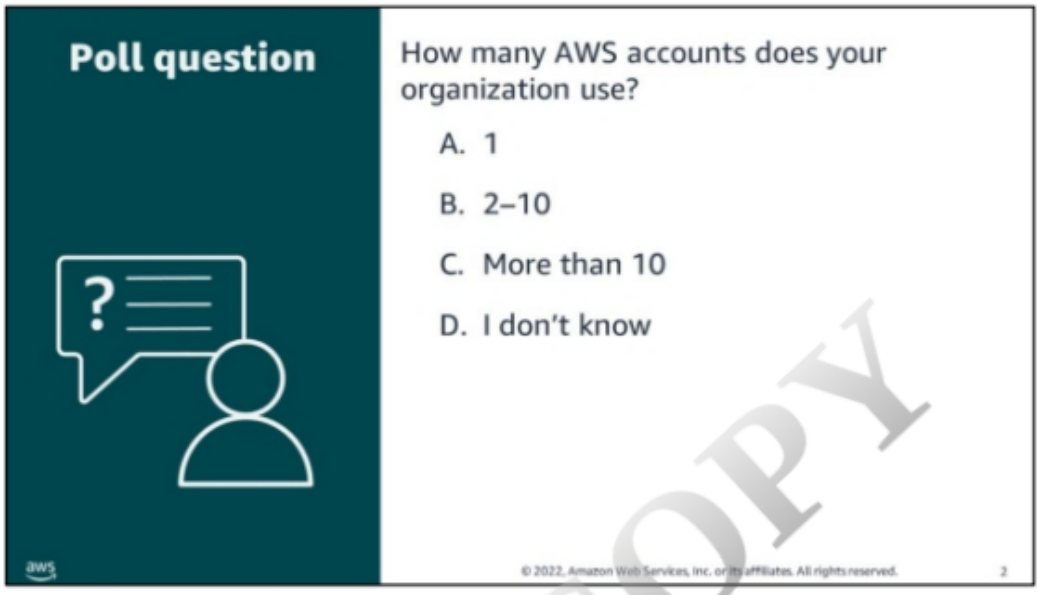
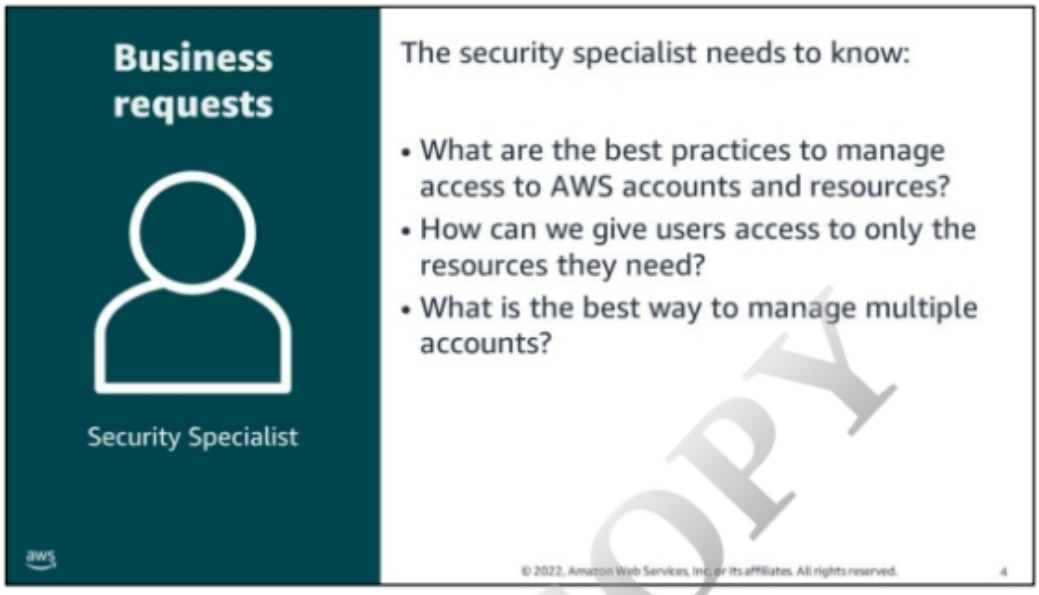
Printed by: 2d35e8483186bd2@placeholder.44518.edu. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Module overview

- Business requests
- Principals and identities
- Security policies
- Managing multiple accounts
- Module review
- Knowledge check

 3

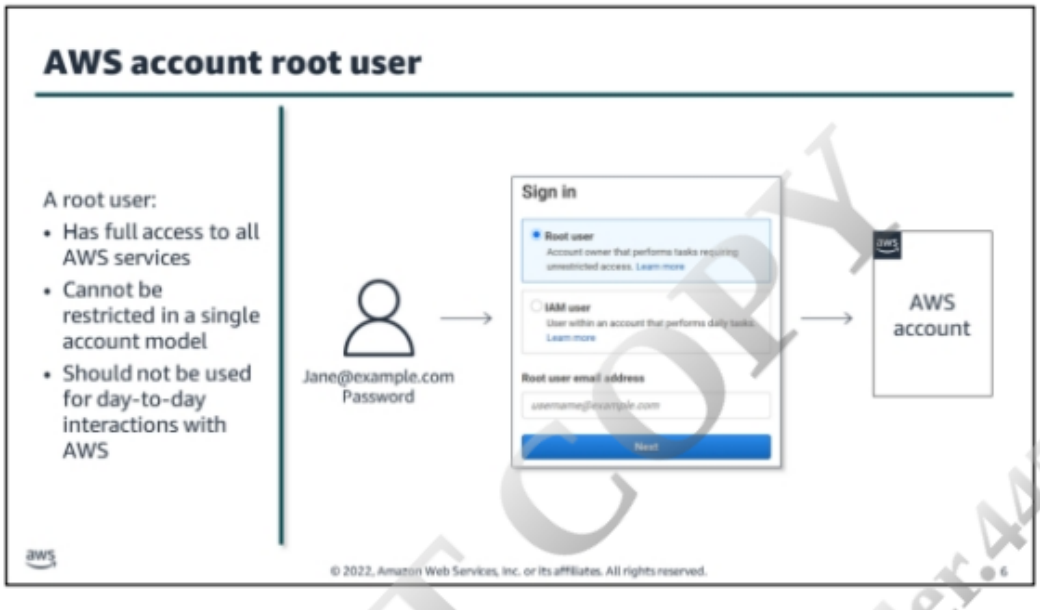


Imagine your security specialist meets with you to discuss how to start building accounts with least privilege in AWS. Here are some questions they are asking about account security.

At the end of this module, you meet with the security specialist and present some solutions.

The security specialist asks, "What are the best practices to manage access to AWS accounts and resources?"

The security team must start setting up accounts. The company wants your advice for how to provide access.



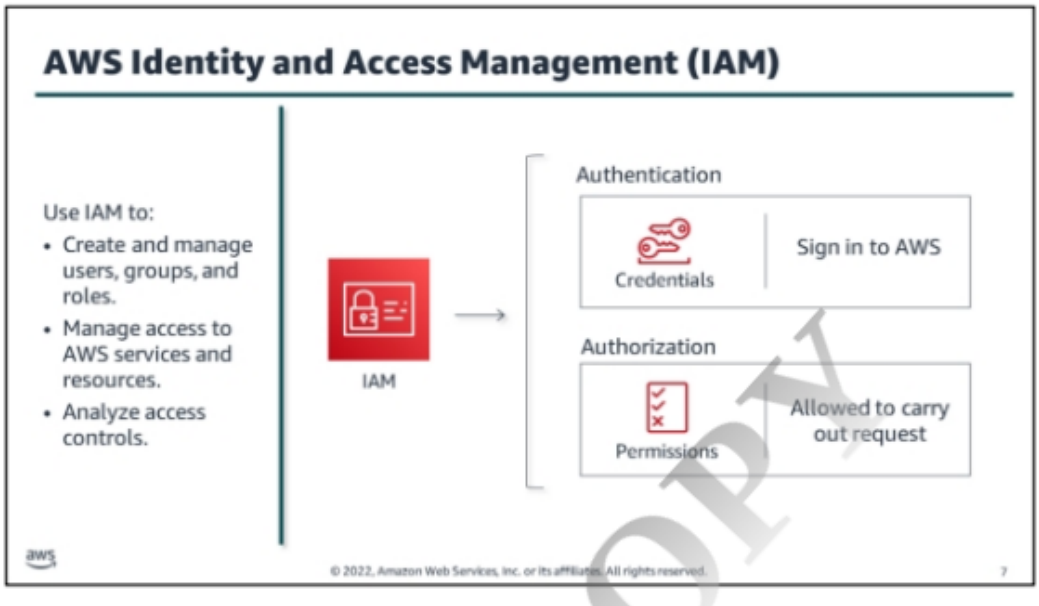
When you first create an AWS account, you begin with a *root user*. This user has complete access to all AWS services and resources in the account. Access the root user identity by signing in with the email address and password provided when you created the account. AWS strongly recommends that you not use root account credentials for day-to-day interactions with AWS. Create users for everyday tasks. You can manage and audit users with relative ease.

Create your additional users and assign permissions to these users following the *principle of least privilege*. Grant users only the level of access they require and nothing more. You can start by creating an administrator user. Manage the account with the administrator user instead of the root user.

As a best-practice, require multi-factor authentication (MFA) for your root user. It provides you with an extra layer of security for your AWS accounts. Use your root user only for tasks that require it.

For more information about the root user, see "AWS account root user" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html).

For information about least privilege and IAM best practices, see "Grant least privilege" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege).



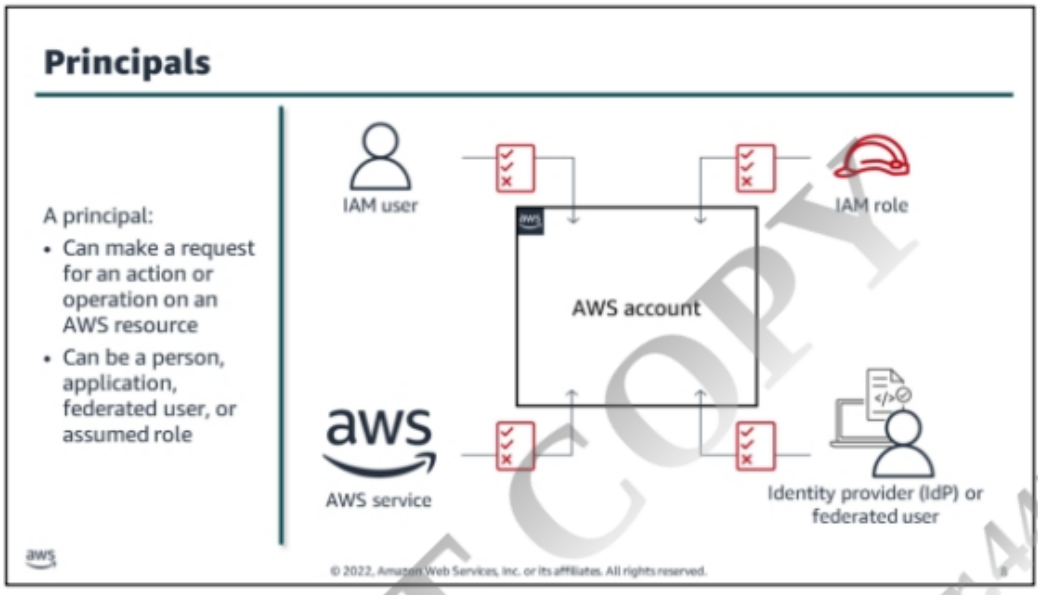
IAM is a web service that helps you securely control access to AWS resources. Use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

Think of IAM as the tool to centrally manage access to launching, configuring, managing, and terminating your resources. You have granular control over access permissions. This control is based on resources and helps you define who has permissions to which API calls.

You manage access in AWS by creating and using security policies. You learn about IAM users, IAM user groups, and roles in this section.

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

For more information about policy types and their uses, see "Policies and permissions in IAM" (https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html).

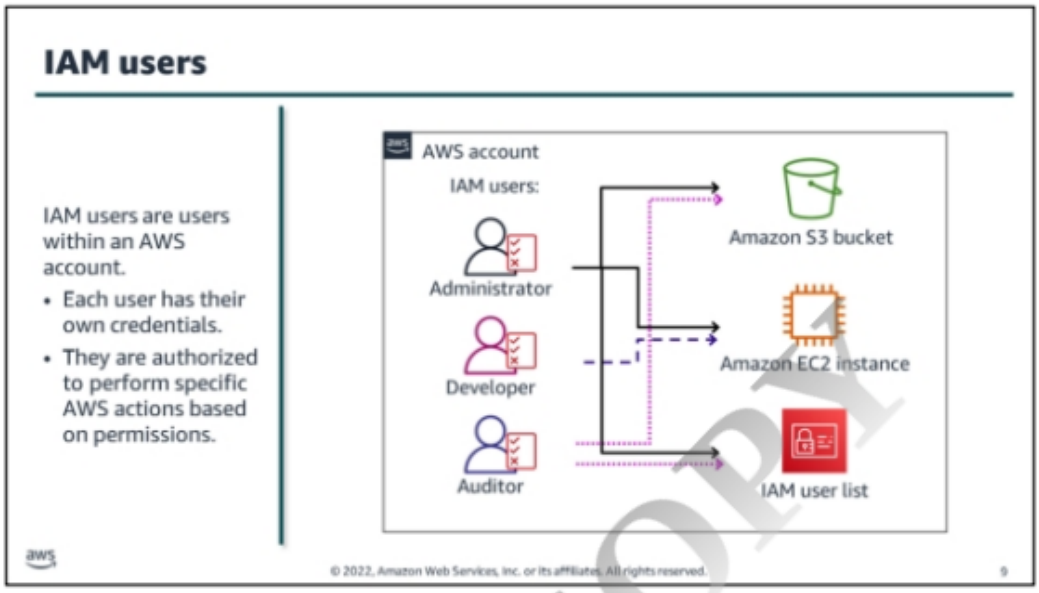


A *principal* is an entity that can request an action or operation on an AWS resource. IAM users and IAM roles are the most common principals you work with, and you learn about them in this lesson.

The principal can also be an AWS service, such as Amazon Elastic Compute Cloud (Amazon EC2), a Security Assertion Markup Language 2.0 (SAML 2.0) provider, or an identity provider (IdP). With an IdP, you manage identities outside of AWS IAM, for example, Login with Amazon, Facebook, or Google. You can give these external identities permissions to use AWS resources in your account.

Federated users are external identities that are not managed directly by AWS IAM.

For more information about federated users, see "Identity federation in AWS" (https://aws.amazon.com/identity/federation/).

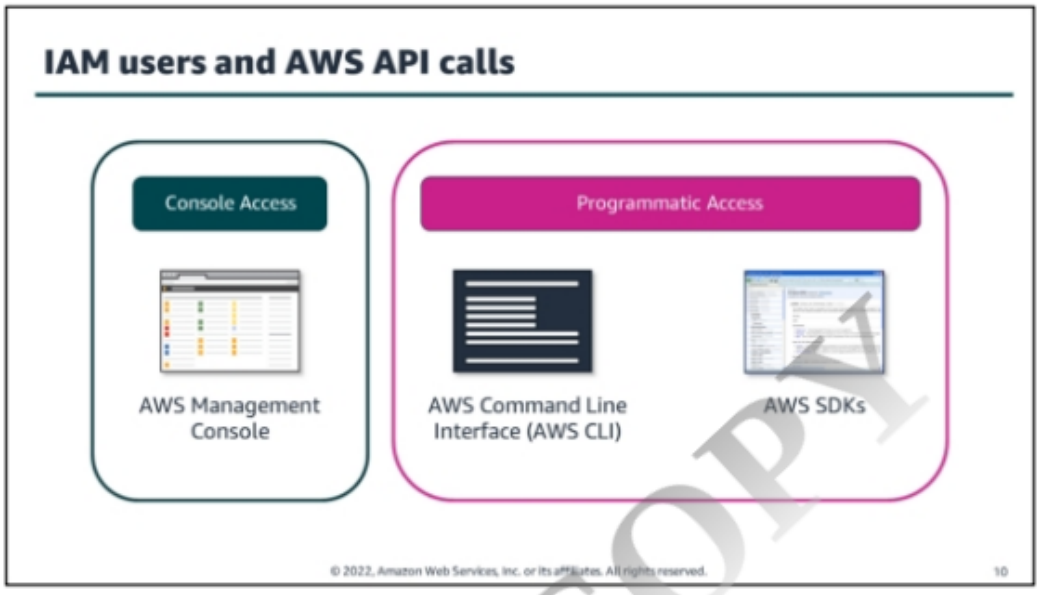


By default, a new IAM user has no permissions to do anything. The user is not authorized to perform any AWS operations or access any AWS resources. An advantage of having individual IAM users is that you can assign permissions individually to each user.

For example, in this diagram you note three IAM users—an administrator, developer, and auditor—and their permissions within an AWS account. The administrator has permissions to access an S3 bucket, an EC2 instance, and a list of IAM users in your account. The auditor has read-only permissions to S3 and AWS IAM, but not EC2. The developer only has permissions to the EC2 instance.

As a best practice, require multi-factor authentication (MFA) for your IAM users and set up an IAM user password policy.

For more information about IAM users, see "IAM users" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html).



Provide the type of credentials required for the type of access that a user will need.

Ways to access AWS services:

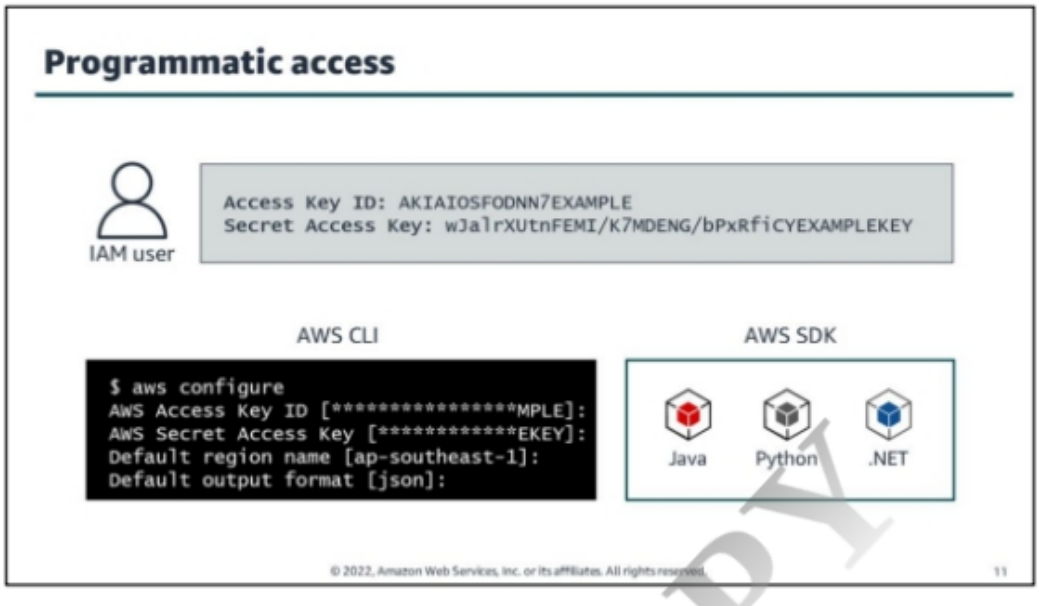
1. AWS Management Console access – Create a password for a user.
2. Programmatic access – The IAM user might need to make API calls, use the AWS CLI, or use the AWS Tools for Windows PowerShell or AWS API tools for Linux. In that case, you will create an access key (access key ID and a secret access key) for that user.

As a best practice, apply the principle of least privilege. This means that you create only the credentials that the user needs. For example, do not create access keys for a user who requires access only through the console.

AWS requires different types of security credentials, depending on how you access AWS.

For more information, see "Understanding and getting your AWS credentials" in the *AWS General Reference* (https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html).

For information about password creation, see "Managing passwords for IAM users" in the *AWS Identity Access and Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_passwords_admin-change-user.html).



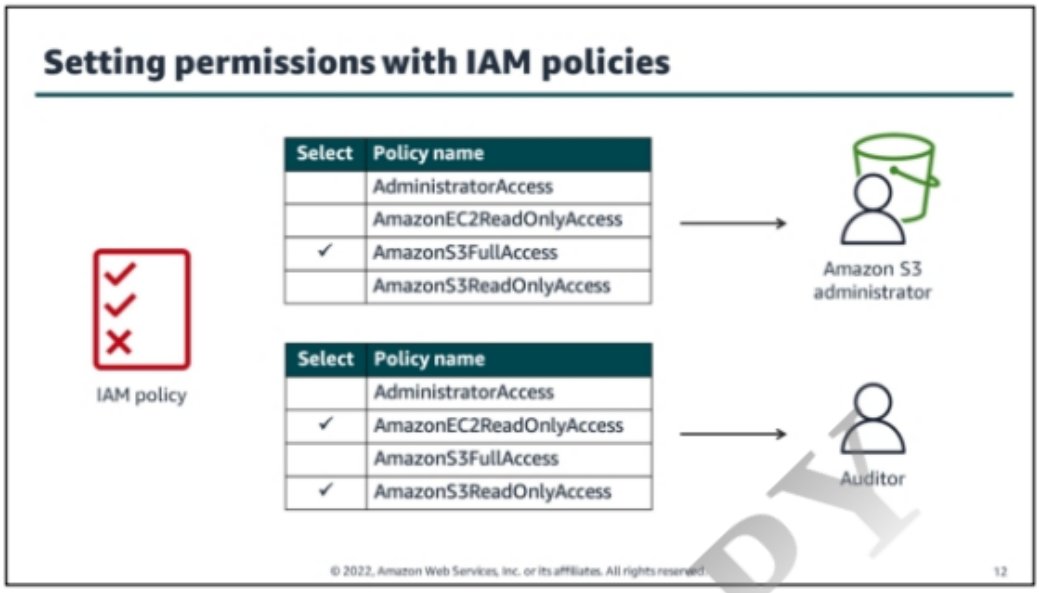
Programmatic access gives your IAM user the credentials to make API calls in the AWS CLI or AWS SDKs. AWS provides an SDK for programming languages such as Java, Python and .NET.

When programmatic access is granted to your IAM user, it will create a unique key pair made up of an access key ID and secret access key. Use your key pair to configure the AWS CLI or make API calls through an AWS SDK.

To set up AWS CLI in your client, enter the aws configure command. The example code shows the four elements required to configure your IAM user in AWS CLI:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name
- Default output format (json, yaml, yaml-stream, text, table)

For more information about configuring your key pair in AWS CLI, see "Configuration basics" in the *AWS Command Line Interface User Guide* (https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html).



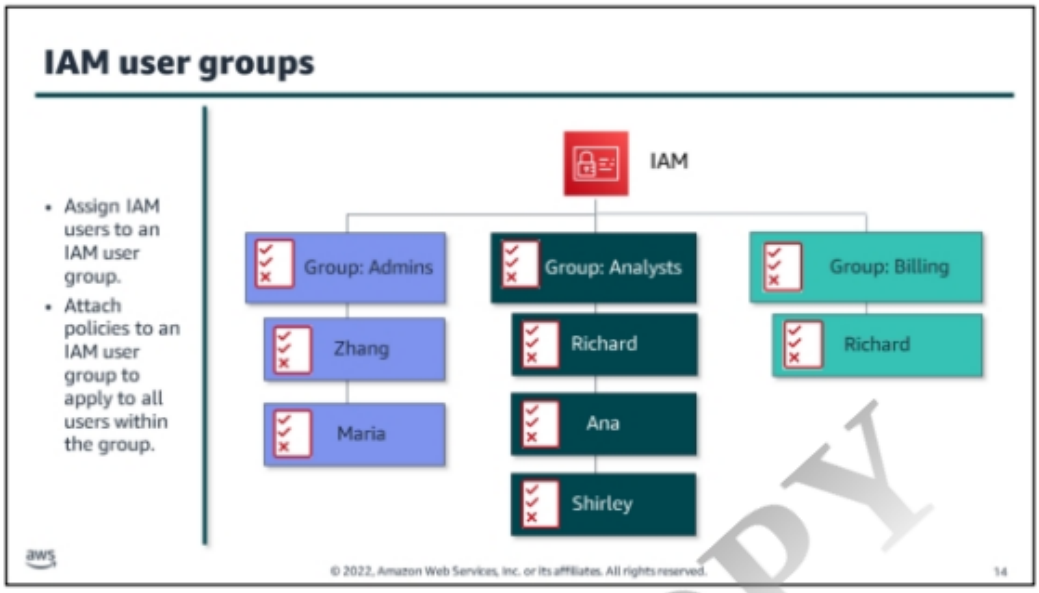
To allow IAM users to create or modify resources and perform tasks, do the following:

1. Create or choose IAM policies that grant IAM users permission to access the specific resources and API actions they will need.
2. Attach the policies to the IAM users or groups that require those permissions.

Users only have the permissions you specify in the policy. Most users have multiple policies. Together, they represent the permissions for that user.

In the diagram, you choose to give the Amazon S3 administrator full access to S3, but you do not grant full access to all services in your AWS account. You attach the AmazonEC2ReadOnlyAccess and AmazonS3ReadOnlyAccess policies to an auditor who needs to know what resources exist in your account. The auditor should not be able to modify or delete anything.

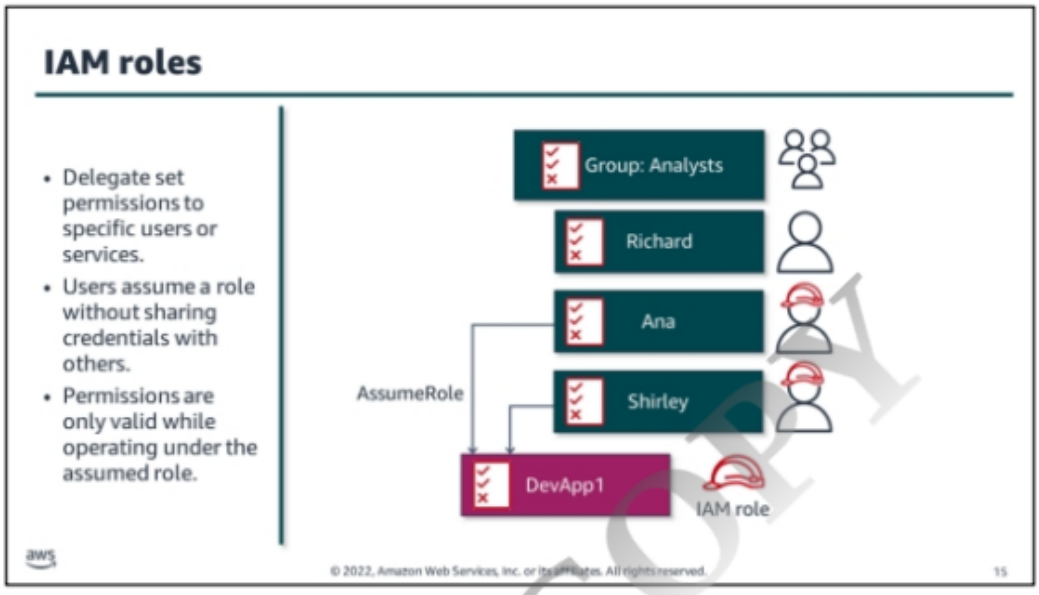
As a best practice, attach only the policies needed to complete the work required by that user.



An IAM user group is a collection of IAM users. With user groups, you can specify permissions for multiple users, which makes it easier to manage the permissions.

A user can be a member of more than one user group. In the diagram, Richard is a member of the Analysts group and the Billing group. Richard gets permissions from both IAM user groups.

For more information about user groups, see "IAM user groups" (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_groups.html).



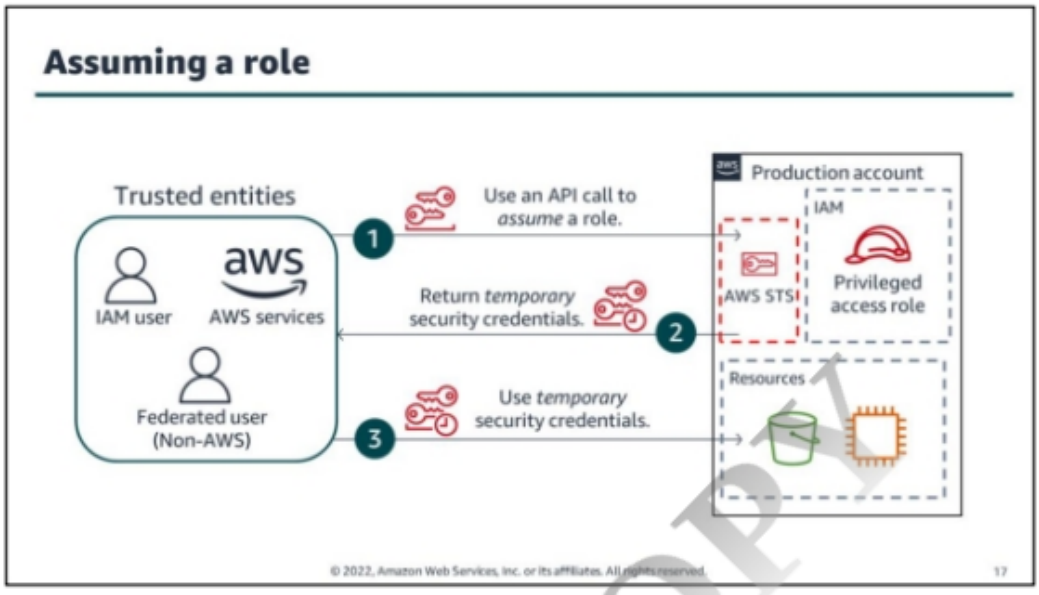
IAM roles deliver temporary AWS credentials. They're easy to manage because multiple employees and applications can use the same role. There are no charges for using roles.

For example, in this diagram the IAM users Richard, Ana, and Shirley are members of the Analysts user group. As members of the Analysts group, these users inherit permissions assigned to the group. There is also an IAM role called DevApp1 that is being used for testing purposes. DevApp1 has its own set of permissions. Ana and Shirley can assume the role and temporarily use the permissions specific to the DevApp1 role.

While they assume this role, Ana and Shirley only have the permissions granted by the role and do not follow their group's inherited permissions.

The following are examples of how you might use IAM roles:

- Cross-account access – Developer Diego requires access to an S3 bucket in the Prod account.
- Temporary account access – Contractor Carlos requires temporary access to an S3 bucket in the Prod account.
- Least privilege – Require Diego to use IAM roles to delete a DynamoDB table.
- Audit – Administrator Ana wants to track who used an IAM role.
- Access for AWS services – Amazon Lex needs to use Amazon Polly to synthesize speech responses for your bot.
- IAM roles for EC2 – An application running on Amazon EC2 requires access to an S3 bucket and a DynamoDB table.
- SAML federation – Administrator Ana wants to use IAM with identities stored in an external IdP.



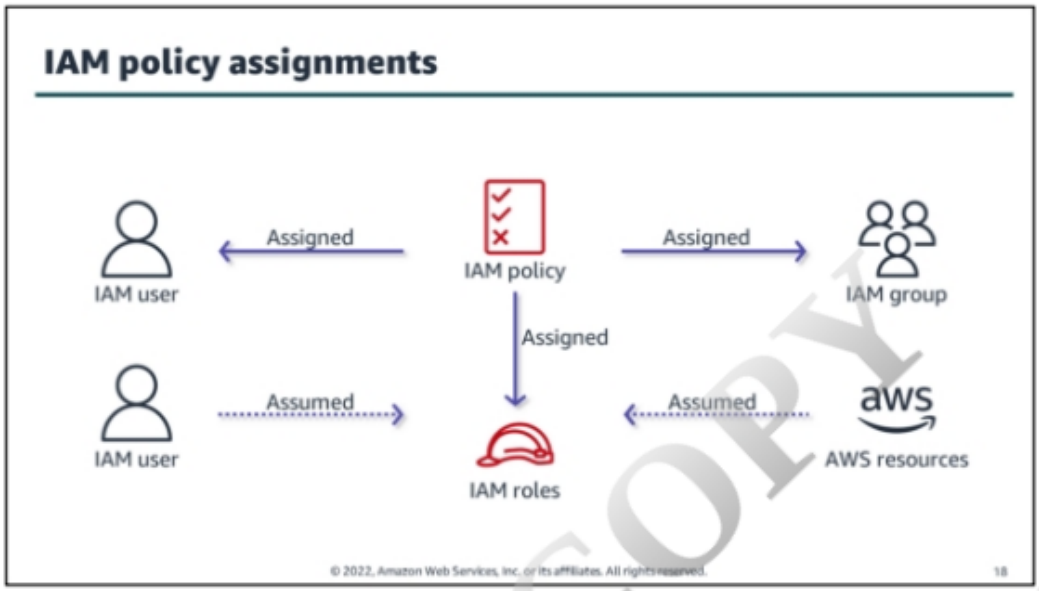
You assume a role using a trusted entity, such as an IAM user, an AWS service, or a federated user.

IAM users assume roles in the AWS Management Console or AWS Command Line Interface (AWS CLI). This action uses the AssumeRole API. AWS services can use the same API call to assume roles in your AWS accounts. Your federated users use either the AssumeRoleWithSAML *or* AssumeRoleWithWebIdentity API calls.

The API call is made to AWS Security Token Service (AWS STS). AWS STS is a web service that provides temporary, limited-privilege credentials for IAM or federated users. It returns a set of temporary security credentials consisting of an access key ID, a secret access key, and a security token. These credentials are then used to access AWS resources. The AssumeRole API is typically used for cross-account access or federation.

For more information about AWS STS, see the *AWS Security Token Service API Reference* (https://docs.aws.amazon.com/STS/latest/APIReference/Welcome.html).

For more information about using IAM roles, see "Using IAM roles" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use.html).



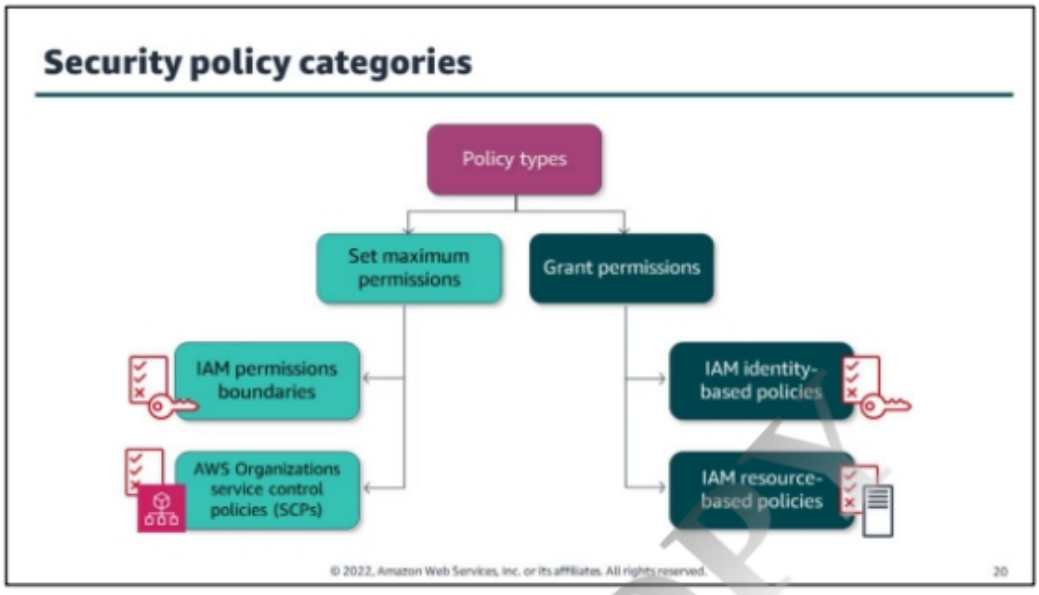
IAM provides you with the tools to create and manage all types of IAM policies (managed policies and inline policies). To add permissions to an IAM identity (IAM user, group, or role), you create a policy, validate the policy, and then attach the policy to the identity. You can attach multiple policies to an identity, and each policy can contain multiple permissions.

You learn more about IAM policies in the next section.

Use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources.

The security specialist asks, "How can we give users access to only the resources they need?"

The security team has users and roles set up. The company wants your advice about setting up permissions in security policies.



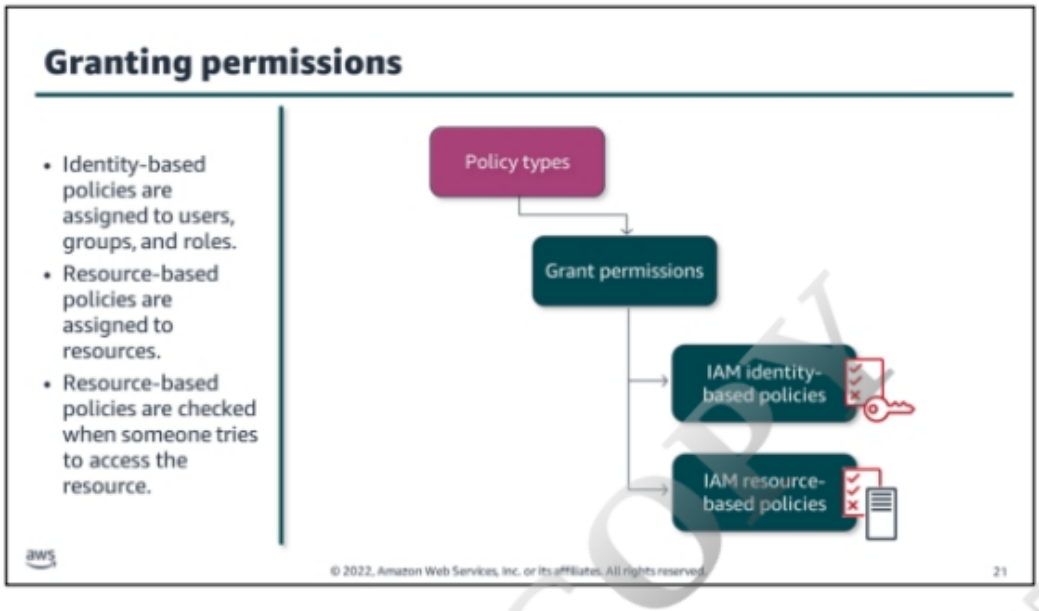
A *policy* is attached to an identity or resource to define its permissions. AWS evaluates these policies when a principal, such as a user, makes a request.

In the diagram, the policy types are responsible for either setting maximum permissions or granting permissions. IAM permissions boundaries and AWS Organizations service control policies (SCPs) help set maximum permissions on actions in your account. IAM identity-based and resource-based policies grant permissions to allow or deny actions in your account.

The following policy types, listed in order of frequency, are available for use in AWS. You learn about each of these policy types in more detail later in this module.

Policy types

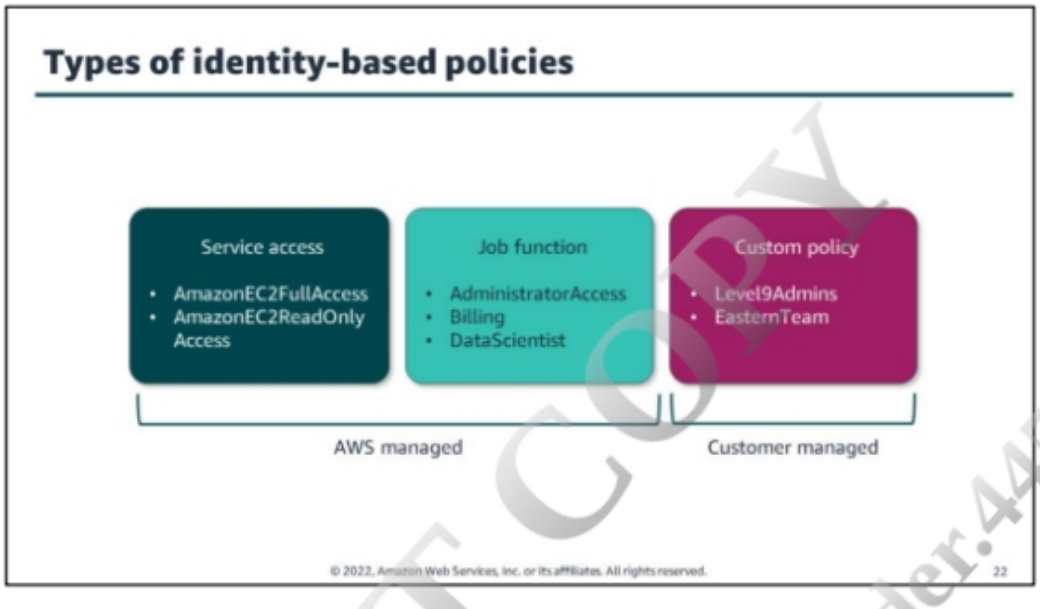
- **Identity-based policies** – Attach managed and inline policies to IAM identities. This includes users, groups to which users belong, and roles.
- **Resource-based policies** – Attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies.
- **AWS Organizations service control policies (SCPs)** – Use Organizations SCPs to define the maximum permissions for account members of an organization or organizational unit (OU).
- **IAM permissions boundaries** - AWS supports *permissions boundaries* for IAM entities (users or roles). Use IAM permissions boundaries to set the maximum permissions that an IAM entity can receive.



Identity-based policies are JSON permissions policy documents that control:

- What actions an IAM identity (users, groups of users, and roles) can perform
- On which resources
- Under what conditions

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. These policies grant the principal permission to perform specific actions on that resource and define under what conditions this applies. Resource-based policies are inline policies. There are no managed resource-based policies.



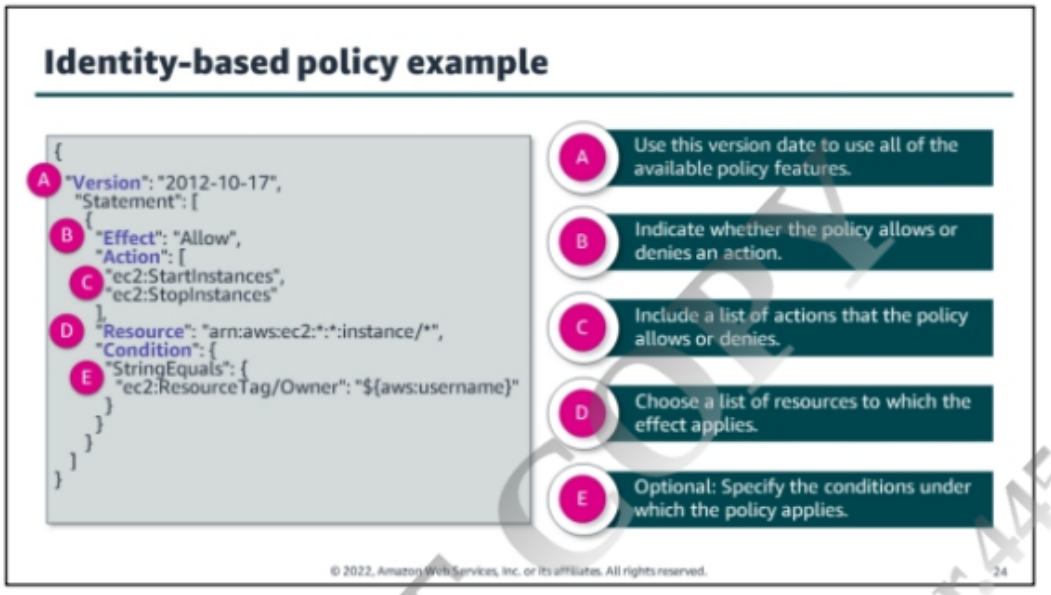
You can choose to use existing AWS policies. Some are managed by AWS. You also have the option to create your own policies.

Identity-based policies can be categorized by the following types:

- **Managed policies** – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. There are two types of managed policies:
  - **AWS managed policies** – Managed policies that are created and managed by AWS. They are built to provide specific service access or permissions for job functions.
  - **Customer managed policies** – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies.
- **Inline policies** – Policies that you add directly to a single user, group, or role. Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity.

Regarding inline or customer managed policies:
An *inline policy* is one that you create and embed directly to an IAM group, user, or role. Inline policies can't be reused on other identities or managed outside of the identity where they exist. As a best practice, use customer managed policies instead of inline policies.

For more information, see "Use customer managed policies instead of inline policies" (https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#best-practice-managed-vs-inline).



A JSON identity-based policy document includes these elements:

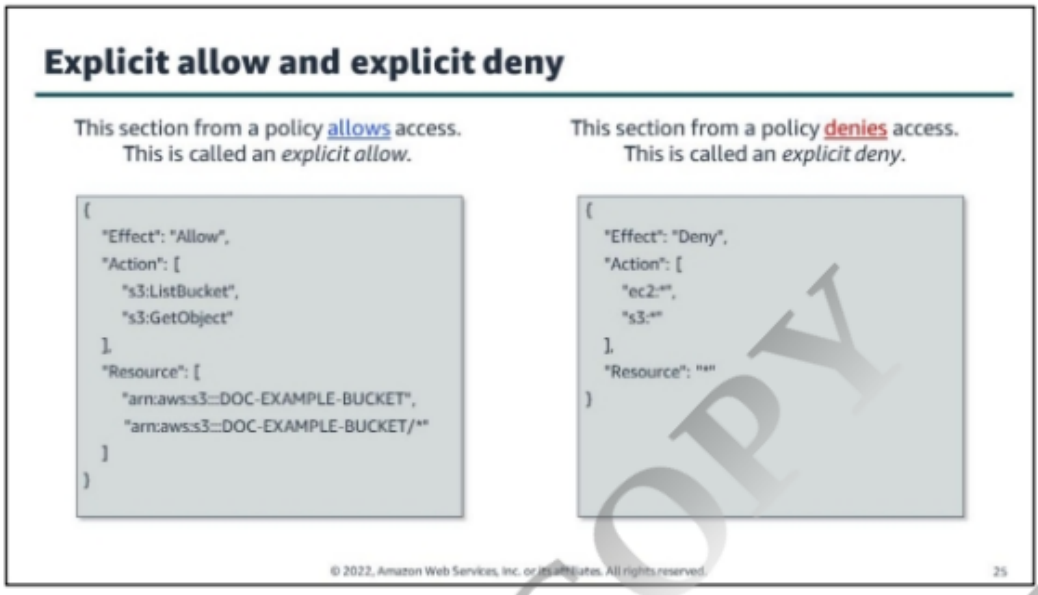
- **Version** – The Version policy element specifies the language syntax rules that are to be used to process a policy. To use all of the available policy features, include the value "2012-10-17" for the version in your policies.
- **Effect** – Use Allow or Deny to indicate whether the policy allows or denies access.
- **Action** (or NotAction) – Include a list of actions that the policy allows or denies.
- **Resource** (or NotResource) – You must specify a list of resources to which the actions apply.
- **Condition** (or NotCondition) – Specify the circumstances under which the policy grants permission.

The NotAction, NotResource, and NotCondition policy elements are not mentioned in this course.

When you attach the example policy statement to your IAM user, for example, that user is allowed to stop and start EC2 instances in your account as long as the condition is met. Here, the EC2 instances your IAM user can control must have a tag with key "Owner" and value equal to the IAM user name.

In the Resource element, the policy lists an AWS Resource Name (ARN) with a wildcard (star) character. Wildcards are used to apply a policy element to more than one resource or action. This policy applies for resources in any account number and Region with any resource ID. It can be reused in multiple accounts without having to rewrite the policy with your AWS account ID.

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

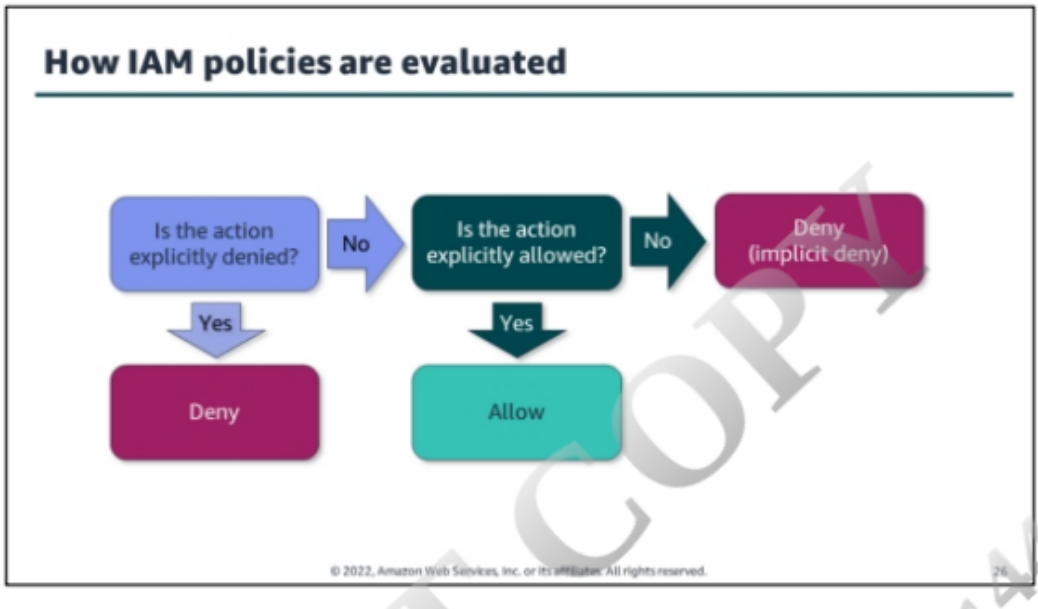


An IAM policy is made up of explicit allow statements, explicit deny statements, or both.

An *explicit allow*, shown in the first policy, authorizes your IAM user, group, or role to take the listed actions against a set of your resources. The policy allows list and get actions on all objects in an S3 bucket called DOC-EXAMPLE-BUCKET. When you use a wildcard character after the bucket name and slash, it covers all objects in that bucket.

An *explicit deny*, shown in the second policy, stops your IAM user, group, or role when trying to take an action listed for a set of your resources. In the second policy example, all actions in Amazon EC2 or Amazon S3 on any resource are denied.

Use allow and deny in your statement to guide what actions your principals can take in your account.

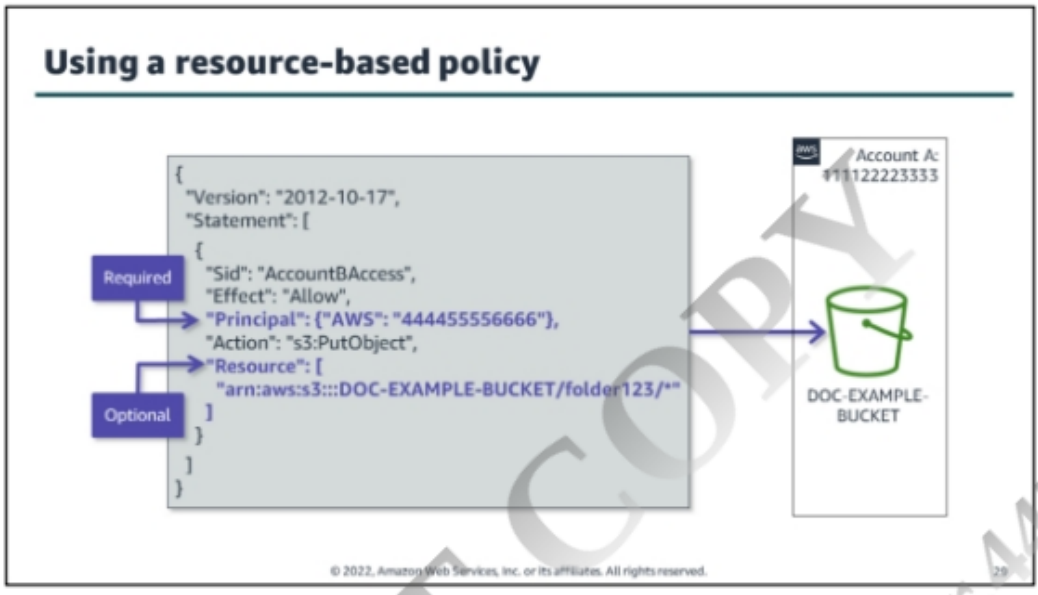


It is important to know the AWS evaluation logic when building IAM policies for your account. This way you can give your users and applications only the access they need.

AWS evaluates all policies that are applicable to the request context. The following is a summary of the AWS evaluation logic for policies within a single account.

- By default, all requests are implicitly denied with the exception of the AWS account root user, which has full access. This is called an *implicit deny*.
- An explicit allow in an identity-based or resource-based policy overrides this default.
- An explicit deny in any policy overrides any allows.

Explicit deny is useful as a safety measure because it overrides explicit allow.



Resource-based policies are attached to a single resource, like an S3 bucket or AWS Lambda function. You learn more about S3 buckets and Lambda functions later in this course.

With resource-based policies, you choose who has access to the resource and what actions they can perform on it.

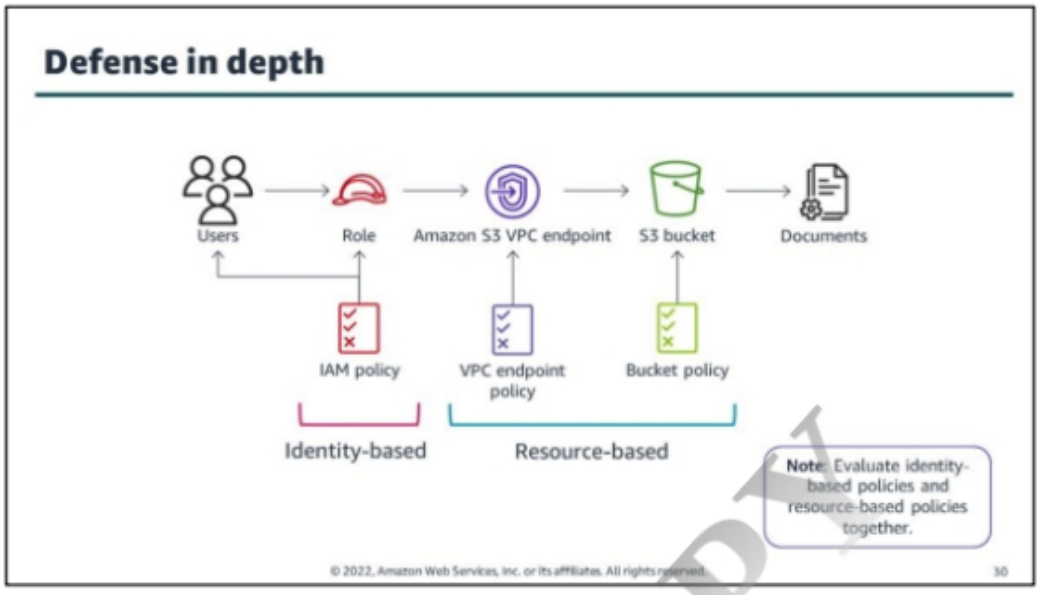
If you create a resource-based policy, you must list the **principal** account, user, role, or federated user to which you want to allow or deny access. If you are creating an IAM permissions policy to attach to a user or role, you cannot include this element. The principal is implied as that user or role.

In your resource-based policies, the **resource** element is optional. If you do not include this element, the resource to which the action applies is the resource to which the policy is attached.

In the example, the principal is an AWS account ID. The set of resources is all objects in the bucket DOC-EXAMPLE-BUCKET that are within the folder called folder123. The bucket policy allows a specific AWS account to upload objects to your bucket's folder.

AWS identity-based policies and resource-based policies are evaluated together. Recall how IAM policies are evaluated. If any explicit deny statement is found in any IAM policy, the action is denied. If at least one allow statement exists with no explicit deny, the action is allowed.

For more information about identity-based policies, see "Identity-based policies and resource-based policies" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_identity-vs-resource.html).

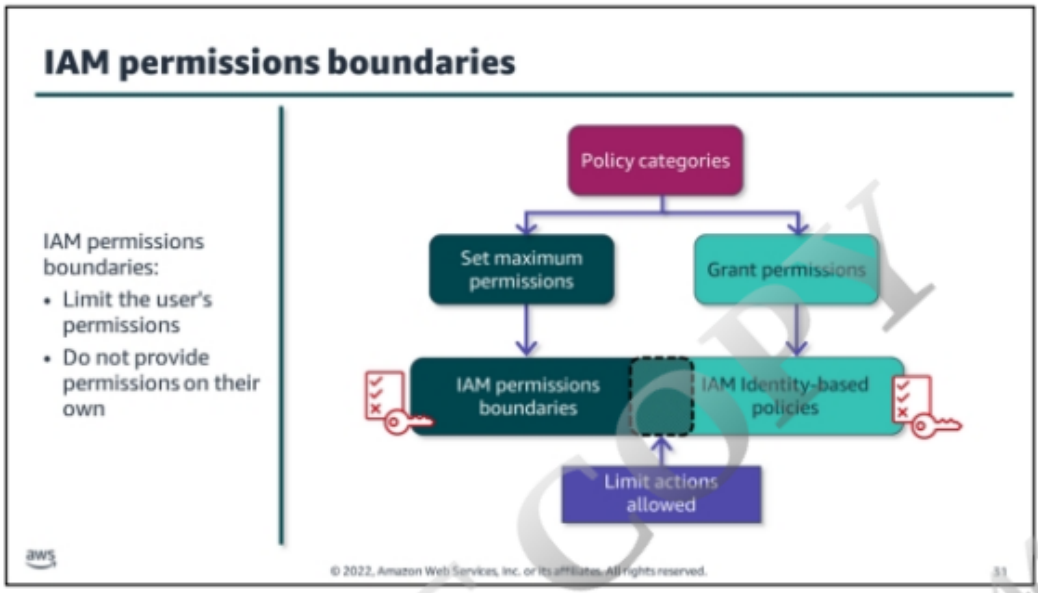


Defense in depth is a strategy focused on creating multiple layers of security.

Apply a defense-in-depth approach with multiple security controls to all layers. For example, apply it to the edge of the network, virtual private cloud (VPC), load balancing, and every instance, compute service, operating system, application, and code. Application security is as critical as instance security.

In the diagram, different users try to access a document in your S3 bucket. Each user needs an identity-based policy assigned to either their user or a role they assume to access AWS. They then navigate through layers of resource-based policies—first a VPC endpoint policy, then a bucket policy for the S3 bucket. Your users are able to access the documents they need for their task. You will learn more about VPC endpoints and S3 buckets later in this course.

For more information, see "Policy evaluation logic" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html).



AWS supports *permissions boundaries* for IAM entities—users or roles. 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.* Permissions boundaries act as a filter.

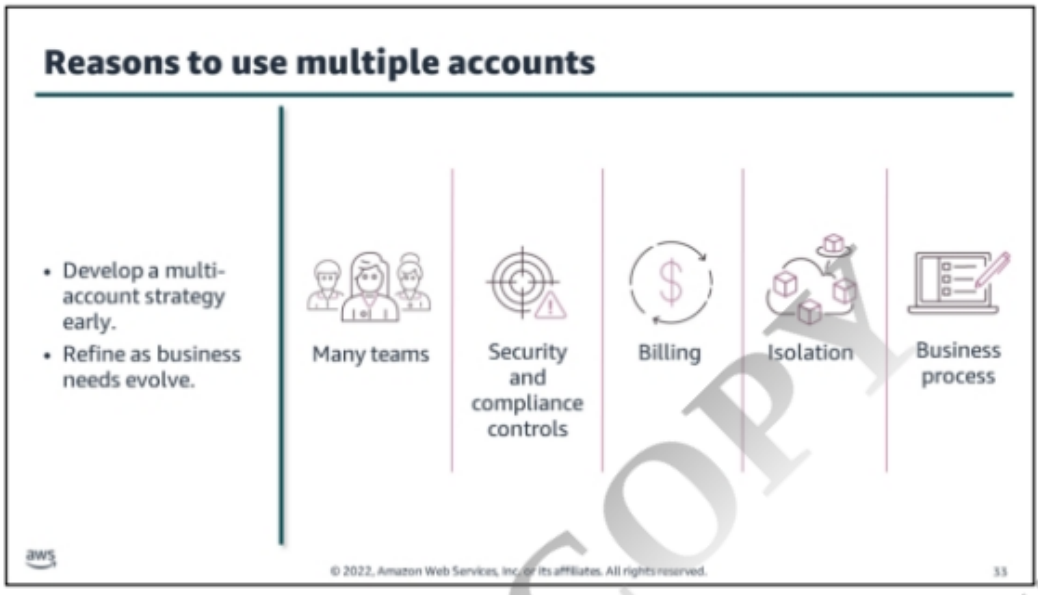
An entity's permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries.

For more information about permissions boundaries, see "Permissions boundaries for IAM entities" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html).

**For Accessibility: Diagram showing the two policy categories, set maximum permissions and grant permissions. Connected to set maximum permissions is IAM permission boundaries. Partially overlapping IAM permission boundaries and connected to grant permissions is IAM identity-based policies. The areas of overlap is labeled "limits actions allowed." End Description.

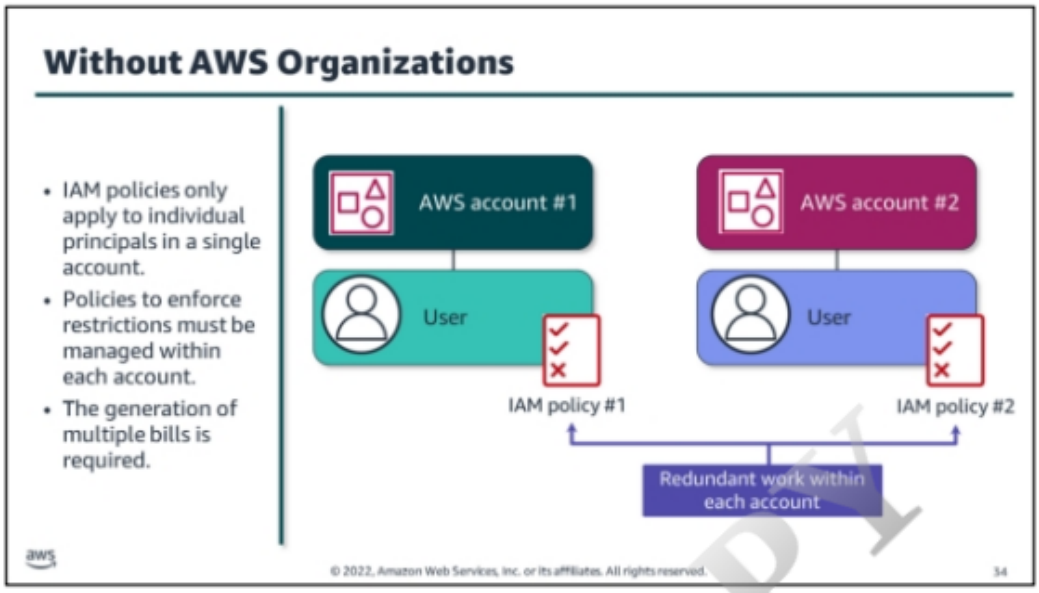
The security specialist asks, "What is the best way to manage multiple accounts?"

The company wants your advice about ways to manage more than one account.

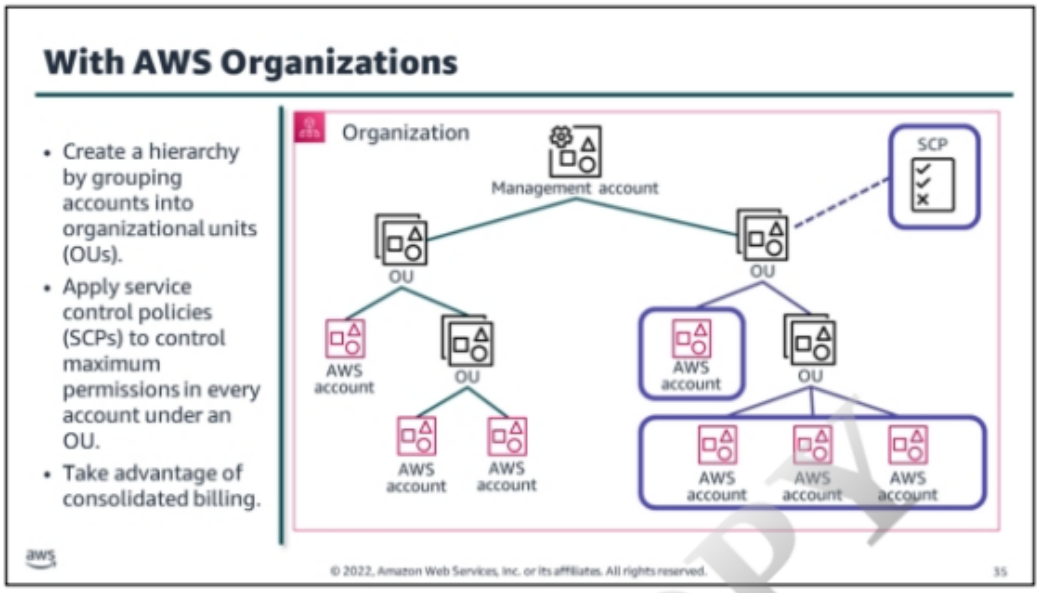


As you expand your use of AWS, you have several reasons that you might want to create a multi-account structure in your organization:

- To group resources for categorization and discovery
- To improve your security posture with a logical boundary
- To limit potential impact in case of unauthorized access
- To simplify management of user access to different environments



Managing multiple accounts is more challenging without AWS Organizations. For example, because IAM policies only apply to a specific AWS account, you must duplicate and manage IAM policies in each account to deploy standardized permissions across all accounts.

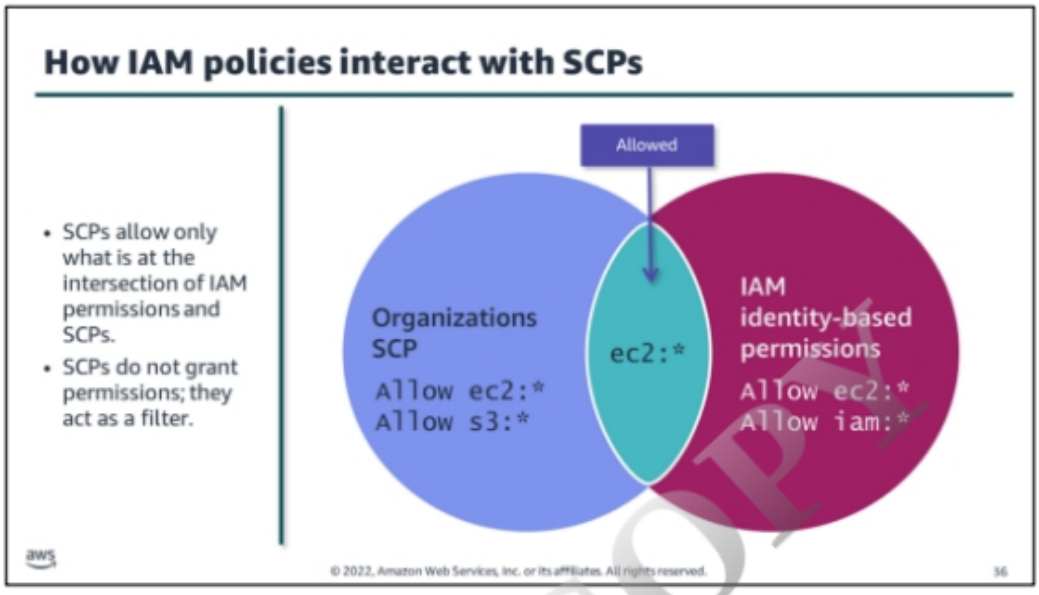


AWS Organizations provides these key features:

- Centralized management of all your AWS accounts
- Consolidated billing for all member accounts
- Hierarchical grouping of your accounts to meet your budgetary, security, or compliance needs
- Policies to centralize control over the AWS services and API actions that each account can access
- Policies to standardize tags across the resources in your organization's accounts
- Policies to control how AWS artificial intelligence (AI) and machine learning (ML) services can collect and store data
- Policies that configure automatic backups for the resources in your organization's accounts
- Integration and support for IAM
- Integration with other AWS services
- Global access
- Data replication that is eventually consistent
- No cost for use

For more information about inheritance for SCPs, see "Inheritance for service control policies" in the *AWS Organizations User Guide* (https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_inheritance_auth.html).

**For accessibility: An AWS organization containing a management account which has two OUs. Each of these OUs has one child AWS account and child OU. Each of these child OUs has multiple child AWS accounts. A policy is applied to a top OU and is active on all child AWS accounts and child OUs. End description.



An SCP is a type of organization policy that you can use to manage permissions in your organization. SCPs:

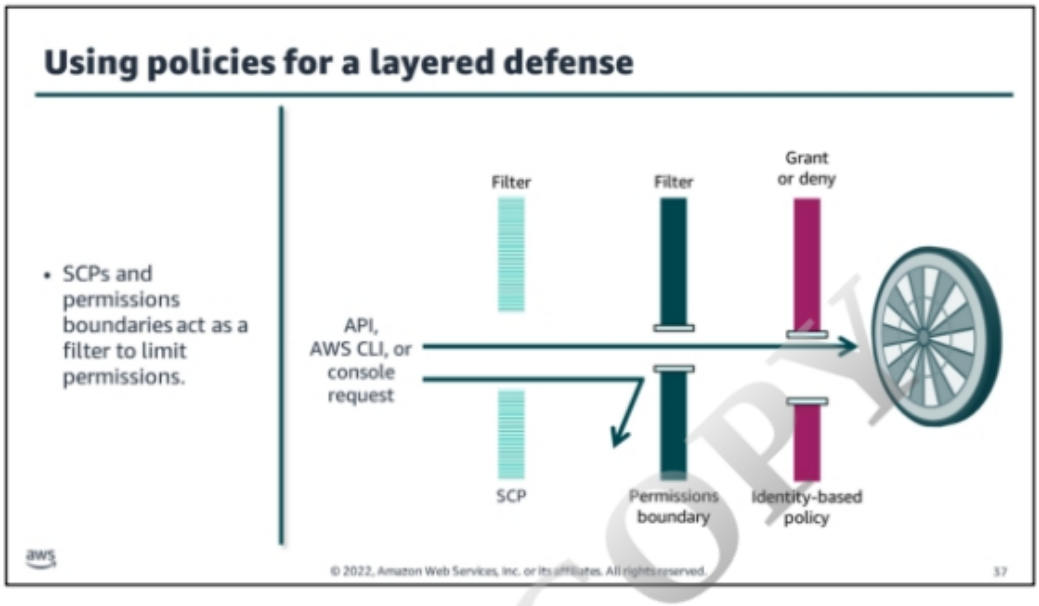
- Offer central control over the maximum available permissions for all accounts in your organization
- Help your accounts stay within your organization's access control guidelines
- Are available only in an organization that has all features turned on

SCPs aren't available if your organization turns on only the consolidated billing features.

Attaching an SCP to an Organizations entity (root, OU, or account) defines a guardrail. SCPs set limits upon the actions that the IAM users and roles in the affected accounts can perform. To grant permissions, you need to attach identity-based or resource-based policies to IAM users, or to the resources in your organization's accounts. When an IAM user or role belongs to an account that is a member of an organization, the SCPs limit the user's or role's effective permissions.

In the example, an SCP allows all EC2 and S3 actions. A collection of IAM identity-based permissions allow all EC2 and IAM actions. The effective allowed permissions for the IAM identity are all EC2 actions. It excludes both S3 and IAM actions because they are not explicitly allowed in both policy types.

For more information about SCPs, see "How to Use Service Control Policies in AWS Organizations" in the *AWS Security Blog* (https://aws.amazon.com/blogs/security/how-to-use-service-control-policies-in-aws-organizations/).



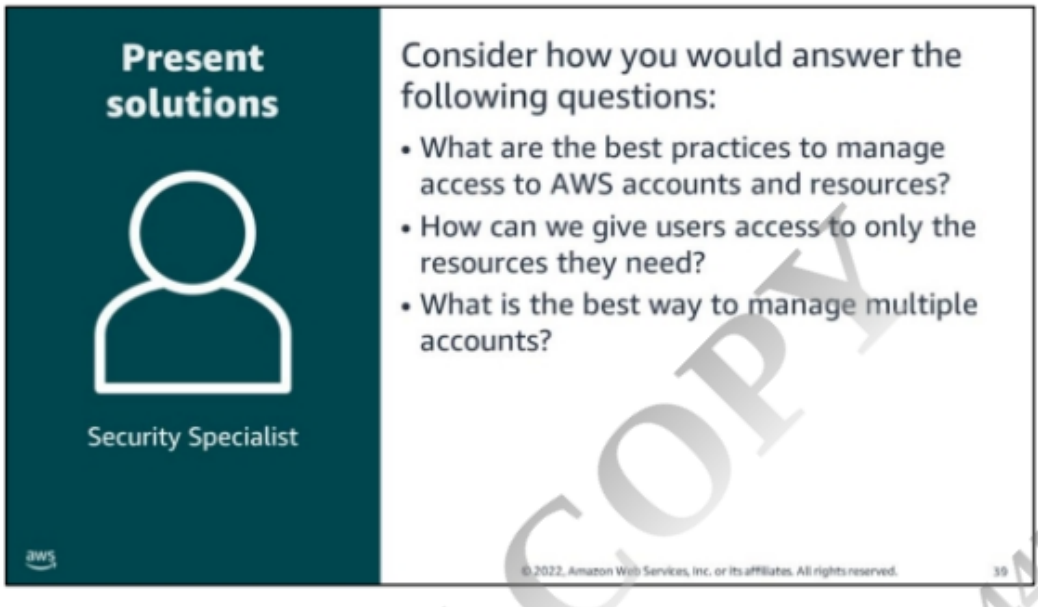
When a principal tries to use the console, the AWS API, or the AWS CLI, that principal sends a request to AWS. With AWS you can configure several resources to determine whether to grant or deny the request.

In this example you observe the following layers of defense:

- First, the action must be allowed by any SCPs configured for the organization.
- Next, the action must be included within any applied permissions boundaries.
- Finally, the identity-based policy must allow and not explicitly deny the action.

In an IAM entity (user or role), a permissions boundary allows it to perform only the actions that are allowed by both its identity-based policies and its permissions boundaries. This adds an additional layer to protect against the creation of an IAM identity-based policy that allows overly permissive actions for that entity.

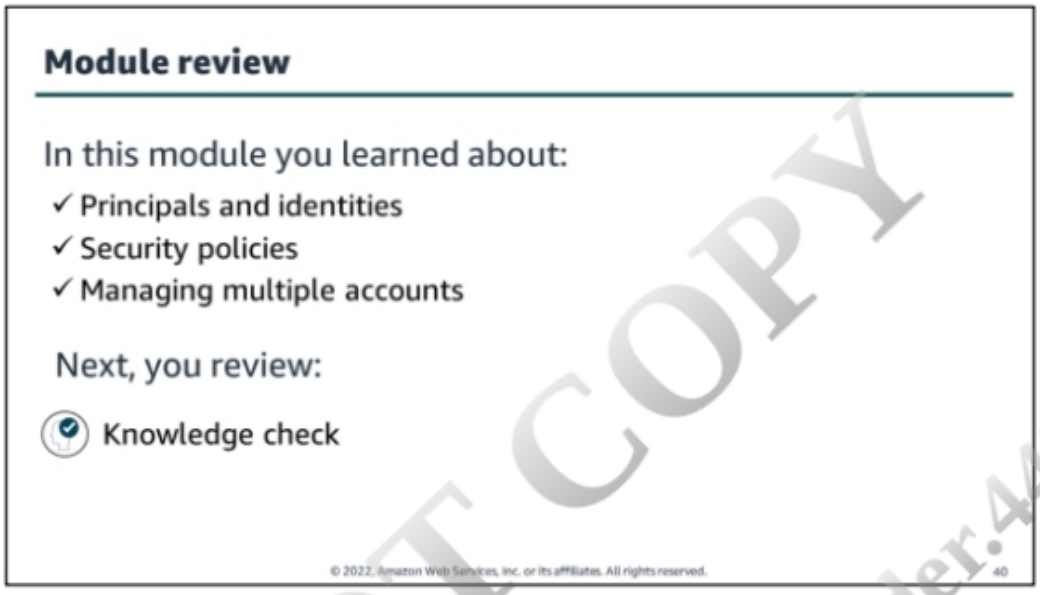
For more information, see "Policy evaluation logic" (https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html).

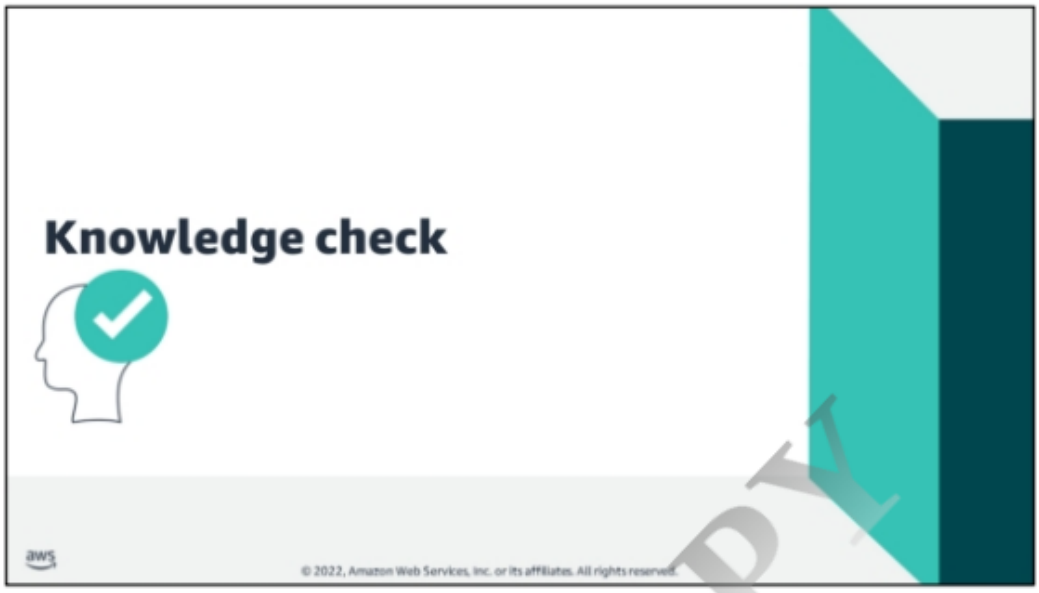


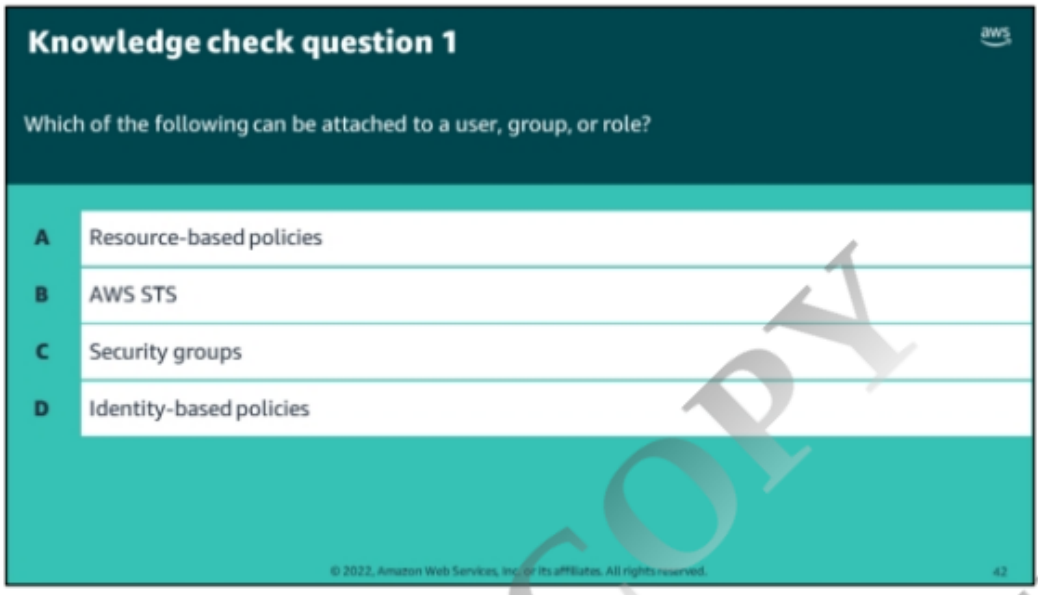
Imagine you are now ready to talk to the security specialist and present solutions that meet their architectural needs.

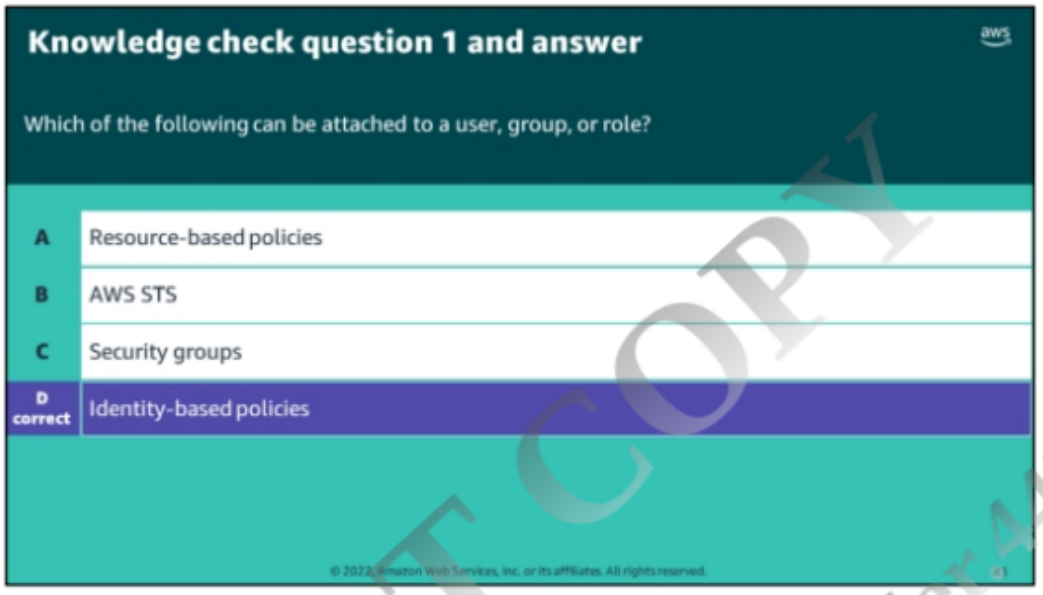
Think about how you would answer the questions from the beginning of the lesson about account security.

Your answers should include the following solutions:
- Create IAM users, user groups, and roles to manage access to AWS accounts and resources.
- Build security policies with allow and deny statements. Use permissions boundaries as a protective layer.
- Use SCPs in AWS Organizations to manage multiple accounts.



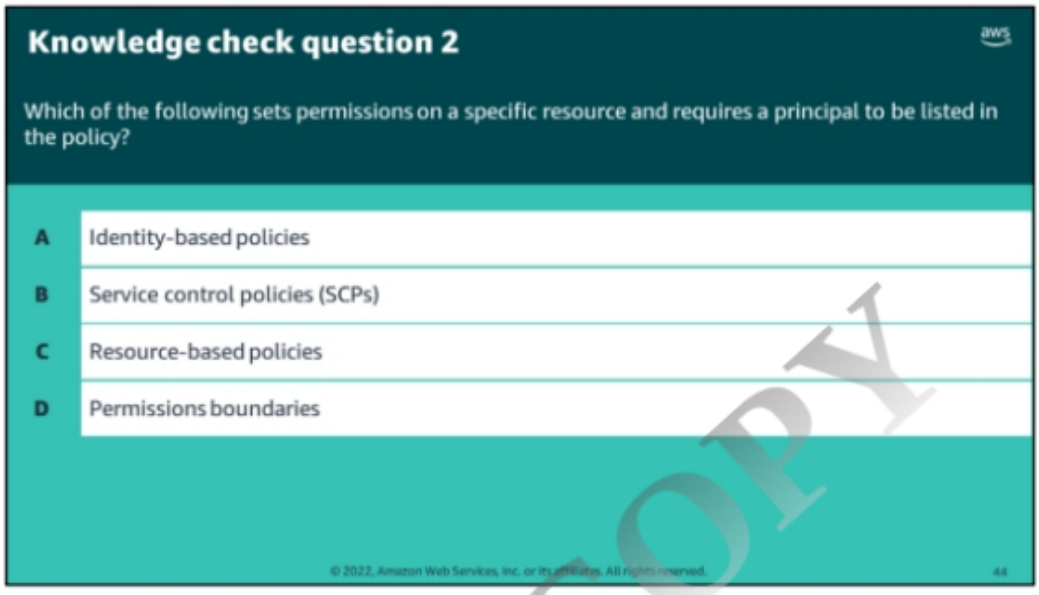


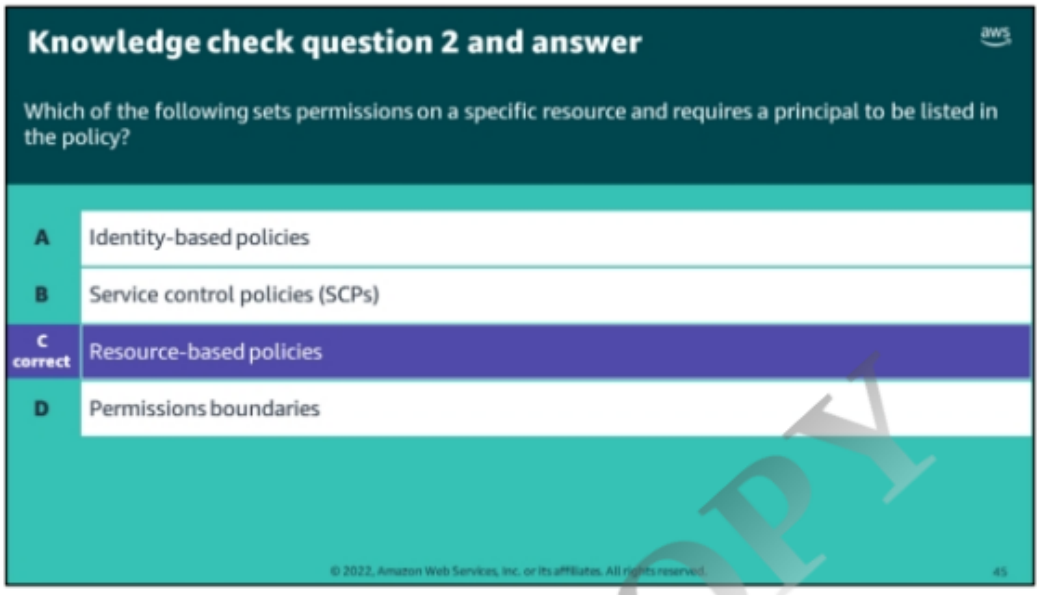




The correct answer is D, identity-based policies.

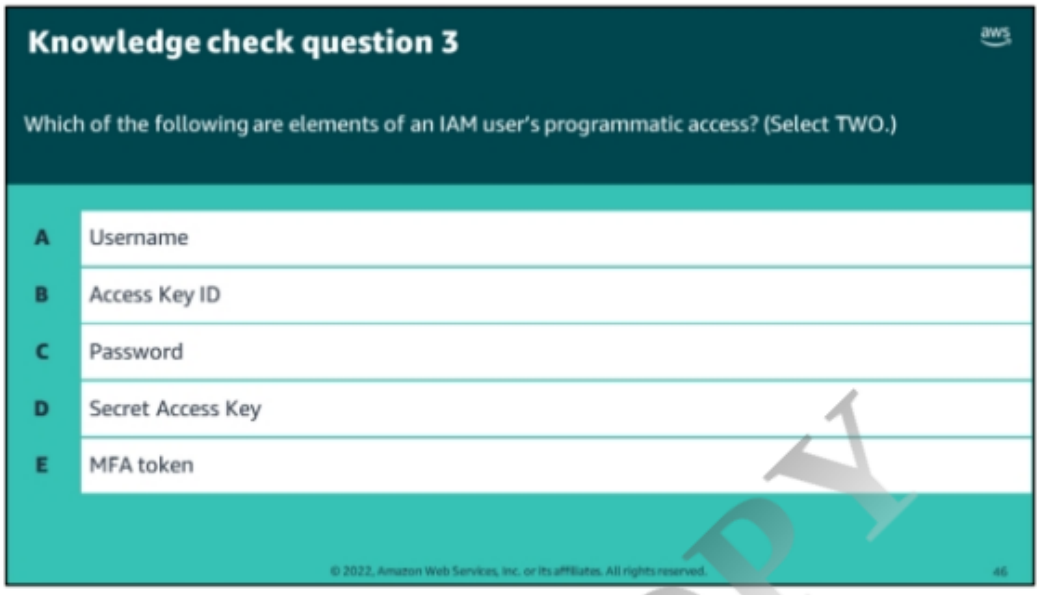
Attach managed and inline policies to IAM identities. These identities include users, user groups, and roles.

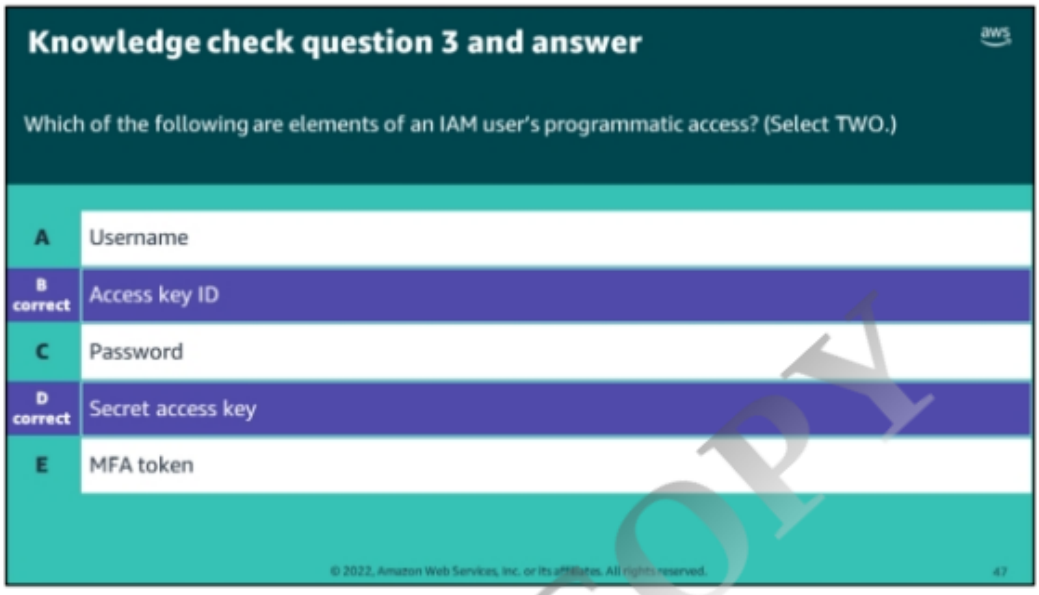




The correct answer is C, resource-based policies.

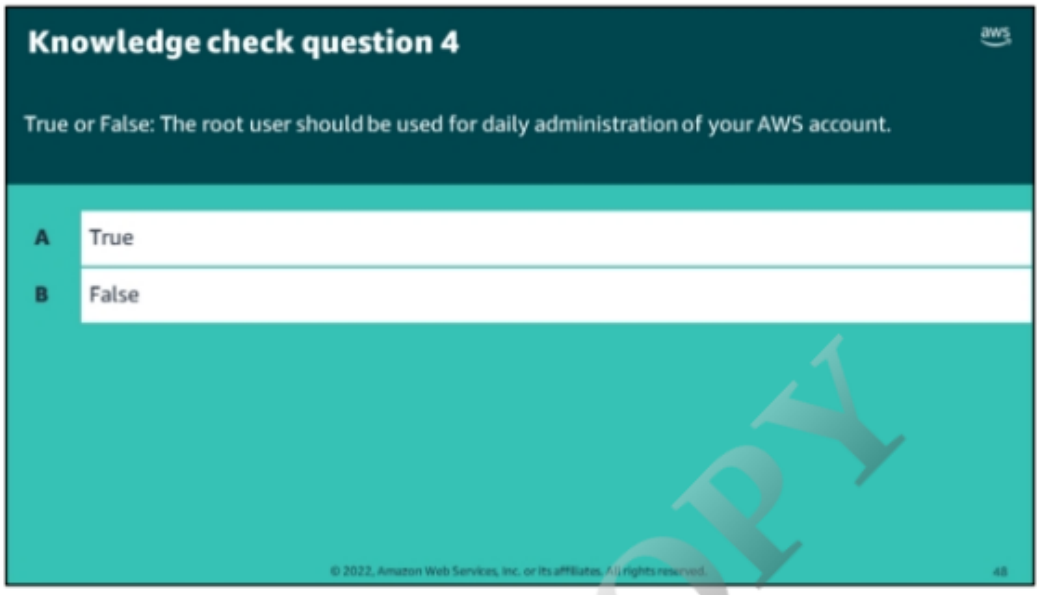
Resource-based policies attach inline policies to resources. The most common examples of resource-based policies are Amazon S3 bucket policies and IAM role trust policies.

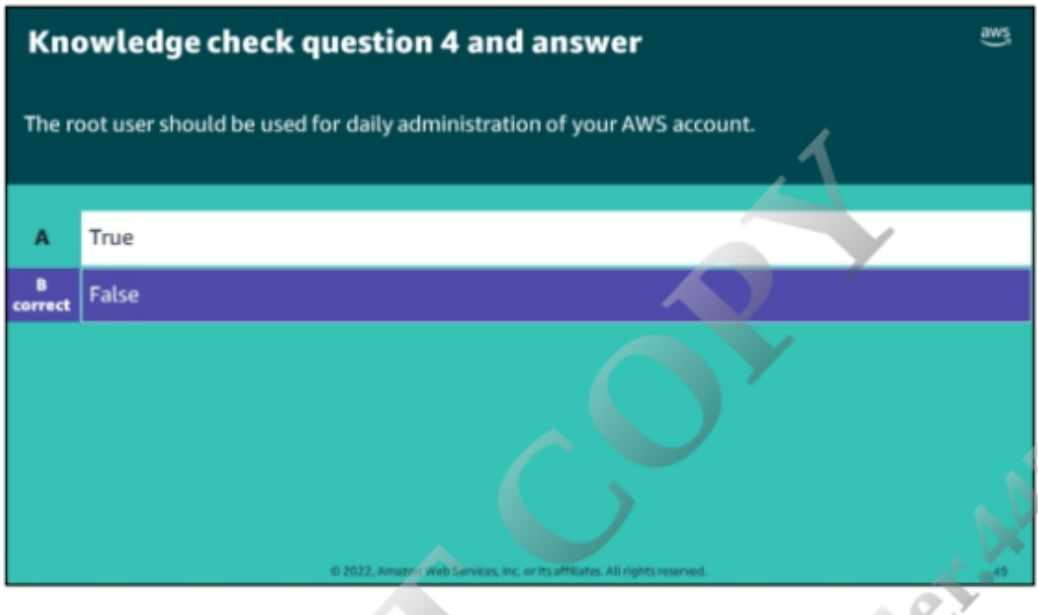




The correct answer is B and D, access key ID and secret access key.

An IAM user's programmatic access does not use a username and password authentication. Instead, they use a unique access key ID and secret access key. MFA tokens cannot be used with long-term credentials (IAM user access keys and root user access keys).

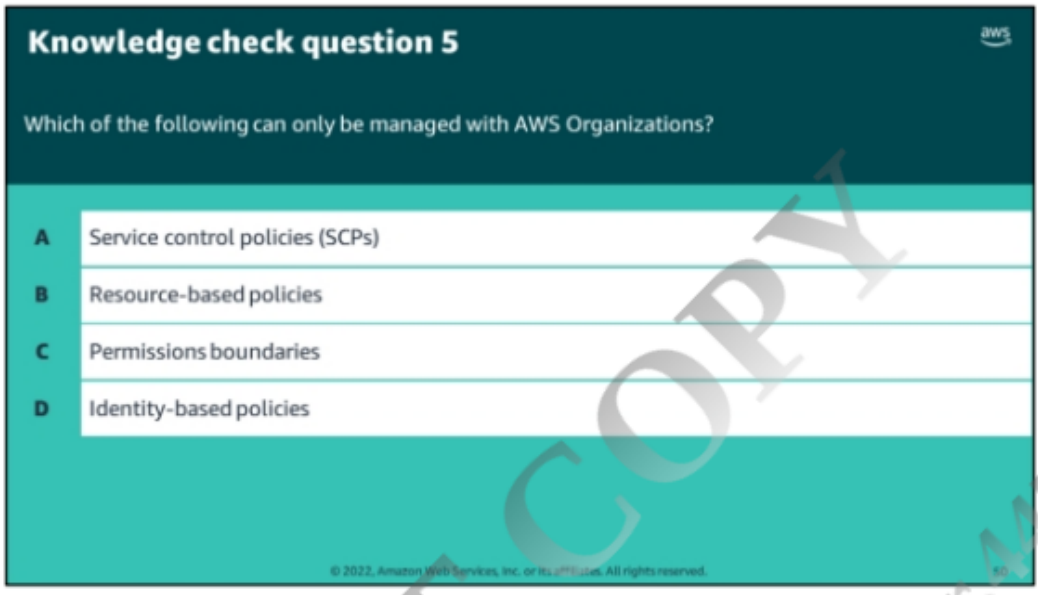


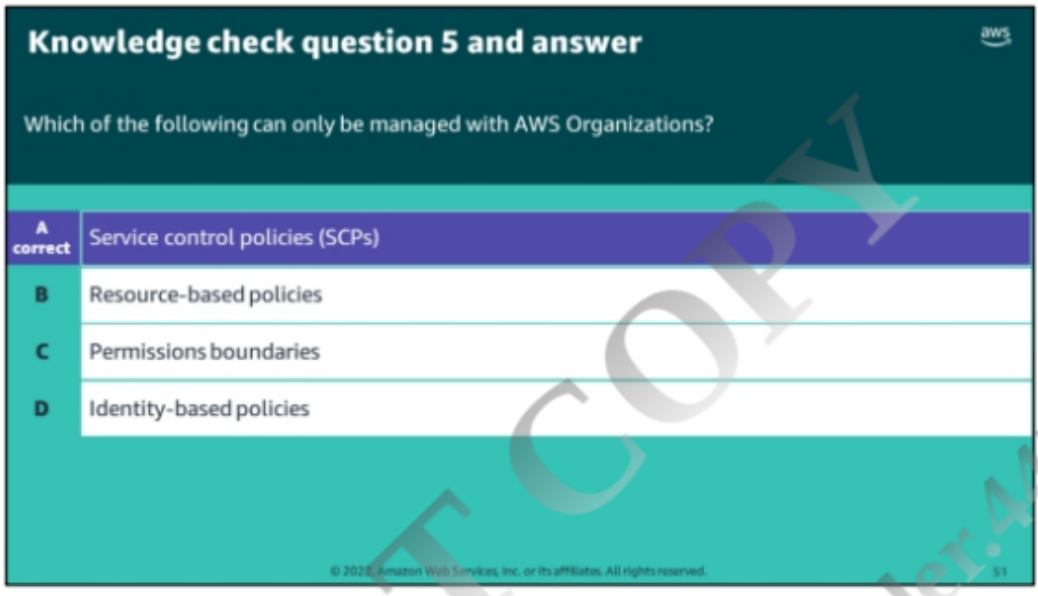


The correct answer is B, false.

You do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then, securely lock away the root user credentials and use them to perform only a few account and service management tasks.

For more information about the root user, see "AWS account root user" in the *AWS Identity and Access Management User Guide* (https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html).





The correct answer is A, service control policies (SCPs).

SCPs offer central control over the maximum available permissions for all accounts in your organization. Your SCPs are managed through AWS Organizations. You can attach an SCP to the organization root, to an organizational unit (OU), or directly to an account.

The administrator must still attach identity-based or resource-based policies to IAM users or roles, or to the resources in your accounts to actually grant permissions. The effective permissions are the logical intersection between what is allowed by the SCP and what is allowed by the identity-based and resource-based policies.