■ AWS IAM

Identity & Access Management

Complete Enterprise Guide

Interview Ready • Production SOP • Architecture • Policies

GUI Steps • POC • GitHub README • Security Best Practices

Zero Trust • Least Privilege • MFA • Cross-Account Access

■ IAM	■ USERS	■ GROUPS	■ POLICIES	■ ROLES	■ MFA

■ Table of Contents

1. What is AWS IAM?

■ Interview Explanation

AWS IAM (Identity and Access Management) is a FREE global AWS service that controls WHO can do WHAT on WHICH AWS resources. It is the security foundation of every AWS account. Without IAM, there is no access control — every action in AWS is authorized through IAM.

The Golden Rule of IAM: By default, everything is DENIED. You must explicitly ALLOW access. An explicit DENY always wins over any ALLOW, no matter how many allows exist.

The Perfect Interview Analogy:

"IAM is like a company's badge access system. The company (AWS account) has buildings (services). Employees (IAM Users) have badges (credentials). Their badge only opens certain doors (policies). A contractor (IAM Role) gets a temporary badge valid for 1 hour (STS token). Security rules (SCPs) apply to entire floors (AWS Accounts) regardless of individual badges."

What IAM Controls:

- Authentication — Verifying IDENTITY: who is making the request (username/password, access keys, MFA).
- Authorization — Verifying PERMISSIONS: what actions are allowed on which resources.
- Delegation — ROLES: temporary credentials for services, applications, or cross-account access.
- Federation — SSOS: enterprise users authenticate via SAML/OIDC (Active Directory, Okta, Google).
- Auditing — CLOUDTRAIL: every API call is logged with who, what, when, from where.

Key Facts for Interviews:

Fact	Value
Cost	100% FREE — no charge for IAM users, roles, or policies
Scope	GLOBAL — not region-specific. IAM resources are universal.
Root User	All-powerful account owner. Should NEVER be used for daily tasks.
Max IAM Users	5,000 per account (soft limit, can be increased)
Max Policies/User	10 managed policies attached per principal
Password Policy	Configurable: length, complexity, rotation, reuse prevention
Access Key Limit	2 active access keys per IAM user at a time
MFA Support	Virtual MFA, Hardware TOTP, FIDO2/WebAuthn Security Keys

2. IAM Core Components

■ ROOT USER

The account owner. Created when you first create an AWS account. Has unrestricted access to everything. NEVER use root for daily tasks. Protect with MFA and hardware security key. Only use root for: account closure, billing settings, support plan changes, and enabling MFA on root itself.

■ *Best Practice: Lock root: enable MFA, delete access keys, create admin IAM user instead.*

■ IAM USERS

Permanent identities representing a person or application. Each user has: username, password (console access), and/or access keys (programmatic access via CLI/API). Users start with zero permissions. Best practice: one user per person, never share credentials.

■ *Best Practice: Avoid IAM users for applications — use IAM Roles instead.*

■ IAM GROUPS

Collections of IAM users. Policies attached to a group apply to ALL users in that group. A user can belong to multiple groups (up to 10). Groups CANNOT contain other groups. Example groups: Developers, Admins, ReadOnly, DataScientists.

■ *Best Practice: Always assign permissions via groups, never directly to individual users.*

■ IAM ROLES

Temporary identity with permissions. No permanent credentials — uses STS to generate short-lived tokens (15 min to 12 hrs). Used by: AWS services (EC2, Lambda), cross-account access, federated users (SSO), and applications. The MOST IMPORTANT IAM concept.

■ *Best Practice: Use roles over users for all service-to-service authentication.*

■ IAM POLICIES

JSON documents defining permissions. Attached to users, groups, or roles. Evaluated at request time to determine Allow or Deny. Structure: Version, Statement, Effect (Allow/Deny), Action (AWS API calls), Resource (ARN), Condition (optional context).

■ *Best Practice: Always use least privilege — start with AWS managed policies, refine with custom.*

■ MFA

Multi-Factor Authentication adds a second layer beyond password. Types: Virtual MFA (Google Authenticator, Authy), TOTP Hardware token, FIDO2 Security Key (YubiKey). Enable on root account IMMEDIATELY. Require via IAM policy condition aws:MultiFactorAuthPresent.

■ *Best Practice: Enforce MFA for all human users, especially those with admin permissions.*

3. IAM Policy Deep Dive

IAM Policies are JSON documents that define permissions. Every AWS API call is checked against all applicable policies. Understanding policy structure is essential for interviews and for writing secure production policies.

Policy JSON Structure — Full Anatomy:

```
{ "Version": "2012-10-17", // Always use this version "Statement": [ // Array of permission statements { "Sid": "AllowS3ReadOnly", // Optional statement ID "Effect": "Allow", // Allow OR Deny "Principal": "*", // WHO (for resource-based policies) "Action": [ // WHAT API actions "s3:GetObject", "s3:ListBucket" ], "Resource": [ // WHICH resources (ARN format) "arn:aws:s3:::my-bucket", "arn:aws:s3:::my-bucket/*" ], "Condition": { // WHEN (optional context) "StringEquals": { "s3:prefix": ["finance/"] }, "Bool": { "aws:MultiFactorAuthPresent": "true" }, "IpAddress": { "aws:SourceIp": "203.0.113.0/24" } } } ] }
```

Policy Elements Explained:

Element	Required?	Values / Notes
Version	Yes	'2012-10-17' — Always use this. The older '2008-10-17' lacks key features.
Statement	Yes	Array of one or more permission statements.
Sid	No	Statement ID — for your reference only. No functional effect.
Effect	Yes	'Allow' or 'Deny'. Explicit Deny ALWAYS overrides any Allow.
Principal	Cond.	WHO the policy applies to. Required in resource-based policies only.
Action	Yes	AWS API operations. Format: service:ActionName. Wildcards: s3:*, *.
Resource	Yes	ARN of resources. Wildcard (*) means all resources (avoid in prod!).
Condition	No	Context restrictions: MFA, IP range, time of day, tags, service, etc.
NotAction	No	Matches everything EXCEPT listed actions. Used in deny statements.
NotResource	No	Matches all resources EXCEPT listed. Used with Deny for guardrails.

Wildcards in Actions:

```
// ■ NEVER do this in production: "Action": "*" // Grants EVERY action on EVERY service // ■■ Be careful with service wildcards: "Action": "ec2:*" // All EC2 actions — very broad // ■ Best practice — specific actions: "Action": [ "ec2:DescribeInstances", "ec2:StartInstances", "ec2:StopInstances" ] // ■ Prefix wildcards are acceptable: "Action": "s3:Get*" // All s3 GET operations
```

Common Condition Keys:

Condition Key	Example Use Case

aws:MultiFactorAuthPresent	Require MFA for sensitive operations like IAM changes
aws:SourceIp / aws:VpcSourceIp	Restrict access to corporate IP range or VPC only
aws:RequestedRegion	Deny actions outside approved regions (data residency)
aws:CurrentTime	Allow access only during business hours
aws:PrincipalTag/Department	ABAC — allow access based on user's department tag
s3:prefix	Restrict S3 ListBucket to specific folder prefix
ec2:ResourceTag/Environment	Allow actions only on instances tagged 'production'
iam:PassedToService	Control which services a role can be passed to
sts:ExternalId	Cross-account role assumption security measure

4. IAM Policy Types

AWS Managed Policies	Created and maintained by AWS. Auto-updated when new services launch. Examples: AdministratorAccess, ReadOnlyAccess, AmazonS3FullAccess, AmazonEC2ReadOnlyAccess. Can be attached to multiple users/groups/roles. Cannot be edited.

■ *Use as starting point, then create Customer Managed for fine-grained control.*

Customer Managed Policies	Created by you. Reusable across multiple principals. Versioned — up to 5 versions stored. You control the policy content. Best for custom organizational requirements.

■ *Preferred for production. Enables reuse, versioning, and centralized policy management.*

Inline Policies	Embedded directly into a single user, group, or role. 1:1 relationship. Deleted when the principal is deleted. No reuse. Harder to audit. Use sparingly — only when a policy must not be shared accidentally.

■ *Avoid inline policies. Use Customer Managed Policies instead for auditability.*

Resource-Based Policies	Attached to AWS resources (S3 buckets, KMS keys, SQS queues, Lambda functions). Define WHO can access the resource. Include a Principal element. Enable cross-account access without assuming a role. Examples: S3 Bucket Policy, KMS Key Policy.

■ *Use for cross-account S3 access. Always include aws:PrincipalOrgID condition for safety.*

Service Control Policies (SCPs)	AWS Organizations feature. Applied to OUs or accounts. Set maximum permissions guardrails. Even Administrator users cannot exceed what SCP allows. SCPs do NOT grant permissions — they only restrict. Applied hierarchically: Root → OU → Account.

■ *Use SCPs to prevent RegionLock bypass, disable unused services org-wide, enforce encryption.*

Permission Boundaries	Advanced feature that sets maximum permissions an IAM entity can have. Used to safely delegate IAM permission management. The effective permissions = intersection of identity-based policy AND permission boundary.

■ *Use when delegating IAM creation to dev teams — prevent privilege escalation.*

Policy Evaluation Logic — The Decision Flow:

```
1. DENY by default (if no policy → DENY)
2. Check for explicit DENY → if found → DENY (cannot override)
3. Check SCPs (AWS Organizations) → if SCP denies → DENY
4. Check Resource-based policies → if Allow → ALLOW (cross-account: need identity too)
5. Check Permission Boundaries → if boundary denies → DENY
6. Check Session Policies (for assumed roles) → restrict if needed
7. Check Identity-based policies → if Allow found → ALLOW
8. If no Allow found → DENY

TL;DR: Explicit Deny > SCP > Resource Policy > Boundary > Session Policy > Identity Policy
```

5. IAM Roles — Deep Dive

IAM Roles are the most powerful and most important IAM concept. Roles provide temporary credentials via AWS STS (Security Token Service). Unlike users, roles have NO permanent credentials — they issue short-lived tokens.

■ AWS Service Roles

Use case: EC2, Lambda, ECS, CodePipeline need to call AWS APIs

Trust Policy (who can assume this role):

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

■ Cross-Account Roles

Use case: Account A users access resources in Account B

Trust Policy (who can assume this role):

```
{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Principal": {"AWS": "arn:aws:iam::111122223333:root"}, "Action": "sts:AssumeRole", "Condition": {"StringEquals": {"sts:ExternalId": "UniqueID-12345"}} }] }
```

■ Federated / SSO Roles

Use case: Enterprise users via SAML 2.0 / OIDC (Okta, AD, Google)

Trust Policy (who can assume this role):

```
{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Principal": {"Federated": "arn:aws:iam::123456789012:saml-provider/MySSO"}, "Action": "sts:AssumeRoleWithSAML" }] }
```

How AssumeRole Works — Step by Step:

```
Step 1: Principal calls sts:AssumeRole API aws sts assume-role \ --role-arn arn:aws:iam::999999999999:role/ProductionRole \ --role-session-name MySession \ --duration-seconds 3600 Step 2: STS validates the trust policy of the target role → Is the caller in the Principal list? → Check Condition Step 3: STS returns temporary credentials: { "AccessKeyId": "ASIAIOSFODNN7EXAMPLE", "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG...", "SessionToken": "AQoDYXdzEJr...", // Required for STS creds "Expiration": "2024-01-15T12:00:00Z" } Step 4: Caller uses these 3 values (Key+Secret+Token) to make AWS API calls Step 5: Credentials auto-expire — no manual revocation needed
```

6. How IAM Works — Authentication & Authorization Flow

```
EVERY AWS API Call goes through this flow: 1. REQUEST arrives at AWS → Who: Principal (User/Role/Service) → What: Action (e.g., s3:GetObject) → Which: Resource (ARN) → Context: IP, time, MFA status, tags 2. AUTHENTICATION → Verify signature (SigV4 signing with access key) → Verify credentials not expired → For console: verify password + MFA 3. AUTHORIZATION — Policy Evaluation a) Check for explicit DENY in any policy → DENY if found b) Check Organization SCPs → must have Allow c) Check resource-based policies d) Check Permission Boundaries e) Check Session Policies f) Check identity-based policies → must have Allow g) If no Allow → implicit DENY 4. RESULT: ALLOW → request proceeds | DENY → AccessDenied error
```

STS Token Components:

Component	Description	Length
Access Key ID	Starts with ASIA (STS) or AKIA (IAM user). Identifies the caller.	20 chars
Secret Access Key	Used to sign requests (SigV4). Never transmitted over the wire.	40 chars
Session Token	Required for STS credentials. Passed as header or query param.	Variable
Expiration	When credentials expire. Default 1hr, max 12hrs for role sessions.	Timestamp

ABAC — Attribute-Based Access Control:

Modern IAM approach using tags to control access dynamically. Instead of writing a new policy for every team/project, you use conditions on resource tags matching principal tags. Scales massively with fewer policies.

```
// ABAC Policy — Developer can only manage EC2s tagged with their team { "Effect": "Allow", "Action": ["ec2:StartInstances", "ec2:StopInstances"], "Resource": "*", "Condition": {"StringEquals": { "ec2:ResourceTag/Team": "${aws:PrincipalTag/Team}" }} }
```

7. How to Create IAM Resources — GUI Step-by-Step

7A — Create IAM User (Console)

1	**Navigate to IAM** AWS Console → Search 'IAM' → IAM Dashboard
2	**Users Section** Left sidebar → Users → Click 'Create user'
3	**User Details** Enter username (e.g., 'john.smith') → Check 'Provide user access to AWS Console' if needed → Set password or auto-generate
4	**Set Permissions** Option A: Add user to group (RECOMMENDED) → Option B: Attach policies directly → Option C: Copy from existing user
5	**Add Tags** Key: Department = Engineering | Key: CostCenter = 1234 | Key: ManagedBy = Terraform
6	**Review & Create** Review all settings → Click 'Create user' → DOWNLOAD credentials CSV immediately (shown only once!)
7	**Enable MFA** Go to user → Security credentials → Assign MFA device → Virtual MFA → Scan QR with Authenticator app → Enter 2 consecutive codes → Assign

7B — Create IAM Role for EC2 (Console)

- **Step 1:** IAM → Roles → Create role
- **Step 2:** Trusted entity type: AWS service → Use case: EC2 → Next
- **Step 3:** Add permissions: search and select 'AmazonS3ReadOnlyAccess' → Next
- **Step 4:** Role name: 'ec2-s3-readonly-role' → Add tags → Create role
- **Step 5:** Attach to EC2: EC2 Console → Instance → Actions → Security → Modify IAM Role → Select role

7C — Create Custom IAM Policy (Console)

- **Step 1:** IAM → Policies → Create policy
- **Step 2:** Visual Editor OR JSON tab → Paste your policy JSON
- **Step 3:** Policy name: 'MyTeamS3Access' → Description → Tags
- **Step 4:** Create policy → Attach to user/group/role

7D — Enable MFA on Root Account (CRITICAL — Do First!)

■■ This is the MOST IMPORTANT step when creating a new AWS account:

- **Step 1:** Sign in as root → Click account name → Security credentials
- **Step 2:** Multi-factor authentication (MFA) → Assign MFA device
- **Step 3:** Device name: 'root-mfa' → Authenticator app → Continue
- **Step 4:** Open Google Authenticator / Authy → Scan QR code
- **Step 5:** Enter MFA code 1 → Wait 30 seconds → Enter MFA code 2 → Add MFA
- **Step 6:** VERIFY: Sign out → Sign back in → Must provide MFA code ■

8. Benefits & Features

■ 100% Free
No charge for IAM users, roles, groups, or policies. Pay only for what IAM protects (AWS services used).

■ Global Service
IAM is not region-specific. Create once, use everywhere across all AWS regions.

■ Zero Trust Security
Default deny everything. Explicit grants required. Least privilege built in. Aligns with Zero Trust architecture.

■ Flexible Identity Types
Support humans (users), services (roles), federated users (SAML/OIDC), and AWSS services natively.

■■ Temporary Credentials
STS issues time-limited tokens. No permanent credentials for services. Automatic expiry reduces attack surface.

■ Multi-Account Control
AWS Organizations + SCPs enforce guardrails across 100s of accounts from a single management account.

■■ ABAC Support
Tag-based access control scales with your org. One policy covers unlimited teams/projects using tags.

■ Federation
Integrate with Active Directory, Okta, Azure AD, Google Workspace via SAML 2.0 or OIDC. SSO for enterprise.

■ Full Audit Trail
Every API call logged in CloudTrail. WHO did WHAT to WHICH resource WHEN and from WHERE. 90-day retention free.

■ MFA Options
Virtual MFA (TOTP apps), Hardware MFA tokens, FIDO2/WebAuthn security keys (most secure).

■ Policy Simulator
Test policies before deploying. Validate what actions a principal can or cannot perform — no real API calls.

■ IAM Access Analyzer
Identifies resources shared externally. Validates policies against 100+ checks. Generates least-privilege policies from CloudTrail.

9. Enterprise Architecture — Multi-Account IAM

```
AWS ORGANIZATIONS
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ MANAGEMENT
ACCOUNT (Root) ■ ■ → Billing, Organizations, Control Tower ■ ■ SCP: DenyAllExceptOrgs,
RequireIMDSv2, EnforceEncryption ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■ ■ ■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■ SECURITY ■ ■ SHARED ■
■ OU ■ ■ SERVICES ■ ■ Log Archive■ ■ Network Hub■ ■ Audit Acct ■ ■ DNS, TGW ■
■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ WORKLOADS OU ■ ■ ■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■ ■ ■ ■ PROD ACCT ■ ■ DEV ACCT ■ ■ ■
■ Strict SCPs■ ■ Loose SCPs ■ ■ ■ ■ IAM Roles: ■ ■ IAM Roles: ■ ■ ■ ■ AppRole ■ ■ DevRole
■ ■ ■ ■ ReadOnly ■ ■ Admin ■ ■ ■ ■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ IDENTITY CENTER (SSO)
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Corporate IdP (Okta/AD) → SAML 2.0 ■
■ ↓ ■ ■ AWS Identity Center ■ ■ Permission Sets → Assigned to Accounts■ ■ Users get roles in
each account ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ CROSS-ACCOUNT ACCESS
PATTERN DEV Account User → sts:AssumeRole → PROD Account Role Requires: Trust Policy (who) +
Permission Policy (what)
```

IAM Identity Center (AWS SSO) Flow:

AWS IAM Identity Center (formerly AWS SSO) is the recommended way to manage human access across multiple AWS accounts. Users authenticate once via your corporate IdP (Okta, Azure AD, Active Directory) and get temporary role credentials per account through the AWS access portal.

10. Proof of Concept (POC) — Cross-Account + ABAC

POC Goal: Allow developers in Account A to access S3 in Account B using ABAC

Step 1: In Account B (target) — Create Cross-Account Role

```
// Trust Policy — allow Account A to assume this role { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Principal": {"AWS": "arn:aws:iam::ACCOUNT_A_ID:root"}, "Action": "sts:AssumeRole", "Condition": { "StringEquals": {"sts:ExternalId": "SecureUniqueToken-XYZ789"}, "StringLike": {"aws:PrincipalArn": "arn:aws:iam::ACCOUNT_A_ID:user/*"} } }] } // Permission Policy — ABAC: access S3 buckets tagged with user team { "Version": "2012-10-17", "Statement": [ { "Sid": "ABACTaggedS3Access", "Effect": "Allow", "Action": ["s3:GetObject", "s3:PutObject", "s3:DeleteObject"], "Resource": "arn:aws:s3:::*/*", "Condition": {"StringEquals": {"s3:ResourceTag/Team": "${aws:PrincipalTag/Team}"}} }, { "Sid": "ListBuckets", "Effect": "Allow", "Action": "s3:ListAllMyBuckets", "Resource": "*" } ] }
```

Step 2: In Account A — Allow Users to AssumeRole

```
{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": "sts:AssumeRole", "Resource": "arn:aws:iam::ACCOUNT_B_ID:role/CrossAccountDevRole", "Condition": {"Bool": {"aws:MultiFactorAuthPresent": "true"}} }] }
```

Step 3: Tag IAM Users with their Team

```
# AWS CLI — tag IAM user with team aws iam tag-user \ --user-name john.smith \ --tags Key=Team,Value=DataEngineering # Tag S3 bucket with same team aws s3api put-bucket-tagging \ --bucket data-engineering-lake \ --tagging '{"TagSet": [{"Key":"Team","Value":"DataEngineering"}]}'
```

Step 4: Assume Role and Test

```
# Assume cross-account role (from Account A) aws sts assume-role \ --role-arn "arn:aws:iam::ACCOUNT_B_ID:role/CrossAccountDevRole" \ --role-session-name john-session \ --external-id "SecureUniqueToken-XYZ789" \ --duration-seconds 3600 # Export temporary credentials export AWS_ACCESS_KEY_ID="ASIA..." export AWS_SECRET_ACCESS_KEY="wJal..." export AWS_SESSION_TOKEN="AQoD..." # Verify access — should work (matching team tag) aws s3 ls s3://data-engineering-lake/ aws s3 cp test.txt s3://data-engineering-lake/test.txt # Verify access denied — wrong team bucket aws s3 ls s3://finance-team-bucket/ # AccessDenied ■
```

11. Standard Operating Procedures (SOP)

■ SOP-IAM-001: New Employee Onboarding

```
1. Create IAM user with email-based username (john.smith@company.com)
2. Add to appropriate IAM group (Engineering/Finance/HR)
3. Force password reset on first login
4. Send onboarding email with console login URL
5. User enables MFA within 24 hours (enforced by policy)
6. Provide access only to required accounts via IAM Identity Center
7. Document in HR/CMDB system
8. Schedule 90-day access review
```

■ SOP-IAM-002: Access Key Rotation

```
1. NEVER rotate root access keys — root should have NO access keys
2. Create new access key (user can have 2 active simultaneously)
aws iam create-access-key --user-name john.smith
3. Update all applications/scripts with new key
4. Wait 24-48 hours to ensure no requests using old key
5. Deactivate old key (not delete — allows rollback)
aws iam update-access-key --access-key-id AKIAXXXXXXXX --status Inactive
6. Monitor CloudTrail for 48 hours — no errors with old key
7. Delete old key: aws iam delete-access-key --access-key-id AKIAXXXXXXXX
8. Rotation frequency: 90 days max (use AWS Config rule: access-keys-rotated)
```

■ SOP-IAM-003: Employee Offboarding

```
IMMEDIATE (within 1 hour of termination):
1. Disable console access (deactivate password)
aws iam update-login-profile --user-name john.smith --no-password-reset-required
2. Deactivate all access keys
aws iam list-access-keys --user-name john.smith
aws iam update-access-key --access-key-id AKIAXX --status Inactive
3. Revoke all active sessions (force logout)
aws iam delete-user-policy ... (remove inline policies)
4. Remove from all IAM groups
5. Deactivate in IAM Identity Center

WITHIN 24 HOURS:
6. Audit and reassign any owned resources
7. Transfer S3 bucket ownership if applicable
8. Delete IAM user after confirming no dependencies
9. Document in security incident log
```

■ SOP-IAM-004: Security Incident — Compromised Credentials

```
IMMEDIATE RESPONSE:
1. Deactivate ALL access keys for compromised user/role IMMEDIATELY
2. Revoke all active sessions: IAM → Users → Revoke sessions
3. If root compromised: Call AWS Support immediately
4. Isolate affected resources with restrictive SCP or explicit deny policy

INVESTIGATION:
5. Query CloudTrail for actions taken with compromised credentials
aws cloudtrail lookup-events --lookup-attributes
AttributeKey=Username,AttributeValue=john.smith
6. Check all regions for new resources created (EC2, IAM users, etc)
7. Review GuardDuty findings

REMEDIATION:
8. Terminate unauthorized resources
9. Issue new credentials with fresh permissions
10. File security incident report within 72 hours
```

■ SOP-IAM-005: Quarterly Access Review

```
1. Export IAM credential report:
aws iam generate-credential-report
aws iam get-credential-report --output json | jq -r .Content | base64 -d
2. Identify: Users with no activity in 90+ days
3. Identify: Access keys not rotated in 90+ days
4. Identify: Users without MFA enabled
5. Run IAM Access Analyzer findings review
6. Review AWS Trusted Advisor IAM recommendations
7. Check for overly permissive policies (Action:* or Resource:*)
8. Remove unused IAM users, roles, and policies
9. Document findings and remediation in Security register
10. Present summary to CISO/Security team
```

12. IAM Security Best Practices

These are AWS-recommended hardening practices. Every item should be implemented in production accounts. Use AWS Config Rules to automate compliance checking.

■ CRITICAL

- ■ Enable MFA on root account immediately — use hardware security key.
- ■ Delete all root account access keys — root should NEVER have programmatic access.
- ■ Do NOT use root account for daily tasks — create individual IAM admin user.
- ■ Enable CloudTrail in all regions — log every API call.
- ■ Enable GuardDuty — detects compromised credentials and anomalous activity.

■ HIGH PRIORITY

- ■ Enforce MFA for all human IAM users — use IAM policy with MFA condition.
- ■ Apply least privilege — start minimum, add permissions only when needed.
- ■ Use IAM Roles for applications, NEVER embed access keys in code or configs.
- ■ Store credentials in Secrets Manager or SSM Parameter Store, never plain text.
- ■ Rotate access keys every 90 days — use AWS Config rule 'access-keys-rotated'.
- ■ Use IAM Access Analyzer to identify unintended external resource access.

■ STANDARD

- ■ Use IAM groups to assign permissions — never attach policies directly to users.
- ■ Use Customer Managed Policies over inline policies for auditability.
- ■ Enable IAM Password Policy: min 14 chars, require uppercase/numbers/symbols.
- ■ Use AWS Organizations SCPs to enforce guardrails across all accounts.
- ■ Use IAM Identity Center (SSO) for human access — no individual IAM users.
- ■ Tag all IAM resources for cost allocation and access control (ABAC).
- ■ Review and remove unused IAM users, roles, and policies quarterly.
- ■ Use IAM Policy Simulator to test policies before deployment.

■ ADVANCED

- ■ Implement Permission Boundaries when delegating IAM management to dev teams.
- ■ Use VPC Endpoint for IAM/STS to keep traffic within AWS network.
- ■ Enable IAM Access Analyzer policy validation in CI/CD pipelines.
- ■ Use service-linked roles for AWS services — they have minimal required permissions.
- ■ Set sts:ExternalId on cross-account roles to prevent confused deputy attacks.
- ■ Implement ABAC (tag-based access) for scalable multi-team environments.
- ■ Set up AWS Config rules: iam-root-access-key-check, iam-user-mfa-enabled, etc.

13. GitHub README (Professional Format)

````
# ■ AWS IAM Enterprise Setup

![AWS](https://img.shields.io/badge/AWS-IAM-blue?logo=amazonaws)
![License](https://img.shields.io/badge/license-MIT-green)
![Terraform](https://img.shields.io/badge/IaC-Terraform-purple)

Production-grade IAM setup: users, groups, roles, policies, and SCPs.

## ■ Prerequisites
- AWS CLI configured with admin credentials
- Terraform >= 1.5.0
- AWS Organizations enabled
- Root account MFA enabled (MANDATORY)

## ■■ What This Creates
- IAM Groups: Admin, Developer, ReadOnly, DataScience
- IAM Roles: CrossAccountRole, EC2AppRole, LambdaRole
- Custom Policies: LeastPrivilegeS3, ABACTagPolicy
- SCPs: DenyLeaveOrg, RequireEncryption, EnforceRegions
- Password policy enforcement
- MFA enforcement policy

## ■ Quick Start
```bash
git clone https://github.com/your-org/aws-iam-enterprise.git
cd aws-iam-enterprise
cp terraform.tfvars.example terraform.tfvars
terraform init && terraform plan && terraform apply
```

## ■ Project Structure
iam/
■■■ groups.tf          # IAM groups
■■■ roles.tf           # IAM roles and trust policies
■■■ policies/          # Custom policy JSON files
■■■ scps/              # Service Control Policies
■■■ password_policy.tf # Account password policy

## ■ Security Features
- Permission Boundaries on delegated roles
- ABAC with department/team tags
- MFA enforcement via condition policy
- IP restriction for admin access
- Automated key rotation reminder via Lambda

## ■ Cost
IAM is FREE. Costs only from services IAM protects.

## ■ Contributing
Fork → Feature branch → PR → Security review → Merge

## ■ License
````

```
MIT License
```

14. Top 25 IAM Interview Q&A;

Q1: What is AWS IAM?	Free global service controlling authentication (who) and authorization (what) for all AWS API calls. Default deny everything.
Q2: IAM Users vs Roles?	Users: permanent credentials for humans/apps. Roles: temporary STS credentials for services/cross-account. Always prefer roles for applications.
Q3: What is an IAM Policy?	JSON document defining Allow/Deny for AWS API actions on resources. Attached to users, groups, or roles. Evaluated at every API call.
Q4: Explicit Deny vs Implicit Deny?	Implicit deny = no policy allows it. Explicit deny = a policy specifically denies it. Explicit deny ALWAYS wins over any allow.
Q5: Policy evaluation order?	Default deny → Explicit deny → SCPs → Resource policies → Permission Boundaries → Session Policies → Identity policies.
Q6: What is STS?	Security Token Service. Issues temporary credentials (AccessKey + SecretKey + SessionToken) when assuming roles. Default 1hr, max 12hrs.
Q7: What are SCPs?	Service Control Policies in AWS Organizations. Set maximum permissions guardrails for accounts. Don't grant permissions — only restrict them.
Q8: What is a Permission Boundary?	Advanced IAM feature. Sets maximum permissions an entity can have. Used to safely delegate IAM management without privilege escalation.
Q9: What is IAM Access Analyzer?	Identifies resources shared externally (S3, roles, KMS). Validates policies. Generates least-privilege policies from CloudTrail. Free basic tier.
Q10: How to enforce MFA?	IAM policy with Condition: {"Bool": {"aws:MultiFactorAuthPresent": "true"}} on sensitive actions.
Q11: What is ABAC?	Attribute-Based Access Control. Uses tags on principals and resources in policy conditions. Scales dynamically — one policy covers unlimited teams.
Q12: Managed vs Inline Policies?	Managed: reusable, versioned, better auditing. Inline: embedded 1:1, deleted with principal. Always prefer Customer Managed Policies.

Q13: What is AssumeRole?	API call to STS to get temporary credentials for a role. Trust policy controls who can assume. Permission policy controls what they can do.
Q14: Cross-account access?	Create role in target account with trust policy pointing to source account. Source account users assume role. Optional: ExternalId for 3rd parties.
Q15: What is IAM Federation?	Allow external identities (Active Directory, Okta, Google) to assume IAM roles via SAML 2.0 or OIDC. No IAM users needed for federated users.
Q16: What is IAM Identity Center?	Formerly AWS SSO. Centrally manage human access to multiple AWS accounts and applications with a single sign-on portal.
Q17: How to rotate access keys safely?	Create new key → update apps → wait 48hrs → deactivate old key → monitor → delete old key. Users can have 2 active keys simultaneously.
Q18: Difference: authentication vs authorization?	Authentication = verifying WHO you are (credentials). Authorization = verifying WHAT you can do (policies).
Q19: What is the IAM Credential Report?	CSV download listing all IAM users and status of passwords, access keys, and MFA. Use for access review and compliance audits.
Q20: Service-Linked Roles?	Special roles with predefined permissions for specific AWS services (e.g., AWSServiceRoleForAutoScaling). Created automatically by services.
Q21: What is sts:ExternalId?	Optional condition in cross-account trust policy. Prevents confused deputy attack where 3rd-party service is tricked into accessing your account.
Q22: Resource-based vs identity-based policies?	Identity-based: attached to user/group/role (who can do what). Resource-based: attached to resource (who can access this resource).
Q23: What is IAM Policy Simulator?	Console tool to test what actions a user/role/group can or cannot perform across services. Tests policies without real API calls.
Q24: How to prevent privilege escalation?	Don't grant iam:CreatePolicy, iam:AttachUserPolicy, iam:PassRole without Permission Boundaries. Audit with IAM Access Analyzer.
Q25: Best IAM practices summary?	MFA everywhere, Least privilege, No root usage, Roles over users, No hardcoded credentials, Regular access reviews, CloudTrail enabled, Rotate keys.

■ YOU ARE NOW IAM CERTIFIED READY ■