AWS IAM: Identity and Access Management

Secure Access Control for AWS Resources - From Basic to Associate Level

Prepared By: Rashi Rana
AWS Corporate Trainer

What is AWS IAM

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

Core Purpose

- **Authentication:** Verify who is trying to access AWS resources
- **Authorization:** Determine what actions authenticated users can perform
- **Access Control:** Manage permissions to AWS services and resources
- **Security:** Implement principle of least privilege

Key Benefits

Centralized Control

Manage access to all AWS services from a single location

Fine-grained Permissions

Control access at the resource and action level

Secure by Default

No Additional Cost

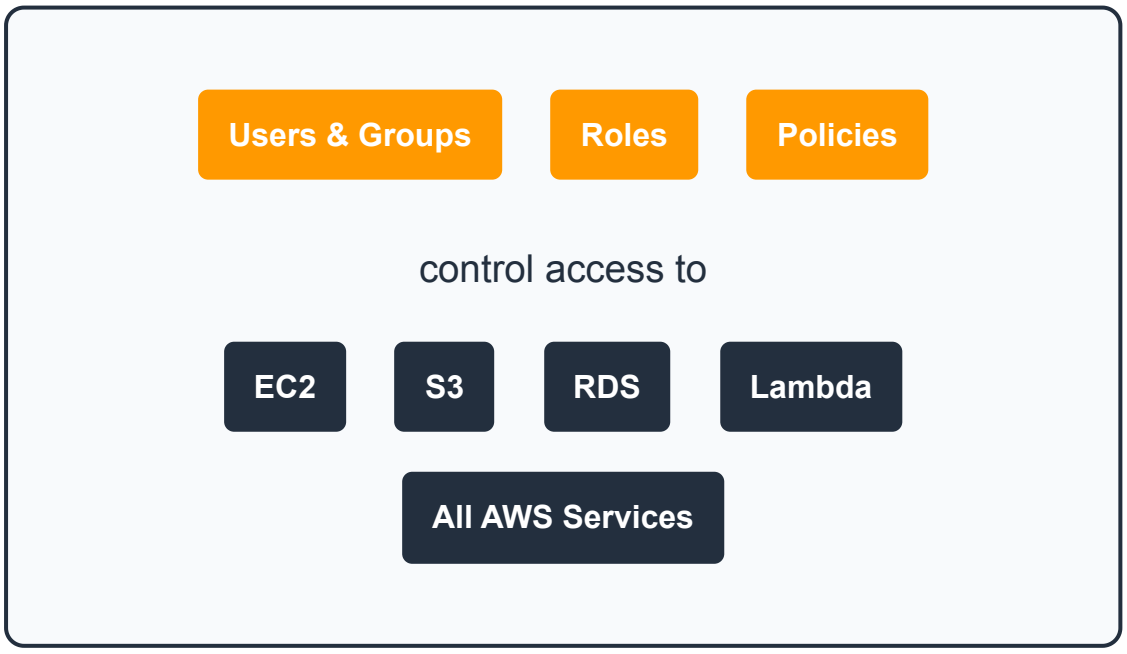
Deny access by default, grant only necessary permissions

IAM is provided at no additional charge

IAM vs Traditional Access Control

Aspect	Traditional Systems	AWS IAM
Scope	Single system or application	All AWS services and resources
Granularity	Role-based, limited flexibility	Action and resource-level control
Integration	Manual setup for each system	Native integration with all AWS services
Scalability	Limited by system capacity	Scales with AWS infrastructure

IAM Architecture Overview



Real-World Use Cases

- **Employee Access:** Grant developers access to development resources only
- **Application Security:** Allow applications to access only required AWS services
- **Cross-Account Access:** Enable secure access between different AWS accounts
- **Temporary Access:** Provide time-limited access for contractors or partners
- **Compliance:** Meet regulatory requirements for access control and auditing

> **Key Principle:** IAM follows the **principle of least privilege** - users and applications should have only the minimum permissions necessary to perform their tasks.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Core Components

AWS IAM consists of four main components that work together to provide comprehensive access control: **Users, Groups, Roles, and Policies**.

The Four Pillars of IAM

Users

Individual identities with unique credentials for accessing AWS resources

Groups

Collections of users that share common permissions and access patterns

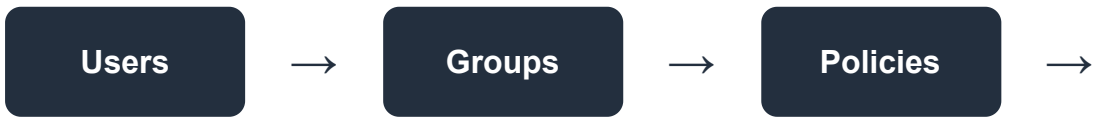
Roles

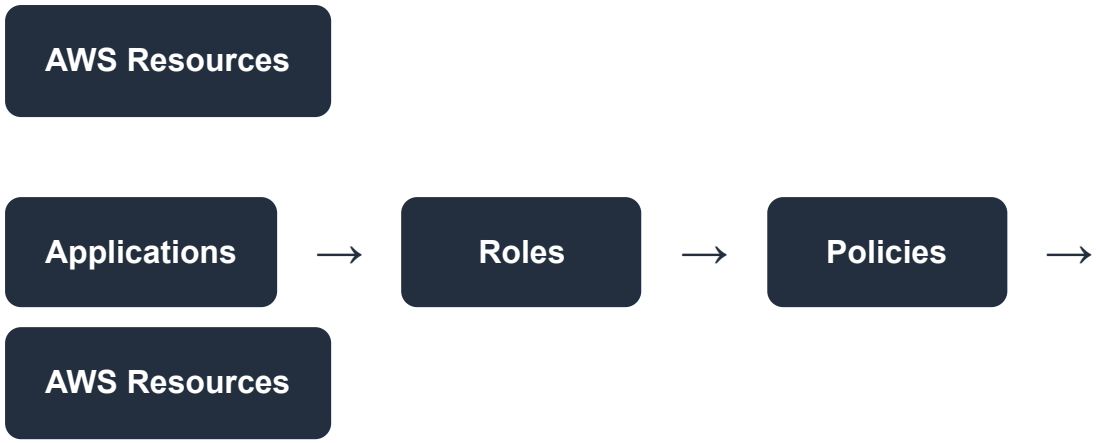
Temporary identities that can be assumed by users, applications, or services

Policies

JSON documents that define permissions and access rules

Component Relationships





Component Comparison

Component	Purpose	Best Used For	Credentials
Users	Individual access	Human users, long-term access	Username/Password, Access Keys
Groups	Organize users	Team-based permissions	Inherit from users
Roles	Temporary access	Applications, cross-account access	Temporary security tokens
Policies	Define permissions	Access control rules	Not applicable

IAM Hierarchy

```
AWS Account (Root) ├── IAM Users │ ├── User 1 (with attached policies) │ ├── User 2 (member of groups) │ └── User N ├── IAM Groups │ ├── Developers Group (with policies) │ ├── Admins Group (with policies) │ └── ReadOnly Group (with policies) ├── IAM Roles │ ├── EC2-S3-Access-Role │ ├── Lambda-Execution-Role │ └── Cross-Account-Role
```

```
└── IAM Policies ├── Managed Policies (AWS/Customer) └── Inline Policies (attached directly)
```

Best Practices for Component Usage

- **Users:** Create individual users for each person, avoid sharing credentials
- **Groups:** Use groups to assign permissions to multiple users efficiently
- **Roles:** Prefer roles over users for applications and services
- **Policies:** Use managed policies for reusability, inline for specific cases

> **Security Note:** Never use the AWS root account for daily tasks. Create IAM users with appropriate permissions instead.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Users

IAM Users represent individual identities within your AWS account. Each user has unique credentials and can be granted specific permissions to access AWS resources.

User Characteristics

- **Unique Identity:** Each user has a unique name within the AWS account
- **Permanent Credentials:** Long-term access keys and passwords
- **Individual Permissions:** Can have policies attached directly
- **Group Membership:** Can belong to multiple groups
- **Console and API Access:** Can access both AWS Management Console and APIs

User Credential Types

Console Password

For AWS Management Console access with optional MFA

Access Keys

For programmatic access via AWS CLI, SDKs, and APIs

SSH Keys

Server Certificates

For AWS CodeCommit Git repository access

For SSL/TLS certificates used with AWS services

User Creation Process

1. **Create User:** Define username and select credential types
2. **Set Permissions:** Attach policies directly or add to groups
3. **Configure Access:** Set up console password and/or access keys
4. **Enable MFA:** Add multi-factor authentication for enhanced security
5. **Provide Credentials:** Securely share login information with the user

User Management Best Practices

Practice	Description	Benefit
Individual Users	Create separate user for each person	Accountability and audit trails
Strong Passwords	Enforce password policy requirements	Prevent unauthorized access
MFA Enabled	Require multi-factor authentication	Additional security layer
Regular Rotation	Rotate access keys periodically	Limit exposure of compromised keys

Practice	Description	Benefit
Least Privilege	Grant minimum required permissions	Reduce security risk

User Limits and Considerations

- **Account Limit:** 5,000 users per AWS account (can be increased)
- **Group Membership:** User can belong to up to 10 groups
- **Managed Policies:** Up to 10 managed policies per user
- **Inline Policy Size:** Maximum 2,048 characters per inline policy
- **Access Keys:** Maximum 2 access keys per user

Common User Scenarios

Developer Access

Console and programmatic access to development resources

Admin User

Full administrative access with MFA requirement

Service Account

Programmatic access only for applications (consider roles instead)

Auditor Access

Read-only access for compliance and monitoring

User Security Considerations

Security Risks:

- Long-term credentials can be compromised
- Shared user accounts reduce accountability
- Unused users with active credentials pose security risks
- Overprivileged users violate least privilege principle

Best Practice: Use IAM roles for applications and services instead of creating users with long-term credentials. Reserve users for human access only.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Groups

IAM Groups are collections of IAM users that make it easier to manage permissions for multiple users. Groups allow you to specify permissions for multiple users at once.

Group Characteristics

- **User Collections:** Contain multiple IAM users with similar access needs
- **Policy Attachment:** Policies attached to groups apply to all members
- **No Credentials:** Groups themselves don't have credentials
- **Nested Groups:** Groups cannot contain other groups
- **Multiple Membership:** Users can belong to multiple groups

Benefits of Using Groups

Simplified Management

Manage permissions for multiple users simultaneously

Consistency

Ensure users with similar roles have identical permissions

Scalability

Organization

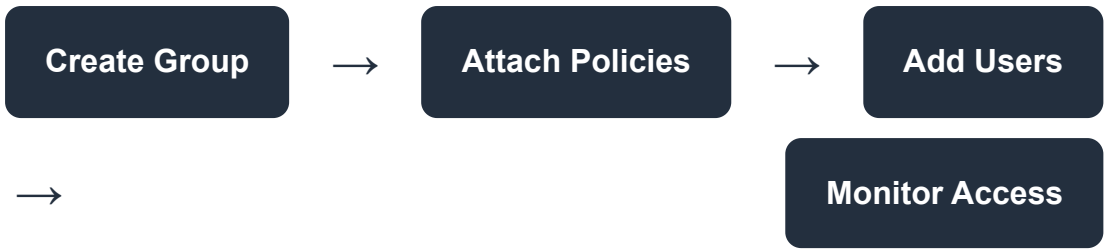
Easy to add/remove users and modify group permissions

Logical grouping based on job functions or departments

Common Group Patterns

Group Name	Purpose	Typical Permissions	Members
Administrators	Full AWS access	AdministratorAccess	IT administrators, DevOps leads
Developers	Development resources	EC2, S3, Lambda, CloudWatch	Software developers, engineers
ReadOnlyUsers	View-only access	ReadOnlyAccess	Auditors, managers, analysts
DatabaseAdmins	Database management	RDS, DynamoDB, ElastiCache	Database administrators
SecurityTeam	Security services	IAM, CloudTrail, GuardDuty	Security engineers, compliance

Group Management Workflow



Group Limits and Constraints

- **Account Limit:** 300 groups per AWS account
- **User Membership:** User can belong to up to 10 groups
- **Managed Policies:** Up to 10 managed policies per group
- **No Nesting:** Groups cannot contain other groups
- **No Direct Login:** Cannot log in as a group

Group vs Individual User Permissions

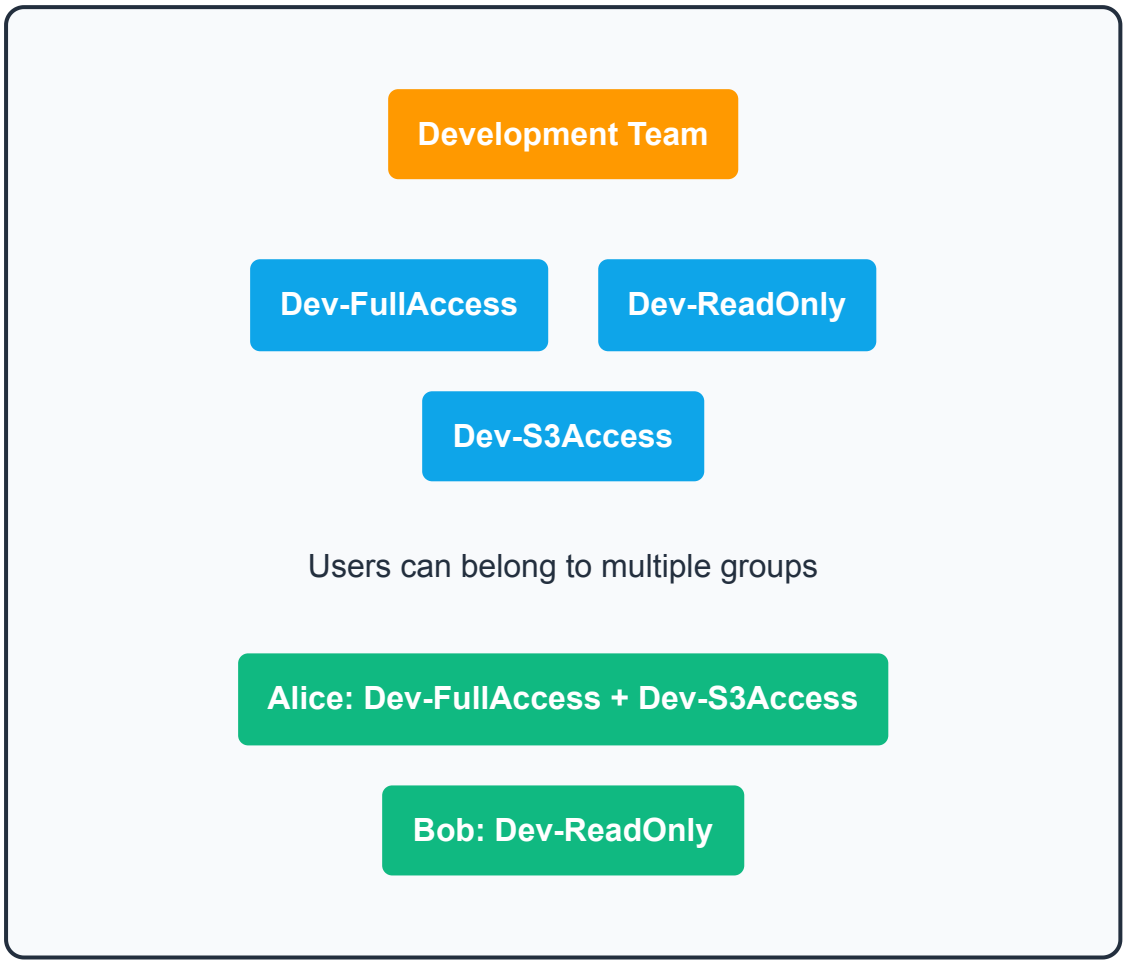
```
Permission Evaluation Order: 1. Explicit Deny in any policy → DENY 2. Explicit Allow in any policy → ALLOW 3. No explicit Allow → DENY (implicit) User's Effective Permissions = User's Inline Policies + User's Managed Policies + All Group Policies (from all groups user belongs to)
```

Best Practices for Groups

- **Job Function Groups:** Create groups based on job roles and responsibilities
- **Environment Groups:** Separate groups for dev, staging, and production access
- **Managed Policies:** Use AWS managed policies when possible for common permissions
- **Regular Review:** Periodically review group memberships and permissions

- **Naming Convention:** Use clear, descriptive names for groups

Example Group Structure



Recommendation: Always use groups to assign permissions to users. This makes permission management much easier and more maintainable as your organization grows.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Roles

IAM Roles are temporary identities that can be assumed by users, applications, or AWS services. Roles provide temporary security credentials and are the preferred way to grant access to AWS resources.

Role Characteristics

- **Temporary Credentials:** Provide short-term access tokens instead of permanent credentials
- **Assumable:** Can be assumed by users, applications, or AWS services
- **No Passwords:** Roles don't have passwords or access keys
- **Cross-Account:** Can be assumed across different AWS accounts
- **Federated Access:** Support for external identity providers

Types of IAM Roles

Service Roles

For AWS services like EC2, Lambda, or ECS to access other AWS services

Cross-Account Roles

Allow users from one AWS account to access resources in another account

Identity Provider Roles

For federated users from external identity providers (SAML, OIDC)

Instance Profiles

Special container for roles that can be attached to EC2 instances

Role Components

Component	Purpose	Example
Trust Policy	Defines who can assume the role	EC2 service, specific AWS account
Permission Policies	Defines what the role can do	S3 read access, DynamoDB write access
Role ARN	Unique identifier for the role	arn:aws:iam::123456789012:role/MyRole
Session Duration	How long credentials are valid	1 hour to 12 hours (default: 1 hour)

Role Assumption Process

Request → **Trust Policy Check** →

Common Role Use Cases

- **EC2 Instance Access:** Allow EC2 instances to access S3, DynamoDB, etc.
- **Lambda Function Execution:** Grant Lambda functions permissions to AWS services
- **Cross-Account Access:** Allow users from Account A to access resources in Account B
- **Application Access:** Enable applications to access AWS services without hardcoded credentials
- **Federated Access:** Allow corporate users to access AWS using existing credentials

Trust Policy Example

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": {
"Service": "ec2.amazonaws.com"
},
"Action": "sts:AssumeRole"
},
{
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::123456789012:user/Alice"
},
"Action": "sts:AssumeRole",
"Condition": {
"Bool": {
"aws:MultiFactorAuthPresent": "true"
}
```

```
  }
  }
  ]
  }
```

Roles vs Users Comparison

Aspect	IAM Users	IAM Roles
Credentials	Permanent (passwords, access keys)	Temporary (security tokens)
Best For	Human users	Applications, services, cross-account
Security	Risk of credential exposure	Automatic credential rotation
Scalability	Limited by user count	Unlimited assumptions

> **Best Practice:** Use IAM roles instead of users for applications, services, and cross-account access. Roles provide better security through temporary credentials and automatic rotation.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Policies

IAM Policies are JSON documents that define permissions. They specify what actions are allowed or denied on which resources and under what conditions.

Policy Types

AWS Managed Policies

Pre-built policies created and maintained by AWS

Customer Managed Policies

Custom policies created and maintained by you

Inline Policies

Policies directly embedded in a single user, group, or role

Resource-Based Policies

Policies attached to resources (S3 buckets, KMS keys, etc.)

Policy Structure

```
{
"Version": "2012-10-17",
"Statement": [
```

```
{
"Sid": "AllowS3ReadAccess",
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:ListBucket"
],
"Resource": [
"arn:aws:s3:::my-bucket",
"arn:aws:s3:::my-bucket/*"
],
"Condition": {
"StringEquals": {
"s3:prefix": "documents/"
}
}
}
]
}
```

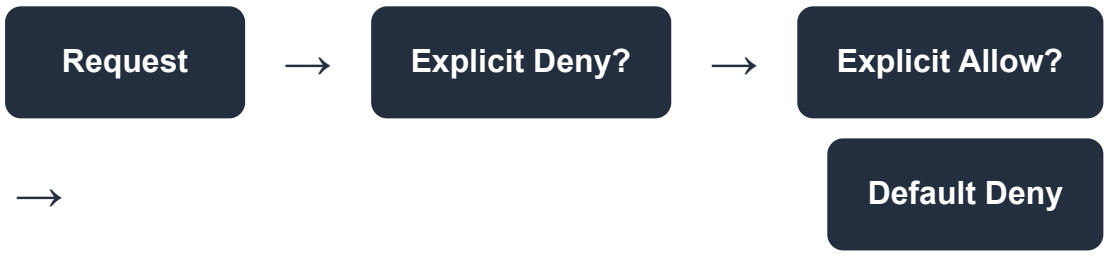
Policy Elements

Element	Required	Description	Example
Version	Yes	Policy language version	"2012-10-17"
Statement	Yes	Array of permission statements	[{...}, {...}]
Effect	Yes	Allow or Deny	"Allow", "Deny"
Action	Yes*	API actions to allow/deny	"s3:GetObject"
Resource	Yes*	Resources the action applies to	"arn:aws:s3:::bucket/*"

Element	Required	Description	Example
Condition	No	When the policy applies	IP address, time, MFA
Principal	No**	Who the policy applies to	User, role, account

* Either Action or NotAction required
** Required for resource-based policies

Policy Evaluation Logic



Common Policy Patterns

- **Service-Specific Access:** Grant access to specific AWS services
- **Resource-Specific Access:** Limit access to specific resources
- **Conditional Access:** Apply conditions based on time, IP, MFA, etc.
- **Cross-Account Access:** Allow access from other AWS accounts
- **Temporary Access:** Time-limited permissions

Policy Best Practices

Least Privilege

Use Managed Policies

Grant only the minimum permissions needed

Prefer AWS managed policies when available

Version Control

Track changes to custom policies

Regular Review

Periodically review and update policies

Example Policies

```
// Read-only S3 access
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:ListBucket"
],
"Resource": "*"
}
]
}

// EC2 start/stop with conditions
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"ec2:StartInstances",
"ec2:StopInstances"
],
"Resource": "*",
"Condition": {
```

```
"StringEquals": {
"ec2:ResourceTag/Environment": "Development"
}
}
}
]
}
```

Important: Always test policies in a safe environment before applying them to production. Use the IAM Policy Simulator to validate policy behavior.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Security Features

AWS IAM provides multiple layers of security features to protect your AWS resources and ensure secure access management.

Multi-Factor Authentication (MFA)

Virtual MFA

Software-based MFA using apps like Google Authenticator or Authy

Hardware MFA

Physical devices like YubiKey or Gemalto tokens

SMS MFA

Text message-based authentication (less secure, not recommended)

U2F Security Keys

FIDO U2F compatible hardware security keys

Password Policies

- **Minimum Length:** Require passwords of specific minimum length
- **Character Requirements:** Require uppercase, lowercase, numbers, symbols
- **Password Expiration:** Force regular password changes

- **Password Reuse:** Prevent reusing previous passwords
- **Account Lockout:** Lock accounts after failed login attempts

Access Keys Security

Security Measure	Description	Best Practice
Key Rotation	Regularly change access keys	Rotate every 90 days or less
Key Deactivation	Disable unused access keys	Deactivate immediately when not needed
Last Used Information	Track when keys were last used	Monitor and remove unused keys
Secure Storage	Store keys securely	Use AWS Secrets Manager or similar

CloudTrail Integration

- **API Logging:** Log all IAM API calls and changes
- **User Activity:** Track user login and access patterns
- **Policy Changes:** Monitor modifications to policies and permissions
- **Compliance:** Maintain audit trails for regulatory requirements

IAM Access Analyzer

External Access Detection

Policy Validation

Validate policies against AWS best practices

Identify resources accessible from outside your account

Unused Access

Find unused permissions and roles

Policy Generation

Generate policies based on actual usage

Credential Report

```
IAM Credential Report includes: - User creation date - Password last used - Password last changed - Password next rotation - MFA active status - Access key 1 & 2 status - Access key last used date - Access key last used service - Certificate status and dates
```

Security Token Service (STS)

- **Temporary Credentials:** Issue short-term access credentials
- **Role Assumption:** Enable secure role switching
- **Federation:** Support external identity providers
- **Cross-Account Access:** Secure access between AWS accounts

IAM Conditions for Enhanced Security

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
```

```
"Action": "*",
"Resource": "*",
"Condition": {
"Bool": {
"aws:MultiFactorAuthPresent": "true"
},
"IpAddress": {
"aws:SourceIp": "203.0.113.0/24"
},
"DateGreaterThan": {
"aws:CurrentTime": "2023-01-01T00:00:00Z"
},
"DateLessThan": {
"aws:CurrentTime": "2023-12-31T23:59:59Z"
}
}
}
]
}
```

Security Recommendation: Enable MFA for all users with console access, especially those with administrative privileges. Use IAM Access Analyzer regularly to review and optimize permissions.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Best Practices

Following IAM best practices is crucial for maintaining a secure and well-managed AWS environment. These practices help minimize security risks and improve operational efficiency.

Fundamental Security Principles

Principle of Least Privilege

Grant only the minimum permissions necessary to perform required tasks

Defense in Depth

Use multiple layers of security controls and monitoring

Zero Trust Model

Never trust, always verify - authenticate and authorize every request

Regular Auditing

Continuously monitor and review access patterns and permissions

Account and Root User Security

- **Secure Root Account:** Enable MFA and use strong password for root account
- **Avoid Root Usage:** Don't use root account for daily tasks

- **Root Access Keys:** Never create access keys for root account
- **Account Contacts:** Keep account contact information up to date
- **Billing Alerts:** Set up billing alerts to detect unusual activity

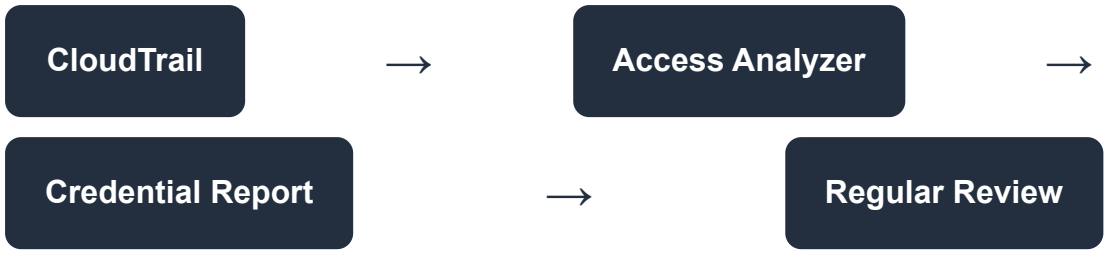
User and Access Management

Practice	Implementation	Benefit
Individual Users	Create separate IAM user for each person	Accountability and audit trails
Use Groups	Assign permissions through groups	Simplified permission management
Strong Passwords	Enforce password policy requirements	Prevent brute force attacks
Enable MFA	Require MFA for console access	Additional authentication layer
Regular Reviews	Audit user access and permissions	Remove unnecessary access

Role and Policy Best Practices

- **Prefer Roles:** Use roles instead of users for applications and services
- **Managed Policies:** Use AWS managed policies when available
- **Policy Versioning:** Version control custom policies
- **Specific Resources:** Avoid using "*" for resources when possible
- **Condition Usage:** Use conditions to restrict access further
- **Policy Testing:** Test policies before production deployment

Monitoring and Auditing



Operational Best Practices

Credential Rotation

Regularly rotate access keys and passwords

Remove Unused Access

Deactivate unused users, roles, and policies

Monitor Activity

Set up alerts for suspicious activities

Document Policies

Maintain clear documentation for custom policies

Common Security Mistakes to Avoid

Security Anti-Patterns:

- **Overprivileged Users:** Granting more permissions than necessary
- **Shared Credentials:** Multiple people using the same user account

- **Hardcoded Keys:** Embedding access keys in application code
- **No MFA:** Not enabling MFA for privileged accounts
- **Unused Resources:** Keeping inactive users and roles
- **Broad Policies:** Using "*" for actions and resources unnecessarily

IAM Security Checklist

1. **Root Account:** Secure root account with MFA, avoid daily use
2. **Users:** Create individual users, enforce strong passwords
3. **MFA:** Enable MFA for all console users
4. **Groups:** Use groups to assign permissions
5. **Roles:** Use roles for applications and cross-account access
6. **Policies:** Apply least privilege principle
7. **Monitoring:** Enable CloudTrail and regular auditing
8. **Maintenance:** Regular cleanup and access reviews

Compliance and Governance

- **Access Reviews:** Quarterly review of user access and permissions
- **Policy Documentation:** Document business justification for policies
- **Change Management:** Implement approval process for IAM changes
- **Incident Response:** Have procedures for compromised credentials
- **Training:** Regular security awareness training for users

Golden Rule: Start with no permissions and add only what's needed. It's easier to grant additional permissions than to remove excessive ones without breaking functionality.

Prepared By: Rashi Rana
AWS Corporate Trainer

Lab: Creating IAM Users and Groups

Hands-on Lab: IAM User and Group Management

Create IAM users, organize them into groups, and assign appropriate permissions using the AWS Management Console.

Lab Objectives

- Create IAM users with different access types
- Set up IAM groups with specific permissions
- Configure password policies and MFA
- Test user access and permissions

Prerequisites

- AWS account with administrative access
- Access to AWS Management Console
- Mobile device for MFA setup (optional)

Step 1: Configure Account Password Policy

1. **Navigate to IAM:** Go to IAM service in AWS Console

2. **Account Settings:** Click on "Account settings" in left navigation
3. **Password Policy:** Click "Change password policy"
4. **Configure Policy:** Set minimum length, character requirements
5. **Save Changes:** Apply the new password policy

Step 2: Create IAM Groups

```
# Groups to create: 1. Administrators - Policy: AdministratorAccess 2. Developers - Policies: AmazonEC2FullAccess, AmazonS3FullAccess 3. ReadOnlyUsers - Policy: ReadOnlyAccess 4. DatabaseAdmins - Policies: AmazonRDSFullAccess, AmazonDynamoDBFullAccess
```

Step 3: Create IAM Users

User Name	Access Type	Group	MFA Required
admin-user	Console + Programmatic	Administrators	Yes
dev-alice	Console + Programmatic	Developers	Yes
readonly-bob	Console only	ReadOnlyUsers	No
dba-charlie	Console + Programmatic	DatabaseAdmins	Yes

Step 4: User Creation Process

1. **Add User:** Click "Add user" in IAM Users section

2. **User Details:** Enter username and select access type
3. **Permissions:** Add user to appropriate group
4. **Tags:** Add tags for organization (optional)
5. **Review:** Review settings and create user
6. **Credentials:** Download credentials CSV file

Step 5: Configure MFA for Admin User

1. **Select User:** Click on admin-user in users list
2. **Security Credentials:** Go to Security credentials tab
3. **Assign MFA:** Click "Assign MFA device"
4. **Device Type:** Choose Virtual MFA device
5. **Setup App:** Use Google Authenticator or similar app
6. **Verify:** Enter two consecutive MFA codes

Step 6: Test User Access

```
# Test console access: 1. Open incognito/private browser window 2. Go to AWS Console sign-in page 3. Use account ID and user credentials 4. Verify access to appropriate services # Test programmatic access: aws configure --profile dev-alice aws s3 ls --profile dev-alice aws ec2 describe-instances --profile dev-alice # Test MFA requirement: aws sts get-session-token --serial-number arn:aws:iam::ACCOUNT:mfa/admin-user --token-code 123456
```

Expected Results

- **Admin User:** Full access to all AWS services with MFA

- **Developer:** Access to EC2 and S3, denied for other services
- **ReadOnly User:** Can view resources but cannot modify
- **DBA User:** Access to RDS and DynamoDB only

Verification Checklist

- ✓ Password policy is enforced for all users
- ✓ Users can only access services permitted by their groups
- ✓ MFA is working for designated users
- ✓ Access keys work for programmatic access
- ✓ Users cannot perform actions outside their permissions

Prepared By: Rashi Rana
AWS Corporate Trainer

Lab: Working with IAM Roles

Hands-on Lab: IAM Roles and Cross-Account Access

Create and configure IAM roles for different use cases including EC2 instance roles and cross-account access.

Lab Objectives

- Create service roles for EC2 instances
- Set up cross-account access roles
- Configure role assumption and trust policies
- Test role functionality and permissions

Step 1: Create EC2 Service Role

1. **Create Role:** Go to IAM → Roles → Create role
2. **Trusted Entity:** Select "AWS service" → "EC2"
3. **Permissions:** Attach AmazonS3ReadOnlyAccess policy
4. **Role Name:** EC2-S3-ReadOnly-Role
5. **Create Role:** Review and create the role

Step 2: Create Custom Policy for Role

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "AllowS3BucketAccess",
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject"
],
"Resource": "arn:aws:s3:::my-app-bucket/*"
},
{
"Sid": "AllowS3BucketList",
"Effect": "Allow",
"Action": "s3:ListBucket",
"Resource": "arn:aws:s3:::my-app-bucket"
},
{
"Sid": "AllowCloudWatchLogs",
"Effect": "Allow",
"Action": [
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:PutLogEvents"
],
"Resource": "*"
}
]
}
```

Step 3: Create Lambda Execution Role

```
# Role Configuration: Role Name: Lambda-S3-CloudWatch-Role Trusted Entity: lambda.amazonaws.com Policies: - AWSLambdaBasicExecutionRole (AWS Managed) - Custom policy created in Step 2 # Trust Policy (automatically created): { "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "Service":
```

```
"lambda.amazonaws.com" }, "Action": "sts:AssumeRole" } ]
}
```

Step 4: Create Cross-Account Access Role

1. **Create Role:** Select "Another AWS account" as trusted entity
2. **Account ID:** Enter the external AWS account ID
3. **External ID:** Add external ID for additional security
4. **MFA Required:** Optionally require MFA for role assumption
5. **Permissions:** Attach appropriate policies
6. **Role Name:** CrossAccount-ReadOnly-Role

Step 5: Test EC2 Instance Role

```
# Launch EC2 instance with role: 1. Launch EC2 instance
2. In "Configure Instance Details": - IAM role: EC2-S3-
ReadOnly-Role 3. Connect to instance via SSH 4. Test AWS
CLI access: # On EC2 instance: aws sts get-caller-
identity aws s3 ls aws s3 cp s3://my-app-bucket/test.txt
./ aws ec2 describe-instances # Should fail - no EC2
permissions
```

Step 6: Test Role Assumption

```
# Assume role from CLI: aws sts assume-role \ --role-arn
arn:aws:iam::ACCOUNT-ID:role/CrossAccount-ReadOnly-Role
\ --role-session-name test-session \ --external-id
unique-external-id # Use temporary credentials: export
AWS_ACCESS_KEY_ID=ASIA... export
AWS_SECRET_ACCESS_KEY=... export AWS_SESSION_TOKEN=... #
Test access with assumed role: aws sts get-caller-
identity aws s3 ls
```

Step 7: Create Role for Cross-Service Access

Role Name	Trusted Service	Use Case	Policies
ECS-Task-Role	ecs-tasks.amazonaws.com	ECS container access	S3, DynamoDB access
CodeBuild-Role	codebuild.amazonaws.com	Build project access	S3, ECR, CloudWatch
API-Gateway-Role	apigateway.amazonaws.com	API Gateway logging	CloudWatch Logs

Role Best Practices Demonstrated

- **Specific Permissions:** Roles have only necessary permissions
- **Service-Specific:** Different roles for different services
- **External ID:** Used for cross-account access security
- **Session Names:** Descriptive names for role sessions
- **Time Limits:** Temporary credentials with expiration

Troubleshooting Common Issues

Common Problems:

- **Trust Policy:** Ensure correct service or account in trust policy
- **Permissions:** Verify role has necessary permissions
- **Instance Profile:** EC2 roles need instance profile attachment
- **External ID:** Must match exactly for cross-account access

- **MFA:** Provide MFA token if required by trust policy

Lab Complete: You've successfully created and tested various IAM roles for different AWS services and use cases. Roles provide secure, temporary access without long-term credentials.

Prepared By: Rashi Rana
AWS Corporate Trainer

Lab: IAM Policies and Permissions

Hands-on Lab: Creating and Testing IAM Policies

Create custom IAM policies with various conditions and test their effectiveness using the IAM Policy Simulator.

Lab Objectives

- Create custom managed policies
- Use policy conditions for enhanced security
- Test policies with IAM Policy Simulator
- Understand policy evaluation logic

Step 1: Create S3 Bucket Access Policy

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "AllowListBucket",
"Effect": "Allow",
"Action": "s3:ListBucket",
"Resource": "arn:aws:s3:::my-company-documents",
"Condition": {
"StringLike": {
"s3:prefix": [
"public/*",
```

```
"shared/*"
]
}
}
},
{
"Sid": "AllowObjectAccess",
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:PutObject"
],
"Resource": [
"arn:aws:s3:::my-company-documents/public/*",
"arn:aws:s3:::my-company-documents/shared/*"
]
},
{
"Sid": "DenyDeleteOperations",
"Effect": "Deny",
"Action": [
"s3:DeleteObject",
"s3:DeleteBucket"
],
"Resource": [
"arn:aws:s3:::my-company-documents",
"arn:aws:s3:::my-company-documents/*"
]
}
]
}
```

Step 2: Create Time-Based Access Policy

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "AllowBusinessHoursAccess",
"Effect": "Allow",
"Action": [
"ec2:StartInstances",
```

```
"ec2:StopInstances",
"ec2:RebootInstances"
],
"Resource": "*",
"Condition": {
"DateGreaterThan": {
"aws:RequestedRegion": "us-east-1"
},
"IpAddress": {
"aws:SourceIp": [
"203.0.113.0/24",
"198.51.100.0/24"
]
},
"StringEquals": {
"ec2:ResourceTag/Environment": "Development"
}
}
}
]
}
```

Step 3: Create MFA-Required Policy

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "AllowViewAccountInfo",
"Effect": "Allow",
"Action": [
"iam:GetAccountPasswordPolicy",
"iam:ListVirtualMFADevices"
],
"Resource": "*"
},
{
"Sid": "AllowManageOwnPasswords",
"Effect": "Allow",
"Action": [
"iam:ChangePassword",
"iam:GetUser"
```

```
],
"Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
"Sid": "AllowManageOwnMFA",
"Effect": "Allow",
"Action": [
"iam:CreateVirtualMFADevice",
"iam:EnableMFADevice",
"iam:ResyncMFADevice"
],
"Resource": [
"arn:aws:iam::*:mfa/${aws:username}",
"arn:aws:iam::*:user/${aws:username}"
]
},
{
"Sid": "DenyAllExceptUnlessSignedInWithMFA",
"Effect": "Deny",
"NotAction": [
"iam:CreateVirtualMFADevice",
"iam:EnableMFADevice",
"iam:GetUser",
"iam:ListMFADevices",
"iam:ListVirtualMFADevices",
"iam:ResyncMFADevice",
"sts:GetSessionToken"
],
"Resource": "*",
"Condition": {
"BoolIfExists": {
"aws:MultiFactorAuthPresent": "false"
}
}
}
]
}
```

Step 4: Test Policies with Policy Simulator

1. **Access Simulator:** Go to IAM → Policy Simulator
2. **Select User/Role:** Choose the entity to test

3. **Select Service:** Choose AWS service (e.g., S3, EC2)
4. **Select Actions:** Choose specific actions to test
5. **Specify Resources:** Enter resource ARNs
6. **Add Conditions:** Set context conditions if needed
7. **Run Simulation:** Execute the simulation

Step 5: Policy Testing Scenarios

Test Scenario	Action	Resource	Expected Result
S3 Read Access	s3:GetObject	arn:aws:s3:::my-company-documents/public/file.txt	Allow
S3 Delete Attempt	s3:DeleteObject	arn:aws:s3:::my-company-documents/public/file.txt	Deny
EC2 Start (Wrong IP)	ec2:StartInstances	arn:aws:ec2:us-east-1:*:instance/*	Deny
IAM without MFA	iam:CreateUser	*	Deny

Step 6: Create Policy with Variables

```
{
"Version": "2012-10-17",
"Statement": [
```

```
{
"Sid": "AllowUserToManageTheirOwnProfile",
"Effect": "Allow",
"Action": [
"iam:GetUser",
"iam:ChangePassword",
"iam:CreateAccessKey",
"iam:DeleteAccessKey",
"iam:ListAccessKeys",
"iam:UpdateAccessKey"
],
"Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
"Sid": "AllowUserToListTheirOwnMFA",
"Effect": "Allow",
"Action": [
"iam:ListMFADevices"
],
"Resource": [
"arn:aws:iam::*:user/${aws:username}",
"arn:aws:iam::*:mfa/${aws:username}"
]
}
]
}
```

Step 7: Validate Policy Syntax

```
# Using AWS CLI to validate policy: aws iam simulate-principal-policy \ --policy-source-arn arn:aws:iam::123456789012:user/testuser \ --action-names s3:GetObject \ --resource-arns arn:aws:s3:::my-company-documents/public/test.txt # Using policy validator: aws iam create-policy \ --policy-name TestPolicy \ --policy-document file://policy.json \ --dry-run
```

Common Policy Conditions

IP Address

aws:SourceIp - Restrict by IP range

Time-based

aws:CurrentTime - Time-based access

MFA

aws:MultiFactorAuthPresent - Require MFA

Resource Tags

aws:ResourceTag - Tag-based access

Policy Troubleshooting Tips

- **JSON Syntax:** Validate JSON format and structure
- **ARN Format:** Ensure correct ARN syntax for resources
- **Action Names:** Use correct service:action format
- **Condition Logic:** Understand AND/OR logic in conditions
- **Policy Size:** Stay within size limits (2048 chars for inline)

Policy Mastery: You've created sophisticated IAM policies with conditions and tested them thoroughly. These skills are essential for implementing fine-grained access control in AWS.

Prepared By: Rashi Rana
AWS Corporate Trainer

IAM Monitoring and Compliance

> Effective monitoring and compliance practices are essential for maintaining secure and compliant AWS environments.

CloudTrail for IAM Auditing

- **API Logging:** Log all IAM API calls and changes
- **User Activity:** Track user login attempts and access patterns
- **Policy Changes:** Monitor modifications to policies and permissions
- **Role Assumptions:** Track when and by whom roles are assumed
- **Failed Attempts:** Log failed authentication and authorization attempts

Key CloudTrail Events for IAM

Event Name	Description	Security Relevance
ConsoleLogin	User login to AWS Console	Track user access patterns
AssumeRole	Role assumption events	Monitor cross-account access
CreateUser	New user creation	Track user provisioning
AttachUserPolicy	Policy attachment to user	Monitor permission changes

Event Name	Description	Security Relevance
CreateAccessKey	Access key creation	Track programmatic access setup

IAM Credential Report

```
Credential Report Contents: - User creation date and age - Password last used and last changed - Password next rotation date - MFA active status and device info - Access key 1 & 2 active status - Access key last used date and service - Access key last rotated date - Cert 1 & 2 active status and dates
```

Automated Compliance Monitoring

AWS Config Rules

Monitor IAM configuration compliance automatically

CloudWatch Alarms

Alert on suspicious IAM activities

AWS Security Hub

Centralized security findings dashboard

AWS GuardDuty

Detect malicious IAM activities

Common Config Rules for IAM

- **iam-password-policy:** Check password policy compliance
- **iam-user-mfa-enabled:** Ensure MFA is enabled for users

- **iam-root-access-key-check:** Detect root access keys
- **iam-user-unused-credentials-check:** Find unused credentials
- **iam-policy-no-statements-with-admin-access:** Detect overly permissive policies

CloudWatch Metrics for IAM

```
# Example CloudWatch alarm for failed console logins aws cloudwatch put-metric-alarm \ --alarm-name "IAM-Failed-Console-Logins" \ --alarm-description "Alert on failed console login attempts" \ --metric-name "ConsoleLoginFailures" \ --namespace "AWS/IAM" \ --statistic "Sum" \ --period 300 \ --threshold 5 \ --comparison-operator "GreaterThanThreshold" \ --evaluation-periods 1
```

Regular Audit Tasks

1. **User Review:** Quarterly review of all IAM users and their access
2. **Permission Audit:** Review and validate user permissions
3. **Credential Rotation:** Ensure regular rotation of access keys
4. **Unused Resources:** Remove unused users, roles, and policies
5. **Policy Review:** Validate custom policies against best practices
6. **MFA Compliance:** Ensure MFA is enabled for appropriate users

Compliance Frameworks

Framework	IAM Requirements	Key Controls
SOC 2	Access controls and monitoring	User access reviews, logging
PCI DSS	Strong access control measures	Unique IDs, MFA, least privilege
HIPAA	Access controls for PHI	User authentication, audit logs
ISO 27001	Information security management	Access management, monitoring

Incident Response for IAM

- **Compromised Credentials:** Immediate deactivation and rotation
- **Unauthorized Access:** Review CloudTrail logs and affected resources
- **Policy Changes:** Investigate unexpected permission modifications
- **Failed Logins:** Analyze patterns and potential brute force attacks
- **Privilege Escalation:** Review role assumptions and policy attachments

Automation Tools for IAM Management

AWS CLI Scripts

Automate routine IAM tasks and audits

Lambda Functions

Event-driven IAM automation

CloudFormation

Infrastructure as Code for IAM resources

Third-party Tools

Specialized IAM management solutions

Monitoring Best Practice: Implement continuous monitoring of IAM activities using CloudTrail, Config, and CloudWatch. Regular audits and automated compliance checks help maintain a secure environment.

Prepared By: Rashi Rana
AWS Corporate Trainer

Summary and Next Steps

Key Concepts Mastered

- Understanding of IAM fundamentals and core components
- Comprehensive knowledge of Users, Groups, Roles, and Policies
- Security features including MFA, password policies, and access keys
- Best practices for secure IAM implementation
- Hands-on experience with IAM management and policy creation
- Monitoring, compliance, and auditing practices
- Cross-account access using IAM roles

Practical Skills Gained

User Management

Create and manage IAM users with appropriate permissions

Group Organization

Organize users into groups for efficient permission management

Role Implementation

Policy Creation

Create and configure roles for various AWS services and use cases

Write custom policies with conditions and test them effectively

Lab Accomplishments

- **User and Group Setup:** Created users, groups, and configured password policies
- **Role Configuration:** Set up service roles and cross-account access
- **Policy Development:** Created custom policies with advanced conditions
- **Security Implementation:** Configured MFA and tested access controls
- **Testing and Validation:** Used Policy Simulator to validate permissions

IAM Security Checklist

1. **Root Account Security:** MFA enabled, no access keys, minimal usage
2. **User Management:** Individual users, strong passwords, MFA where needed
3. **Permission Strategy:** Least privilege principle, group-based permissions
4. **Role Usage:** Roles for applications, cross-account access
5. **Policy Management:** Regular review, managed policies preferred
6. **Monitoring:** CloudTrail enabled, regular credential reports

7 **Compliance:** Regular audits, automated compliance checks

Associate-Level Exam Topics Covered

Domain	Topics Covered	Exam Weight
Security	IAM users, groups, roles, policies, MFA	30%
Identity & Access Management	Access control, federation, cross-account	16%
Management & Governance	CloudTrail, monitoring, compliance	12%

Advanced Topics for Further Learning

- **AWS Organizations:** Multi-account management and Service Control Policies
- **AWS SSO:** Centralized single sign-on solution
- **Identity Federation:** SAML, OIDC, and external identity providers
- **Permission Boundaries:** Advanced permission management
- **AWS Secrets Manager:** Secure credential storage and rotation
- **AWS Systems Manager:** Parameter Store for configuration management

Real-World Implementation Tips

Start Small

Begin with basic user/group structure, expand gradually

Document Everything

Maintain clear documentation of policies

and access patterns

Regular Reviews

Schedule quarterly access reviews and cleanup

Automation

Automate routine tasks and compliance checks

Common Pitfalls to Avoid

Avoid These Mistakes:

- Using root account for daily operations
- Creating overly permissive policies
- Sharing user credentials between team members
- Hardcoding access keys in application code
- Ignoring unused users and roles
- Not enabling CloudTrail for audit logging

Recommended Next Steps

1. **Practice:** Set up IAM in your own AWS account and experiment
2. **Certification:** Prepare for AWS Certified Solutions Architect Associate
3. **Advanced Learning:** Explore AWS Organizations and advanced security services
4. **Real Projects:** Apply IAM knowledge in actual cloud projects

5. **Stay Updated:** Follow AWS security best practices and new features

Additional Resources

- **AWS Documentation:** IAM User Guide and Best Practices
- **AWS Training:** Security specialty and advanced courses
- **AWS Well-Architected:** Security pillar guidelines
- **AWS Security Blog:** Latest security updates and practices
- **Community:** AWS forums and user groups

> **Congratulations!** You've mastered AWS IAM fundamentals and are well-prepared for associate-level AWS certifications. Continue practicing with real-world scenarios and stay updated with AWS security best practices.

Prepared By: Rashi Rana
AWS Corporate Trainer