AWS Academy Cloud Security Foundations
Securing Access to Cloud Resources Student Guide
Version 1.0.0

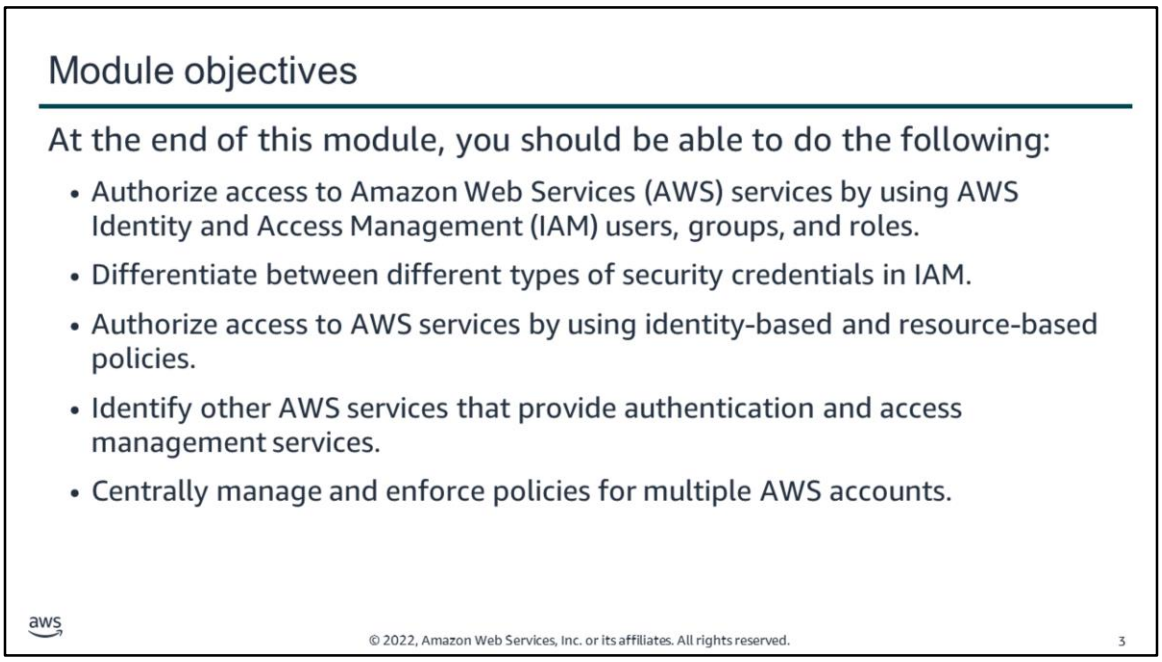
100-ACSECF-10-EN-SG

AWS Academy Cloud Security Foundations

Contents

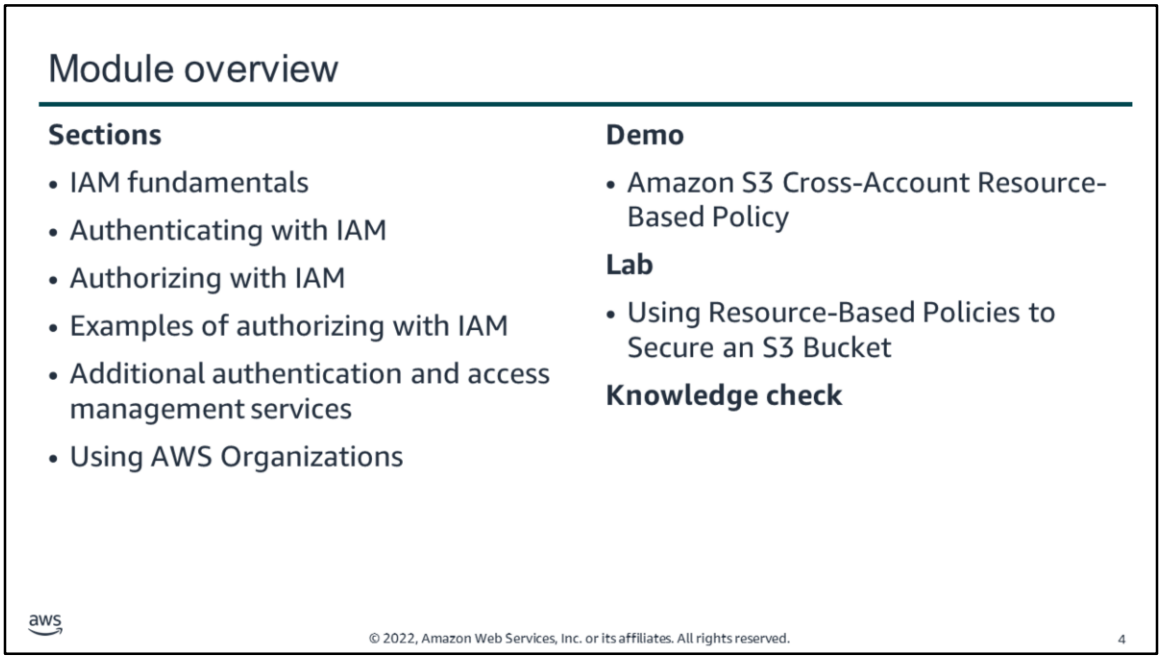
Welcome to the Securing Access to Cloud Resources module.

This first section provides an introduction to the module.



At the end of this module, you should be able to do the following:

- Authorize access to Amazon Web Services (AWS) services by using AWS Identity and Access Management (IAM) users, groups, and roles.
- Differentiate between different types of security credentials in IAM.
- Authorize access to AWS services by using identity-based and resource-based policies.
- Identify other AWS services that provide authentication and access management services.
- Centrally manage and enforce policies for multiple AWS accounts.

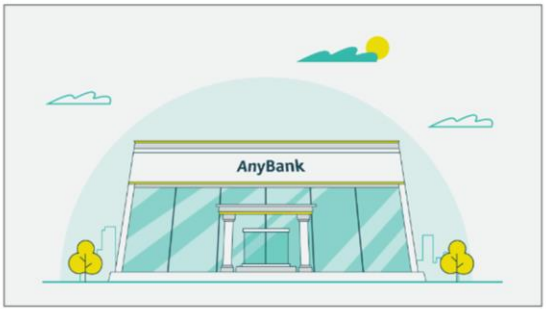


This module includes the following sections:
- IAM fundamentals
- Authenticating with IAM
- Authorizing with IAM
- Examples of authorizing with IAM
- Additional authentication and access management services
- Using AWS Organizations

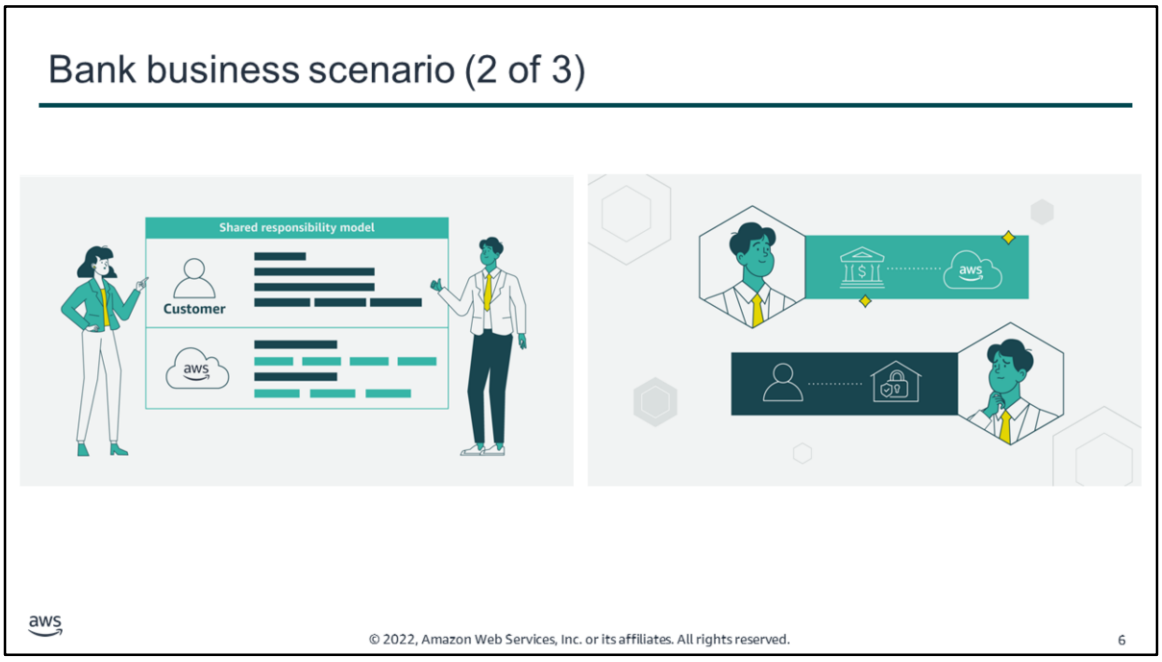
This module also includes the following:
- Demonstration: Amazon S3 Cross-Account Resource-Based Policy
- Lab: Using Resource-Based Policies to Secure an S3 Bucket

Finally, you will be asked to complete a knowledge check that will test your understanding of key concepts covered in this module.

Bank business scenario (1 of 3)

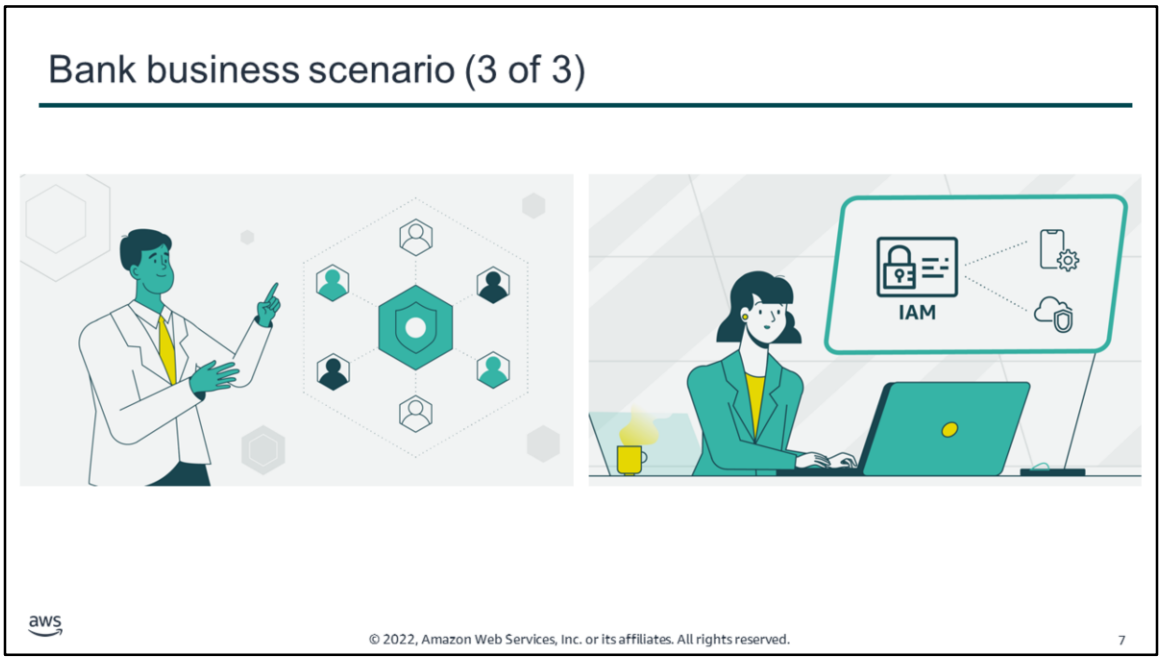


Let’s discuss how the concepts in this module are applicable to the bank business scenario.



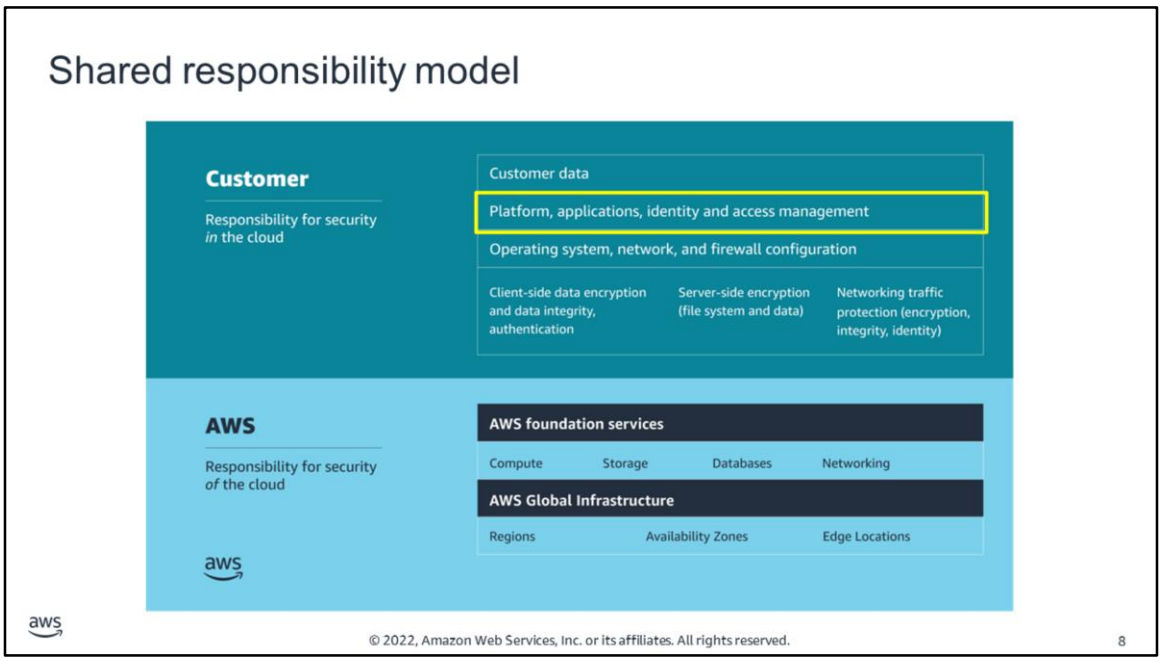
María's initial meeting with John went well. John has a strong grasp of the fundamentals of cloud security, and they want to use the shared responsibility model as a focal point for their discussions. John agrees that migrating to the AWS Cloud could be a smart move to modernize the bank's business operations.

However, he has reservations about how the bank could adequately secure online access for their members while also maintaining internal security protocols.



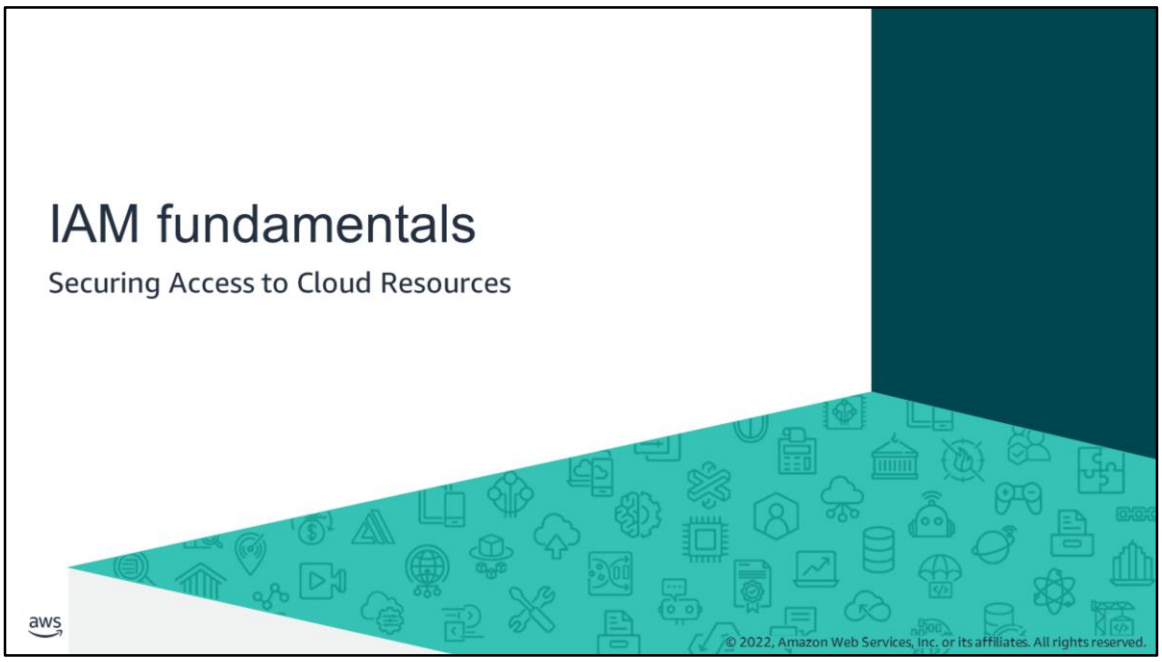
John is particularly focused on restricting internal access to critical resources as much as possible.

In preparation for their next meeting, María decides to focus on the AWS services for identity and access management, including services that could be used to roll out a planned mobile app. She also plans to discuss security best practices, such as the principle of least privilege, to address his concerns about access.

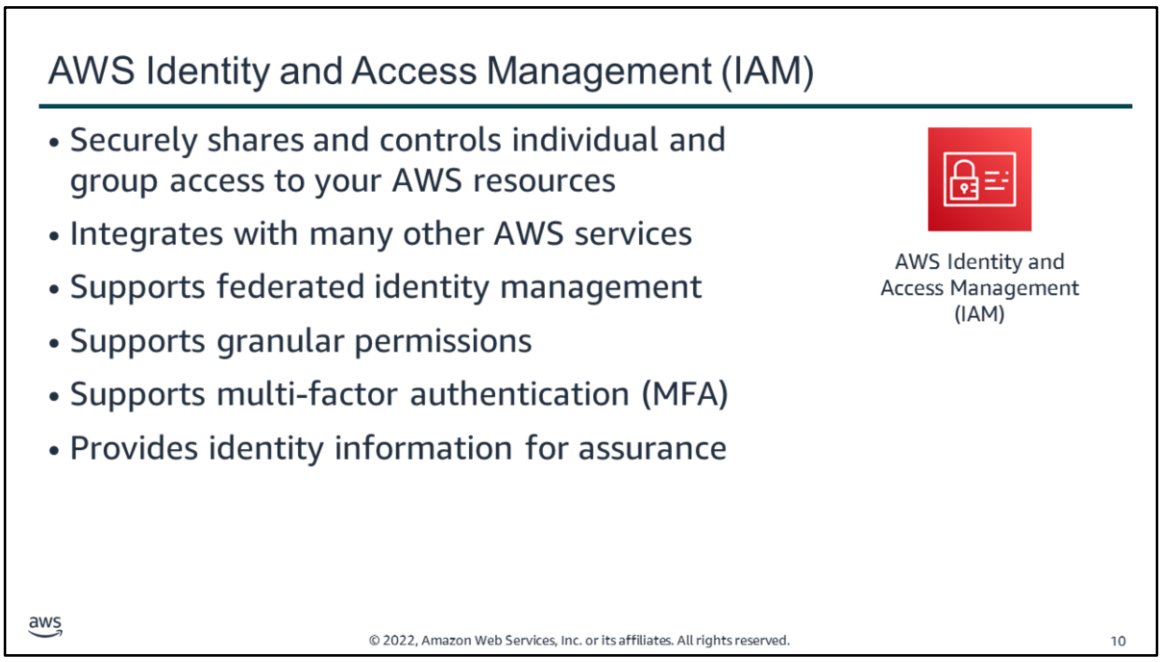


For accessibility: Shared responsibility model listing customer and AWS responsibilities. Customer is responsible for security in the cloud. This includes customer data. Platform, applications, identity and access management. Operating system, network, and firewall configuration. Client-side data encryption and data integrity, authentication. Server-side encryption of file system and data. Networking traffic protection, to include encryption, integrity, and identity. AWS is responsible for security of the cloud. This includes the AWS foundation services for compute, storage, databases, and networking. And the AWS Global Infrastructure, to include Regions, Availability Zones, and Edge Locations. **End of accessibility description.**

This module focuses on the platform, applications, and identity and access management portion of the shared responsibility model, which the customer is responsible to secure.



This section covers fundamental information about the AWS Identity and Access Management (IAM) service.



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

IAM is integrated into most AWS services. You can define access controls from one place in the AWS Management Console, and they will take effect throughout your AWS environment.

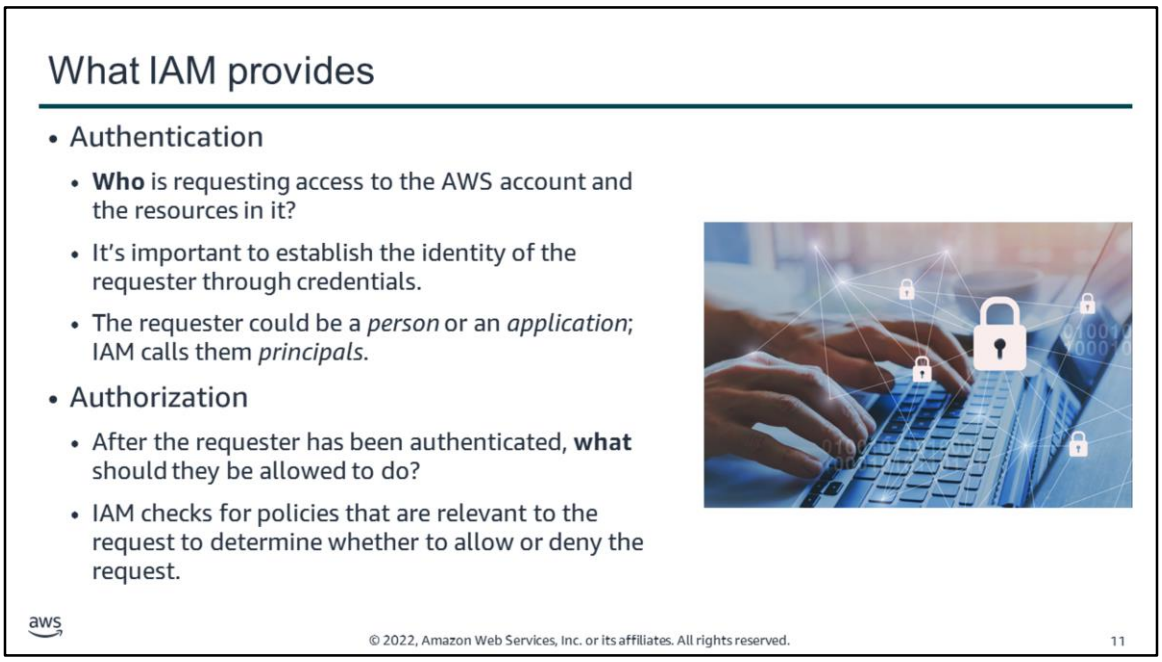
You can use IAM to grant users, groups, and applications granular access to the console and to AWS service application programming interfaces (APIs) by using existing identity systems. AWS supports federation from corporate systems such as Microsoft Active Directory and standards-based identity providers. This capability is beneficial when it comes to integrating new cloud deployments with existing on-premises infrastructure.

IAM also supports multi-factor authentication (MFA). If MFA is activated and an IAM user attempts to log in, then the user is prompted for an authentication code. The authentication code is delivered to a specially configured AWS MFA hardware device or to a software-based AWS MFA device, such as Google's Authenticator application.

By using AWS CloudTrail, IAM can also support information assurance by providing log records that include the identity information of users that request resources in your account.

For more information, see What Is IAM? in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html.

For more information, see AWS Services that Work with IAM in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html.



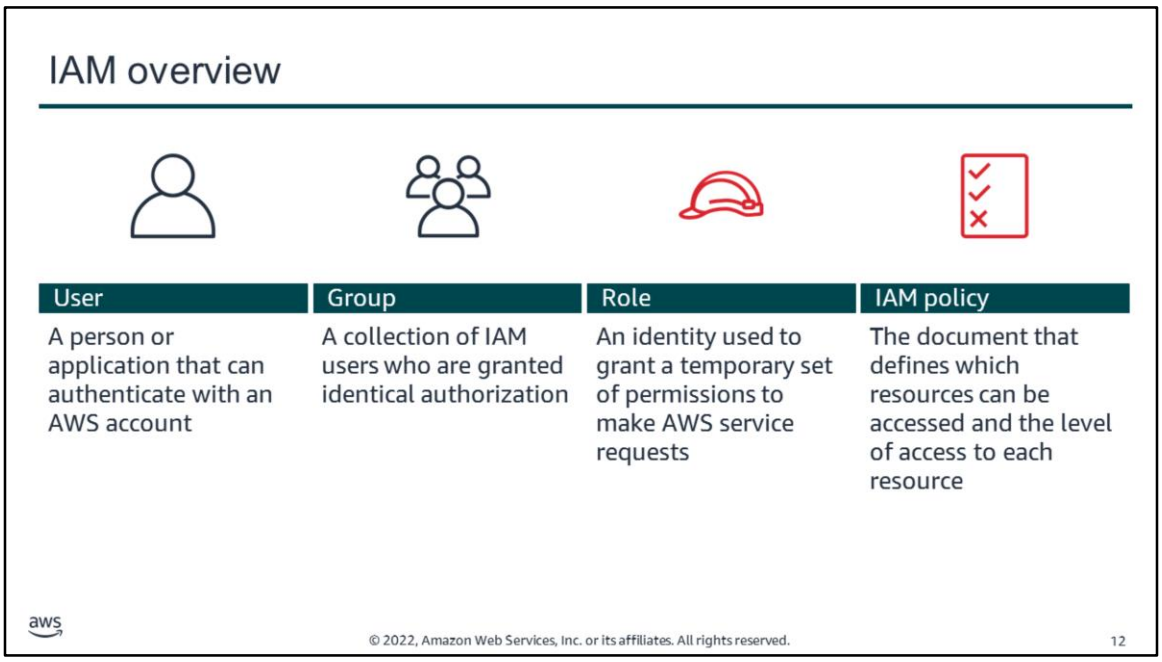
With IAM *authentication*, you can use the identity of requesters to control *who* can use your AWS resources. With IAM *authorization*, you can also use access management to control *what* resources they can use and in what ways.

To understand the fundamental aspects of authentication and authorization, consider a banking analogy. Think of a bank that allows their customers (users) to access their accounts online. Suppose that you are a bank customer. You have money in a checking account at that bank, and you want to pay a bill online. Should any random person be able to log in to your bank account and pay their bills with the money in your account? No, a random person shouldn't be able to use the money in your account.

Before the bank allows you to access your checking account, the bank must ensure that the person who is accessing the account is you. The bank wants to *authenticate* that you are who you claim to be. They typically accomplish this authentication by requiring you to enter your user name and a password that is supposedly known only to you. For extra security, they might have configured two-factor authentication. Thus, in addition to typing in a password, you also must receive a code on your phone and enter that code.

Now, suppose that you are successfully logged in to the bank website (authenticated). Can you pay your bill by using the money in some *other customer's* account? No, you shouldn't be able to use another customer's money to pay your bill. You don't have the *authorization* to access accounts that don't belong to you. However, you are authorized to access the money in your account, and you are authorized to pay bills by using your money.

How does this analogy relate to IAM? Similar to how a bank must secure access to resources (your money), you—as an AWS account owner—must secure access to your AWS account. You want whatever data that you store in your account to be secure so that others can't access it. Likewise, you want to secure the application logic of anything that you build on AWS, so that others can't modify it. IAM provides many features that can help you to accomplish these security objectives.



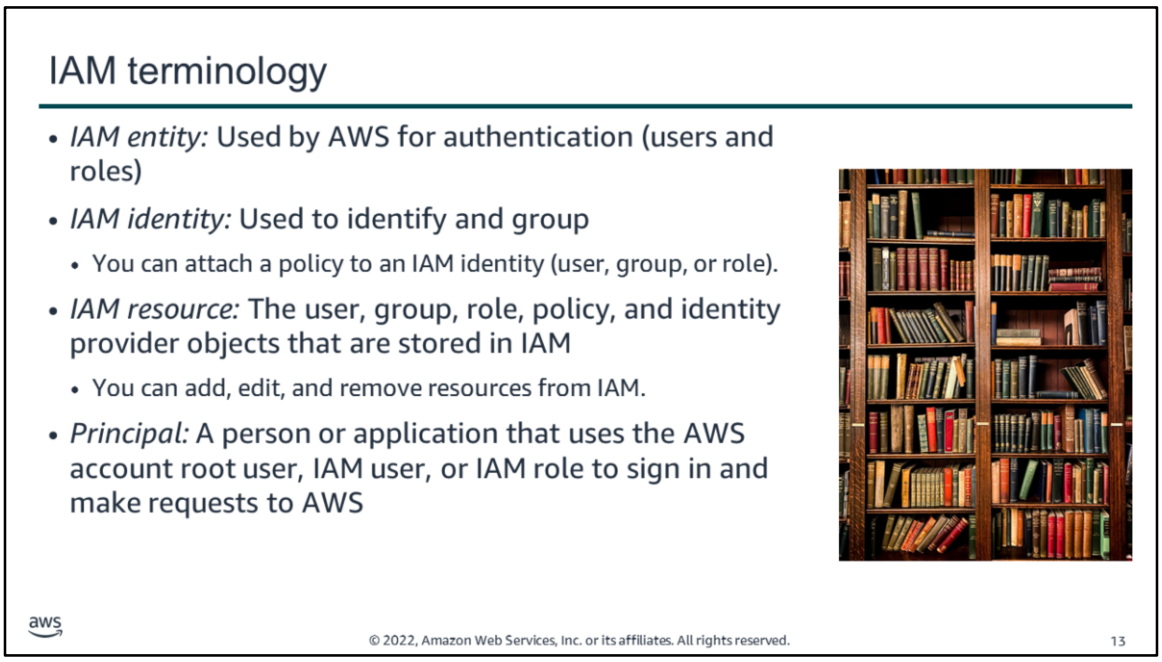
You can use IAM to control access to your AWS resources. You achieve this access control by creating *users*, *groups*, and *roles*. You can also enforce access control by attaching *policies*.

For IAM, these terms are defined as follows:

- An *IAM user* is an entity that you create in AWS to represent a person or application that interacts with AWS account services and resources. A user is given a permanent set of credentials, which stay with the user until a forced rotation occurs. With new AWS accounts, the *root user* account is the first IAM user account established.
- An *IAM group* is a collection of IAM users. You can use groups to grant the same set of permissions to multiple users.
- An *IAM role* is similar to a user in that it's an AWS identity that you can attach permission policies to. These will determine what the identity can and can't do in AWS. However, a role doesn't have long-term credentials (such as a password or access keys) that are associated with it. Instead, when a person or application assumes a role, they are provided with temporary security credentials for the role session. IAM roles will be covered later in this module.
- An *IAM policy* is a document that explicitly lists permissions. The policy can be attached to an IAM user, an IAM group, an IAM role, or any combination of these resources. IAM policies will be discussed in more depth later in this this module.

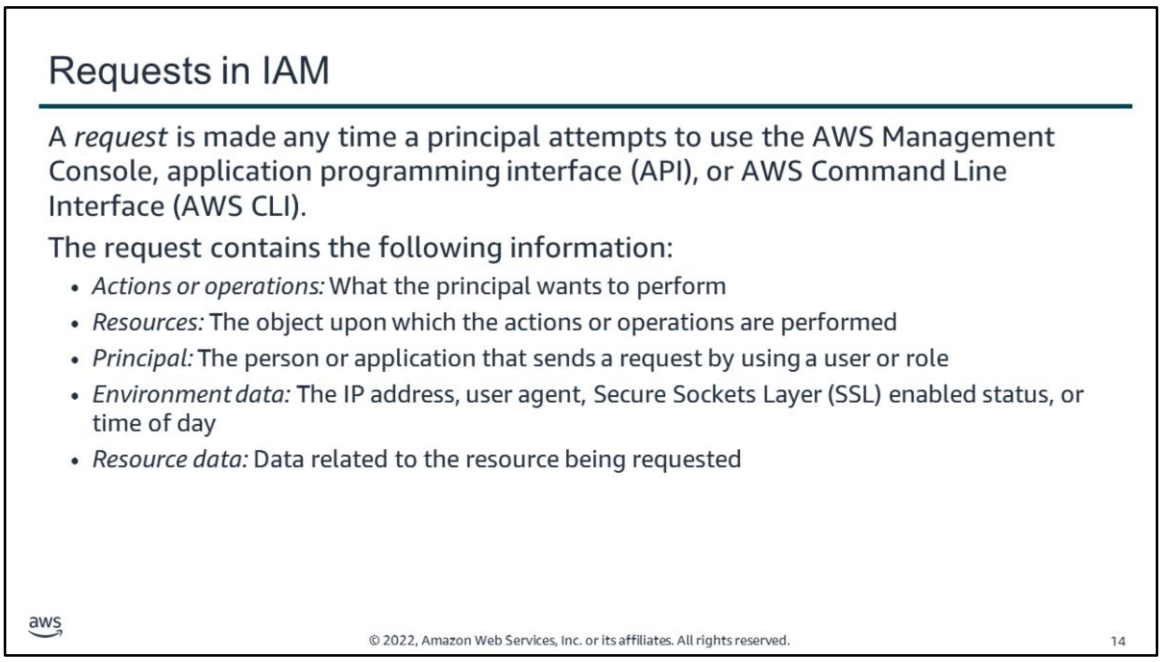
To configure long-term access, a best practice is to attach IAM policies to IAM groups, and then assign IAM users to these IAM groups. An IAM user who is a member of an IAM group inherits the permissions that are attached to that group. You can also attach IAM policies directly to an IAM user to further customize the access that's granted through the group.

For more information, see Security Best Practices in IAM in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html.



The following terms are specific to IAM, and you will use them throughout this course:

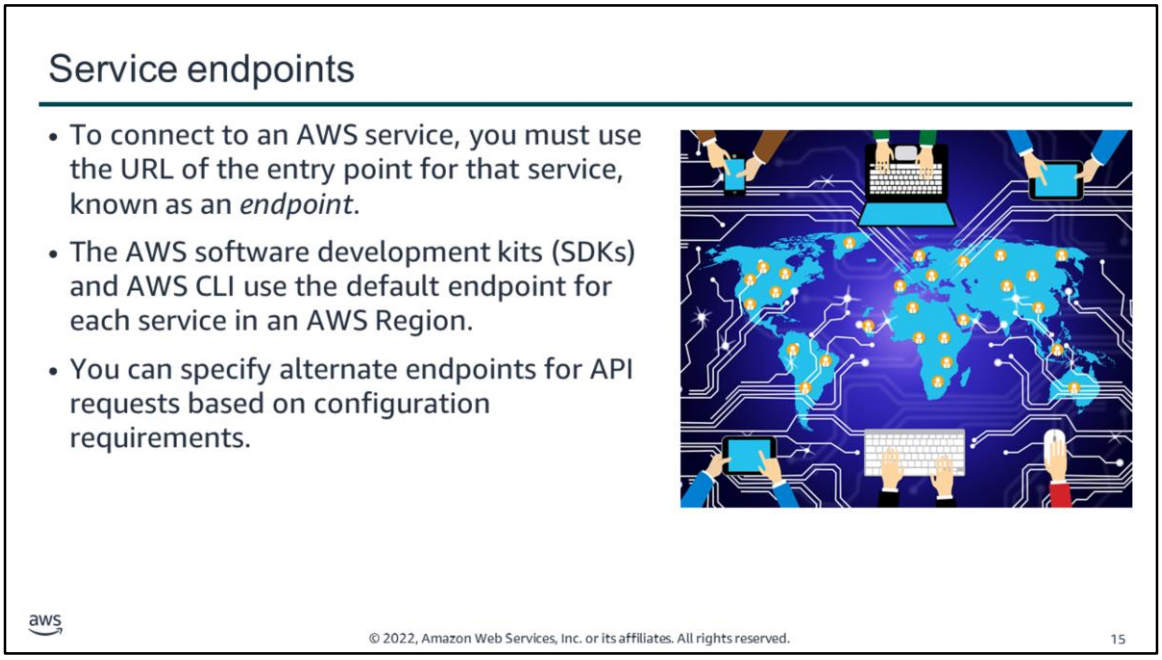
- *IAM entities* are the IAM resource objects that AWS uses for authentication. These include IAM users and roles. IAM user entities give you the ability to sign in to the console. This can be for interactive tasks and to make programmatic requests to AWS services using the API or the AWS Command Line Interface (AWS CLI).
- *IAM identities* are the IAM resource objects that are used to identify and group. You can attach a policy to an IAM identity. These include users, groups, and roles.
- *IAM resources* are the users, groups, roles, policies, and identity provider objects that are stored in IAM. As with other AWS services, you can add, edit, and remove resources from IAM.
- A *principal* is a person or application that uses the AWS account root user, an IAM user, or an IAM role to sign in and make requests to AWS. Principals include assumed roles and federated users—external identities that do not have an AWS account. The principal is authenticated as the AWS account root user or an IAM entity to make requests to AWS.



A request is made any time a principal attempts to use the console, API, or AWS CLI. The request contains the following info:

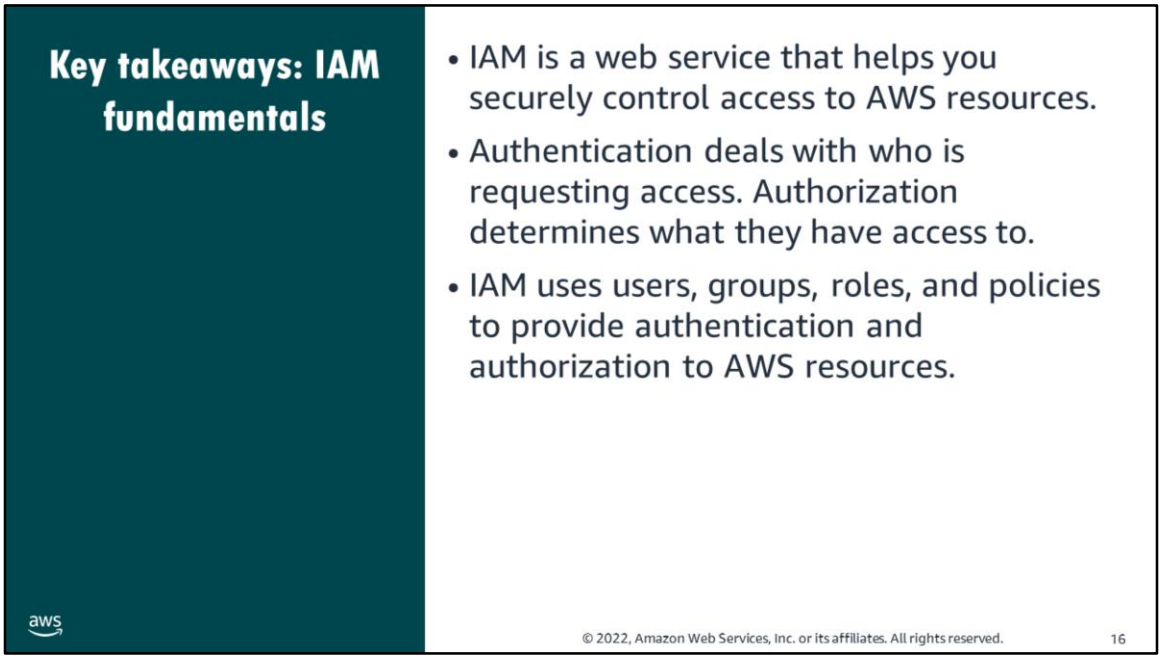
- *Actions* or *operations* are what the principal wants to perform.
- *Resources* are the object or objects upon which the actions or operations are performed.
- The *principal* is the person or application using a user or role to send the request.
- *Environment data* consists of the IP address, user agent, Secure Sockets Layer (SSL) enabled status, or time of day.
- *Resource data* is the data related to the resource being requested.

AWS gathers all the request information into a *request context*, which is then used to evaluate and authorize (or prevent) the request.



To connect to an AWS service, you must use the URL of the entry point of that service, known as an *endpoint*. The AWS software development kits (SDKs) and AWS CLI use the default endpoint for each service in an AWS Region. An example of an Amazon Elastic Compute Cloud (Amazon EC2) endpoint is ec2.us-east-1.amazonaws.com, which is the US East (N. Virginia) service endpoint for Amazon EC2. Using regional endpoints is extremely helpful when making direct API calls for individual AWS services within the same Region.

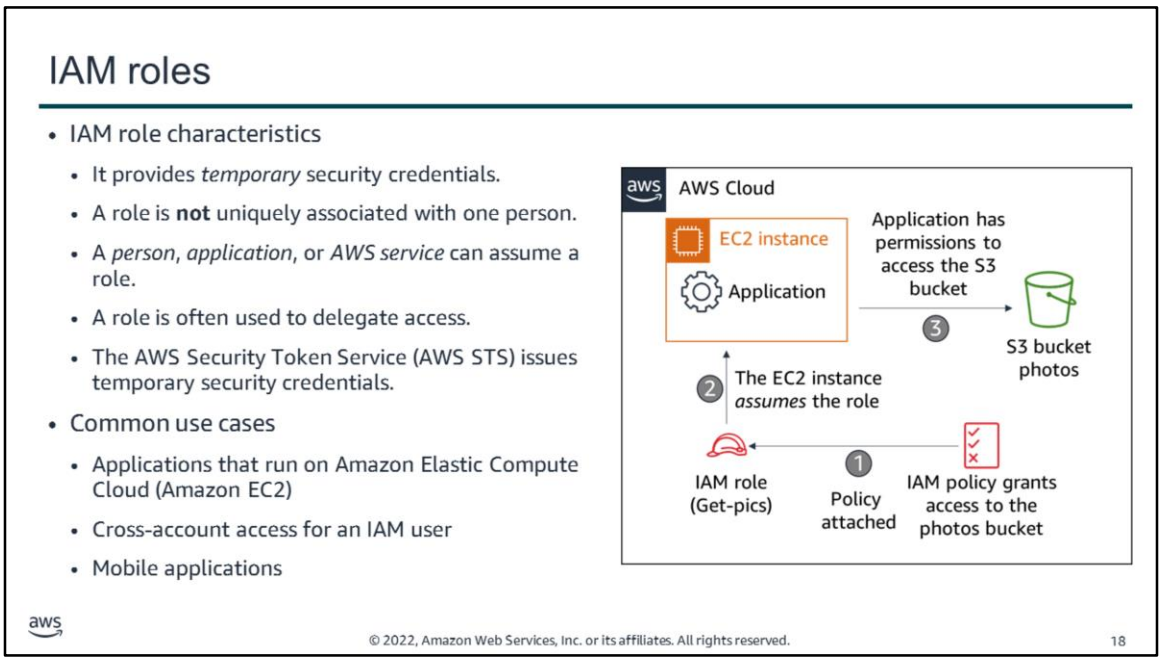
You can create endpoint policies and attach them to endpoints, but the endpoint policies will not override or replace IAM user policies or service-specific policies. An endpoint policy is a separate policy to only control access from the endpoint to the specified service. You can only attach one endpoint policy to an endpoint, but you can modify the policy at any time. These policies are similar to IAM policies but differ in that they must contain a principal element, and the size of the endpoint policy cannot exceed 20,480 characters.



Key takeaways from this section of the module include the following:

- IAM is a web service that helps you securely control access to AWS resources.
- Authentication deals with who is requesting access. Authorization determines what they have access to.
- IAM uses users, groups, roles, and policies to provide authentication and authorization to AWS resources.

This section describes using IAM for authentication.



An IAM *role* is an IAM identity that provides the ability to define a specific set of permissions to access the resources that a user or service needs. An IAM role is similar to an IAM user in that you can attach permissions policies to it. However, you don't attach the permissions to an IAM user or group. Instead, you attach the permissions to a role, and the user or the service *assumes* the role as needed.

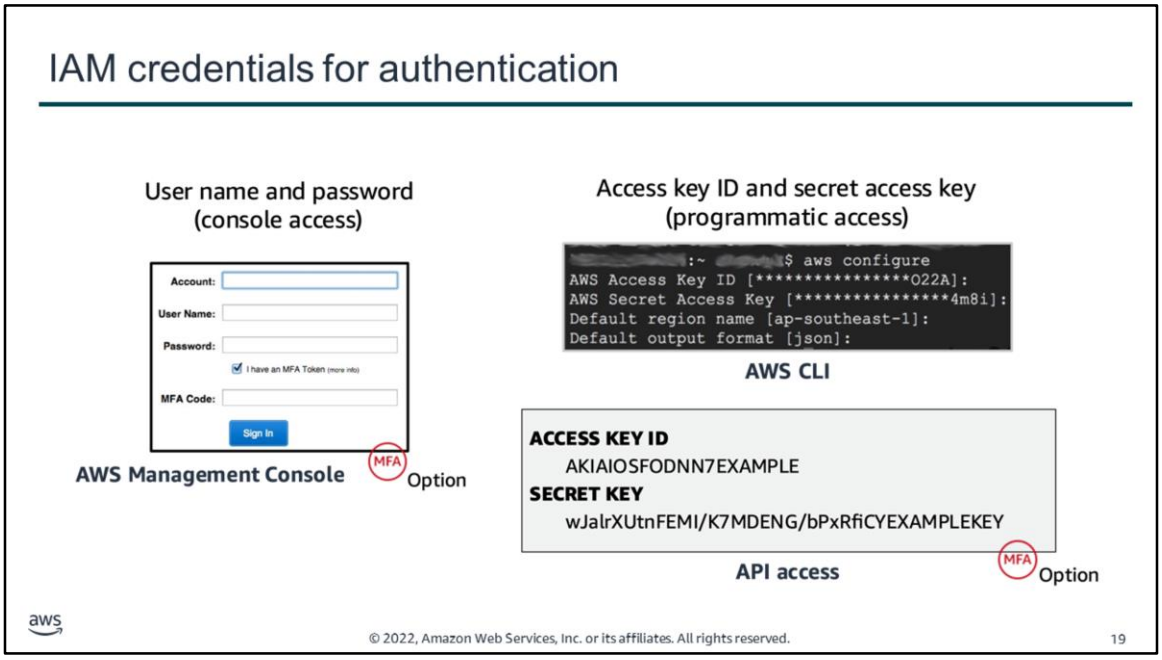
When a user assumes a role, the user's prior permissions are temporarily forgotten. AWS returns temporary security credentials that the user or application can then use to make programmatic requests to AWS for the duration of the session. The AWS Security Token Service (AWS STS) issues the temporary security credentials.

When you use IAM roles, you don’t need to grant long-term security credentials to each entity that requires access to a resource.

For a service like Amazon EC2, applications or AWS services can programmatically assume a role at runtime. The principal that assumes the role might be an IAM user, group, or role from another AWS account, including accounts that you do not own.

The following example illustrates a situation when you might need to use temporary security credentials to provide a role for an EC2 instance:

1. An administrator creates the Get-pics role in IAM, defines a policy that grants access to an Amazon Simple Storage Service (Amazon S3) bucket named photos, and attaches the policy to the role.
2. An EC2 instance assumes the Get-pics role.
3. The EC2 instance is granted temporary permissions to access the S3 bucket named photos.



When you interact with AWS by using the console, AWS CLI, or AWS SDKs, you must provide AWS credentials.

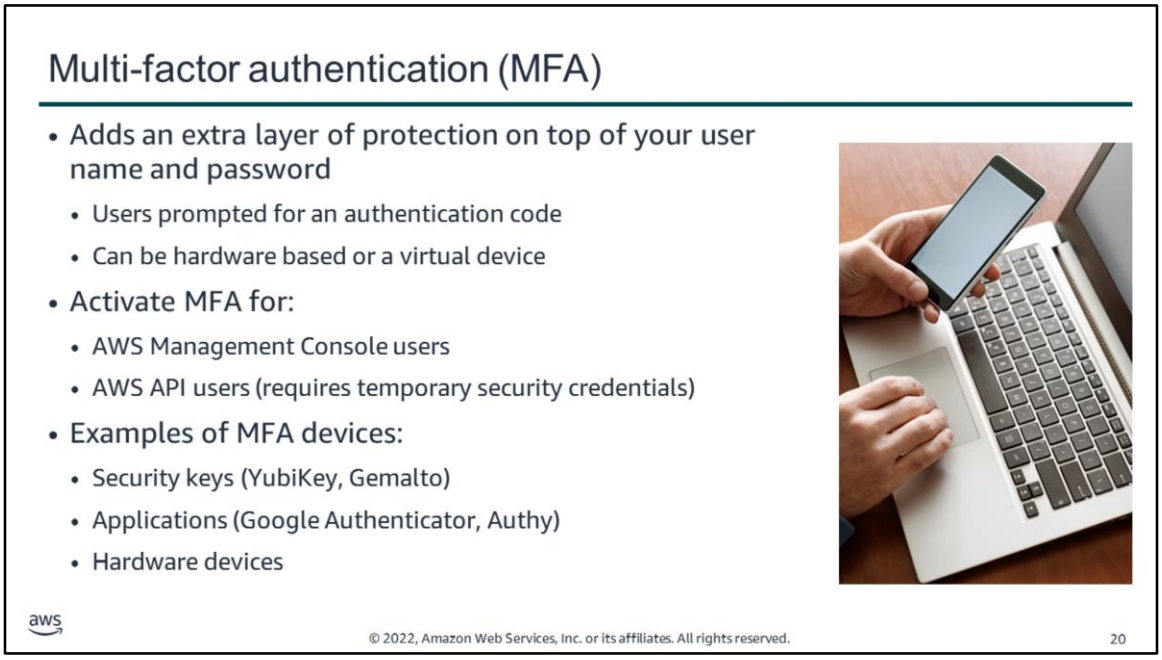
Two primary types of credentials are used for authentication. Which credential type you use depends on how you access AWS:

- To authenticate from the console, you must sign in with your *user name and password*.
- To authenticate programmatically through the AWS CLI, SDKs, and APIs, you must provide an AWS access key. The AWS access key is the combination of an *access key ID and a secret access key*.

The situation will dictate the authentication method. In most circumstances, a human user will use the AWS CLI method, whereas an automated program will use the SDK or API method.

You might also be required to provide additional security information. For example, AWS recommends that you use MFA to increase the security of your account.

For more information, see AWS Security Credentials in the *AWS General Reference Guide* at https://docs.aws.amazon.com/general/latest/gr/aws-security-credentials.html.

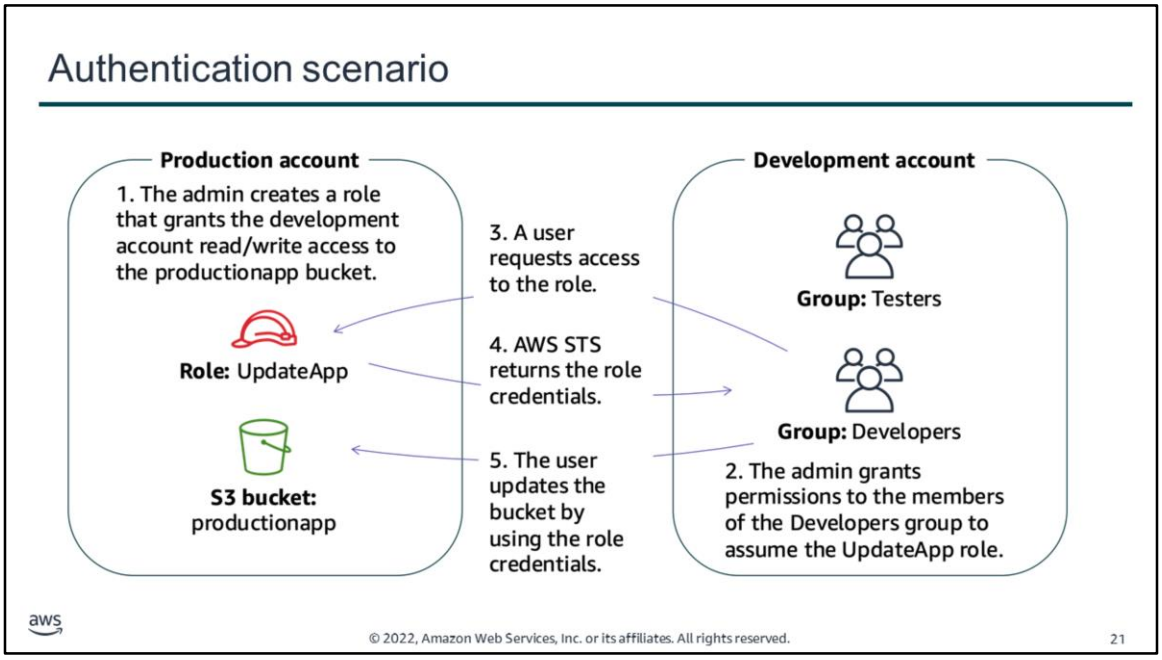


Multi-factor authentication (MFA) is a best practice that adds an extra layer of protection on top of your user name and password. MFA requires two or more factors to achieve authentication. Factors include the following:

- Something you *know*, such as a password
- Something you *have*, such as a cryptographic token
- Something you *are*, such as your fingerprint

By default, MFA is not activated.

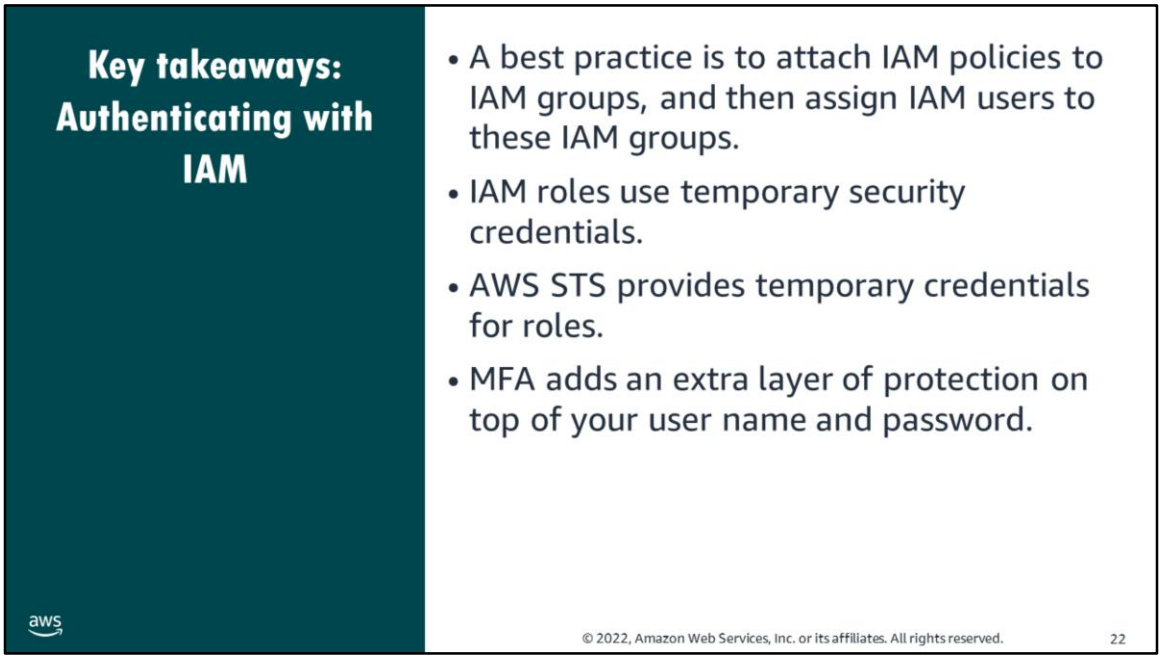
When MFA is enabled and a user signs in to the AWS Management Console, they are prompted for their user name and password (the first factor, what they know). They are then prompted for an authentication response from their AWS MFA device (the second factor—what they have). Examples of MFA devices include security key devices, such as YubiKey and Gemalto devices, and virtual devices, such as Google Authenticator or Authy.



In this scenario, your organization has created multiple AWS accounts to isolate your production environment from your development environment. Remember, an AWS account is a container for your AWS resources, so the production account and development account are essentially two completely separate environments. After testing an update within the development environment, you need to grant members of the Developers group temporary access to update the productionapp S3 bucket.

You take the following steps to grant this temporary access:

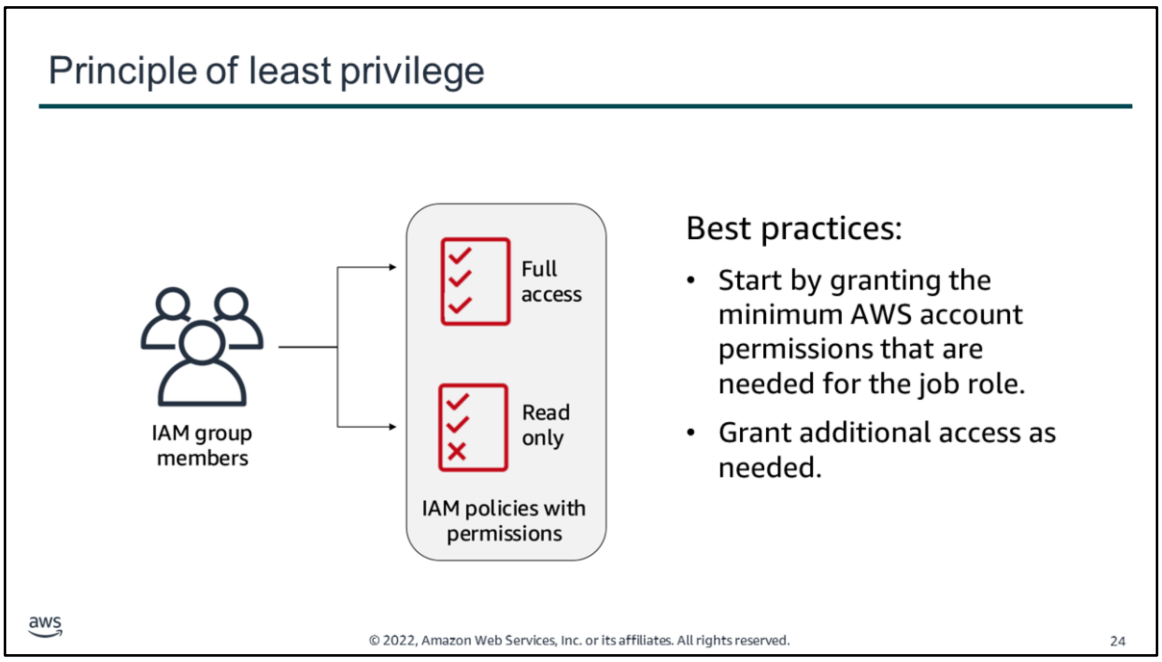
1. An administrator within the production account creates a role that grants the development account read/write access to the production account. To do this, you need to define a trust policy that specifies the development account as a principal. This will authorize users from the development account to assume the UpdateApp role (AWS account authentication).
2. Within the development account, an administrator grants members of the Developers group permissions to assume the UpdateApp role. Because this is a role, which is temporary, the Developers will be granted permissions to call AWS STS to provide a temporary security token. When this is complete, any IAM user that belongs to the Developers group will be allowed to assume the UpdateApp role as needed.
3. The user requests to assume the UpdateApp role through the console, API, or AWS CLI.
4. Upon receiving the role request, AWS STS returns temporary credentials to the requestor after verification is complete.
5. By using the temporary credentials that AWS STS provided, the user from the Developers group is permitted to update the productionapp bucket in the production account.



Key takeaways from this section of the module include the following:

- A best practice is to attach IAM policies to IAM groups, and then assign IAM users to these IAM groups.
- IAM roles use temporary security credentials.
- AWS STS provides temporary credentials for roles.
- MFA adds an extra layer of protection on top of your user name and password.

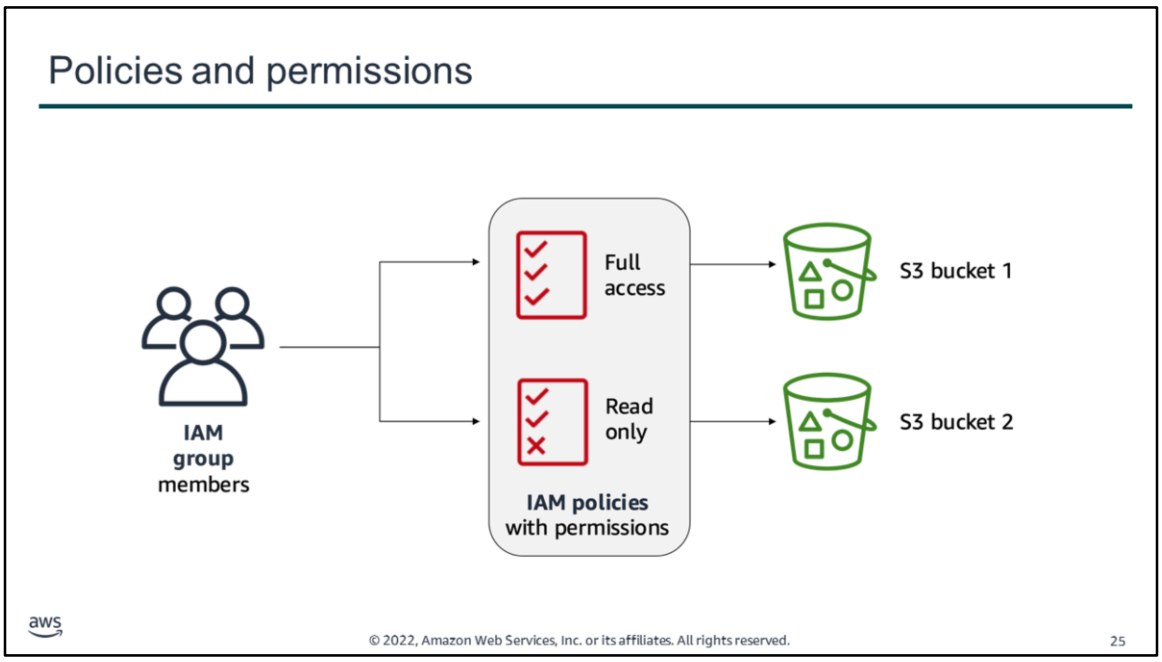
This section describes using IAM for authorization.



When you grant permissions to users, groups, roles, and resources, follow the standard security advice of strong authentication and the *principle of least privilege*. This practice means that you grant only the permissions that are needed to perform a task.

It's more secure to start with a minimum set of permissions and grant additional permissions as needed. This provides better security than starting with permissions that are too permissive and then trying to restrict them later. To define the correct set of permissions, you must do some research to determine what access is needed to accomplish a specific task.

When you create IAM policies, determine what your users need to do, and then craft policies that allow them to perform only those tasks. Similarly, create policies for individual resources that identify precisely who is allowed to access the resource, and allow only the minimal permissions for those users. For example, perhaps developers should be allowed to create EC2 instances in production environments but not to stop or terminate the instances.



For accessibility: Diagram of IAM group member receiving access from IAM policies and permissions. IAM group members are assigned full access to S3 bucket 1. IAM group members are also assigned read only access to S3 bucket 2. **End of accessibility description.**

You can control access to your AWS resources by using IAM policies that grant or deny permissions. In this example, the users in the IAM group can read, write, and delete objects in bucket 1. However, they can only read the objects in bucket 2.

By default, an authenticated IAM user, group, or role can't access anything in your account until you grant them *permissions*. You grant permissions by creating a *policy*. Policies are objects that define permissions for the identity or resource they're associated with. Most policies are defined and stored in a JavaScript Object Notation (JSON) document. Policies define the effect, actions, resources, and optional conditions under which an entity can invoke API operations to the AWS account. By default, any actions or resources that are not explicitly allowed are denied. When an IAM principal (user or role) makes a request, AWS evaluates that request based on the permissions in these policies to determine whether access will be allowed or denied.

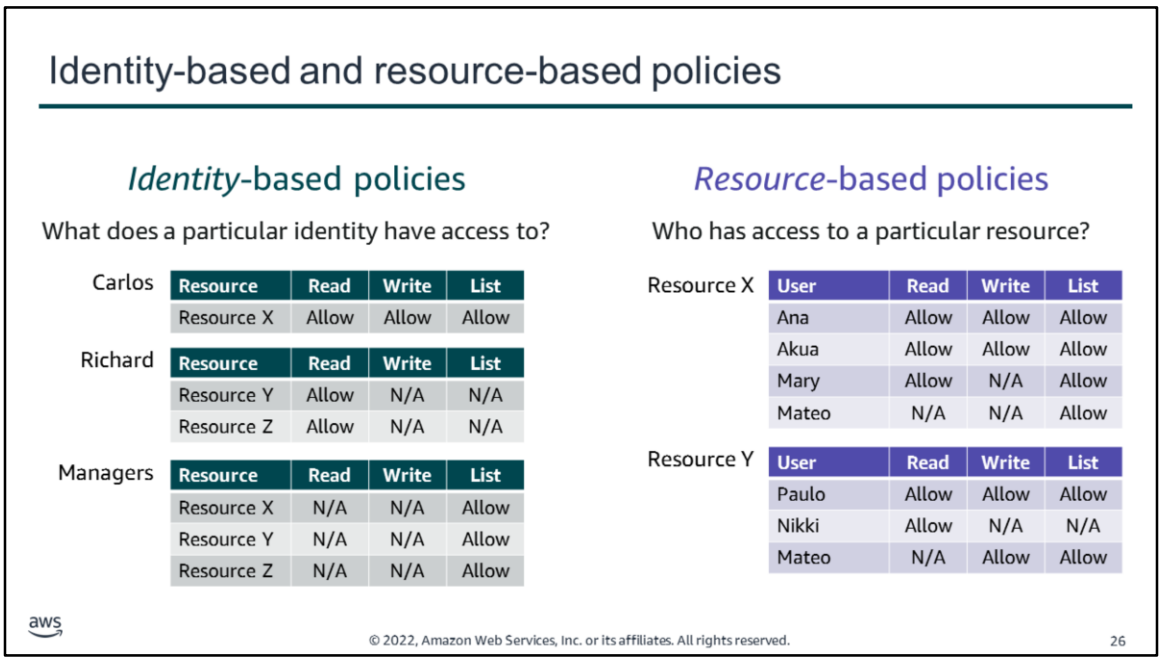
AWS currently supports six types of policies:

- **Identity-based policies:** These policies are attached to IAM identities and grant permissions to the identity.
- **Resource-based policies:** These are attached to resources and grant permissions to the principal specified in the policy.
- **Permissions boundaries:** These define the maximum permissions that the identity-based policies can grant to an entity. Permissions boundaries do not grant permissions.
- **AWS Organizations service control policy (SCP):** This defines the maximum permissions for account members of an organization or organizational unit (OU). You can use SCPs to limit the permissions of identity-

based or resource-based policies, but they cannot be used to grant permissions.

- **Access control lists (ACLs):** You can use an ACL to control which principals in other accounts can access the resource to which the ACL is attached. You cannot use an ACL to grant permissions to entities that reside within the same account. ACLs are the only policy type that doesn't use the JSON document format.
- **Session policies:** These are used with the AWS CLI or AWS API to create a temporary session for a role or federated user. You can use session policies to limit the permissions that a role or a user's identity-based policies grant to a session. Session policies limit permissions but cannot grant them.

For more information, see Policies and Permissions in IAM in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.



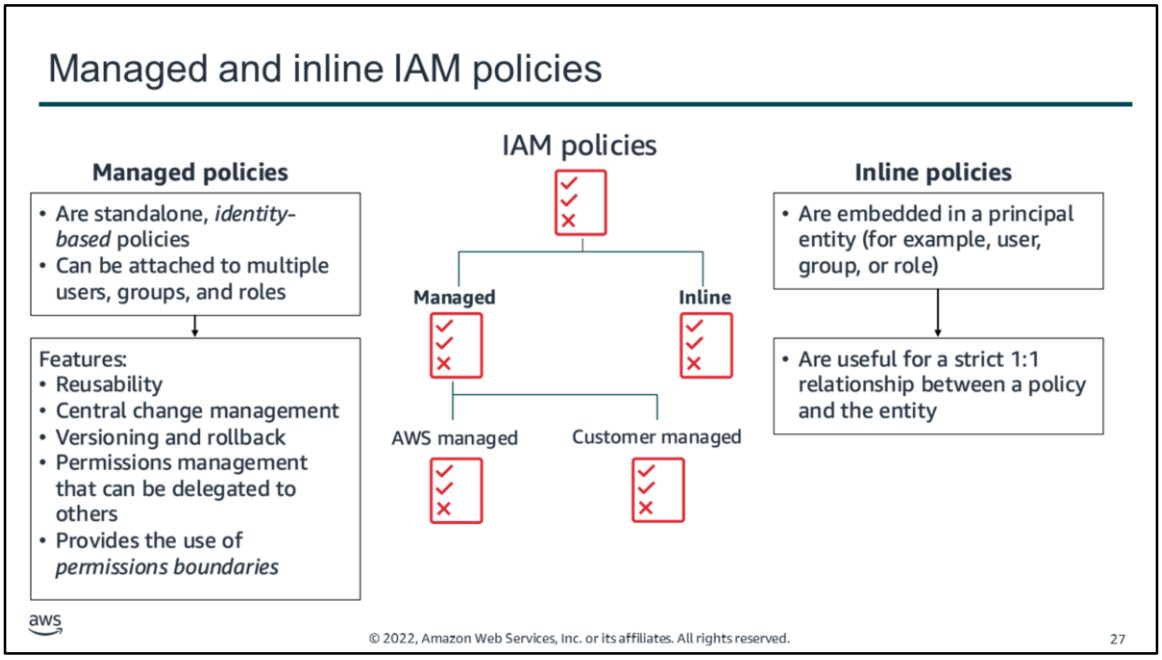
Note: In the tables on the slide, N/A means not applicable because the policy doesn't specify permissions for that particular action.

To reiterate, two types of IAM policies can grant permissions: identity-based policies and resource-based policies. The difference between the two types of policies is a result of where they are applied, which depends on the type of access that you're authorizing or restricting.

Identity-based policies are attached to an IAM user, group, or role. They indicate what that identity can do. For example, you could grant a user the ability to access an Amazon DynamoDB table.

Resource-based policies are attached to a resource. They indicate what a specified user (or group of users) is permitted to do with the resource. For example, you can use resource-based policies to grant access to an S3 bucket or to grant cross-account access between two trusted AWS accounts.

For more information, see Identity-Based Policies and Resource-Based Policies in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_identity-vs-resource.html.



For accessibility: Diagram showing that IAM policies are divided into two categories, managed and inline. Managed policies are then further divided into AWS managed policies and customer managed policies. **End of accessibility description.**

IAM policies can be managed or inline.

Managed policies

- Managed policies are standalone, identity-based policies that you can attach to multiple users, groups, and roles.
- *AWS managed policies* are created and managed by AWS.
- *Customer managed policies* are created and managed by you.
- Managed policies provide several features, including the following:
  - Reusability
  - Central change management
  - Versioning and rollback
  - The ability to delegate permissions management to other users
- Managed policies also provide the use of *permissions boundaries*. This is an advanced feature that allows a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity.

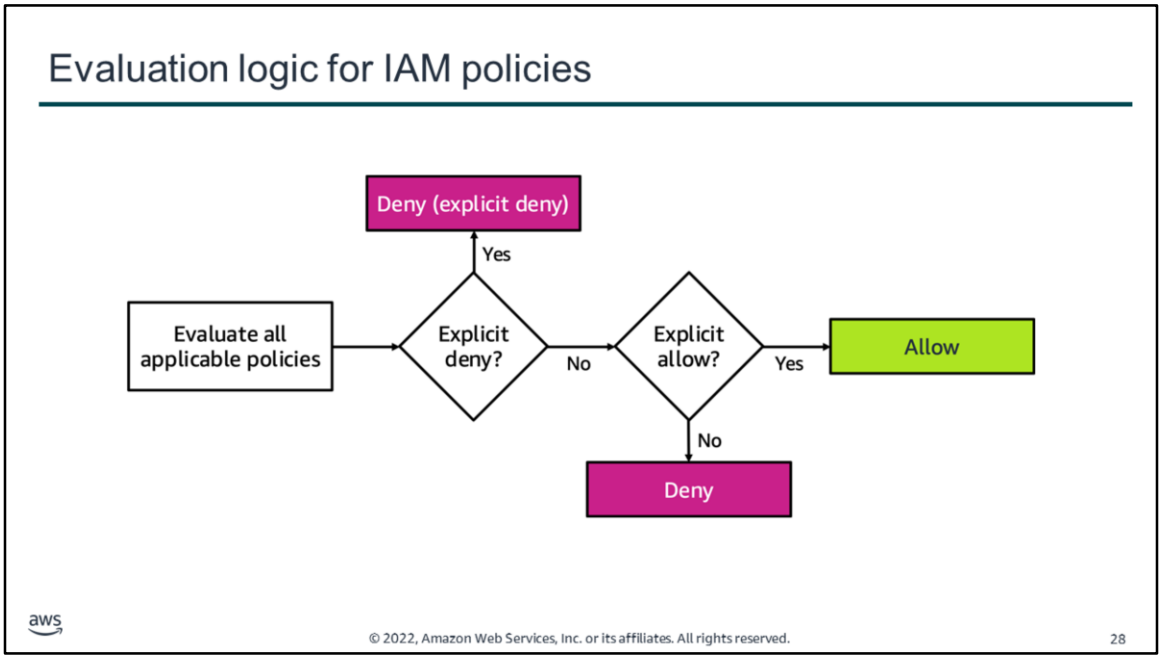
Inline policies

- Inline policies are embedded in a principal entity such as a user, group, or role. That is, the policy is an inherent part of the entity.
- You can use the same policy for multiple entities, but those entities do not share the policy. Instead, each entity has its own copy of the policy, which makes it impossible to centrally manage inline policies.

- Inline policies are useful when you want to maintain a strict one-to-one relationship between a policy and the principal entity to which it is applied. This is because inline policies cannot be inadvertently attached to the wrong entity.

Use cases will vary, and policies should be selected based on the situation. In most cases, AWS recommends the used of managed policies instead of inline policies.

For more information, see Managed Policies and Inline Policies in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html.



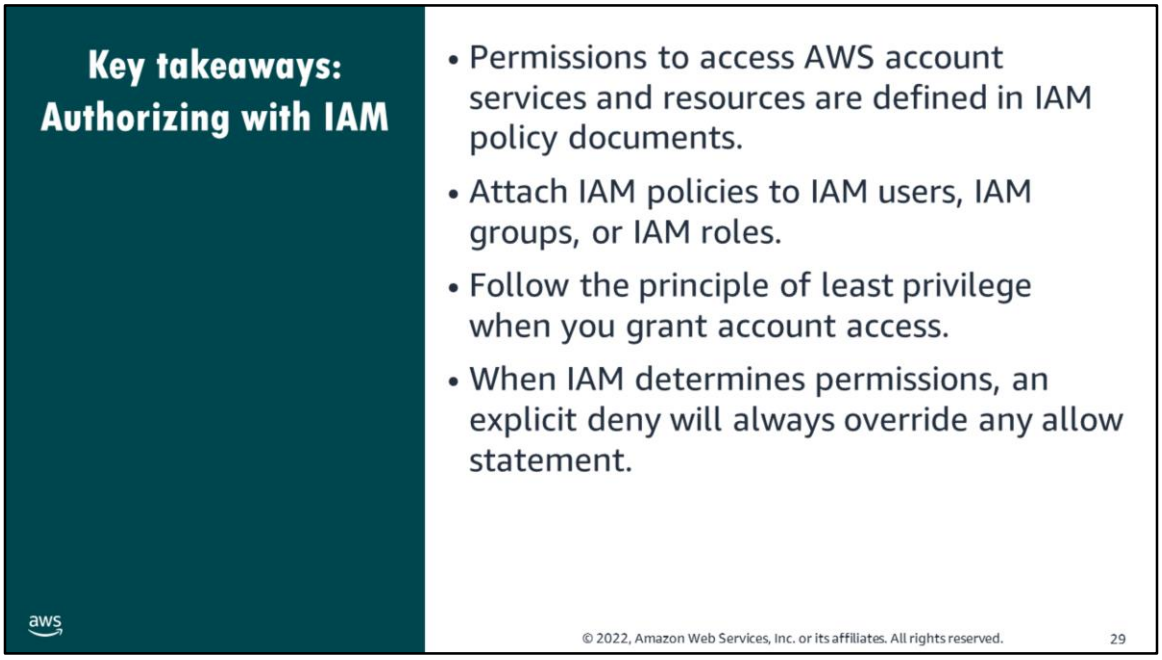
For accessibility: Diagram of evaluation logic for IAM policies. Policies are evaluated first for an explicit deny. If an explicit deny exists, then access is denied. If no explicit deny exists, the policy is then evaluated for an explicit allow. If an explicit allow exists, access is granted, but if no explicit allow exists, access is denied. **End of accessibility description.**

This diagram shows the logic that AWS uses when evaluating IAM policies. AWS evaluates all applicable policies and goes through this evaluation logic:

- By default, all requests are denied.
- An explicit allow overrides the default deny.
- An explicit deny overrides any explicit allow.

The order that the policies are evaluated in has no effect on the outcome of the evaluation. All policies are evaluated, and the result is always that the request is either allowed or denied. Suppose that a conflict occurs (one policy allows an action and another policy denies an action). Then the most restrictive policy (that is, the policy that denies the action) is applied.

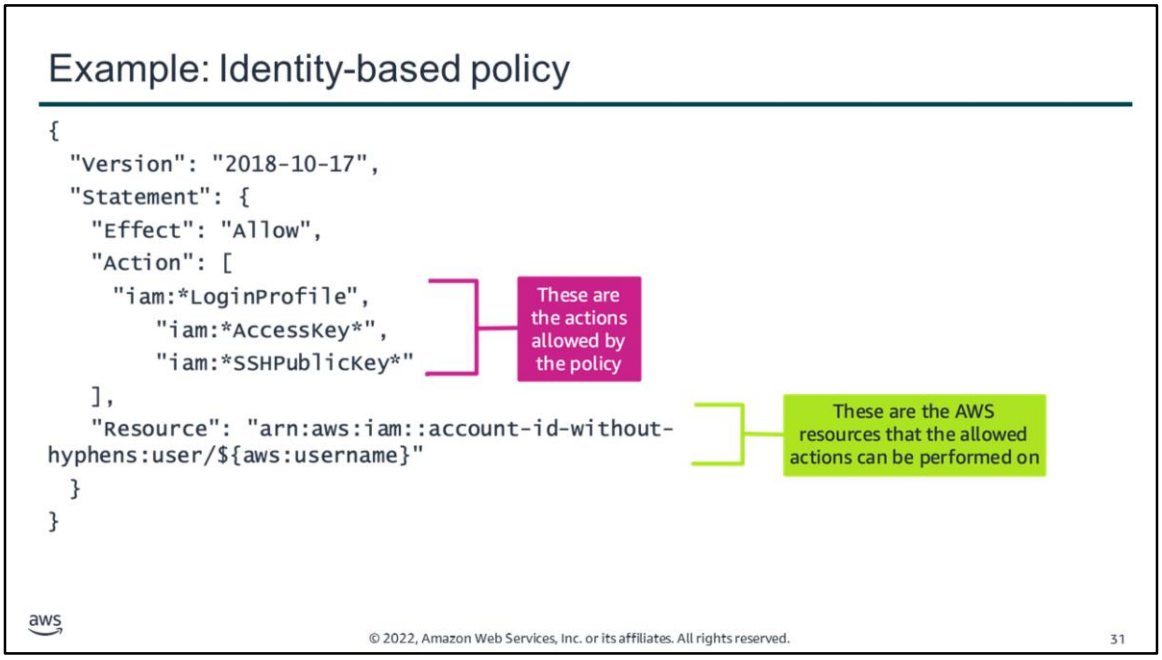
For additional information about evaluating and testing policies, see Testing IAM Policies with the IAM Policy Simulator in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_testing-policies.html.



Key takeaways from this section of the module include the following:

- Permissions to access AWS account services and resources are defined in IAM policy documents.
- Attach IAM policies to IAM users, IAM groups, or IAM roles.
- Follow the principle of least privilege when you grant account access.
- When IAM determines permissions, an explicit deny will always override any allow statement.

This section provides examples of how IAM is used for authorization.

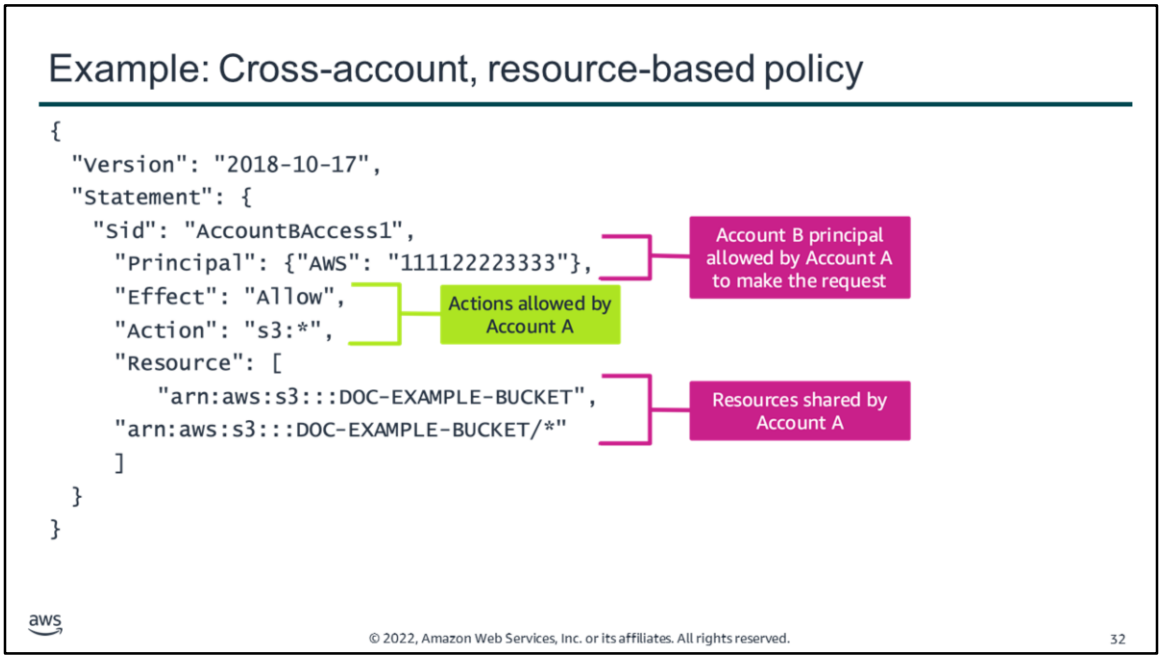


For accessibility: Code snippet from an identity-based policy. In this example, the policy is written to allow three actions to be performed on a specific AWS IAM account. **End of accessibility description.**

After you apply an identity-based policy to a user, group, or role, the policy allows or denies the entity the ability to perform specified actions.

In the example shown, if the entity is a user, the policy allows the user to do the following:
- Create, delete, get, or update their own password by using IAM LoginProfile actions
- Create, delete, list, or update their own access key by using IAM AccessKey actions
- Create, delete, get, list, or update their own Secure Shell (SSH) keys by using IAM SSHPublicKey actions

The actions in the policy include wildcards, which are indicated with an asterisk (*). This format provides a convenient way to include a set of related actions. Notice that the policy gets the AWS user name dynamically, as defined in the "Resource“ key.

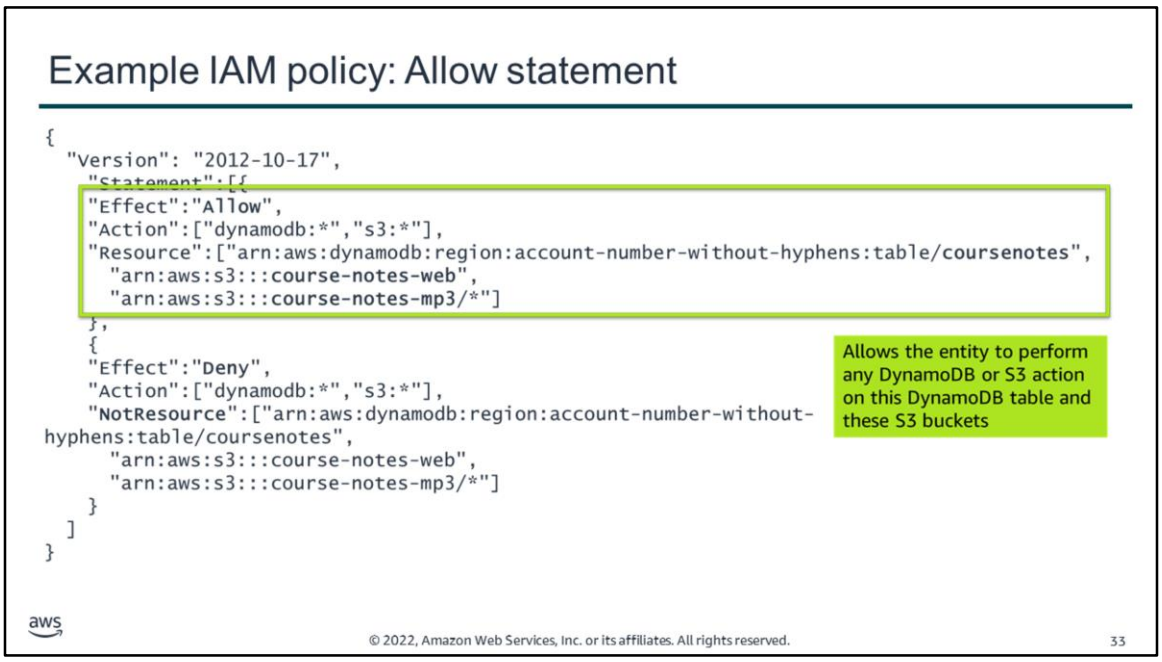


For accessibility: Code snippet from a cross-account, resource-based policy. In this example, the policy written by Account A allows a specific principal in Account B to perform any S3 action in a specific S3 bucket in Account A. **End of accessibility description.**

This example shows a resource-based policy that allows some cross-account access.

In this scenario, Account A created a resource-based policy. The policy grants Account B access to perform *any* Amazon S3 API operation—which is indicated by the asterisk (*)—on Account A's S3 bucket (named DOC-EXAMPLE-BUCKET).

This S3 bucket policy doesn’t specify any IAM users, groups, or roles. Instead, it specifies Account B (AWS account number 111122223333). Account B should create an IAM user policy to allow a user in Account B to access Account A's bucket.

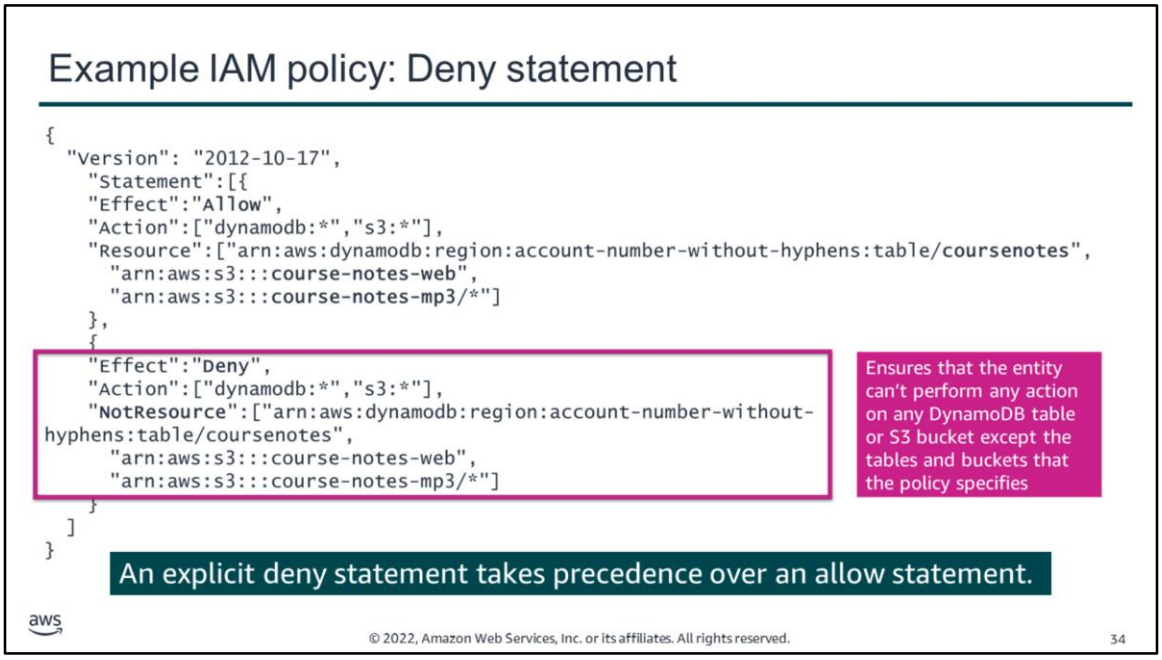


For accessibility: Code snippet of an allow statement portion of an IAM policy. The highlighted portion show a configuration that allows an entity to perform any DynamoDB or S3 action on a specific DynamoDB table and two S3 buckets. **End of accessibility description.**

To understand how IAM policy logic works, consider this example. The first half of this policy gives users access to only the following resources:

- DynamoDB table named coursenotes
- S3 bucket named course-notes-web
- S3 bucket named course-notes-mp3 and all the objects that it contains

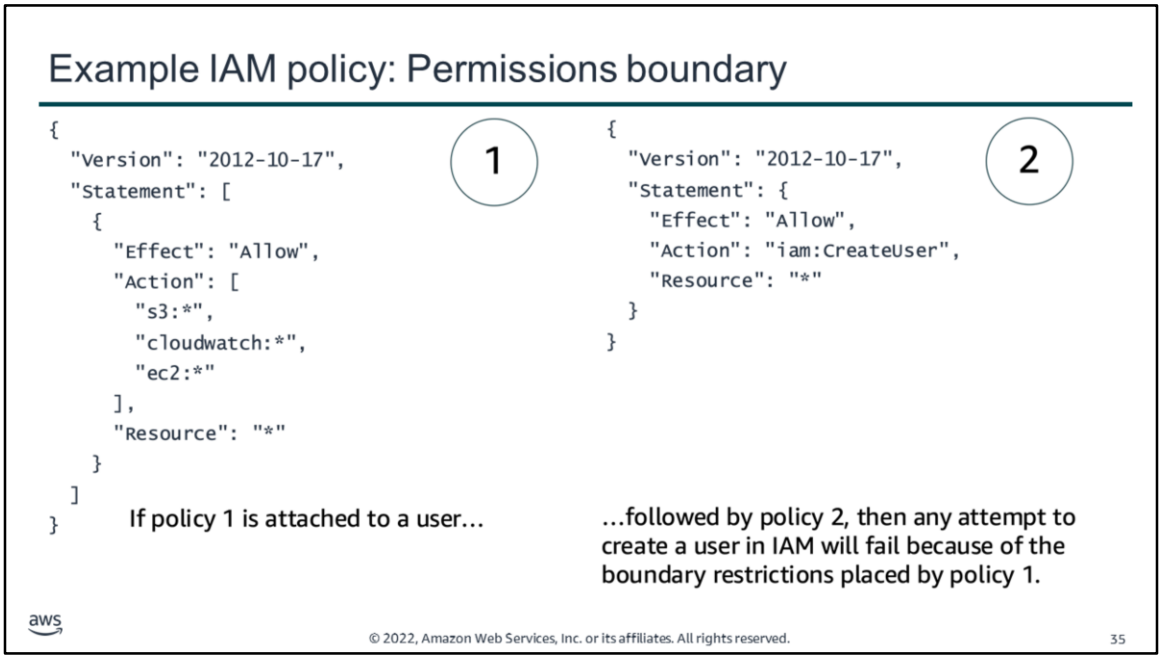
The next slide will discuss the second statement in the policy.



For accessibility: Code snippet of a deny statement portion of an IAM policy. The highlighted portion show a configuration that denies an entity to perform any DynamoDB or S3 action on any resource other than the specific DynamoDB table and S3 buckets specified in the allow section. **End of accessibility description.**

The second half of this policy is an explicit deny. Here, the explicit deny ensures that the entity can't perform any action on any DynamoDB table or S3 bucket. The only exceptions are the tables or buckets that are specified in the policy statement. Even if permissions are granted in another policy, the explicit deny overrides these permissions. An explicit deny statement takes precedence over an allow statement.

In its totality, this IAM policy allows access to specific DynamoDB and Amazon S3 resources. It then explicitly denies access to any other DynamoDB or Amazon S3 resources in the account.



For accessibility: Code snippets of a permissions boundary. The first snippet allows three types of actions to be completed, on in three specific areas S3, CloudWatch, and EC2. The second snippet allows only the IAM:CreateUser action on any resource. **End of accessibility description.**

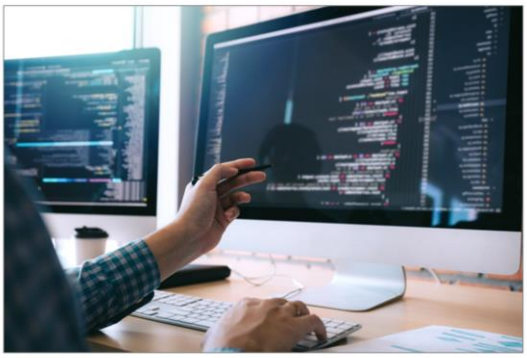
For reference, a permissions boundary is an advanced feature for using a managed policy to set the maximum permissions that an identity-based policy can grant to an IAM entity. An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies **and** its permissions boundaries.

Assume policy 1 is attached to IAM user ExampleUser. This policy allows the user to manage only Amazon S3, Amazon CloudWatch, and Amazon EC2 resources. The policy restricts all other services (including IAM).

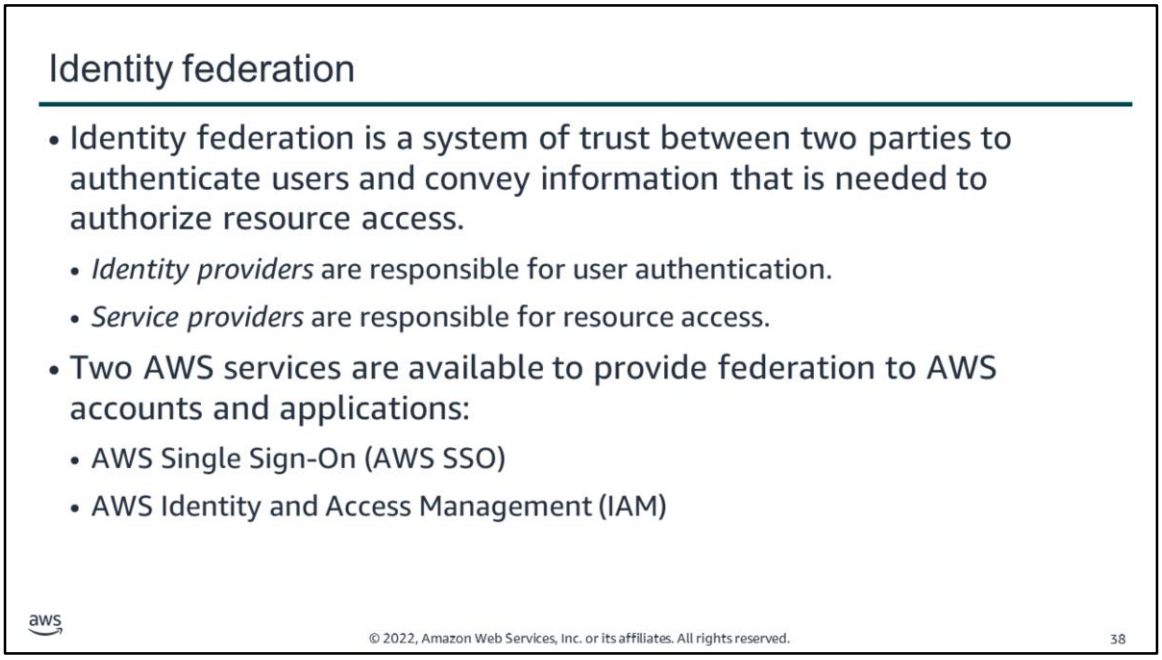
If IAM policy 2 is then attached to ExampleUser, and ExampleUser attempts to perform the iam:CreateUser operation, the operation fails. This is due to the restrictions placed by policy 1. Modifying the boundary policy to allow management of the IAM service would allow ExampleUser to run the iam:CreateUser operation successfully.

For more information about boundary policies, see Permissions Boundaries for IAM Entities in the *AWS Identity and Access Management User Guide* at https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html.

Demonstration: Amazon S3 Cross-Account Resource-Based Policy



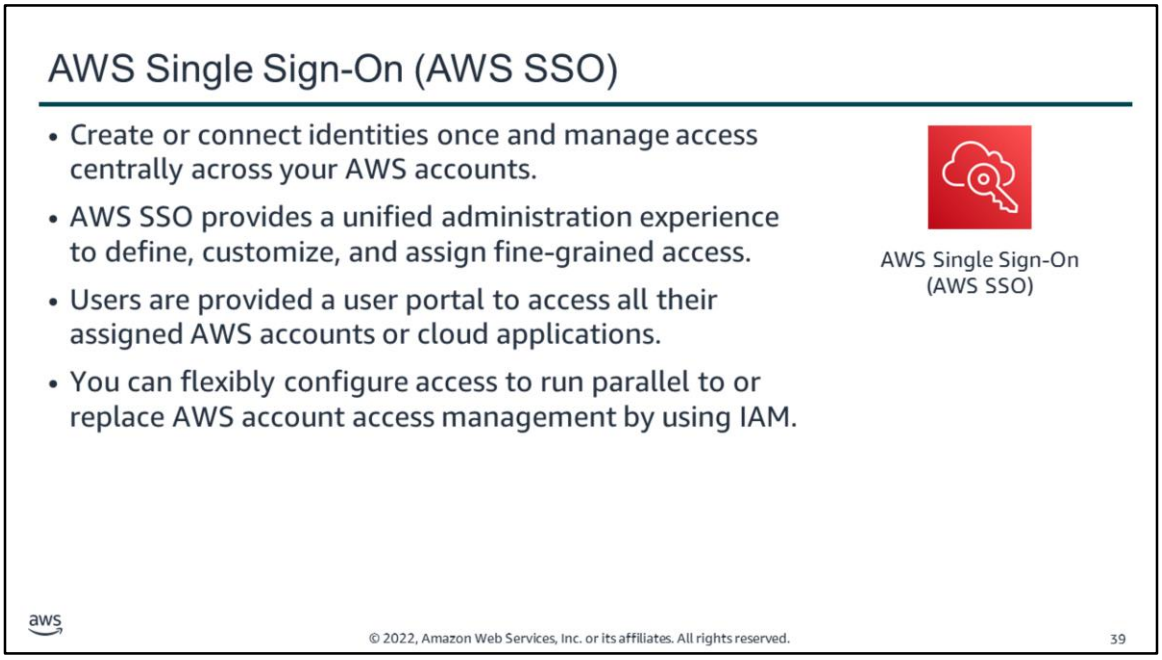
This section covers additional authentication and access management services.



Identity federation is a system of trust between two parties to authenticate users and convey information needed to authorize access to resources. Identity providers (IdPs) are responsible for user authentication. Service providers (SPs), such as services or applications, are responsible for controlling access to resources. Through administrative agreement and configuration, the SP trusts the IdP to authenticate users, and grants them access to the requested resources.

Two AWS services are available to provide federation to AWS accounts and applications: AWS Single Sign-On (AWS SSO) and IAM. If you are using a single centralized directory, AWS SSO is a great option to employ. If you are using multiple directories within your organization, or if you wish to use attribute-based permissions, consider IAM.

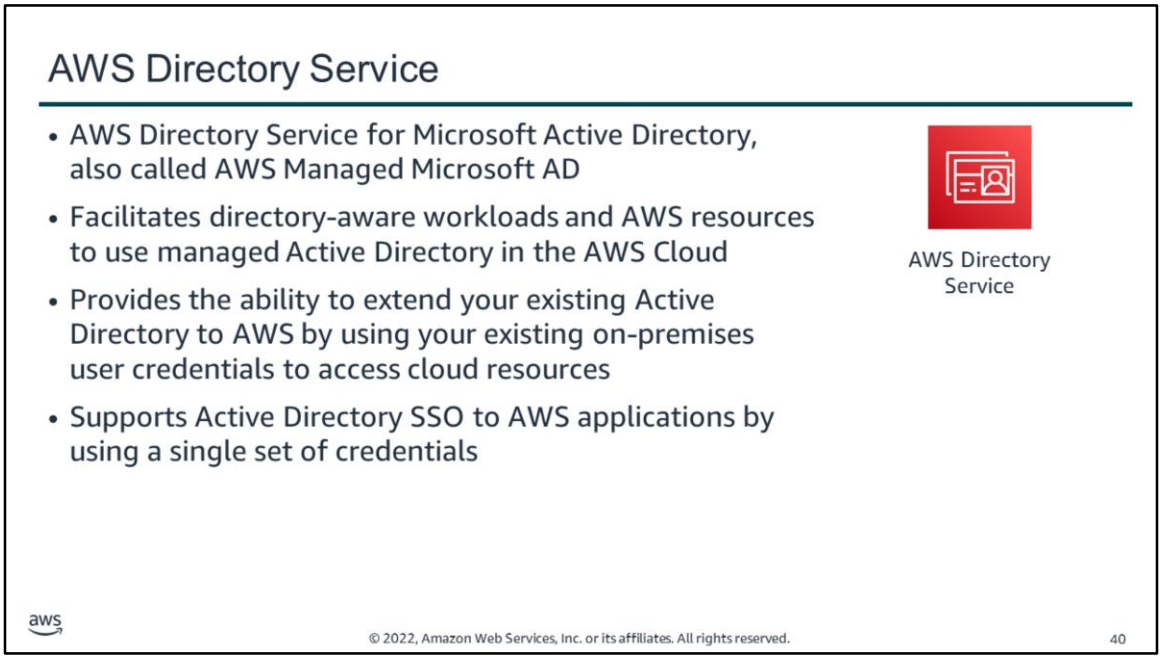
For more information, see Identity Federation in AWS at https://aws.amazon.com/identity/federation.



With AWS Single Sign-On (AWS SSO), you can create or connect identities once in AWS, and centrally manage access across your AWS accounts. AWS SSO provides a unified administration experience to define, customize, and assign fine-grained permissions based on common job functions.

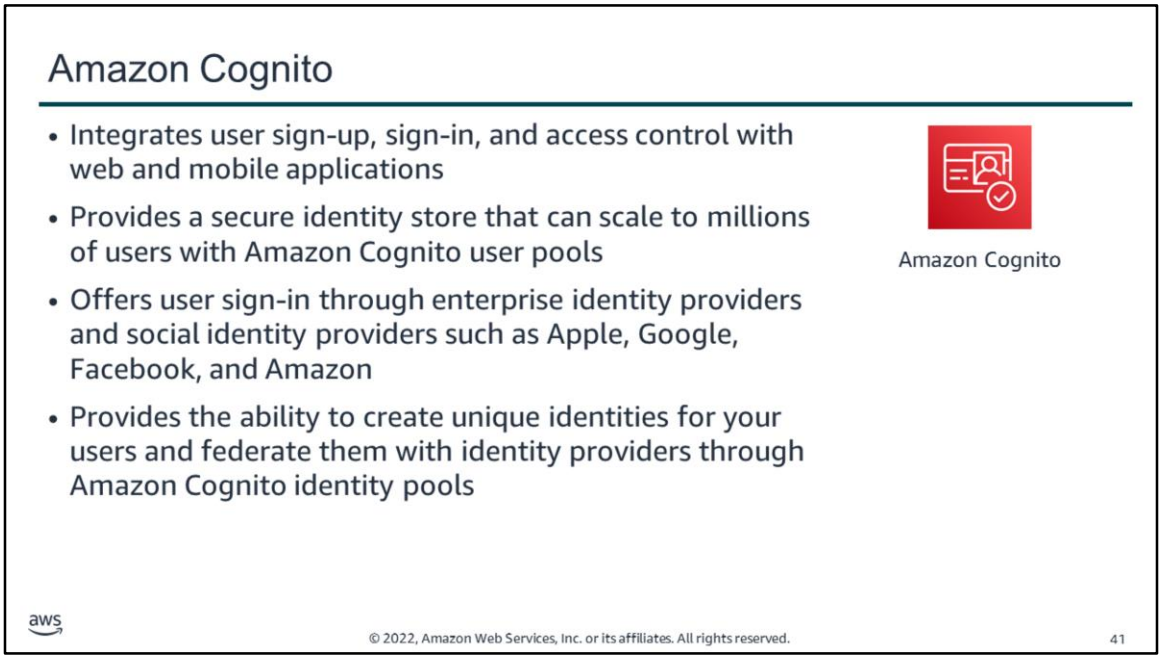
Users in your AWS SSO environment can use their directory credentials to access their user portal. Users can access all their assigned AWS accounts or cloud applications. You can flexibly configure access to run parallel to or replace AWS account access management by using IAM. AWS SSO supports commonly used cloud applications such as Microsoft 365 and Salesforce. The service provides application integration instructions that eliminate the need for administrators to learn the configuration nuances of each cloud application.

For more information, see What is AWS Single Sign-On? in the *AWS Single Sign-On User Guide* at https://docs.aws.amazon.com/singlesignon/latest/userguide/what-is.html.



AWS Directory Service for Microsoft Active Directory is also commonly referred to as AWS Managed Microsoft AD. AWS Managed Microsoft AD makes it possible for directory-aware workload and AWS resources to use managed Active Directory in the AWS Cloud. The service is built on Microsoft Active Directory, which eliminates the requirement to synchronize or replicate your existing Active Directory data to the cloud. With this service, you can use your existing on-premises user credentials to access cloud resources, which simplifies the process to extend your existing Active Directory to AWS.

For more information, see AWS Directory Service: Managed Microsoft Active Directory in AWS at https://aws.amazon.com/directoryservice.

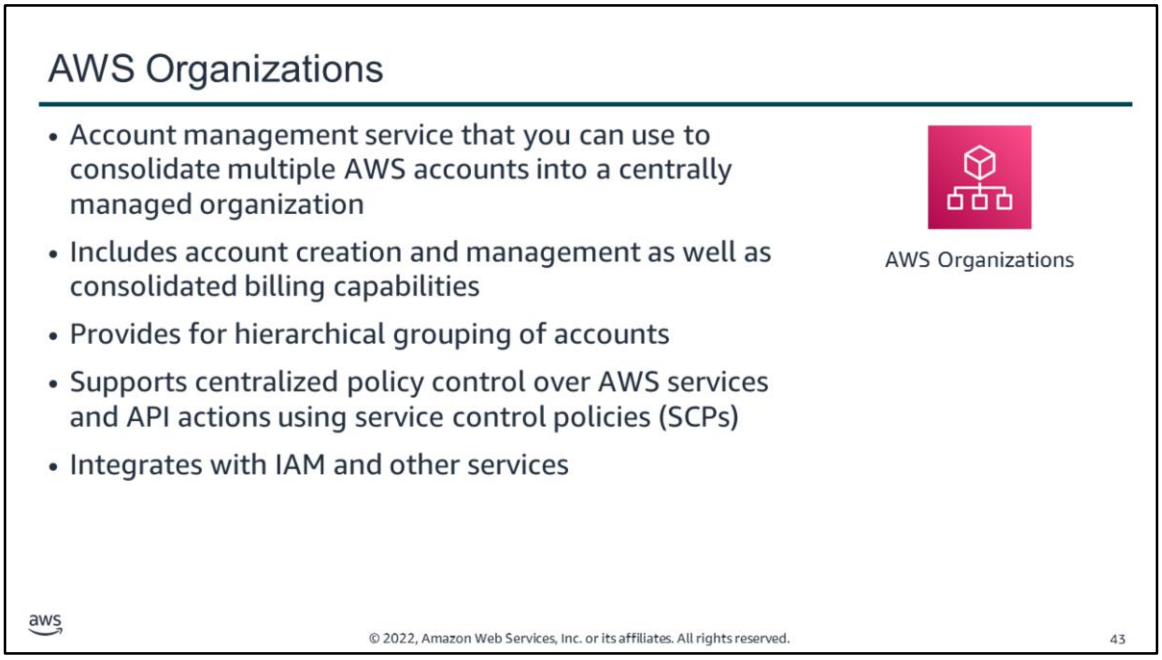


Amazon Cognito is a service that provides authentication, authorization, and user management (sign-up, sign-in, and access control) for mobile and web applications. The service provides a secure identity store, which can help you to scale effectively for millions of users. Amazon Cognito supports direct sign-in through user name and password. But it also supports third-party authentication through social identity providers such as Apple, Google, Facebook, and Amazon.

Amazon Cognito relies on two main components to provide its services to you: user pools and identity pools. *User pools* are directories that provide sign-up and sign-in options for your application users. This integrates with social identity providers and supports security features such as MFA and phone verification. With *identity pools*, you can grant your users access to AWS services through temporary, limited-privilege AWS credentials.

For more information, see What Is Amazon Cognito? in the *Amazon Cognito Developer Guide* at https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html.

This section describes the AWS Organizations service.



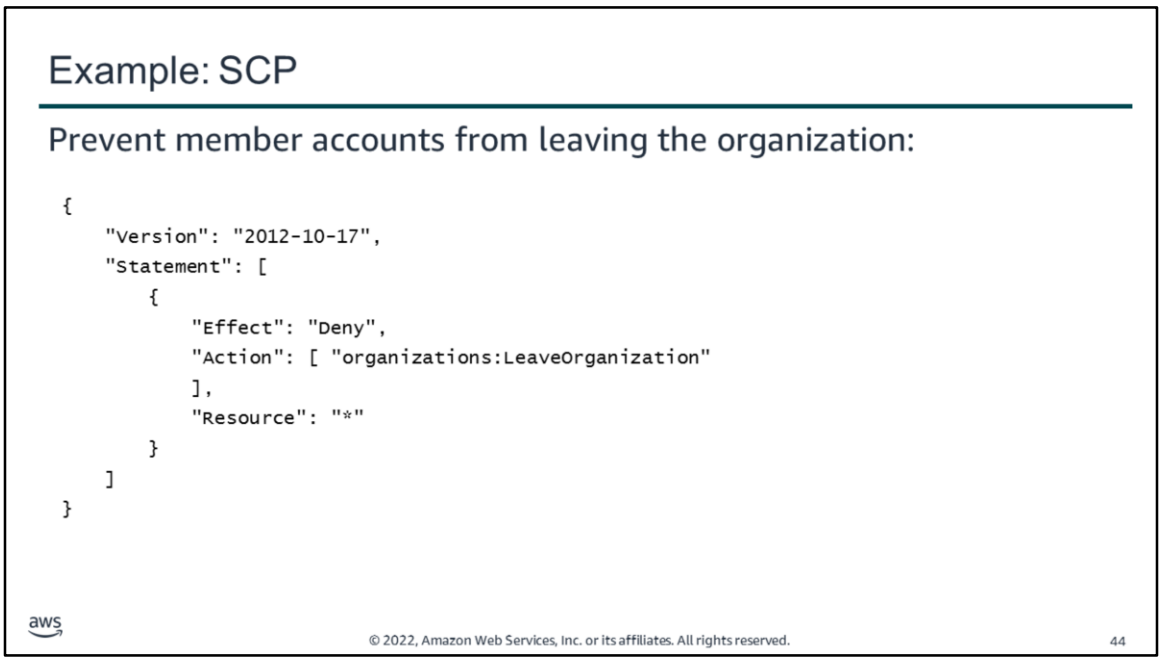
AWS Organizations is an account management service that you can use to create an *organization* where you can consolidate multiple accounts and centrally manage them. AWS Organizations provides centralized account creation and management, as well as consolidated billing capabilities. With these features, you can manage your security, compliance, and budgetary needs more efficiently.

Organizations also provides the ability to hierarchically group your accounts in *organizational units (OUs)* and attach different access policies to each. This provides the ability to create and customize fine-grained policies, which you can target to a single OU or attach to multiple OUs. You can nest OUs within other OUs up to a depth of five levels, which helps you to structure your hierarchy as you prefer.

Another key feature of Organizations is the use of *service control policies (SCPs)* to specify the maximum permissions for member accounts in your organization. This helps you ensure that your accounts stay within your organization's access control guidelines. You cannot use SCPs to grant any permissions. They can only restrict access to a service, resource, or API action for any member account, user, or role that you determine. Any restrictions put in place by an SCP will remain in place even if the restricted actions are implicitly allowed elsewhere.

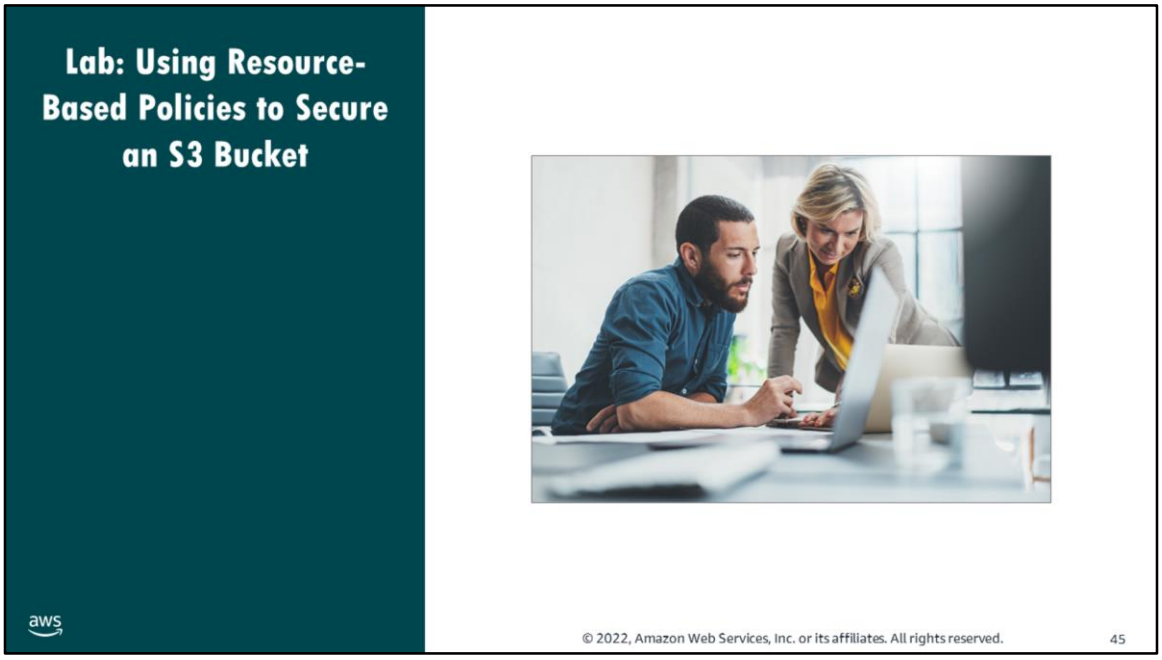
Organizations builds upon IAM by expanding the granular control that IAM provides to the account level. It does this by giving you control over what users and roles in an account or a group of accounts can do. This additional layer of control ensures that users can access only what both Organizations policies and IAM policies allow. If either service blocks an operation, the user will not be able to access that operation.

For more information, see What is AWS Organizations? in the *AWS Organizations User Guide* at https://docs.aws.amazon.com/organizations/latest/userguide/orgs_introduction.html.

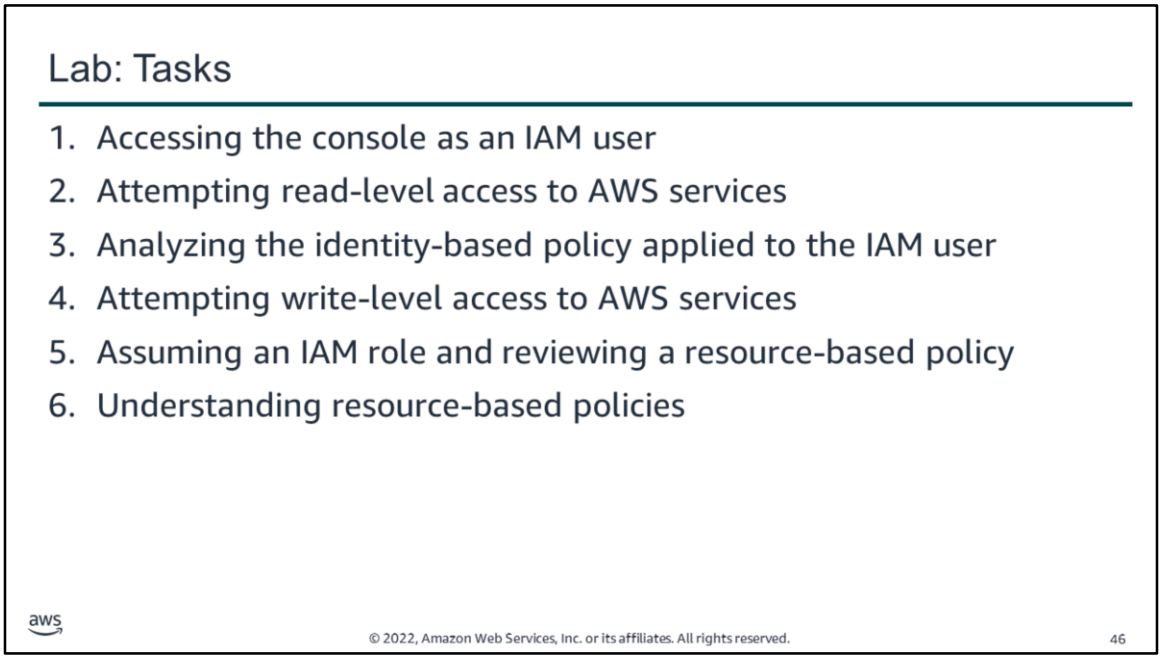


This SCP example prevents member accounts from leaving the organization. The effect of the policy statement is to explicitly deny the organizations:LeaveOrganization action, which prevents member accounts from leaving.

For more information, see Service Control Policies (SCPs) in the *AWS Organizations User Guide* at https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html.



You will now complete the Using Resource-Based Policies to Secure an S3 Bucket lab.

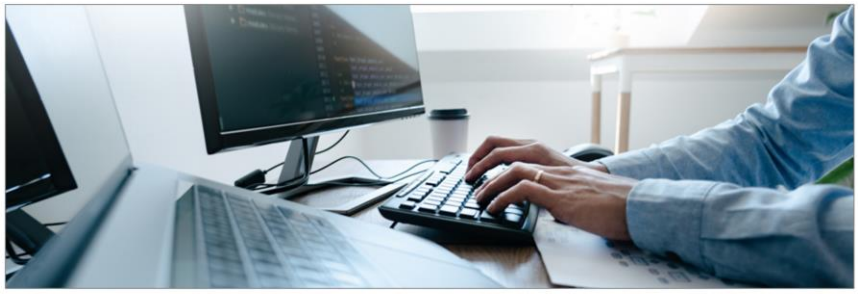


In this lab, you will complete the following tasks:

1. Accessing the console as an IAM user
2. Attempting read-level access to AWS services
3. Analyzing the identity-based policy applied to the IAM user
4. Attempting write-level access to AWS services
5. Assuming an IAM role and reviewing a resource-based policy
6. Understanding resource-based policies

Begin Lab: Using Resource-Based Policies to Secure an S3 Bucket

Duration: 60 minutes

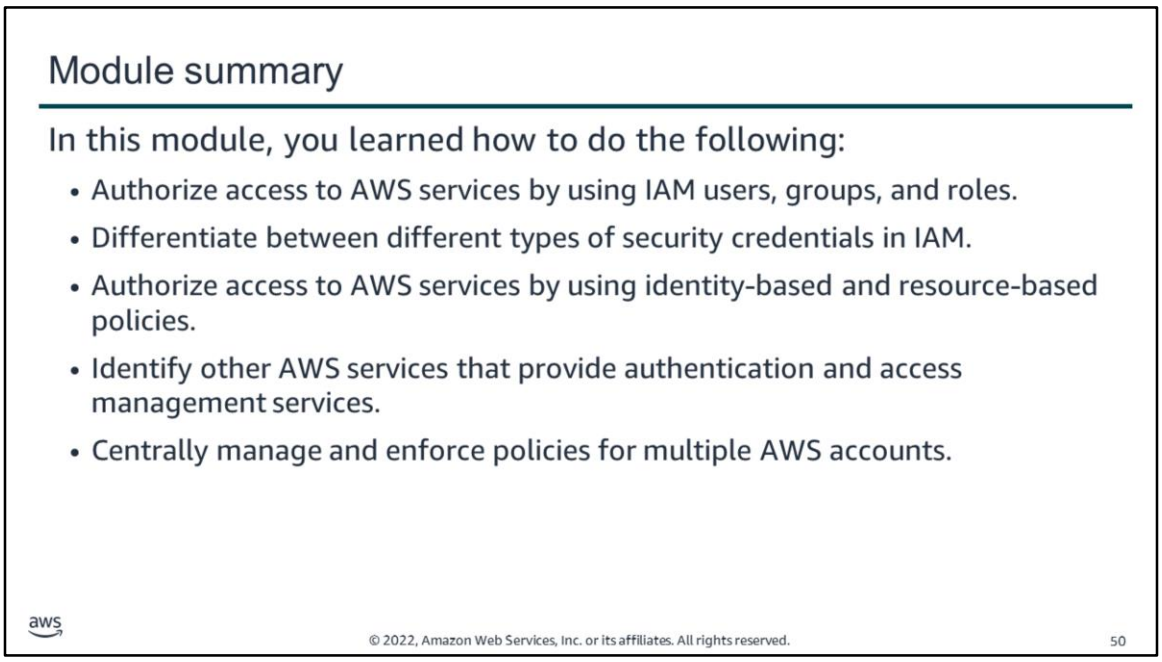


47

It's now time to start the lab.

After you complete the lab, your educator might choose to lead a conversation about the key takeaways from the lab.

It’s now time to review the module, and wrap up with a knowledge check and discussion of a practice certification exam question.



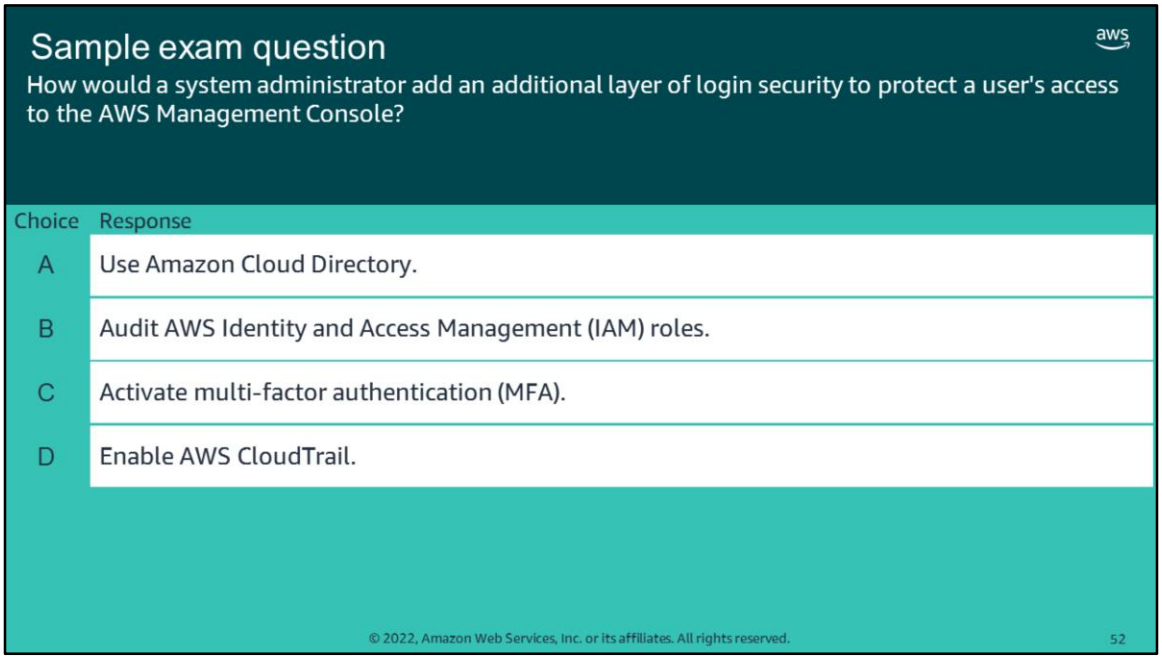
In this module you learned how to do the following:

- Authorize access to AWS services by using IAM users, groups, and roles.
- Differentiate between different types of security credentials in IAM.
- Authorize access to AWS services by using identity-based and resource-based policies.
- Identify other AWS services that provide authentication and access management services.
- Centrally manage and enforce policies for multiple AWS accounts.

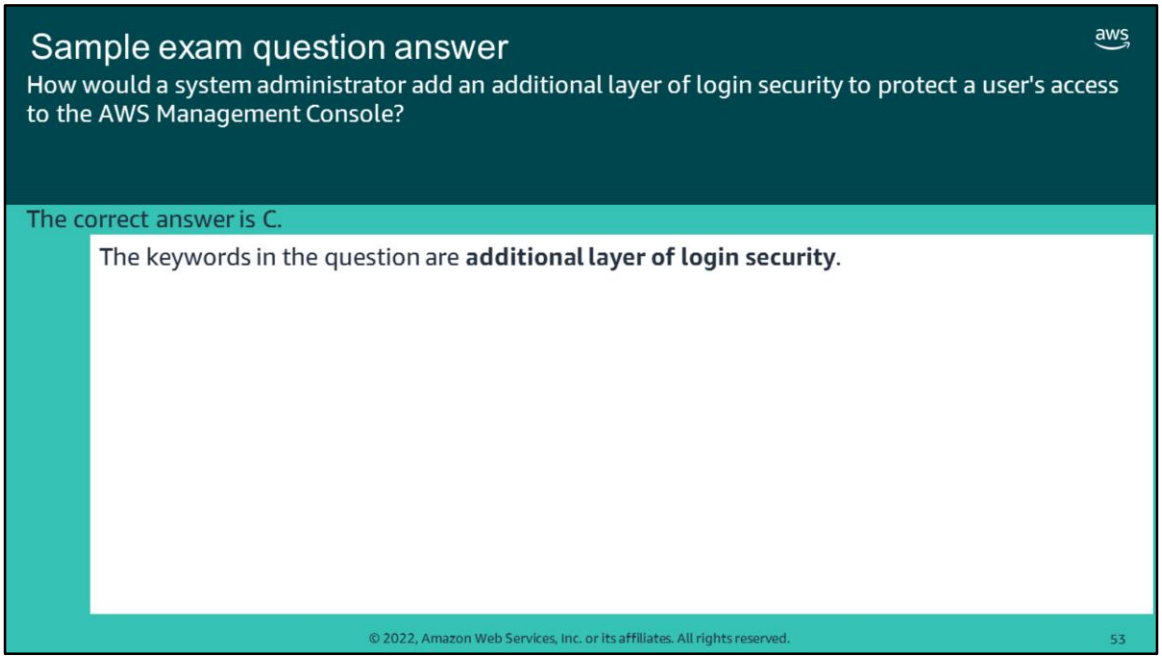
Complete the knowledge check

51

It is now time to complete the knowledge check for this module.



Look at the answer choices and rule them out based on the keywords.



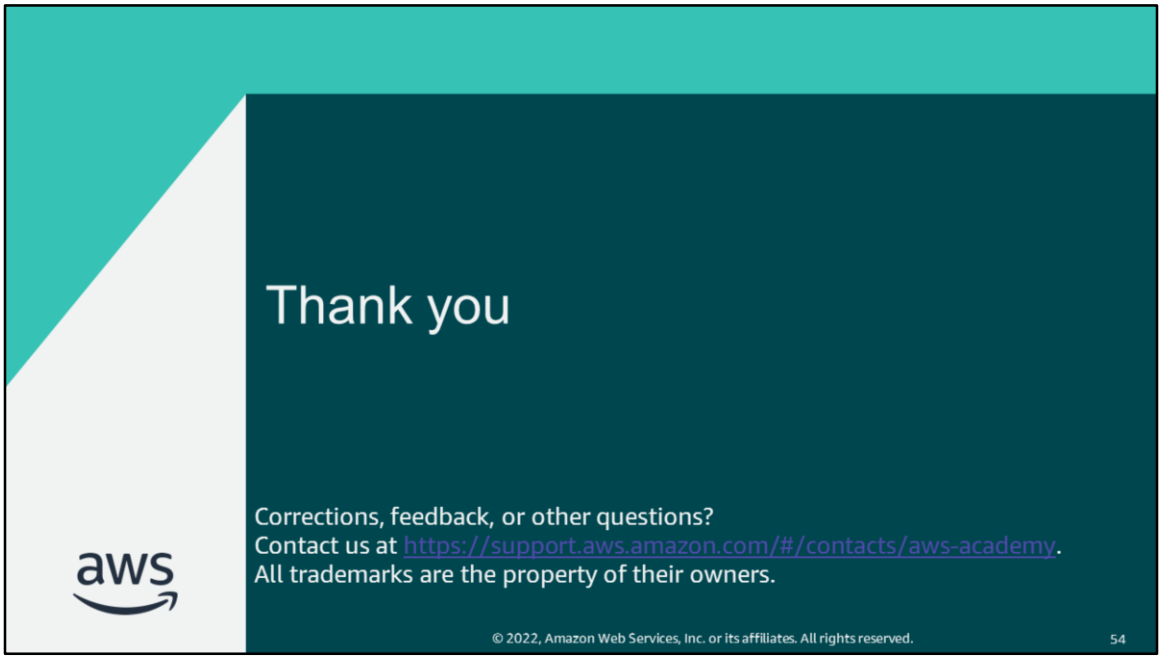
The key point to focus on is the desire to add an additional layer of login security. MFA is the only option that provides this.

The correct answer is C.

MFA is a simple best practice that adds an extra layer of protection on top of a user name and password. With MFA activated, when a user signs in to the AWS Management Console, they will be prompted for their user name and password (the first factor—what they know). They will also be prompted for an authentication code from their MFA device (the second factor—what they have). Taken together, these multiple factors provide increased security for AWS account settings and resources.

Incorrect answers:

- Answer A: Using Cloud Directory would not add any additional layers of login security to the AWS Management Console.
- Answer B: An audit of IAM roles will show you the IAM users that currently have roles assigned, but this does not add any additional security.
- Answer D: With CloudTrail, you can log the date, time, and identity of users accessing your directory data. However, this does not provide additional login security.



Thank you for completing this module.