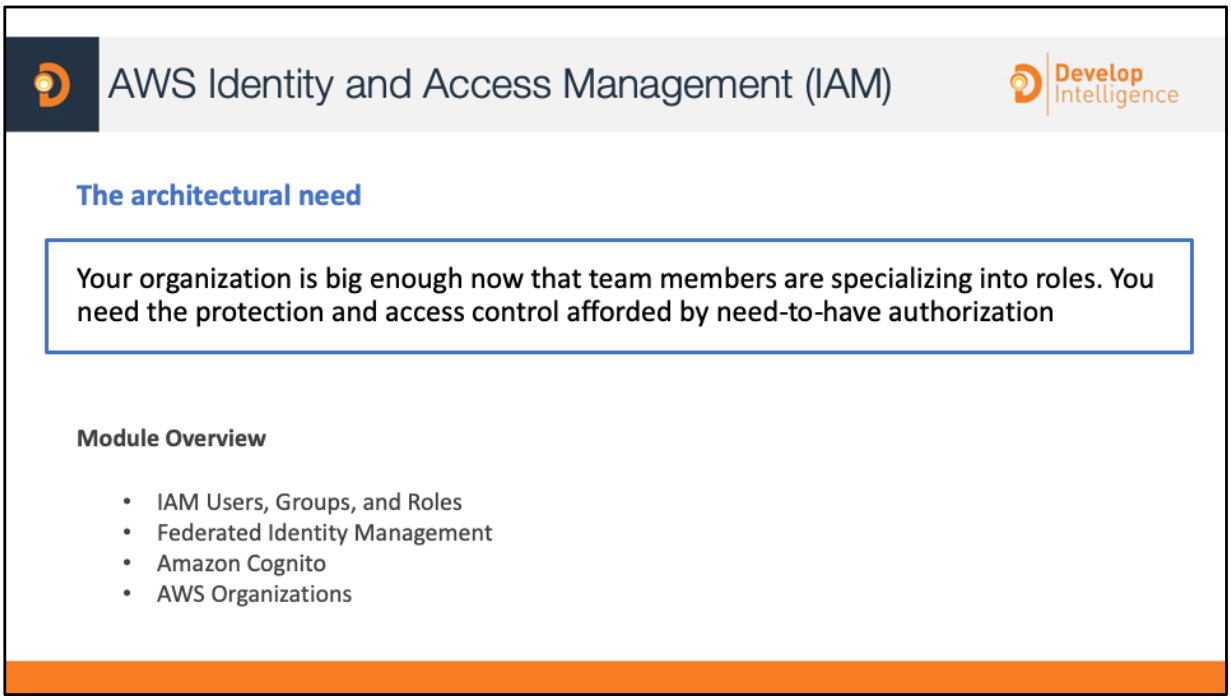
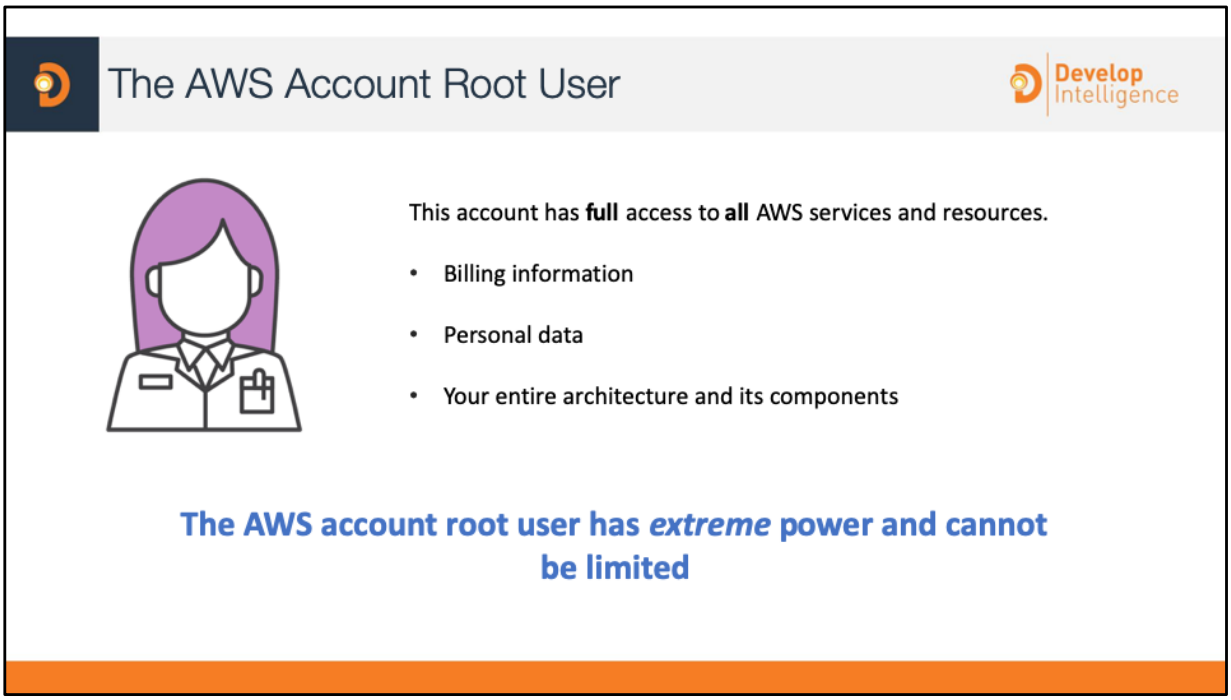
AWS Identity and Access Management (IAM)

Develop Intelligence

- Module 7

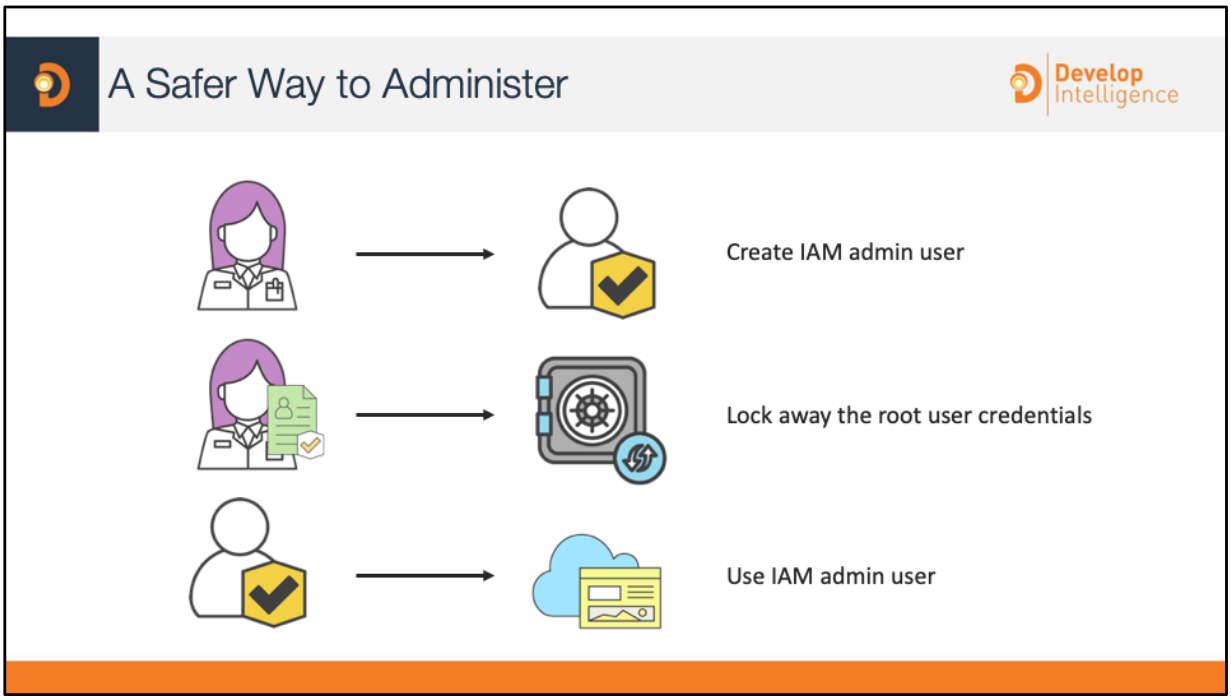


- Some users need access to the AWS Management Console, others only need to use AWS Command Line Interface (AWS CLI).
- Different environments (Dev/Test/Production) have different sets of access requirements.
- On-premises authentication is managed with federation with SSO.
- The security operations team needs to have visibility into who's touching the data in the AWS Cloud.
- External auditors need access to the logs but nothing else.
- The mobile app needs to authenticate thousands of users.



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. This identity is called the *AWS account root user*, and is accessed by signing in with the email address and password provided when you created the account.

For more information about the AWS account root user, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html



The AWS account root user has full access to all resources in the account, and you cannot control the privileges of the root account credentials.

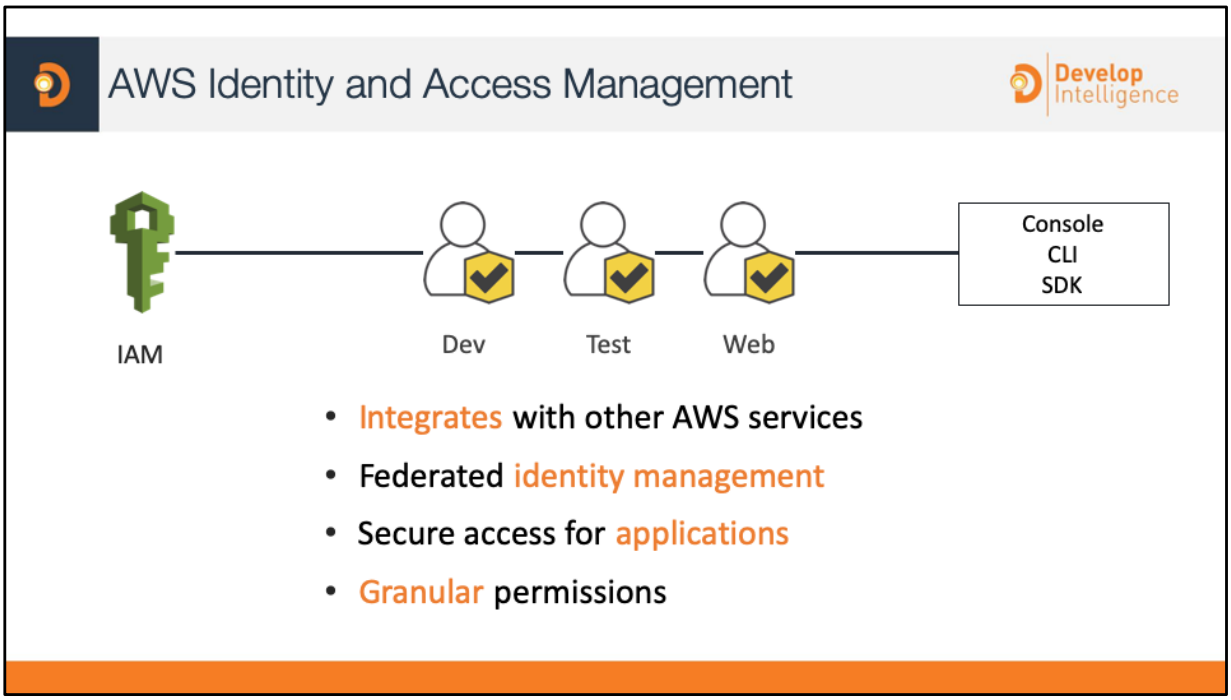
AWS strongly recommends that you not use root account credentials for day-to-day interactions with AWS. IAM users can be managed and audited with relative ease. If an IAM account principal (discussed later) needs more privileges, they can be added. Likewise, if privileges need to be removed or revoked, it can be done with minimal impact on an environment.

Use IAM to create additional users and assign permissions to these users, following the least privilege principle.

Now that you have secured the most important account in your AWS profile, and have an administrator account, create additional accounts for ease of access and security reasons. Least privilege should be the norm.

While it is possible to create accounts that have privileges similar to those of the AWS account root user, it is better to create admin accounts that control only the functionality needed. Do you need your DBA to be able to provision EC2 instances? If the answer is no, then provision accounts accordingly.

It can be helpful to have various accounts and account types. Permissions can be added and removed as needed.

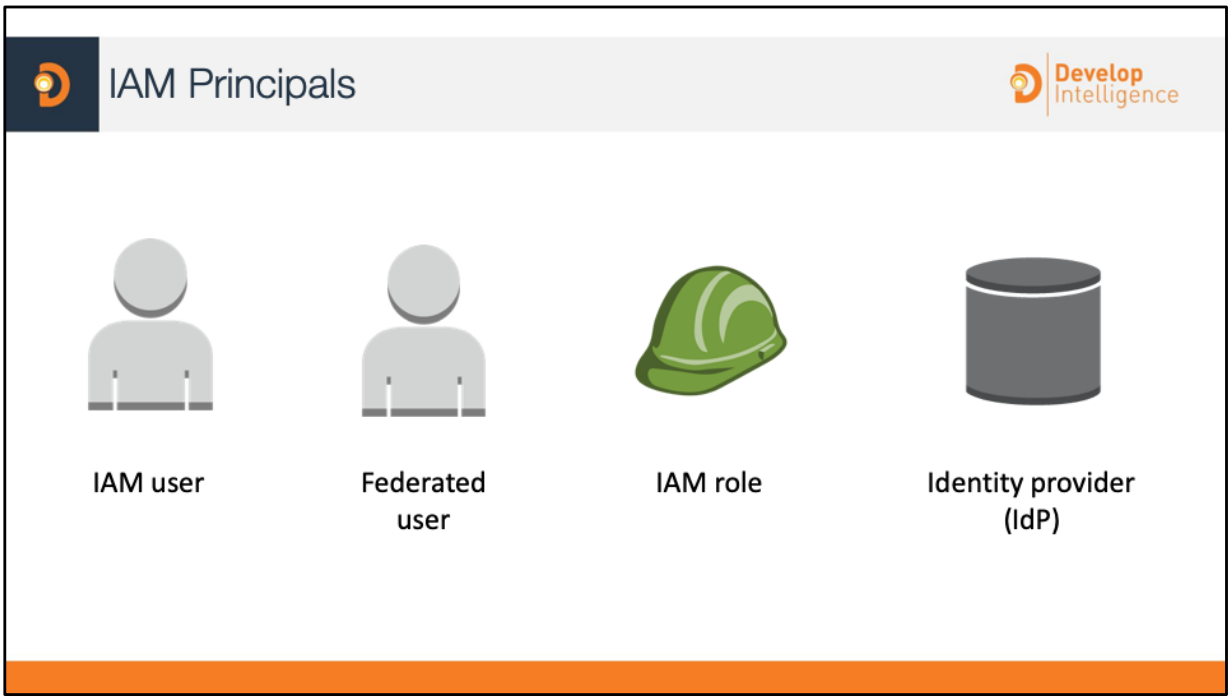


You can create users in the AWS identity management system, assign users individual security credentials (such as access keys, passwords, multi-factor authentication (MFA) devices), or request temporary security credentials to provide users access to AWS services and resources. You can specify permissions to control which operations a user can perform.

For federated identity management, you request security credentials with configurable expirations for users who managed by corporate directory. This provides secure access to employees and applications, so that they can access resources in an AWS account without creating an IAM user account for them. You specify the permissions for these security credentials to control which operations a user can perform.

IAM is service that helps you control access to your AWS resources. IAM allows you to create user accounts and credentials with varying degrees of account permissions and authorization.

IAM can be accessed from the console, the AWS CLI, the AWS SDK, and a secure API endpoint.

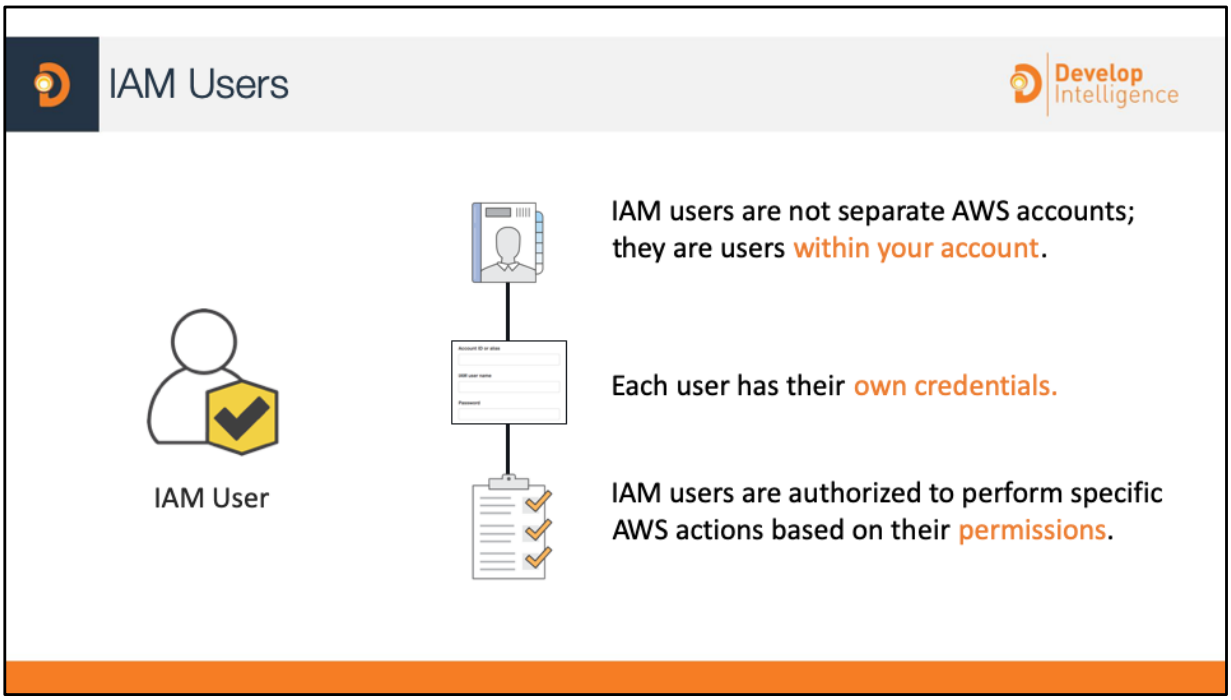


A *principal* is an entity that can take an action on an AWS resource. Over time, you can allow users and services to assume a role. You can support federated users or programmatic access to allow an application to access your AWS account. Users, roles, federated users, and applications are all AWS principals.

The principal can be an AWS IAM user, an AWS service such as Amazon EC2, a SAML provider, or an identity provider (IdP).

An IdP can be used instead of creating IAM users in your AWS account. With an IdP, you manage your user identities outside of AWS (such as Login with Amazon, Google, and Facebook) and give these external user identities permissions to use AWS resources in your account.

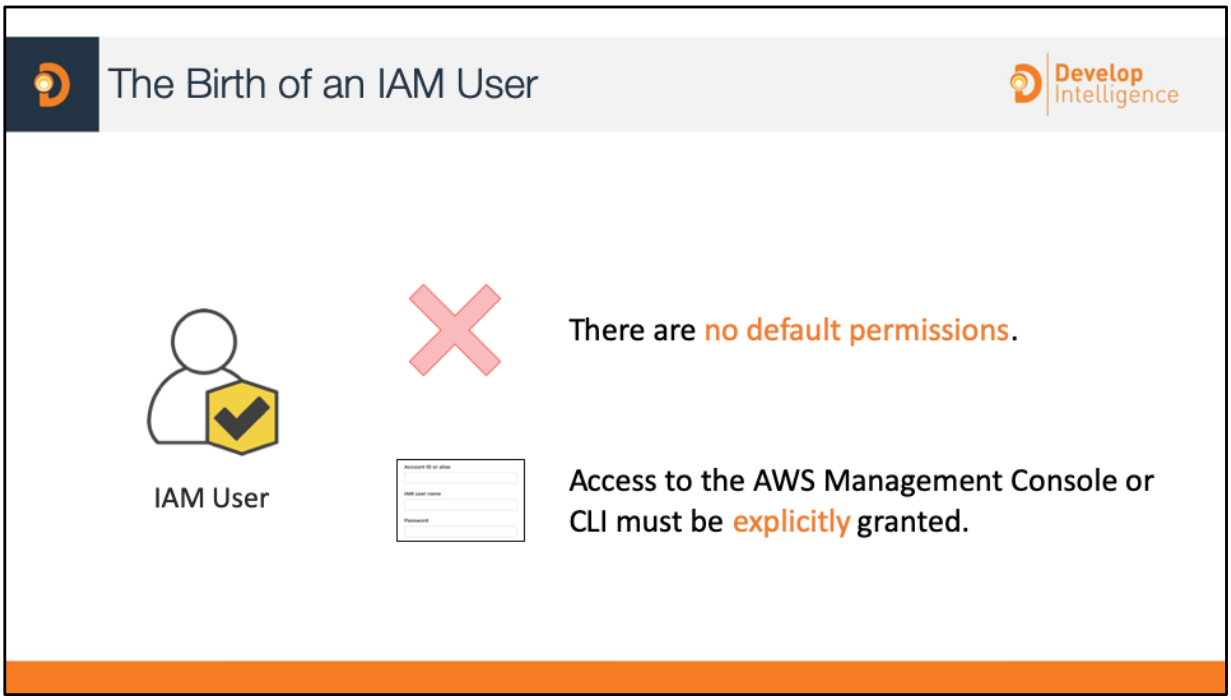
Image Sources:
Login with Amazon - https://login.amazon.com/button-guide
Continue with Facebook - https://developers.facebook.com/docs/facebook-login/web/login-button
Sign in with Google - https://developers.google.com/identity/sign-in/web/build-button



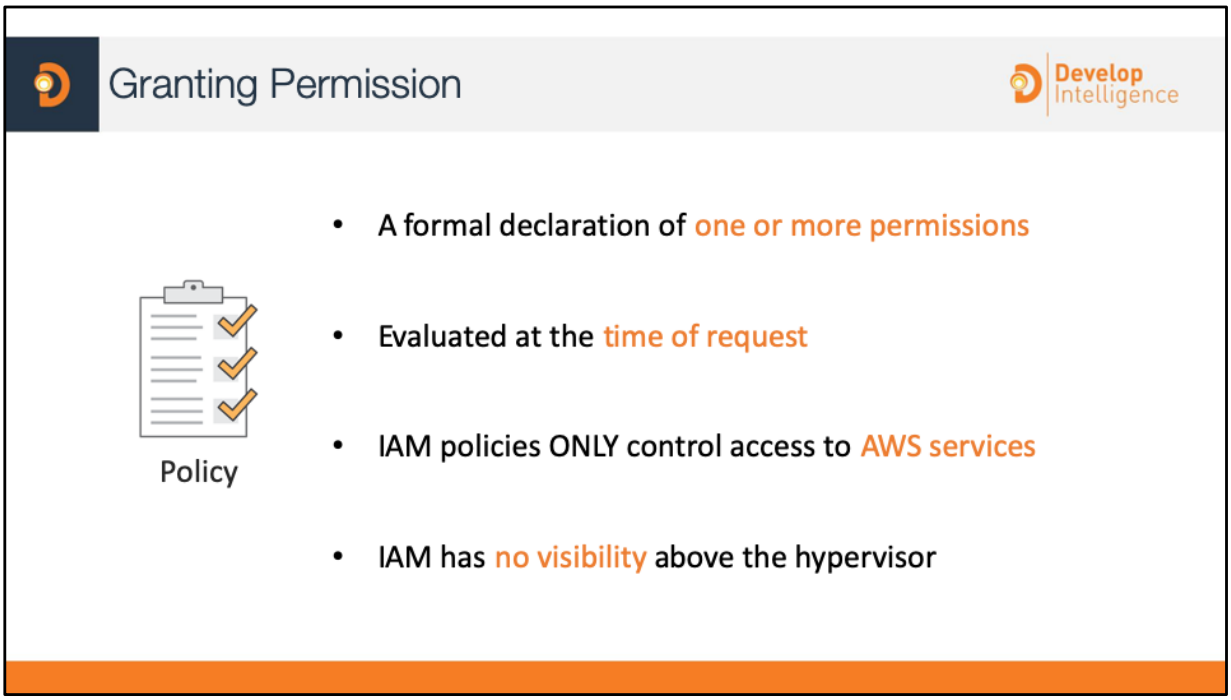
IAM users are not separate accounts; they are principals within your account. Each IAM user can have its own password for access to the console. You can also create an individual access key for each user so that the user can make programmatic requests to work with resources in your account. It is possible to have access to the console without having access to the CLI. The reverse is also true. It is possible to have access to the CLI without having access to the console.

Newly created IAM users have no default credentials to use to authenticate themselves and access AWS resources. You first need to assign security credentials to them for authentication and then attach permissions that authorize them to perform any AWS actions or to access any AWS resources. The credentials you create for users are what they use to uniquely identify themselves to AWS.

IAM IdPs can be used instead of IAM users in your AWS account. With an IdP, you can manage your user identities outside of AWS (such as Amazon.com, Google, and Facebook) and give these external user identities permissions to use AWS resources in your account.



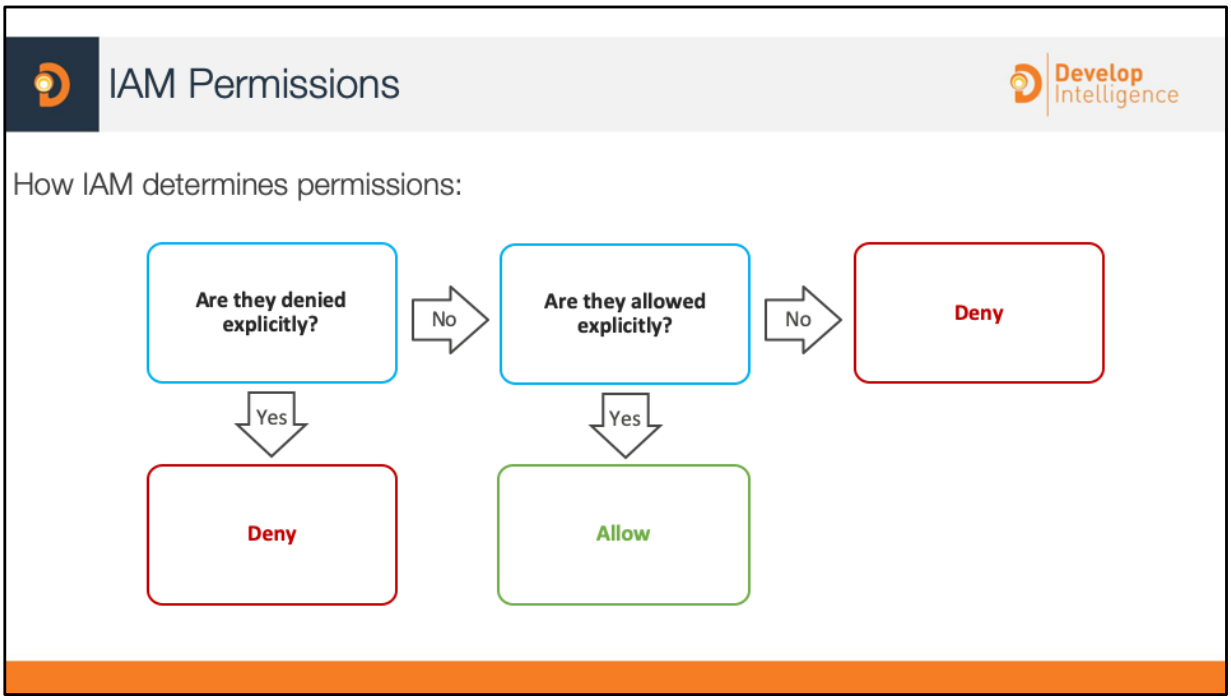
There are no default permissions for IAM principals. AWS does not recommend giving everyone admin rights. We urge you to follow the least-privilege principle.



A *policy* is an entity in AWS that, when attached to an identity or resource, defines their permissions. AWS evaluates these policies when a principal, such as a user, makes a request.

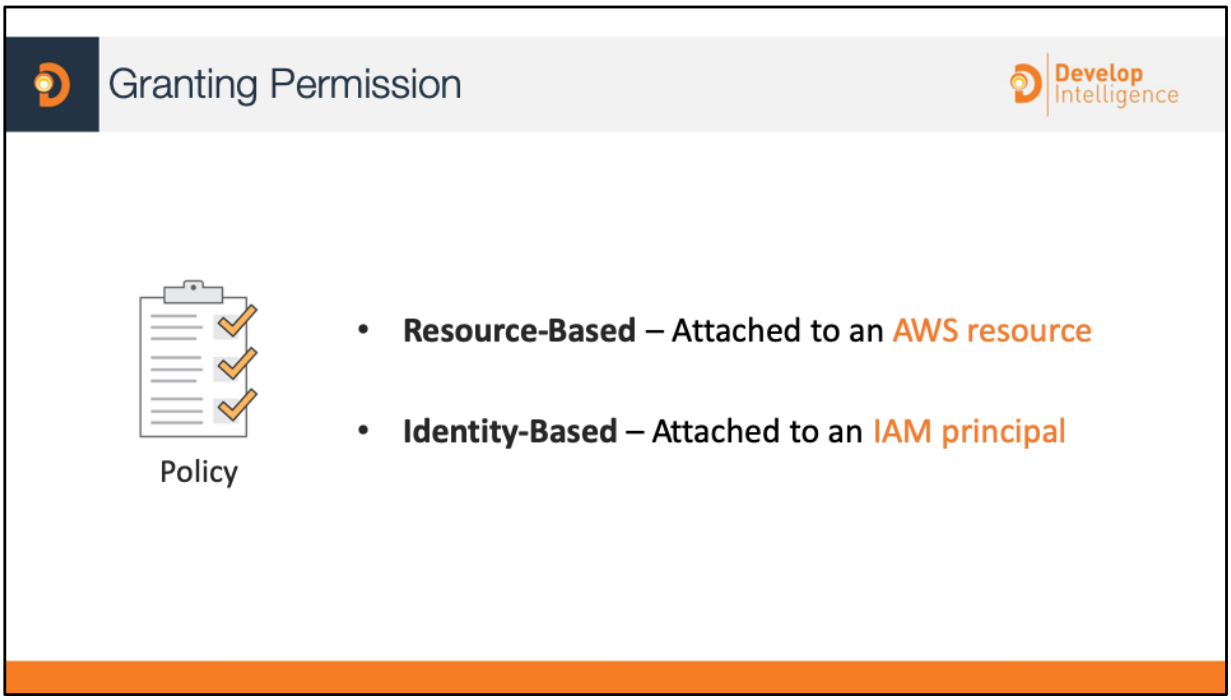
IAM policies only control access to AWS services. IAM has no visibility above the hypervisor layer and it cannot reach outside of AWS.

If you want OS support, use LDAP, SAML, or other identity management system.

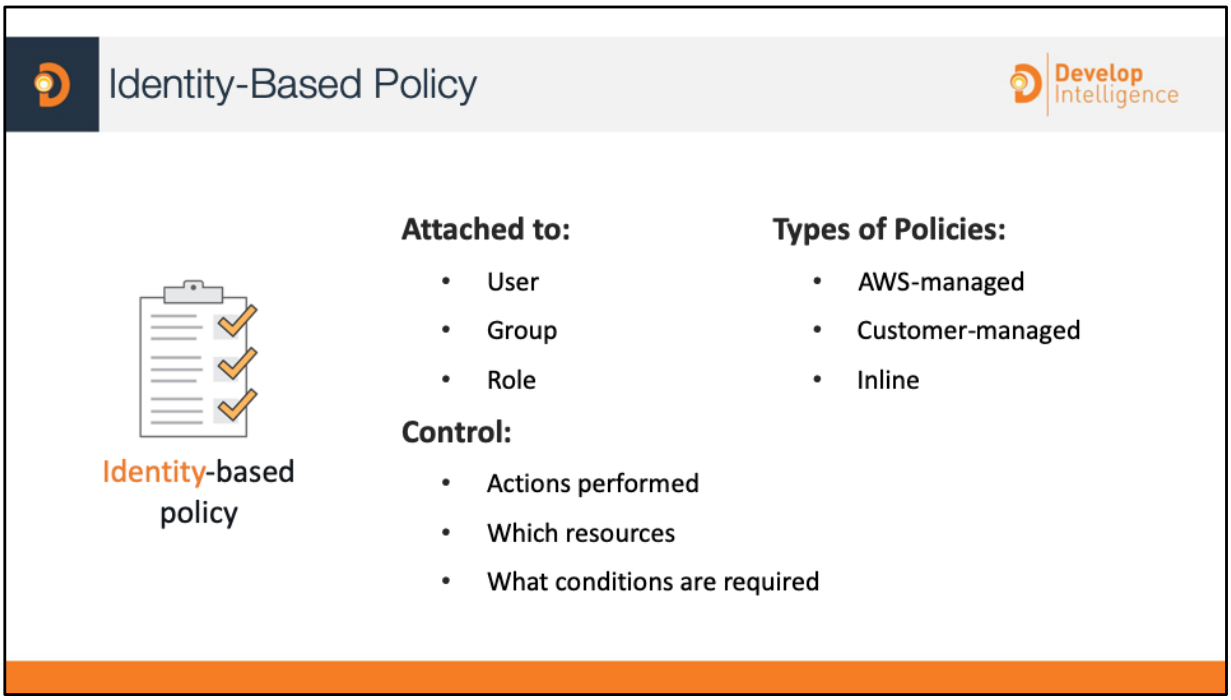


Policies give you the opportunity to fine-tune privileges granted to IAM users, groups, and roles. Because policies are stored in JSON format, they can be used in conjunction with a version control system. It's a good idea to define least-privilege access to each user, group, or role. Then, you can customize access to specific resources using an authorization policy.

When determining whether permission is allowed, IAM first checks for an explicit denial policy. If one does not exist, it then checks for an explicit allow policy. If neither an explicit deny or explicit allow policy exists, IAM reverts to the default: implicit deny.



Permissions in the policies determine whether the request is allowed or denied. Policies are stored in AWS as JSON documents attached to principals as *identity-based policies*, or to resources as *resource-based policies*.



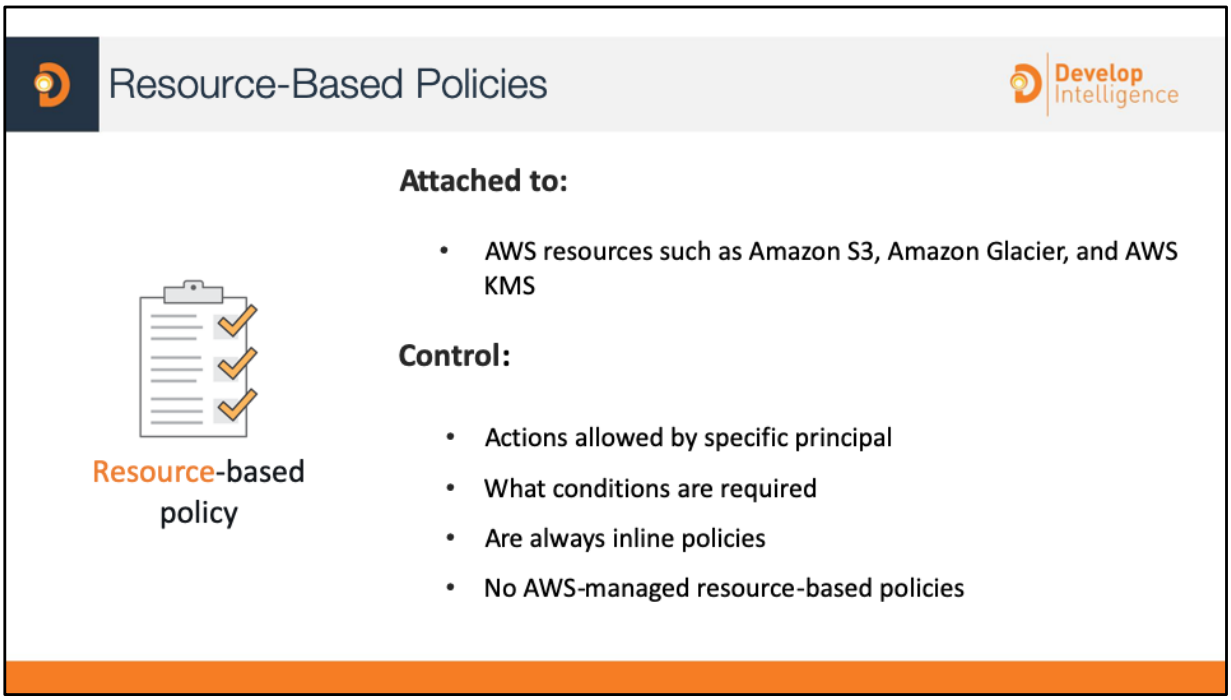
Identity-Based Policies
Identity-based policies are permission policies that you can attach to a principal (or identity), such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions.

Identity-based policies can be further categorized as follows:

Managed policies are standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. You can use two types of managed policies:

- **AWS managed policies** are managed policies that are created and managed by AWS. If you are new to using policies, we recommend that you start by using AWS managed policies.
- **Customer managed policies** are 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. You can create and edit an IAM policy in the visual editor or by creating the JSON policy document directly.

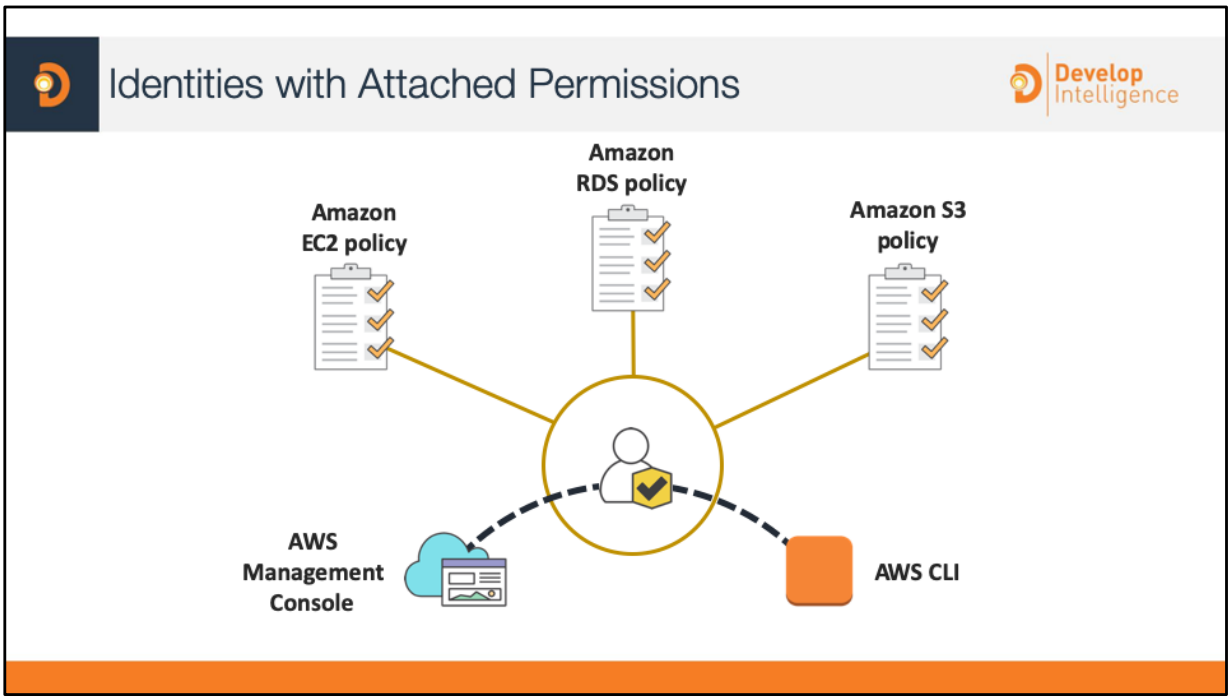
Inline policies are policies that you create and manage and that are embedded directly into a single user, group, or role.



Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket.

These policies control which actions a specified principal can perform on that resource and under what conditions. Resource-based policies are inline policies, and there are no managed resource-based policies.

Although IAM identities are technically AWS resources, you cannot attach a resource-based policy to an IAM identity.



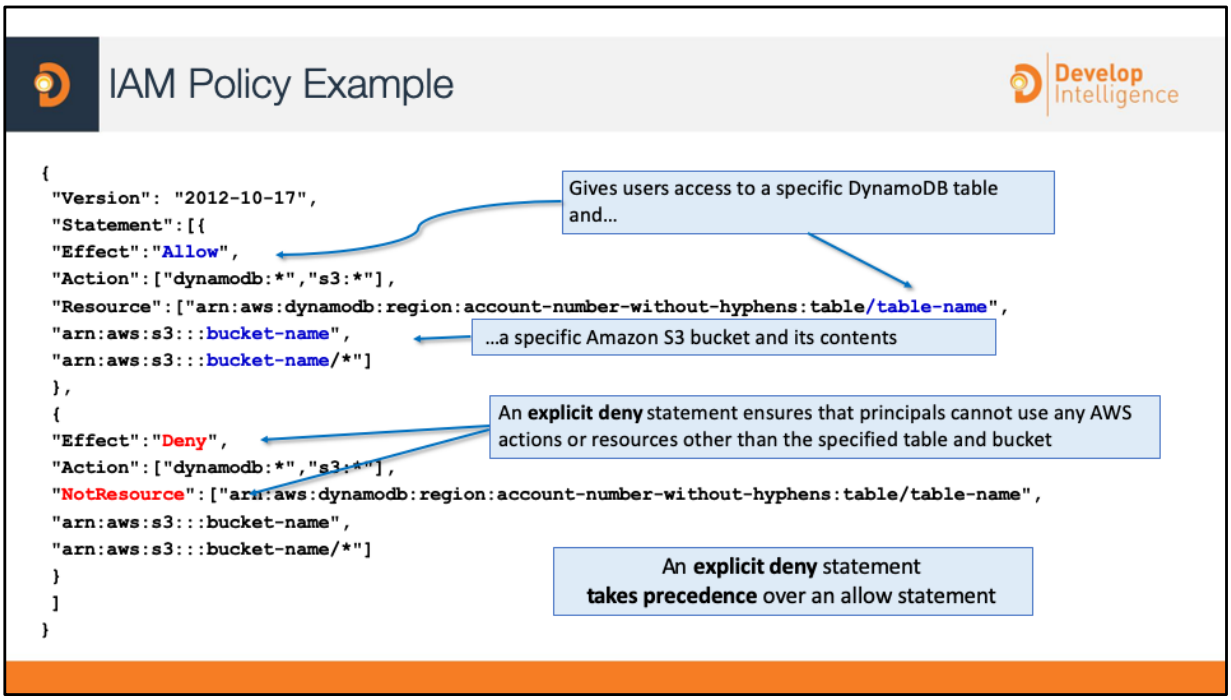
An IAM user is really just an identity (principal) with associated permissions.

You might create an IAM user to represent an application that must have credentials in order to make requests to AWS.

An application might have its own identity in your account and its own set of permissions to access AWS services. This is similar to how, in modern operating systems, processes have their own identities and permissions. If an application or even an EC2 instance has permissions to access something like an S3 bucket, then there's no need to embed credentials in code.

An IAM policy is a formal statement of one or more permissions.

- You attach a policy to any IAM entity.
- Policies authorize the actions that may (or may not) be performed by the entity.
- A single policy can be attached to multiple entities.
- A single entity can have multiple policies attached to it.



Important: Do not edit the Version statement. It refers to the engine that processes the IAM policy.

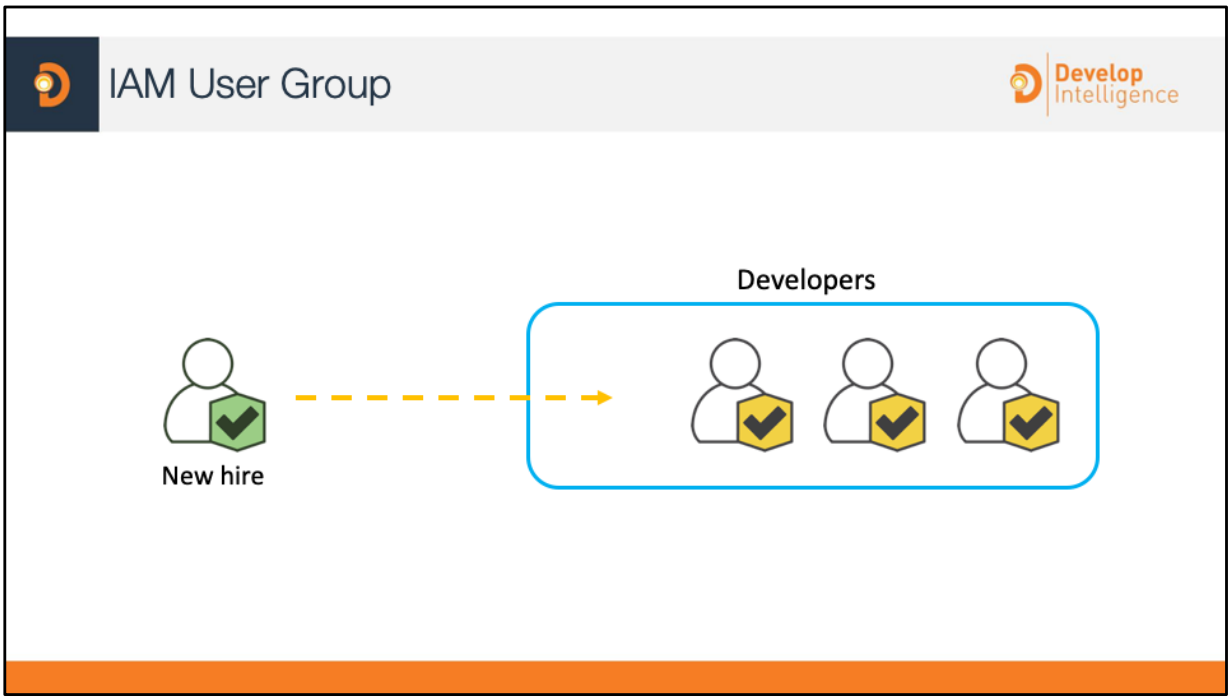
The asterisks in the Action section are wild cards. In this case, they allow all actions for the services DynamoDB and Amazon S3. You can also use wild cards with partial names. For example, s3:List* will match ListAllMyBuckets, ListBucket, ListBucketByTags, ListBucketMultipartUploads, ListBucketVersions, and ListMultipartUploadParts.

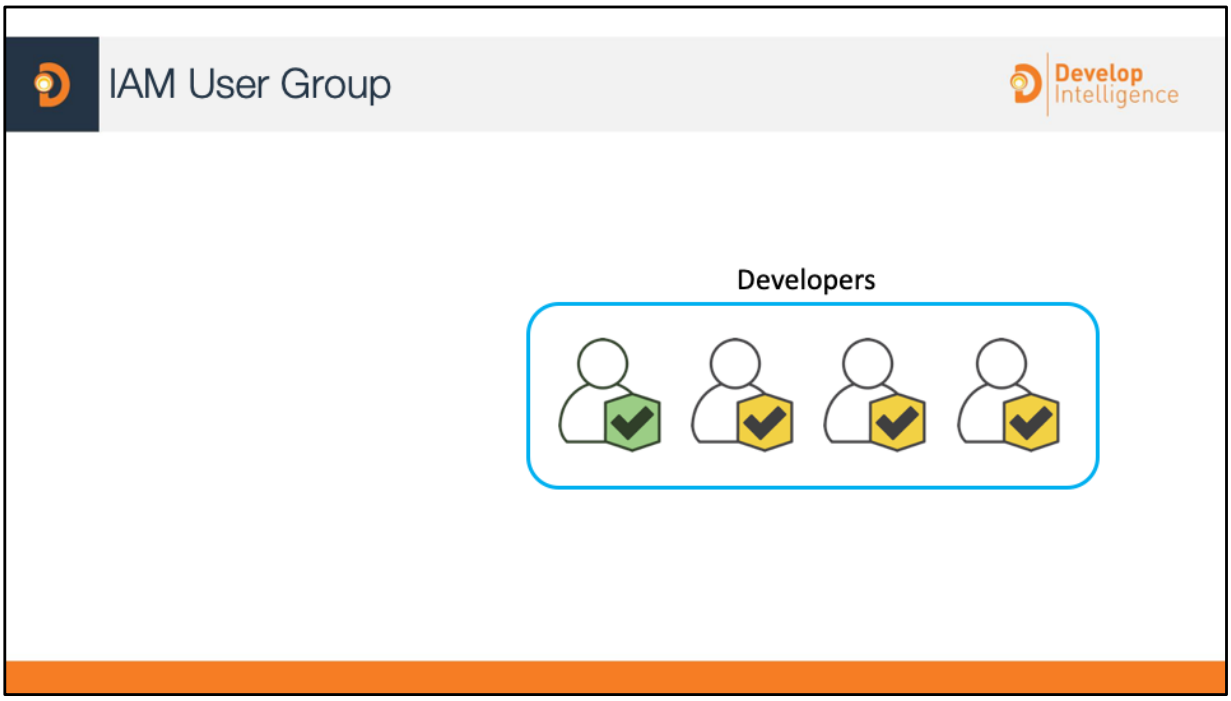
In the Deny section, NotResource can be confusing if you've never seen it before. NotResource is an advanced policy element that explicitly matches everything except the specified list of resources.

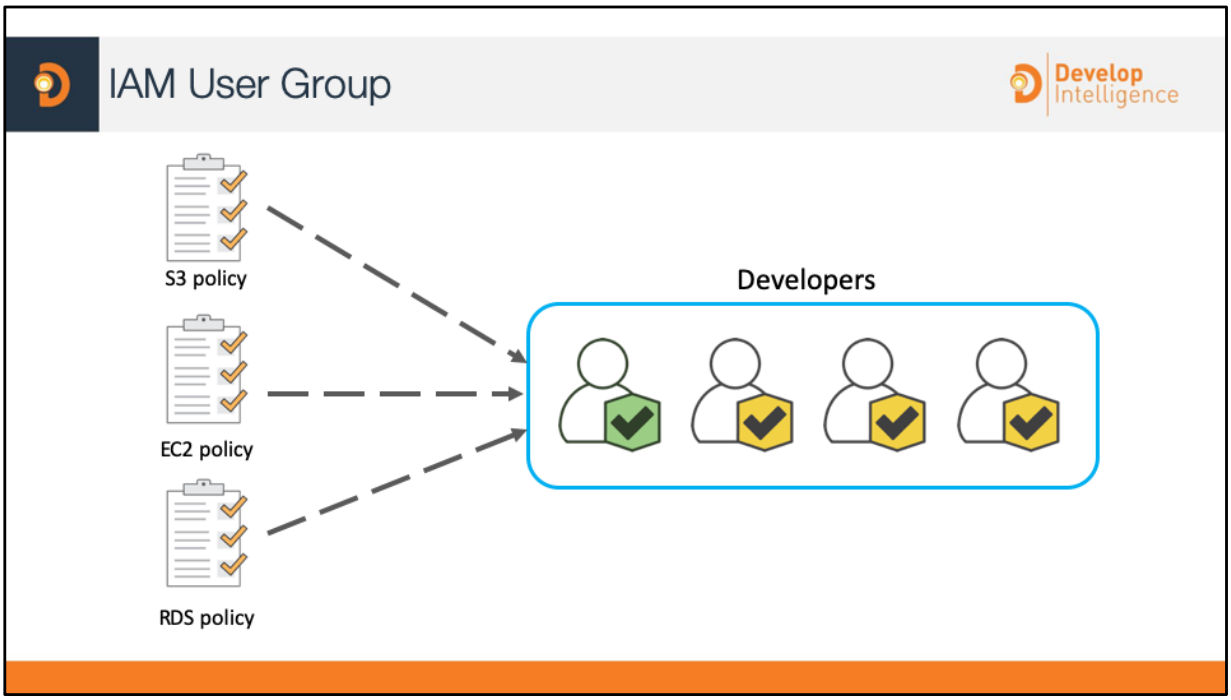
Here, NotResource means explicitly deny *everything other than* what's listed here. If you were to make a change to the top half of the policy, you'd have to make a corresponding change to the deny statement.

Why do this if there's an implicit deny? For some, it's important to lock down access to prevent inadvertent the granting of permission. (It's easy to assume malicious intent, but it may also be a user who has good intentions but poor understanding.) Be aware that this creates added complexity. That's not necessarily a bad thing, but complexity can look like additional work.

Using NotResource can result in a shorter policy by listing only a few resources that should not match, rather than including a long list of resources that will match. When using NotResource, keep in mind that resources specified in this element are the *only* resources that are limited.

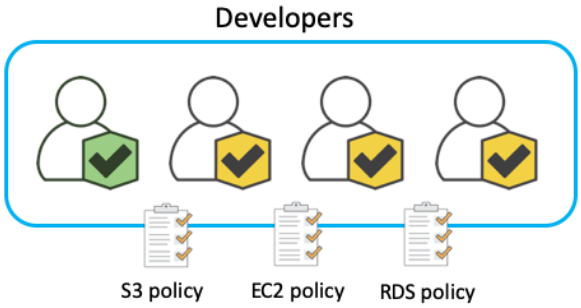

IAM User Group
Develop Intelligence
Developers
New hire

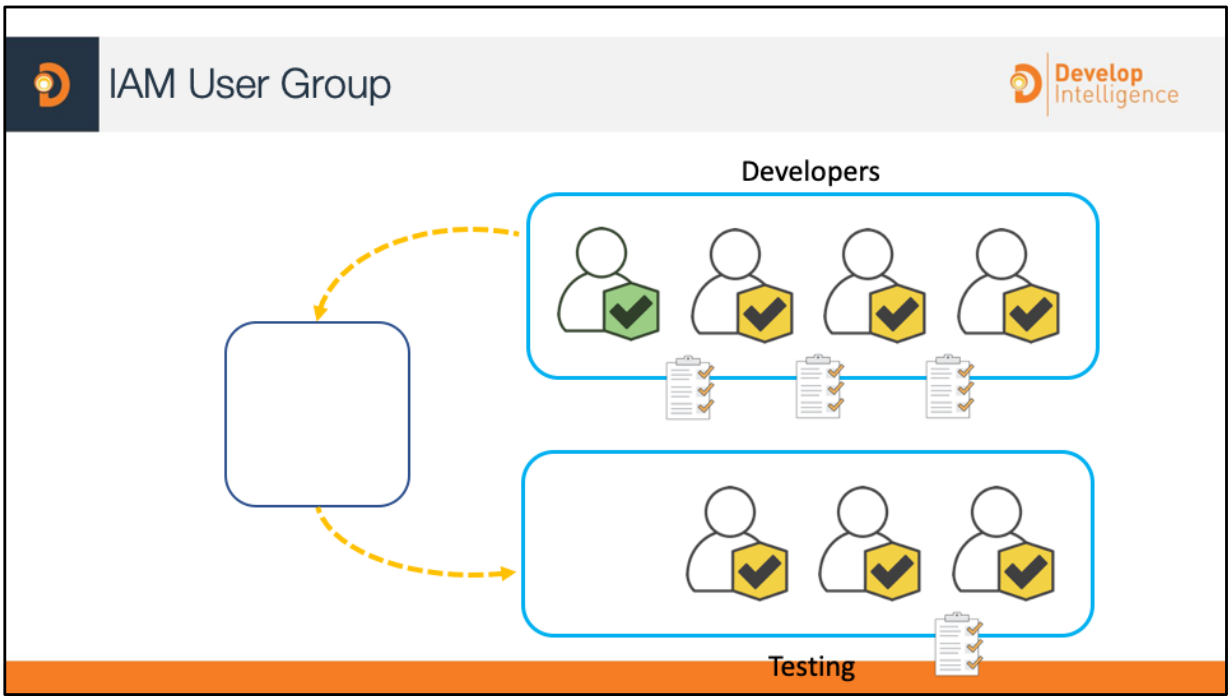

IAM User Group
Develop Intelligence
Developers

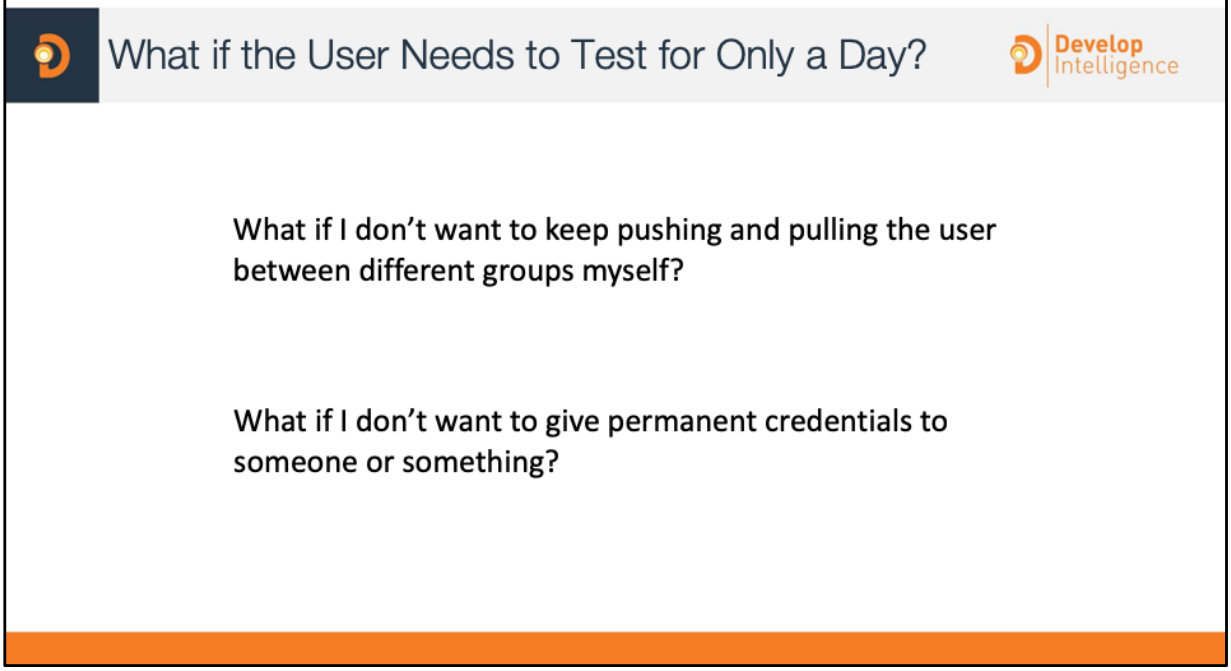

IAM User Group
Develop Intelligence
S3 policy
EC2 policy
RDS policy
Developers

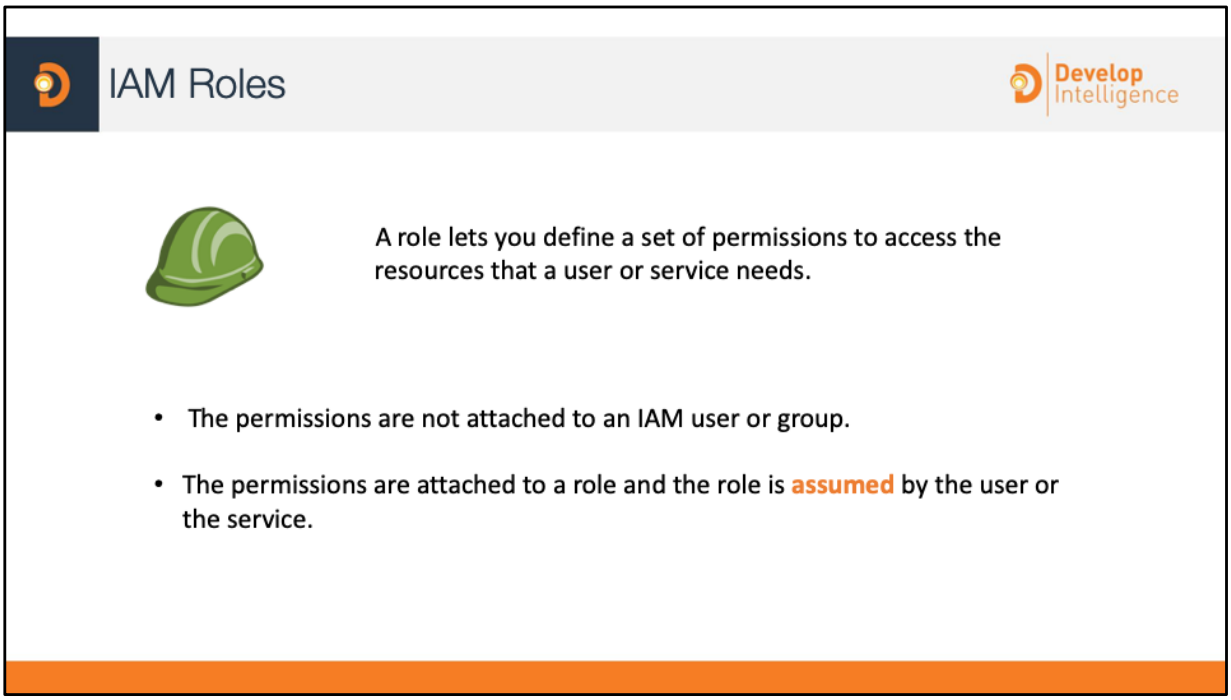
IAM User Group

Develop Intelligence




IAM User Group
Develop Intelligence
Developers
Testing


What if the User Needs to Test for Only a Day?
Develop Intelligence
What if I don't want to keep pushing and pulling the user between different groups myself?
What if I don't want to give permanent credentials to someone or something?



A *role* lets you define a set of permissions to access the resources that a user or service needs, but the permissions are not attached to an IAM user or group. The permissions are attached to a role and the role is *assumed* by the user or the service.

Roles eliminate the need to create multiple accounts for individual users.

When a user assumes a role, the existing permissions are temporarily forgotten. AWS returns temporary security credentials that the user or application uses to make programmatic requests to AWS.

Because of this, you don't have to share long-term security credentials (for example, by creating an IAM user) for each entity that requires access to a resource.

For a service such as Amazon EC2, applications, or AWS services can programmatically assume a role at run-time.

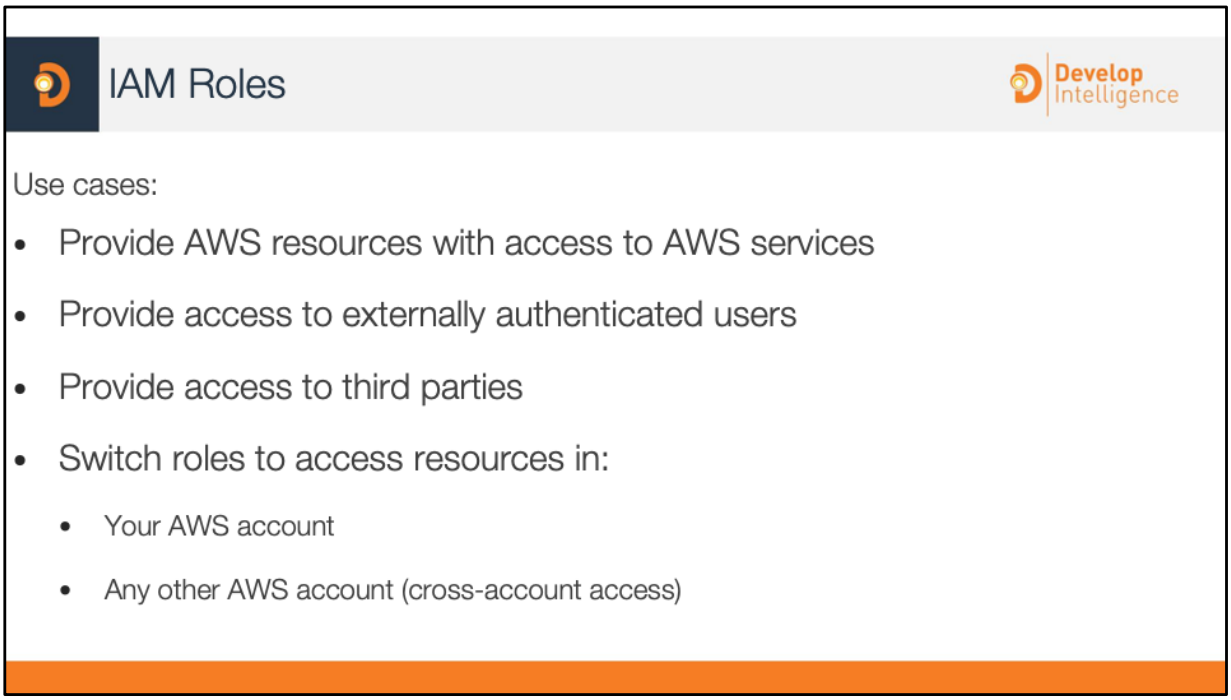
You create a role in the AWS account that contains the resources that requires access. When you create the role, you specify two policies: trust and access.

- The **trust** policy specifies who is allowed to assume the role (the trusted entity, or principal).
- The **access** (or *permissions*) policy defines which actions and resources the principal is allowed to use.

This is useful if your organization already has its own identity system, such as a corporate user directory. Another use case is with a mobile app or web application that requires access to AWS resources. With an identity provider, authentication is managed externally. This helps keep your AWS account secure because you don't have to distribute or embed long-term security credentials in your application.

For more information, see
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers.html

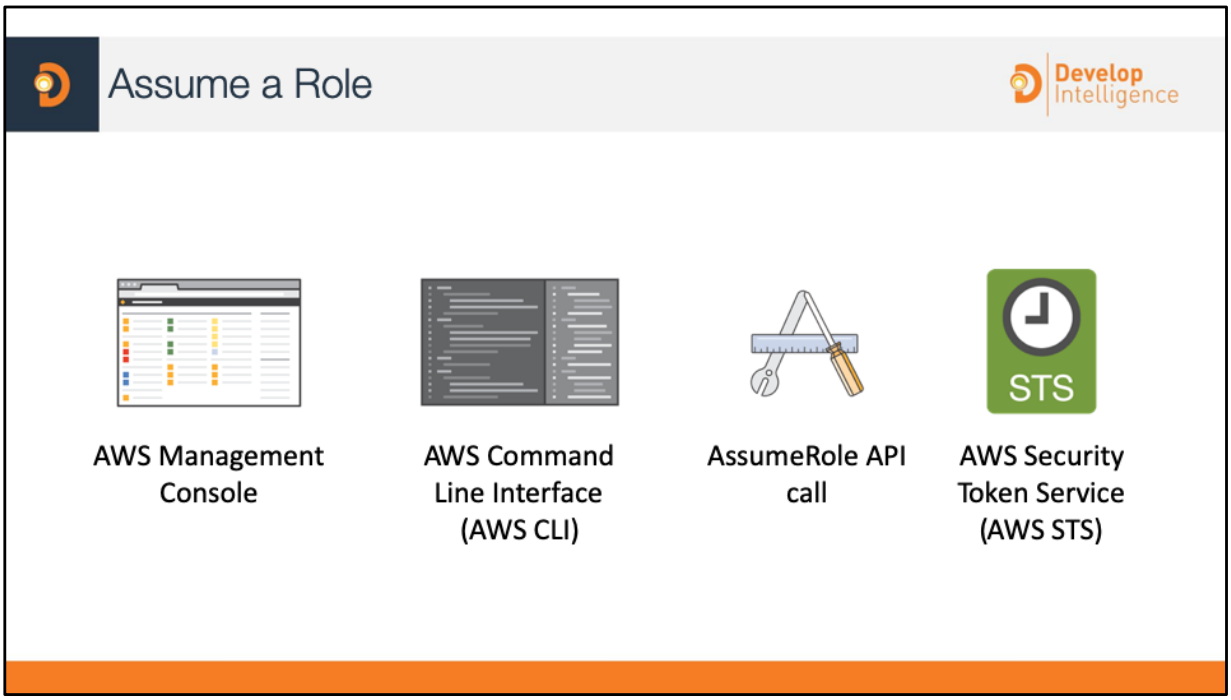
The principal can also be an IAM user, group, or role from other AWS accounts, including the ones not owned by you. This is a bit of an over-simplification of the process, but by creating a role for external account access, you don’t have to manage usernames and passwords for third parties. Requests come in must match your requirements for a role. If you no longer want people to have access, you can modify/delete the role. That means you don’t need to create and manage accounts for people outside of your organization.



The simplest way to use roles is to grant your IAM users permissions to switch to roles that you create within your own or another AWS account. They can switch roles easily using the IAM console. That enables them to use permissions that you don't ordinarily want them to have, and then exit the role to surrender those permissions. This can help prevent accidental access to or modification of sensitive resources.

Federated users sign-in using credentials provided by an identity provider (IdP). AWS then provides the IdP with temporary credentials associated with a role to pass on to the user for inclusion in subsequent AWS resource requests. Those credentials provide the permissions granted to the assigned role. This can be helpful if you want to use existing identities in a corporate directory or a third-party IdP.

When third parties need access to your organization's AWS resources, you can use roles to delegate access to them. For example, a third party might provide a service for managing your AWS resources. With IAM roles, you can grant these third parties access to your AWS resources without sharing your AWS security credentials. Instead, they can assume a role that you created to access your AWS resources.



Roles can be assumed using the console, the CLI, the AssumeRole API, and the AWS Security Token Service (AWS STS), which is a web service that provides temporary, limited-privilege credentials for IAM users or for users authenticated using federation.

The AssumeRole action returns a set of temporary security credentials consisting of an access key ID, a secret access key, and a security token. Typically, AssumeRole is used for cross-account access or federation.

AWS STS supports AWS CloudTrail, which records AWS calls for AWS accounts and delivers log files to an Amazon S3 bucket.

CloudTrail logs all authenticated API requests (made with credentials) to IAM and AWS STS APIs. CloudTrail also logs non-authenticated requests to the AWS STS actions, AssumeRoleWithSAML and AssumeRoleWithWebIdentity and logs information provided by the identity provider.

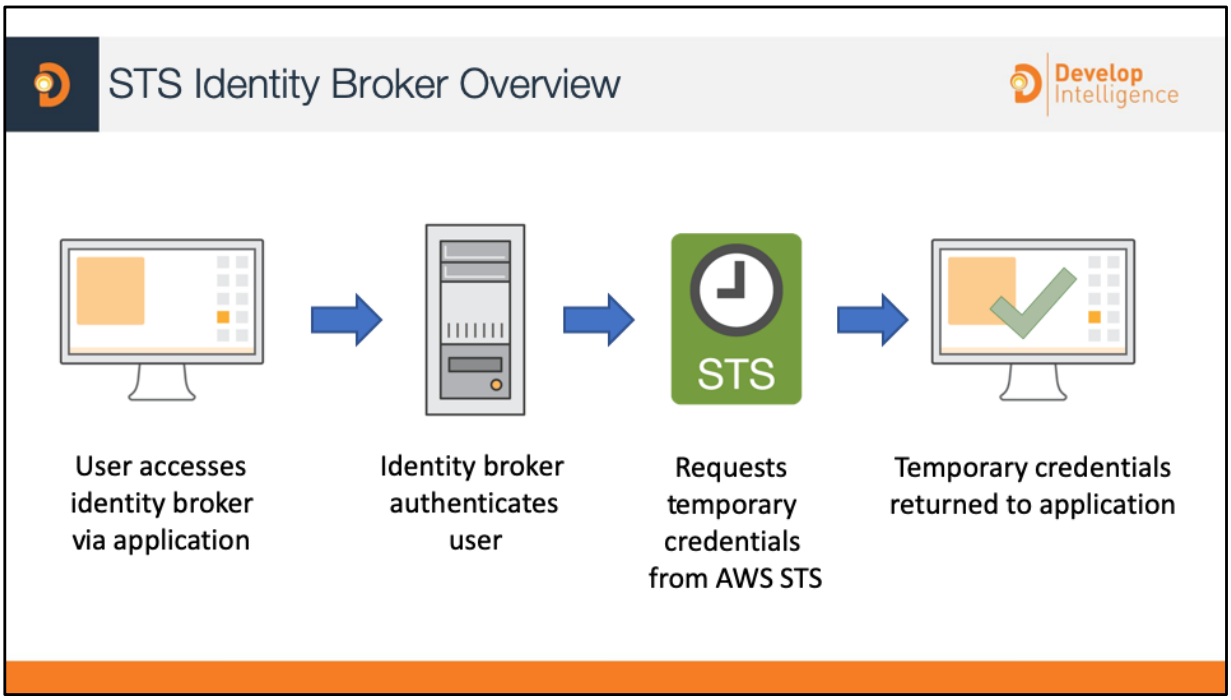
Use this information to map calls made by a federated user with an assumed role back to the originating external federated caller.

In the case of AssumeRole, it is possible to map calls back to the originating AWS service or to the account of the originating user.

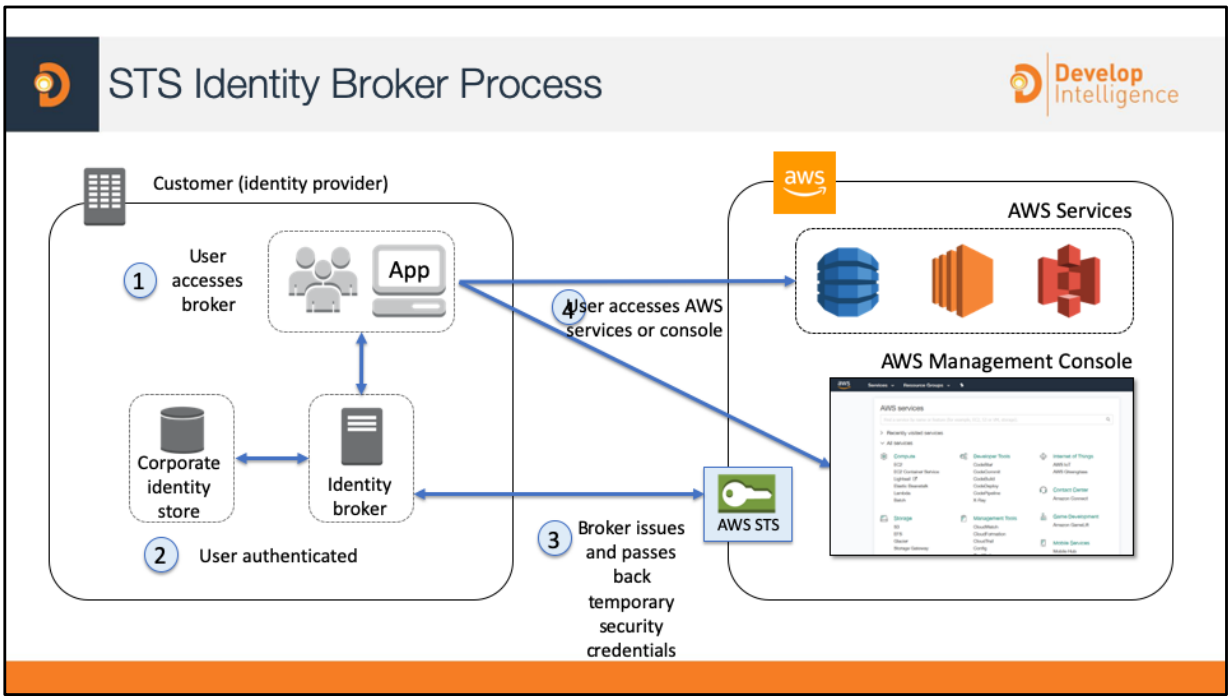
The userIdentity section of the JSON data in the CloudTrail log entry contains the information needed to map the AssumeRole request with a specific federated user.

For more information, see:

- https://docs.aws.amazon.com/STS/latest/APIReference/Welcome.html
- https://docs.aws.amazon.com/IAM/latest/UserGuide/cloudtrail-integration.html



There are four primary steps in using AWS STS to create temporary credentials for an application that's using a third-party authentication service.



In this scenario:

- The identity broker application has permissions to access the AWS STS API to create temporary security credentials.
- The identity broker application can verify that employees are authenticated within the existing authentication system.
- Users get a temporary URL that gives them access to the console (which is referred to as single sign-on).

IAM user groups from another AWS account:
You can establish cross-account access by using IAM roles. In the trusting account, the resource must be in a service that supports roles.

IAM users within the current account:
For mission-critical permissions that IAM users might not frequently use, you can separate those permissions from their normal day-to-day permissions by using roles. Users have to actively assume a role, which can prevent them from accidentally performing disruptive actions.

For example, you might have Amazon EC2 instances that are critical to your organization. Instead of directly granting administrators permission to terminate the instances, you can create a role with those privileges and allow administrators to assume the role.

Administrators won't have permission to terminate those instances; they must first assume a role. By using a role, administrators must take an additional step to assume a role before they can stop an instance that's critical to your organization.

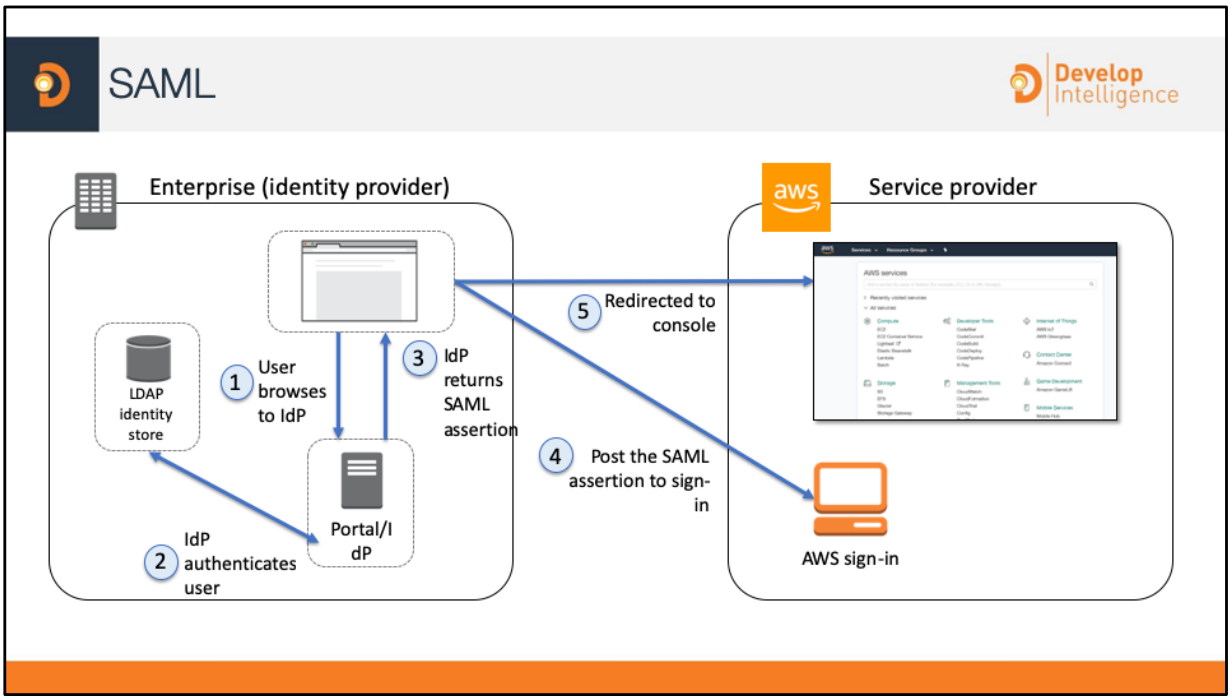
Third Parties:
When third parties require access to your organization's AWS resources, you can use roles to delegate API access to them. For example, a third party might provide a service for managing your AWS resources. With IAM roles, you can grant these third parties access to your AWS resources without sharing your AWS security credentials. Instead, they can assume a role that you created to access your AWS resources.

Third parties must provide you the following information for you to create a role that they can assume:

- The AWS account ID that the third party's IAM users use to assume your role. You specify their AWS account ID when you define the trusted entity for the role.
- An external ID that the third party can associate with your role. You specify the ID that was provided by the third party when you define the trusted entity for the role.
- The permissions that the third party requires in order to work with your AWS resources. You specify these permissions when defining the role's permission policy. This policy defines what actions they can take and what resources they can access.
- After you create the role, you must share the role's Amazon Resource Name (ARN) with the third party. They require your role's ARN in order to assume the role.

Identity Broker:

- Used to query AWS STS
- Determines user from a web request
- Uses AWS credentials (service account) to authenticate to AWS
- Issues temporary security credentials to access AWS APIs (through AWS STS)
- AWS permissions are configured by the administrator of the identity broker
- Configurable timeout: 1-36 hours
- For more information (including sample IIS authentication proxy C# code), see http://aws.amazon.com/code/1288653099190193.

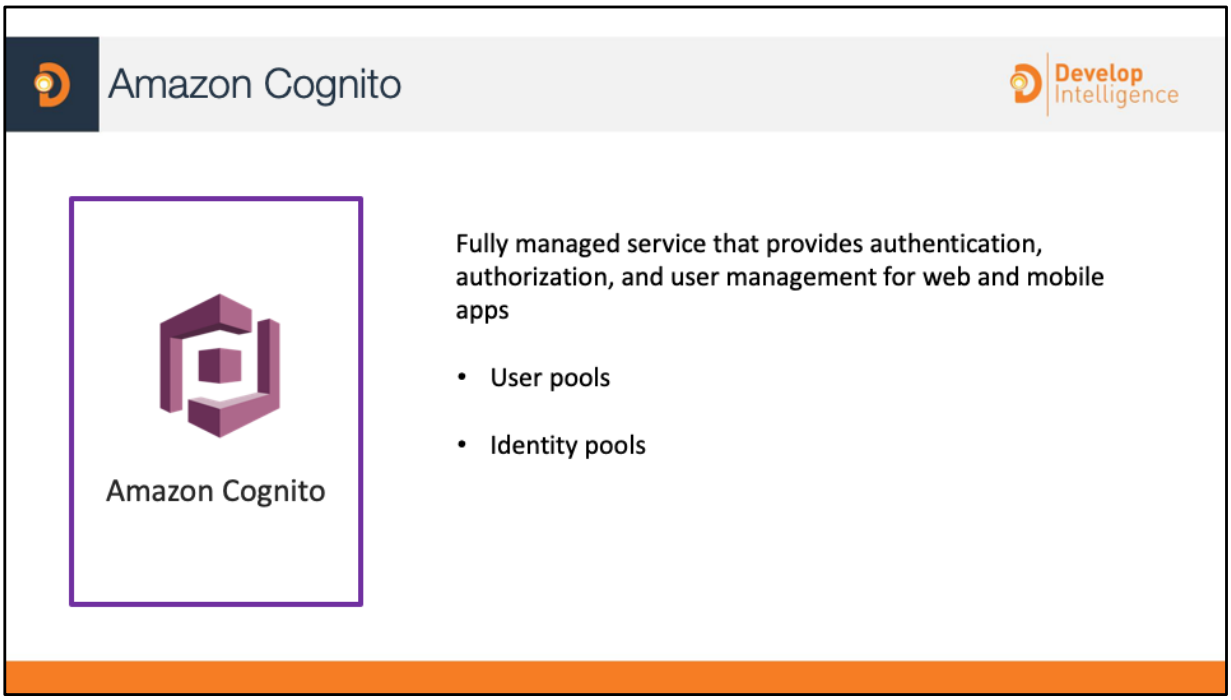


From the user's perspective, the process happens transparently. The user starts at your organization's internal portal and ends up at the console, without ever having to supply any AWS credentials.

1. **User browses to a URL.** A user in your organization browses to an internal portal in your network. The portal also functions as the IdP that handles the SAML trust between your organization and AWS.
2. **User is authenticated**. The identity provider (IdP) authenticates the user's identity against AD.
3. **User receives authentication response**. The client receives a SAML assertion (in the form of authentication response) from the IdP.
4. **Client posts to sign-in passing AuthN**. The client posts the SAML assertion to the new AWS sign-in endpoint. Behind the scenes, sign-in uses the AssumeRoleWithSAML API to request temporary security credentials and construct a sign-in URL.
5. **Client is redirected to the console**. The user's browser receives the sign-in URL and is redirected to the console.

For more information, see:

- https://aws.amazon.com/blogs/security/enabling-federation-to-aws-using-windows-active-directory-adfs-and-saml-2-0/
- https://aws.amazon.com/blogs/security/aws-federated-authentication-with-active-directory-federation-services-ad-fs/



Amazon Cognito is a fully-managed service that provides authentication, authorization, and user management for web and mobile apps. Users can sign in directly with a user name and password or through a third party such as Facebook, Amazon, or Google.

The two main components of Amazon Cognito are *user pools* and *identity pools.*

- **User pools** are user directories that provide sign-up and sign-in options for your app users.
- **Identity pools** enable you to grant your users access to other AWS services. Identity pools and user pools can be used separately or together.

A user pool is a user directory in Amazon Cognito. With a user pool, users can sign-in to a web or mobile app through Amazon Cognito, or federate through a third-party identity provider (IdP).

All members of the user pool have a directory profile that can be accessed through an SDK.

User pools provide:

- Sign-up and sign-in services
- A built-in, customizable web UI to sign in users
- Social sign-in with Facebook, Google, and Login with Amazon, and through SAML and OIDC identity providers from your user pool
- User directory management and user profiles
- Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification
- Customized workflows and user migration through AWS Lambda triggers

For more information about user pools, see https://docs.aws.amazon.com/cognito/latest/developerguide/getting-started-with-cognito-user-pools.html

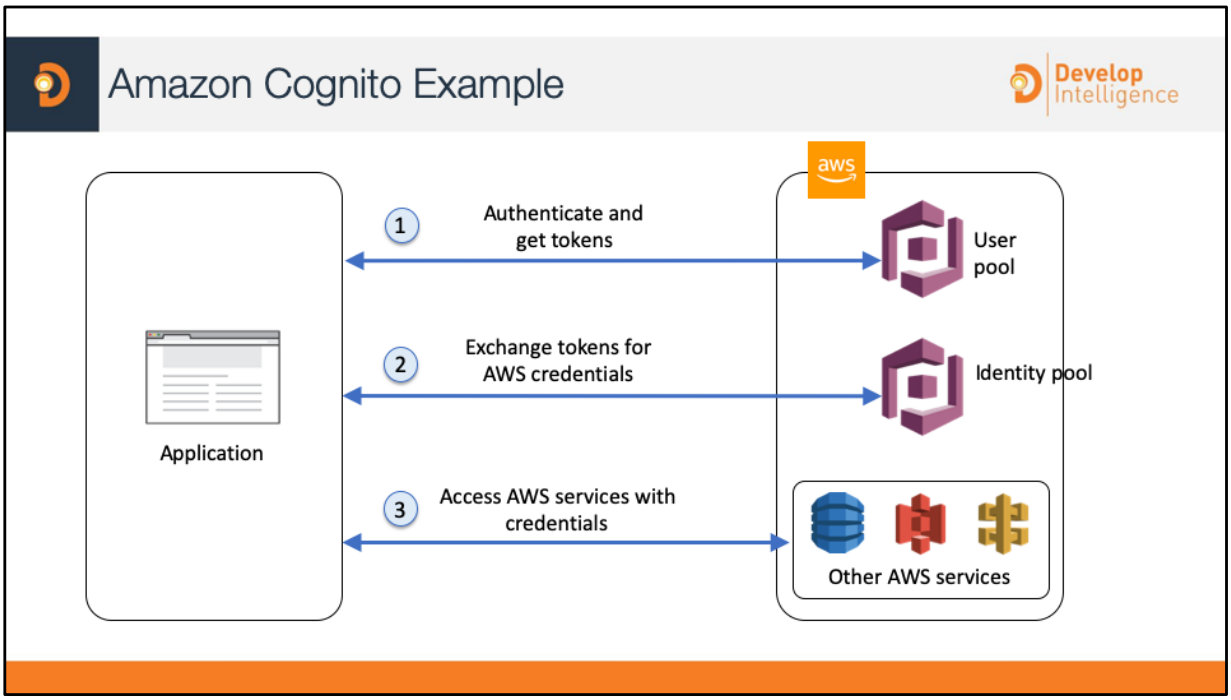
Amazon Cognito identity pools enable the creation of unique identities and permission assignment for users.

With an identity pool, users can obtain temporary AWS credentials to access AWS services or access resources through Amazon API Gateway.

Identity pools provide temporary AWS credentials for users who are guests (unauthenticated/anonymous) as well as the following identity providers:

- Amazon Cognito user pools
- Social sign-in with Facebook, Google, and Login with Amazon
- OpenID Connect (OIDC) providers
- SAML identity providers
- Developer authenticated identities

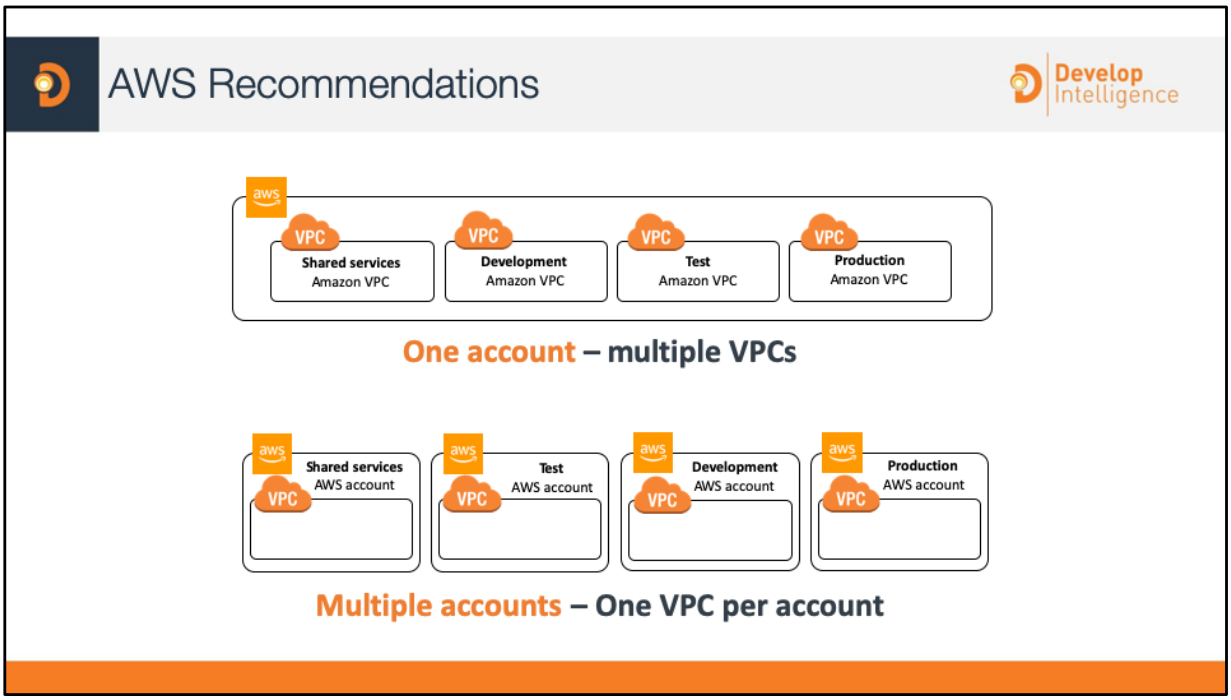
To save user profile information, an Amazon Cognito identity pool must be integrated with an Amazon Cognito user pool.



In this scenario, the goal is to authenticate a user and then grant that user access to another AWS service.

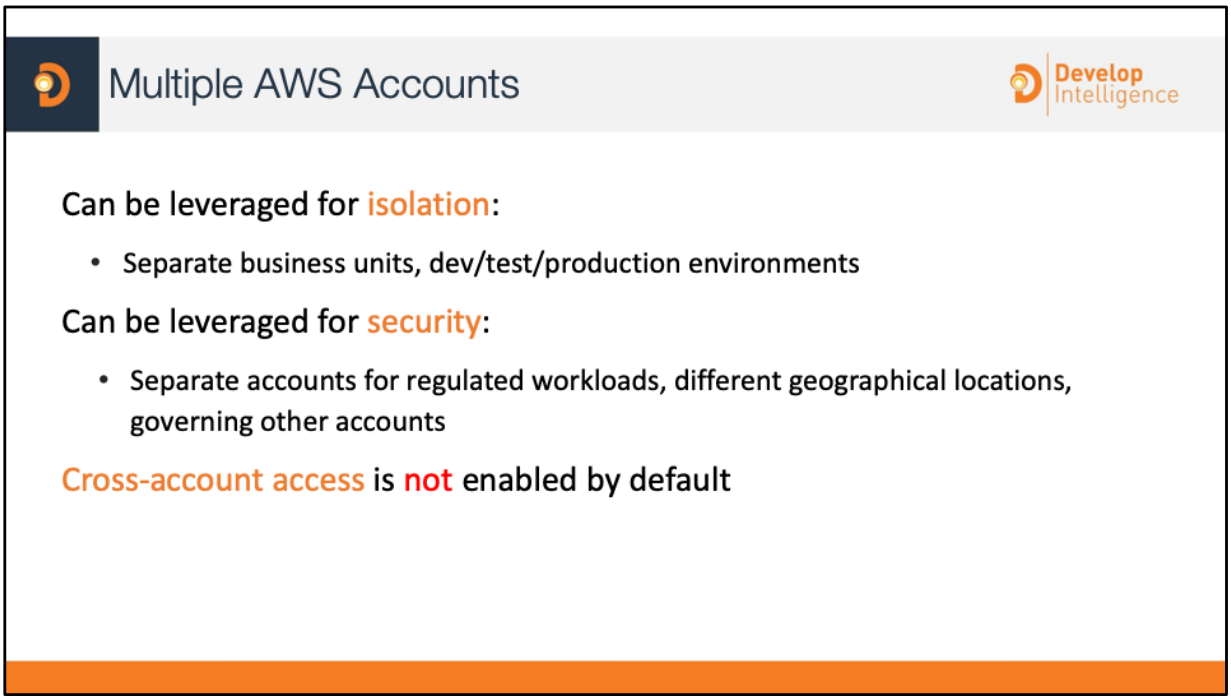
- In the first step, the app user signs in through a user pool and, after successfully authenticating, receives user pool tokens.
- Next, the app exchanges the user pool tokens for AWS credentials through an identity pool.
- Finally, the app user uses those AWS credentials to access other AWS services.


AWS “In the Wild”
Develop Intelligence
How many AWS accounts does your organization need?
aws
Dev
aws
Test
aws
Production



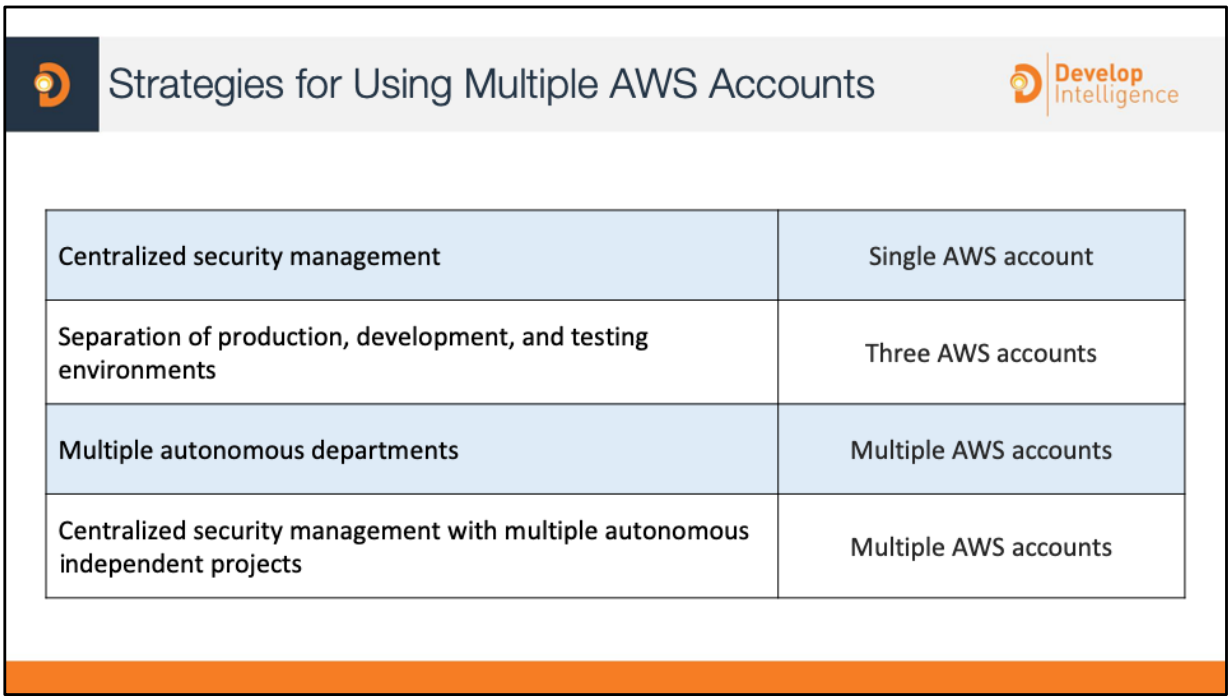
The two primary architectural patterns recommended by AWS are **multi-VPC** (in a single AWS account) and **multi-account**.

In a multi-account system, each account has a single VPC in it. In practice, organizations (large and small) create multiple accounts. They need to manage, maintain, and audit them.



Many AWS customers create multiple AWS accounts for their organizations, such as individual accounts for various business units or separate accounts for their development, test, and production resources.

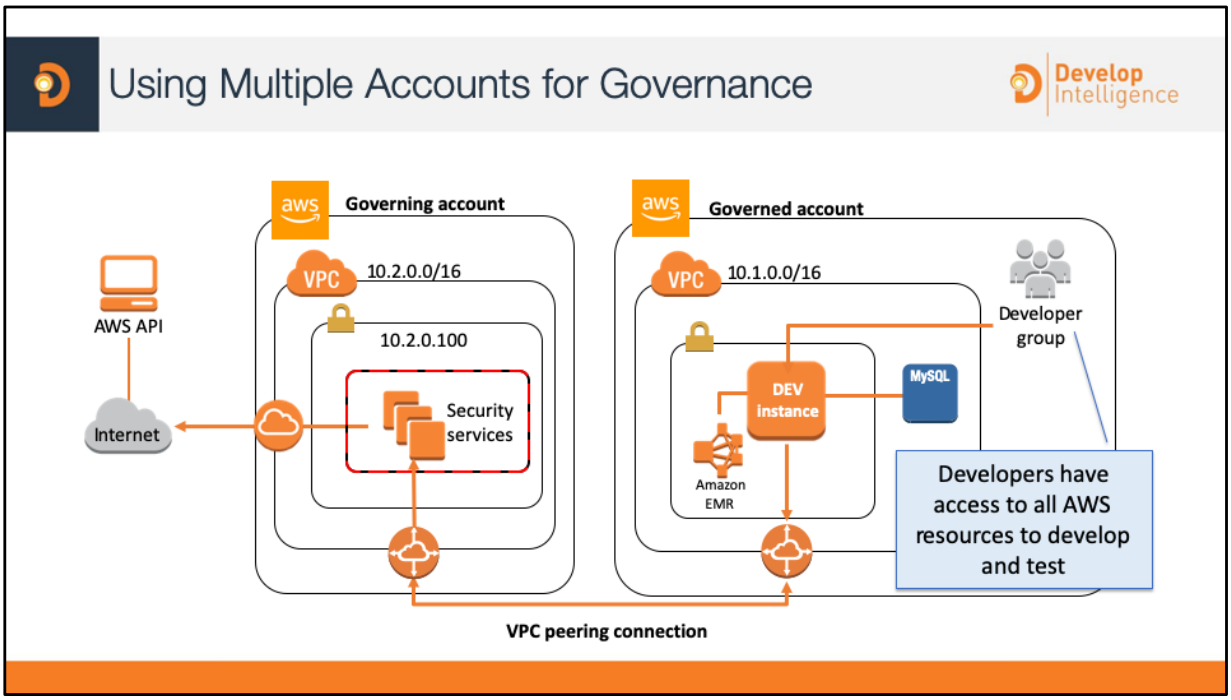
Using separate AWS accounts (usually with consolidated billing) for development and production resources allows customers to cleanly separate different types of resources and can also provide some security benefits.



You can design your AWS account strategy to maximize security and follow your business and governance requirements.

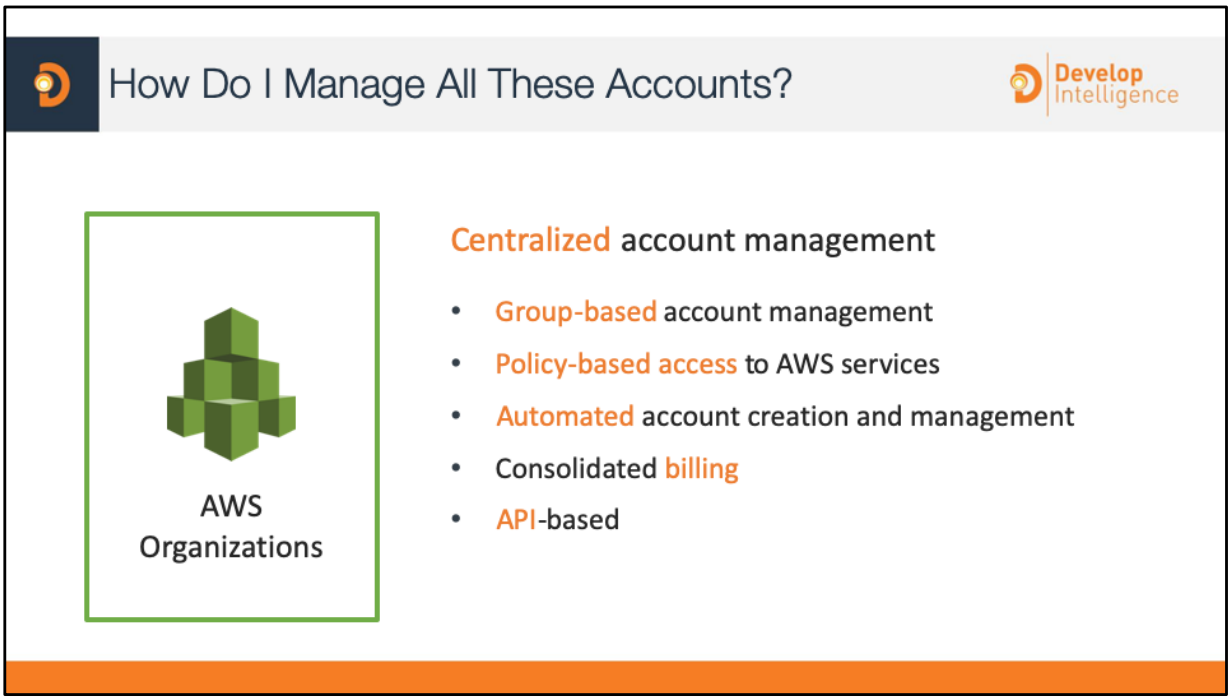
If you prefer centralized information security management with minimum overhead, you could opt for a single AWS account. Alternatively, if your business maintains separate environments for production, development, and testing, you could configure three AWS accounts—one for each environment. Also, if you have multiple autonomous departments, you could also create separate AWS accounts for each autonomous part of the organization.

When you use multiple accounts, a more efficient strategy is to create a single AWS account for common project resources (such as DNS services, Active Directory, and CMS) and separate accounts for the autonomous projects/departments. This allows you to assign permissions and policies under each department/project account and grant access to resources across accounts.



Many large companies use multiple accounts for security and governance. In this approach, two or more AWS accounts are created, with one designated as a Governing account and the others designated as Governed accounts. The solution is architected to isolate all management resources to the Governing account's network. All ingress and egress traffic to the Governed account passes through the security-specific services in the Governing account. This allows for an additional layer of security configured in the Governing account for enhanced security and governance purposes.

The Governed account should still be architected following the security best practices. The Governing account is used to provide an extra layer of security that can be managed centrally.



AWS Organizations is a managed service for account management. An **organization** is an entity that you create to consolidate, centrally view, and manage all of your AWS accounts. In Organizations, an organization has the functionality that is determined by the feature set that you enable.

Centrally manage policies across multiple AWS accounts

Organizations helps you manage policies for multiple AWS accounts. Use the service to create groups of accounts and then attach policies to a group to ensure the correct policies are applied across the accounts.

Organizations enables you to centrally manage policies across multiple accounts, without requiring custom scripts and manual processes.

Group-based Account Management

Using Organizations, you can create groups of AWS accounts. You can create separate groups of accounts to use with development and production resources, and then apply different policies to each group.

Policy-based Access to AWS Services

With Organizations, you can create Service Control Policies (SCPs) that centrally control AWS service use across multiple AWS accounts. SCPs put bounds around the permissions that IAM policies can grant to entities in an account, such as IAM users and roles. Entities can only use the services allowed by both the SCP and the IAM policy for the account. For example, If you want to restrict access to AWS Direct Connect, the SCP must allow access before IAM policies will work. You can apply policies to a group of accounts or all the accounts in your organization.

Automate AWS Account Creation and Management

Use the Organizations APIs to automate the creation and management of new AWS accounts. The Organizations APIs can create new accounts programmatically, and to add them to a group. The policies attached to the group are automatically applied to the new account. For example, you can automate the creation of sandbox accounts for developers and grant entities in those accounts access only to the necessary AWS services.

Consolidate billing across multiple AWS accounts

AS Organizations enables you to set up a single payment method for all the AWS accounts in your organization through consolidated billing. With consolidated billing, you can see a combined view of charges incurred by all your accounts. Consolidated billing also allows you to take advantage of pricing benefits from aggregated usage such as volume discounts for Amazon EC2 and Amazon S3.

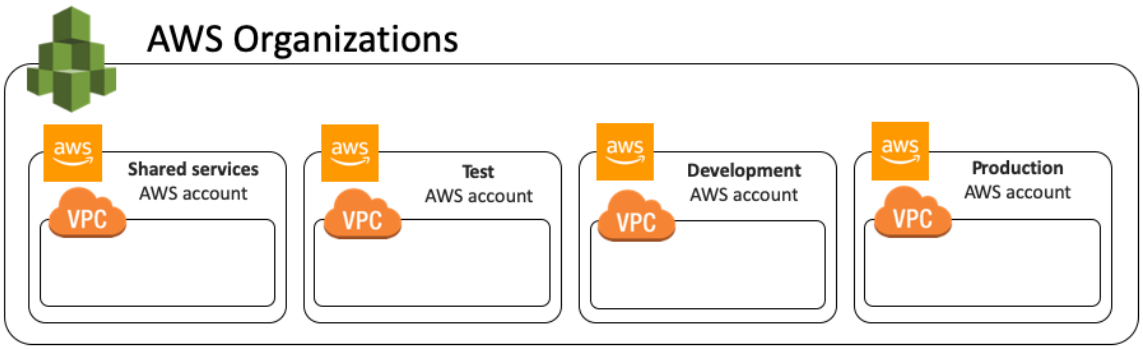
API level control of AWS Services

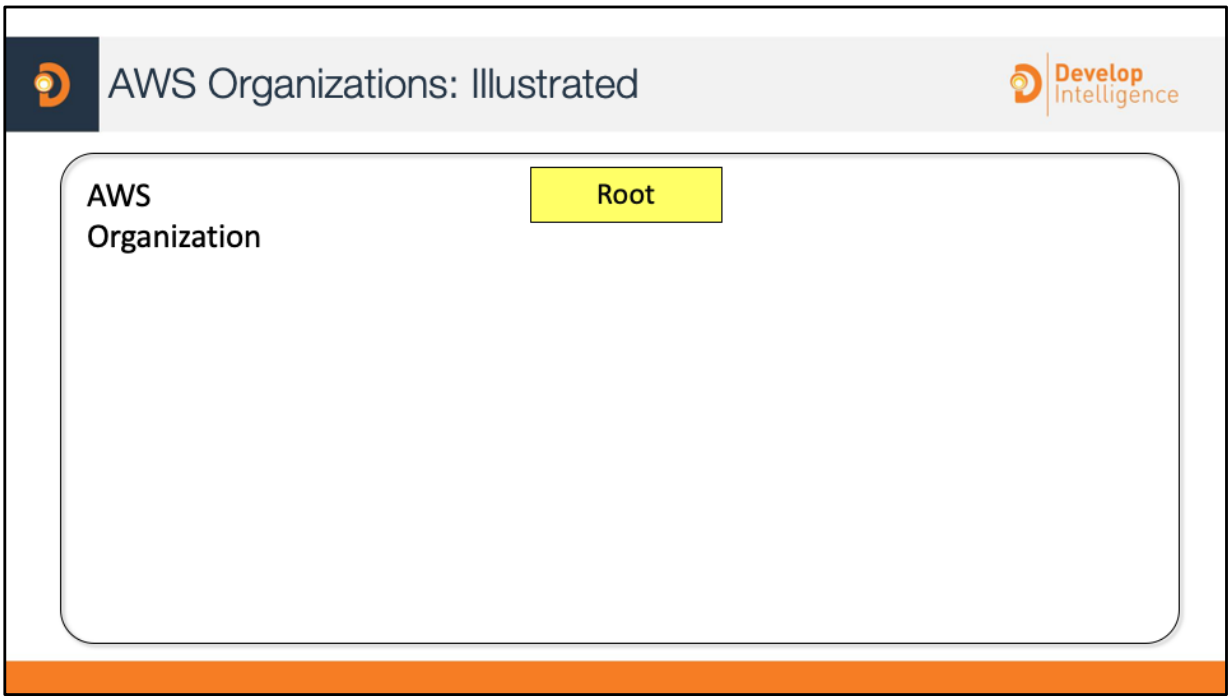
With Organizations, you can use SCPs to manage the use of AWS services at an API level. For example, you can apply a policy to a group of accounts to only allow IAM users in those accounts to read data from Amazon S3 buckets.

Using the Organizations APIs, you can create and add new accounts to a group. Policies attached to a group are automatically applied to accounts added to the group.

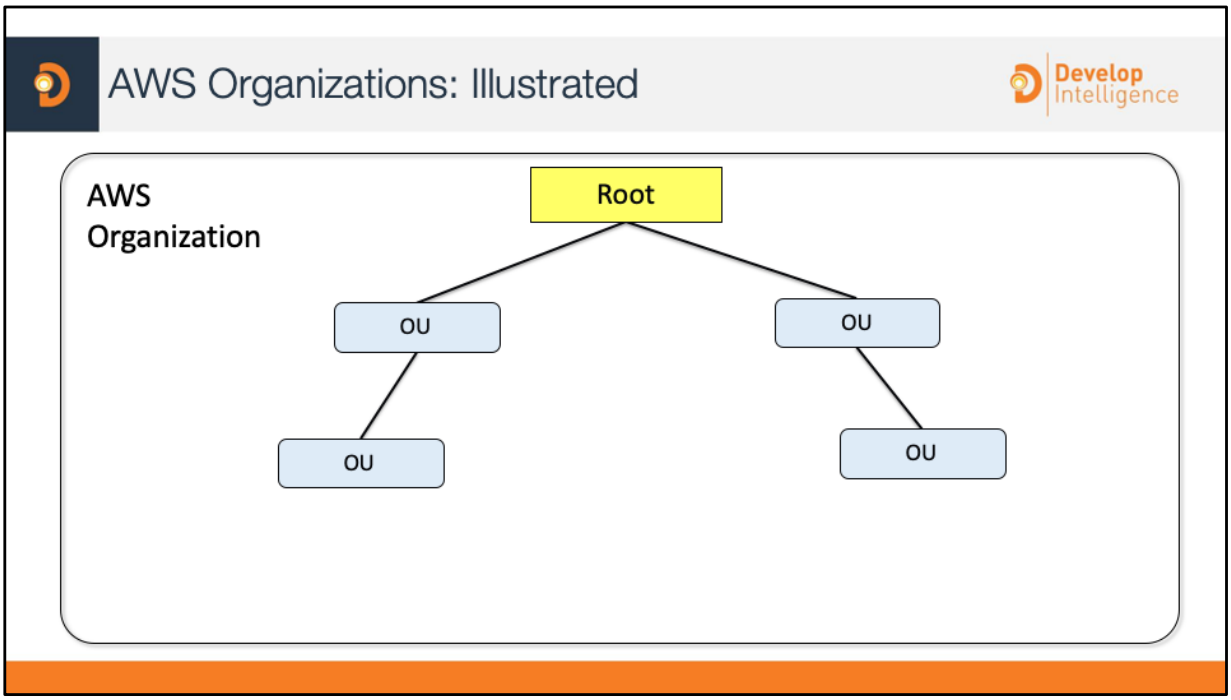
AWS Recommendations

Develop Intelligence

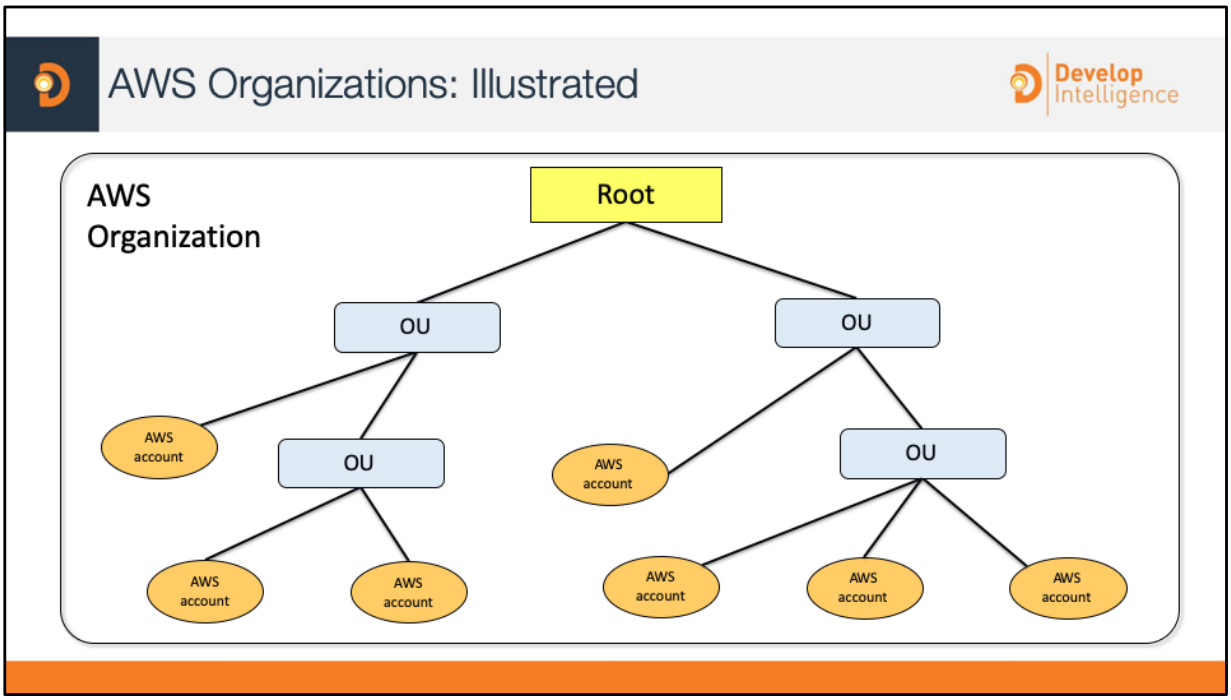




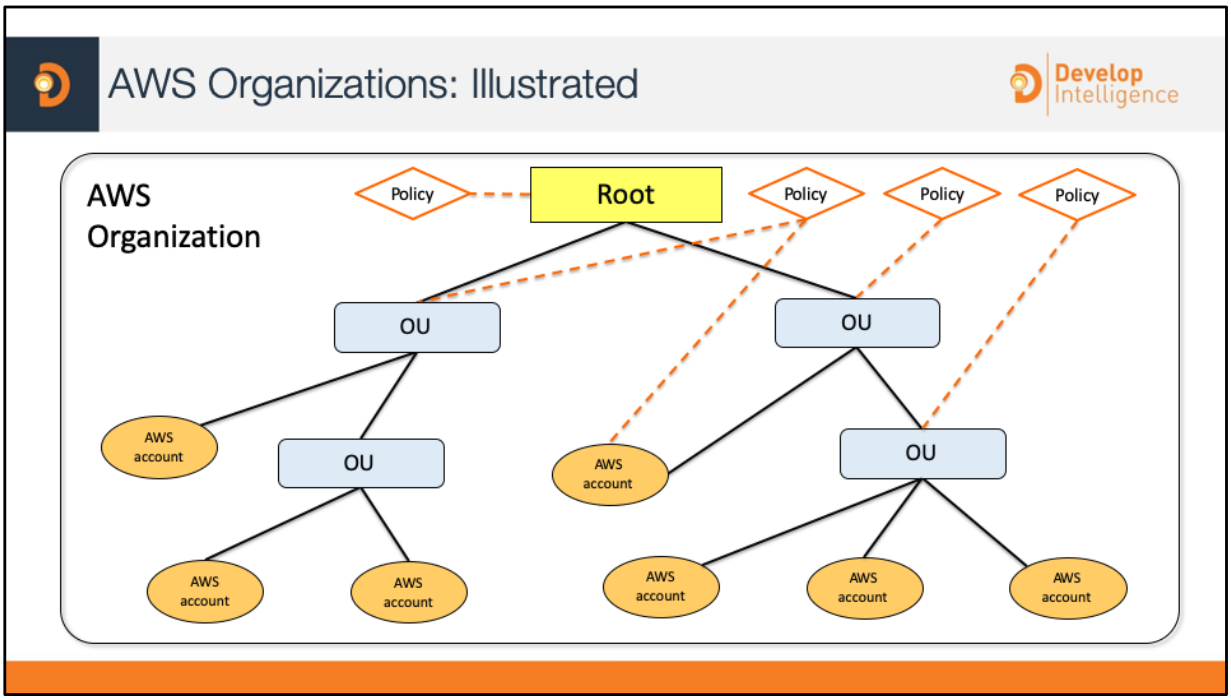
In this example, I have an organization has seven accounts that need to be organized into four Organizational Units (OUs) under the root.



Here, I've added four Organizational Units (OUs) to my Organization. Two sit directly under the root. Then, I have a single OU in each of my primary OUs.

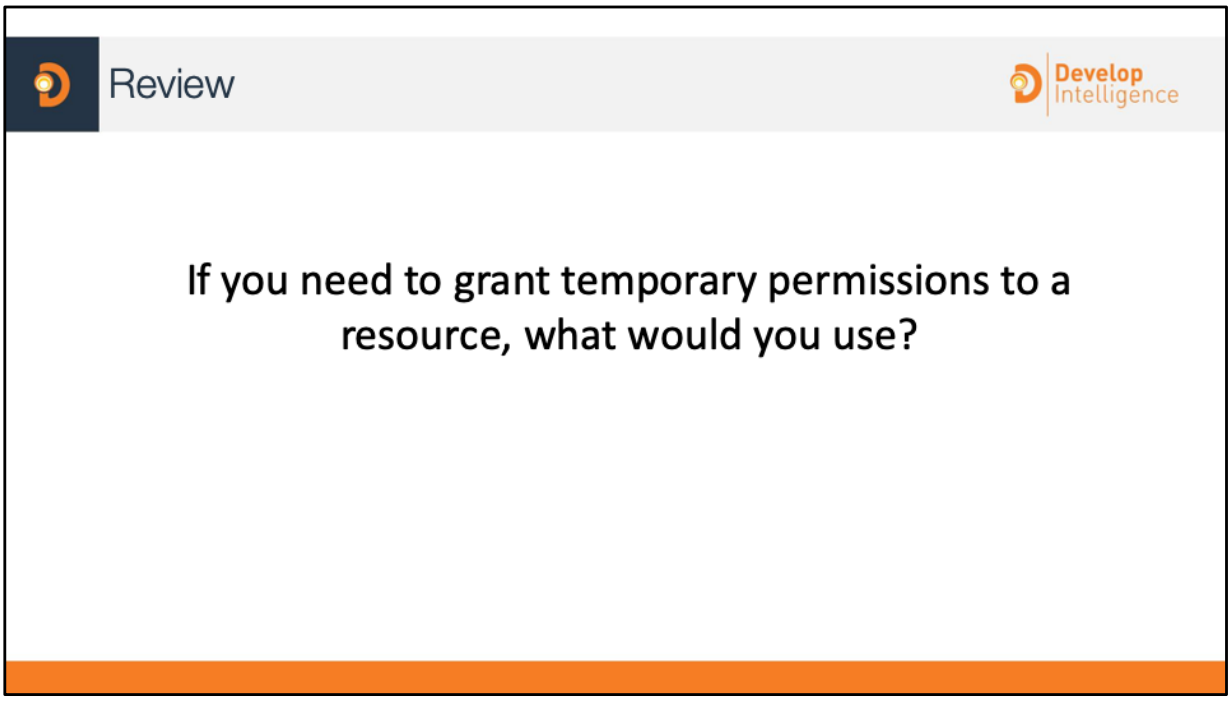


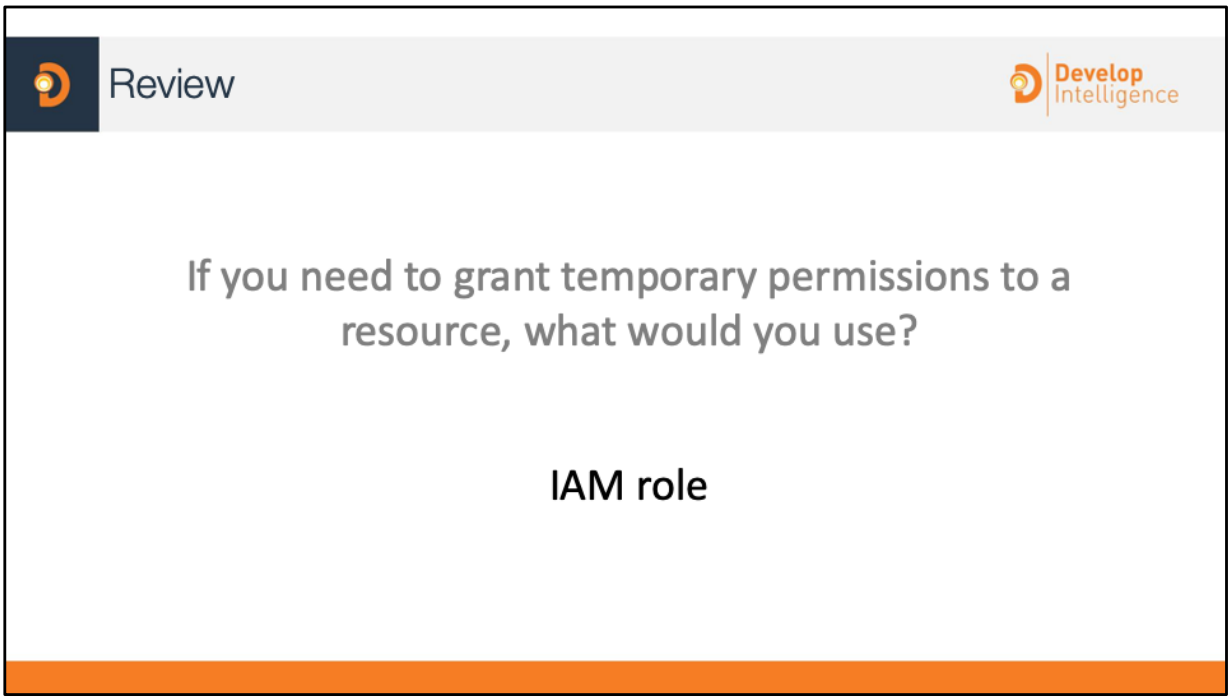
All seven of my AWS accounts are added to my organization and put into the appropriate OU.

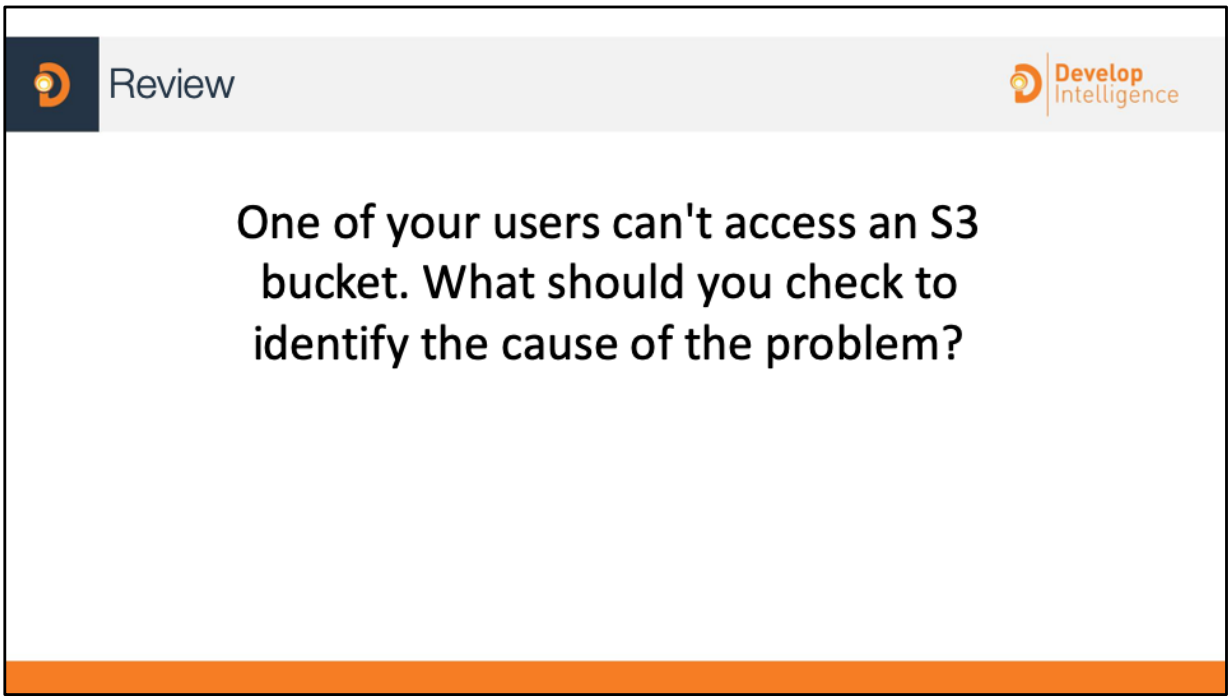


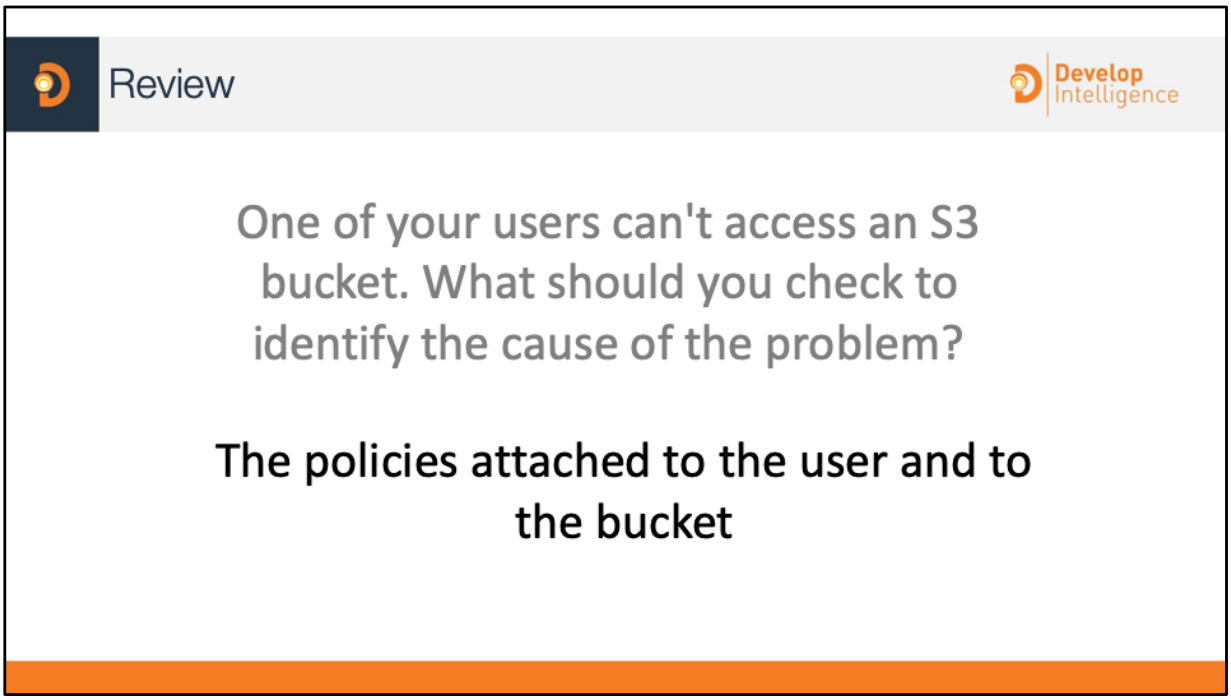
Once the accounts are added, I can apply SCPs to my organization.

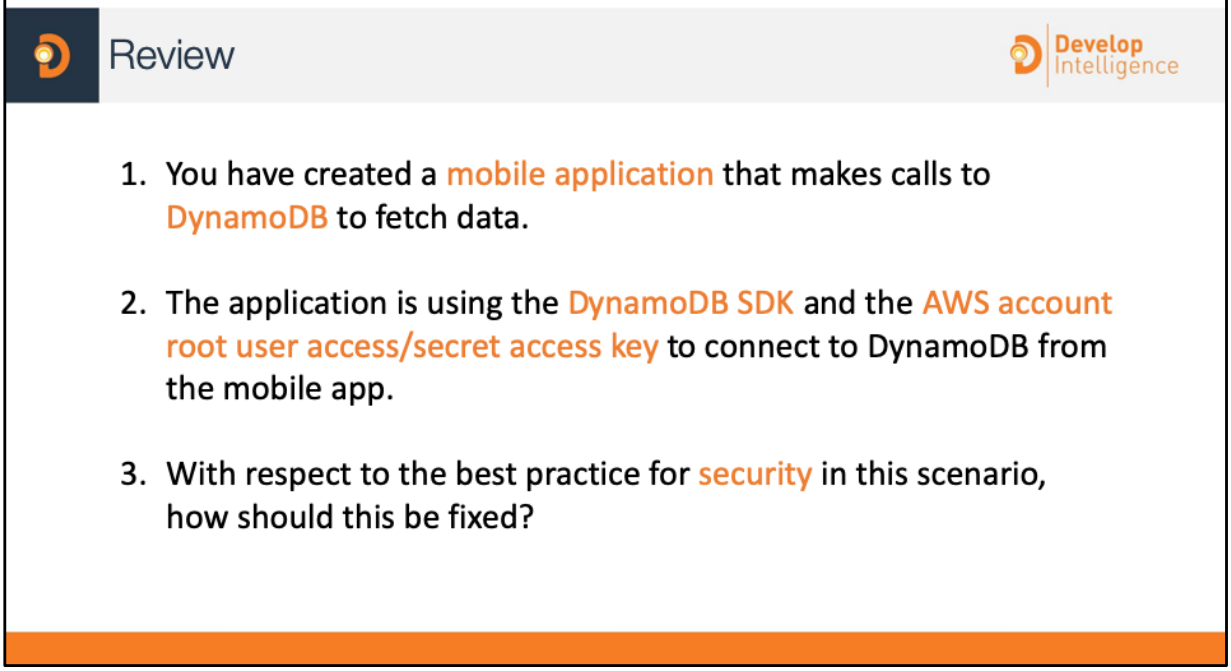
In this example, the root has an SCP attached to it. This policy will apply to all of the OUs and accounts in the organization. An SCP can be applied to one or more OUs or individual accounts.

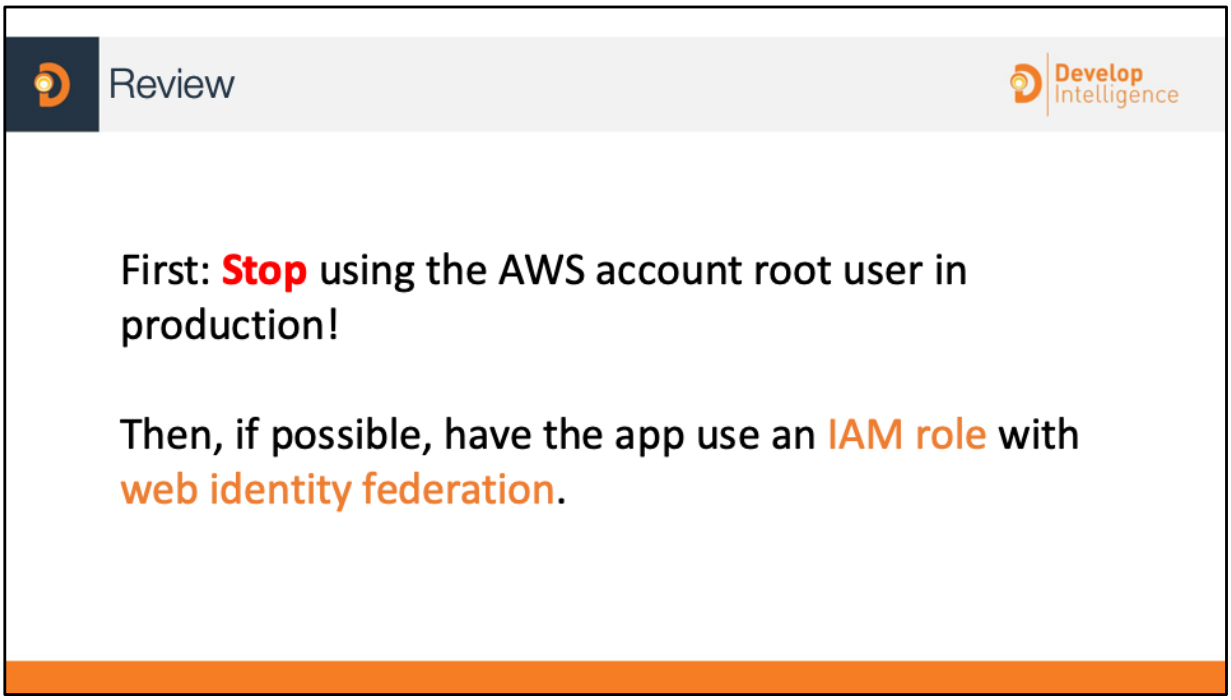

Review
Develop Intelligence
If you need to grant temporary permissions to a resource, what would you use?


Review
Develop Intelligence
If you need to grant temporary permissions to a resource, what would you use?
IAM role


Review
Develop Intelligence
One of your users can't access an S3 bucket. What should you check to identify the cause of the problem?


Review
Develop Intelligence
One of your users can't access an S3 bucket. What should you check to identify the cause of the problem?
The policies attached to the user and to the bucket


Review
Develop Intelligence
1. You have created a mobile application that makes calls to DynamoDB to fetch data.
2. The application is using the DynamoDB SDK and the AWS account root user access/secret access key to connect to DynamoDB from the mobile app.
3. With respect to the best practice for security in this scenario, how should this be fixed?


Review
Develop Intelligence
First: Stop using the AWS account root user in production!
Then, if possible, have the app use an IAM role with web identity federation.