services** (e.g., push to ECR, deploy to ECS).
- 📁 To fetch or store files in **S3 buckets across accounts** during automation.
- 🧪 To test apps in **isolated environments** (each with different roles).

What are service control policies (SCPs)? When are they used?

✅ What are Service Control Policies (SCPs)?

- SCPs are part of **AWS Organizations**.
- They set **permission guardrails** for AWS accounts.
- SCPs **do not grant** permissions — they **only restrict** what can be done.
- Think of them as **"master switch" rules** across accounts.
- They apply to **all IAM users, groups, and roles** in the account.

✅ When Are SCPs Used?

- 🔒 To **enforce security boundaries** across multiple accounts.
- 🚫 To **block risky services** (e.g., prevent use of EC2, IAM in dev accounts).
- ✅ To allow only specific regions (e.g., "Only use `ap-south-1`").
- 📦 To limit services based on **environment** (e.g., no RDS in test account).
- 🛡️ To ensure **compliance** and **centralized control** in large organizations.
- 👨‍💻 To prevent account admins from overriding **org-wide policies**.

Even if an IAM user has full `AdministratorAccess`, **SCP can still block** that action.

What are permission boundaries in IAM?

Q1: What is the main purpose of a permission boundary in IAM?

A. To give full admin access to an IAM user
B. To define the maximum permissions an IAM role or user can get
C. To restrict access to AWS billing
D. To block access to specific IP addresses

✅ **Correct Answer:** B

How do IAM roles work with cross-account access?

Q2: What allows an IAM role in Account A to be used by a user in Account B?

A. IAM policy in Account B
B. CloudTrail logging
C. Trust policy in Account A's role
D. Billing dashboard

✅ **Correct Answer:** C

Which of the following is a valid use case for `AssumeRole` in DevOps?

A. Monitoring CPU usage
B. Deploying resources to another AWS account securely
C. Managing CloudFront cache
D. Encrypting EBS volumes

✅ **Correct Answer:** B

What are Service Control Policies (SCPs)? When are they used?

Q6: What is true about SCPs in AWS Organizations?

A. SCPs give users full access to all services
B. SCPs set permission guardrails across accounts
C. SCPs are for billing settings only
D. SCPs are IAM policies applied to S3 buckets

✅ **Correct Answer:** B

What happens if an SCP blocks an action but an IAM policy allows it?

A. The action is allowed
B. The IAM policy wins
C. The action is denied
D. AWS ignores both policies

✅ **Correct Answer:** C