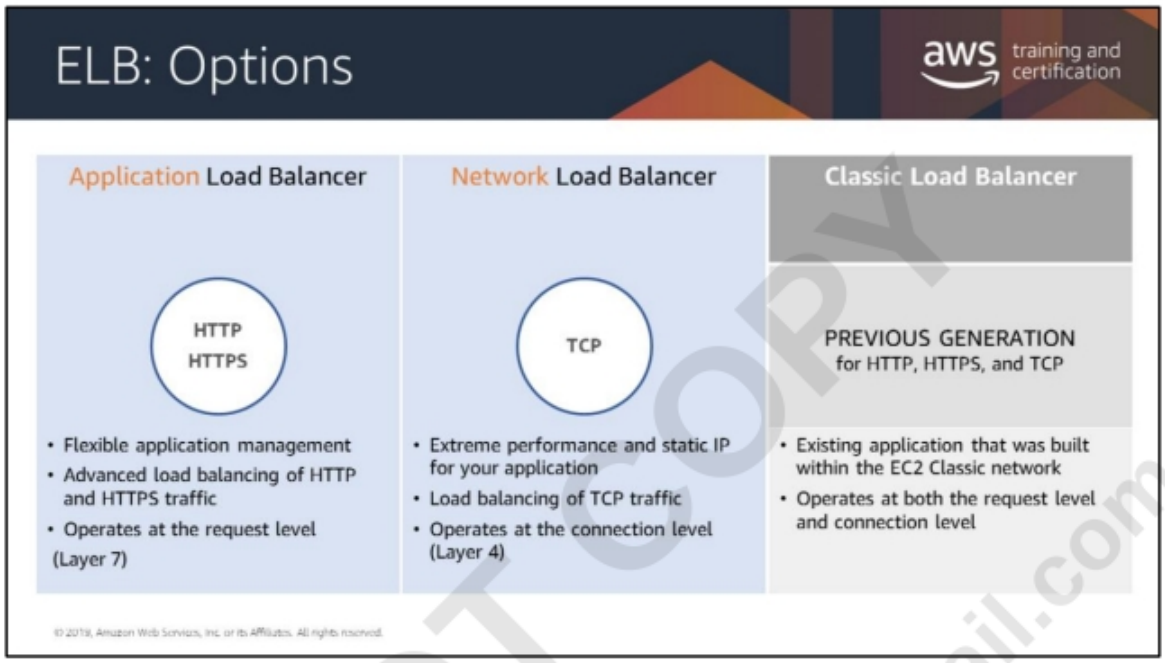
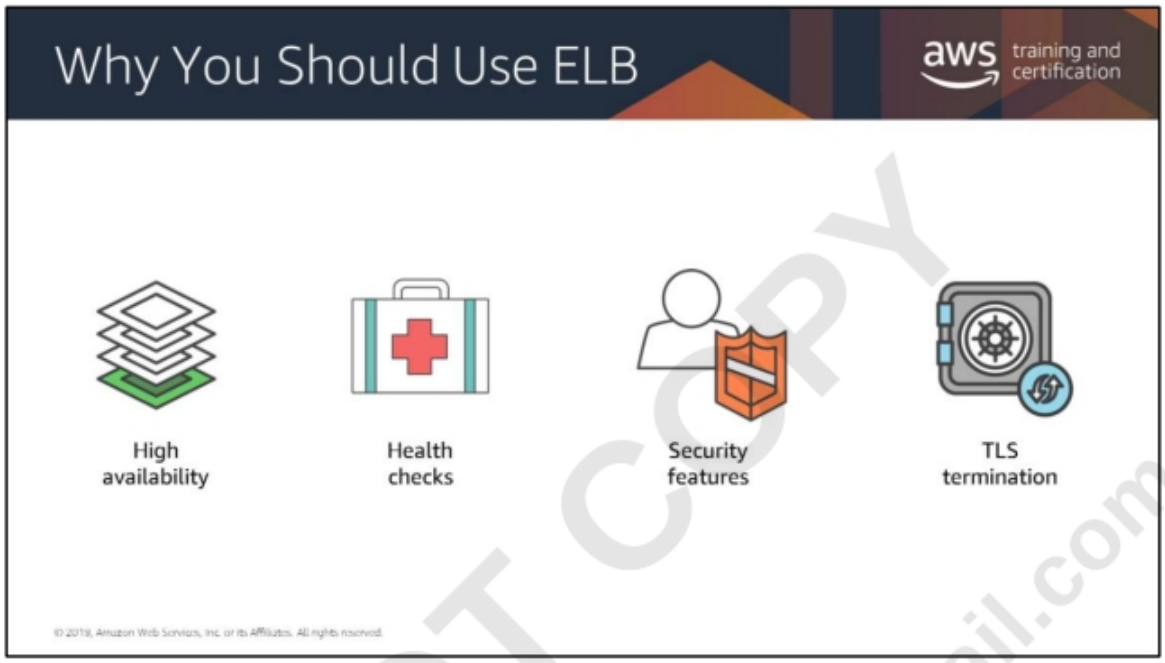




High availability
ELB automatically distributes traffic across multiple targets—Amazon EC2 instances, containers and IP addresses—in a single Availability Zone or multiple Availability Zones.

Health checks
To discover the availability of your Amazon EC2 instances, the load balancer periodically sends pings, attempts connections, or sends requests to test the Amazon EC2 instances. These tests are called *health checks*. Each registered Amazon EC2 instance must respond to the target of the health check with an HTTP status code 200 to be considered healthy by your load balancer.

Security features
ELB load balancers provisioned within an Amazon VPC can leverage its network security features, such as security groups.

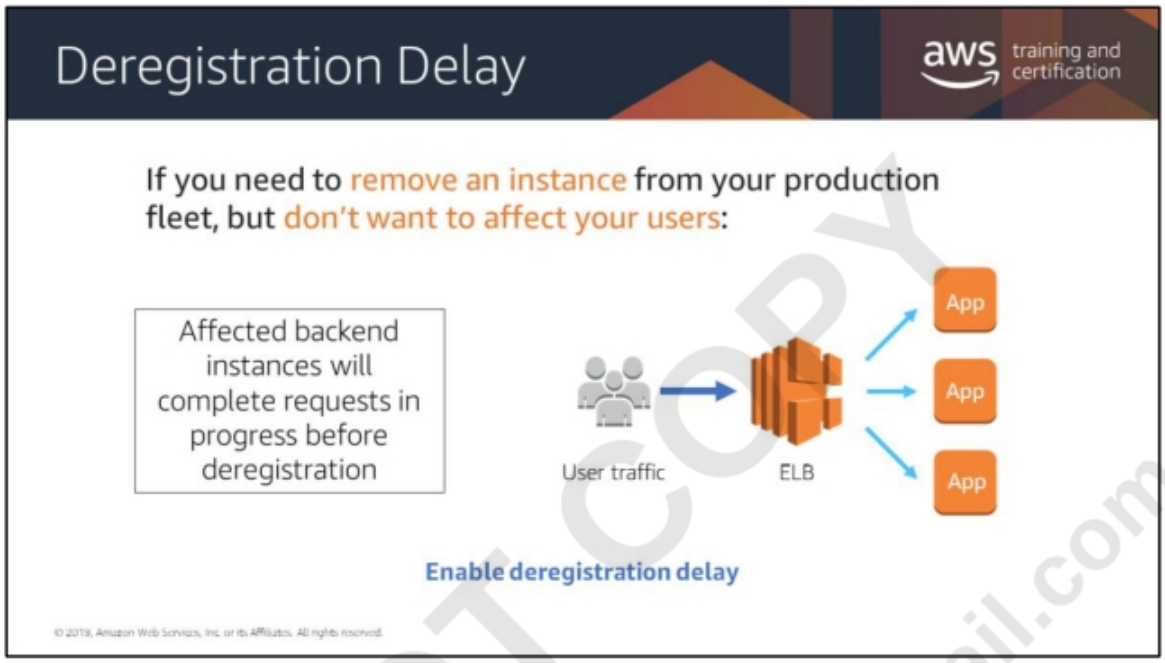
Transport Layer Security Termination
ELB provides integrated certificate management and SSL decryption, allowing you the flexibility to centrally manage the SSL settings of the load balancer and offload CPU-intensive work from your application.

Layer 4 or Layer 7 load balancing

You can balance HTTP/HTTPS applications for Layer 7-specific features, or use strict Layer 4 load balancing for applications that rely purely on the TCP protocol.

For a side-by-side feature comparison, see https://aws.amazon.com/elasticloadbalancing/details/#compare.

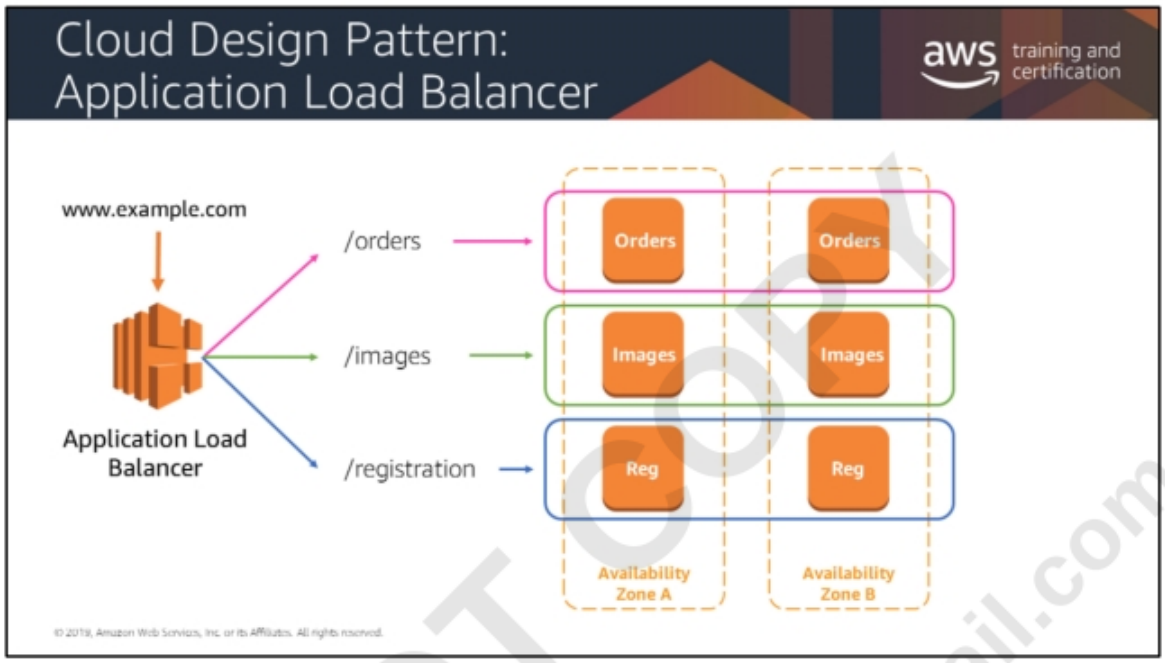
/



When you enable deregistration delay on a load balancer, any backend instances that you deregister will complete requests that are in progress before deregistration. Likewise, if a backend instance fails health checks, the load balancer will not send any new requests to the unhealthy instance. It will allow existing requests to be completed, while ensuring that in-flight requests continue to be served. That means you can perform maintenance like deploying software upgrades or replacing back-end instances without affecting your customers' experience.

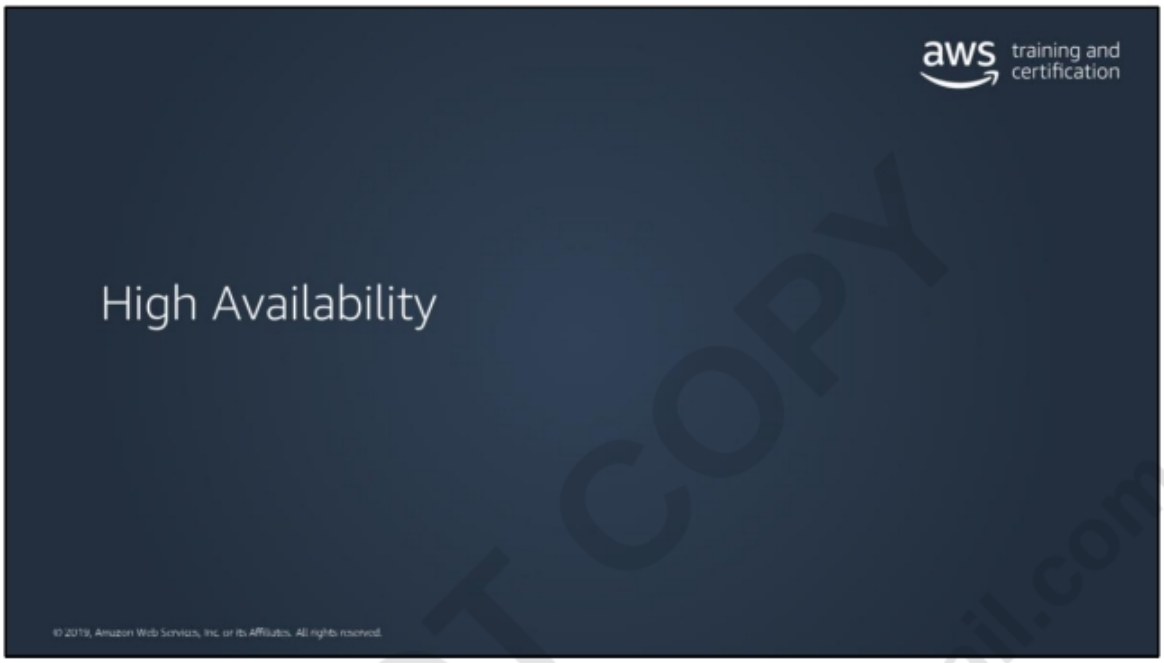
Deregistration delay is also integrated with Auto Scaling, which makes it even easier to manage the capacity behind your load balancer. When deregistration delay is enabled, Auto Scaling will wait for outstanding requests to be completed before terminating instances.

For more information, see:
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html#deregistration-delay (Click deregistration delay under contents.)



For more information, see https://aws.amazon.com/blogs/devops/introducing-application-load-balancer-unlocking-and-optimizing-architectures/

Application Load Balancers now Support Advanced Request Routing.
See https://aws.amazon.com/about-aws/whats-new/2019/03/application-load-balancers-now-support-advanced-request-routing/

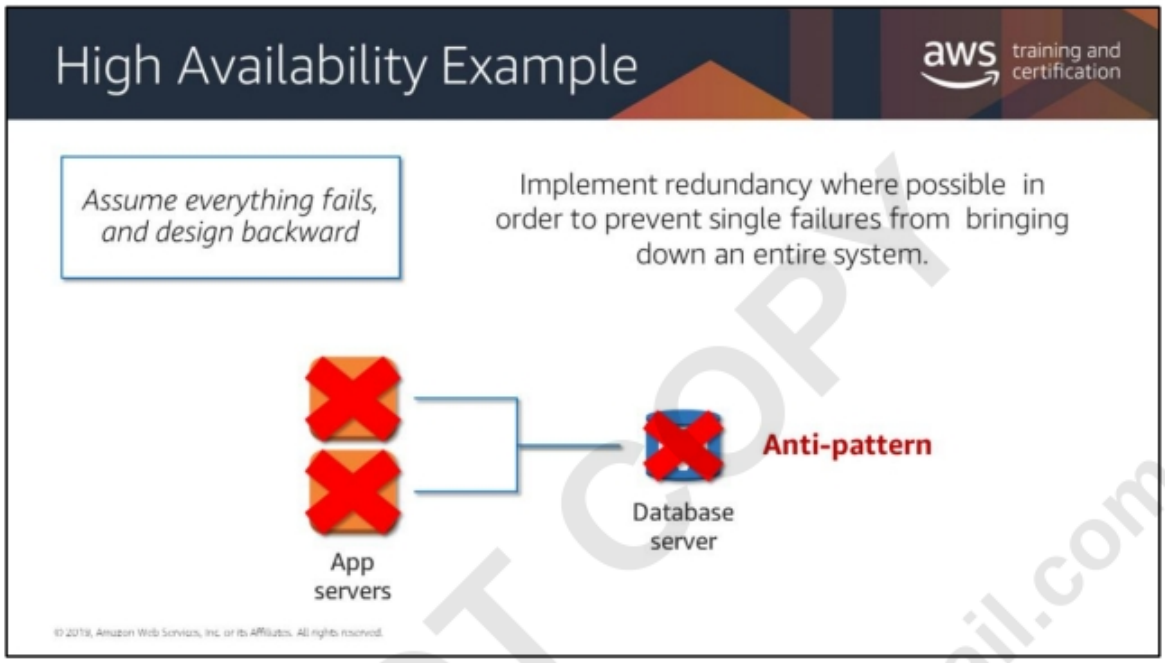


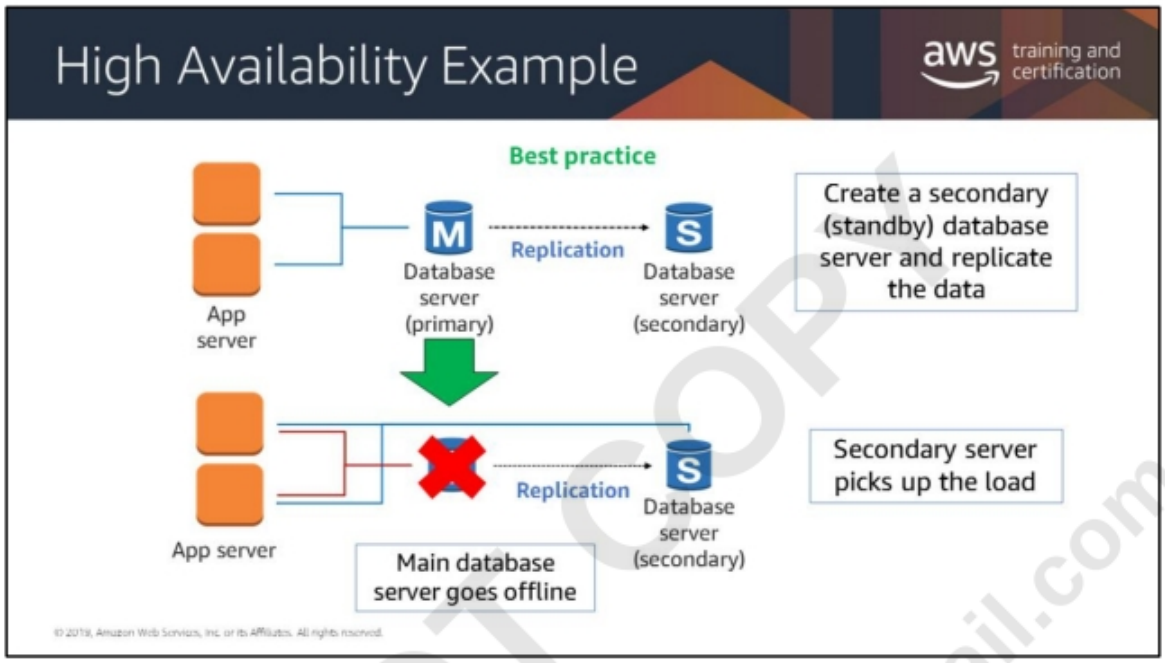
What is High Availability?

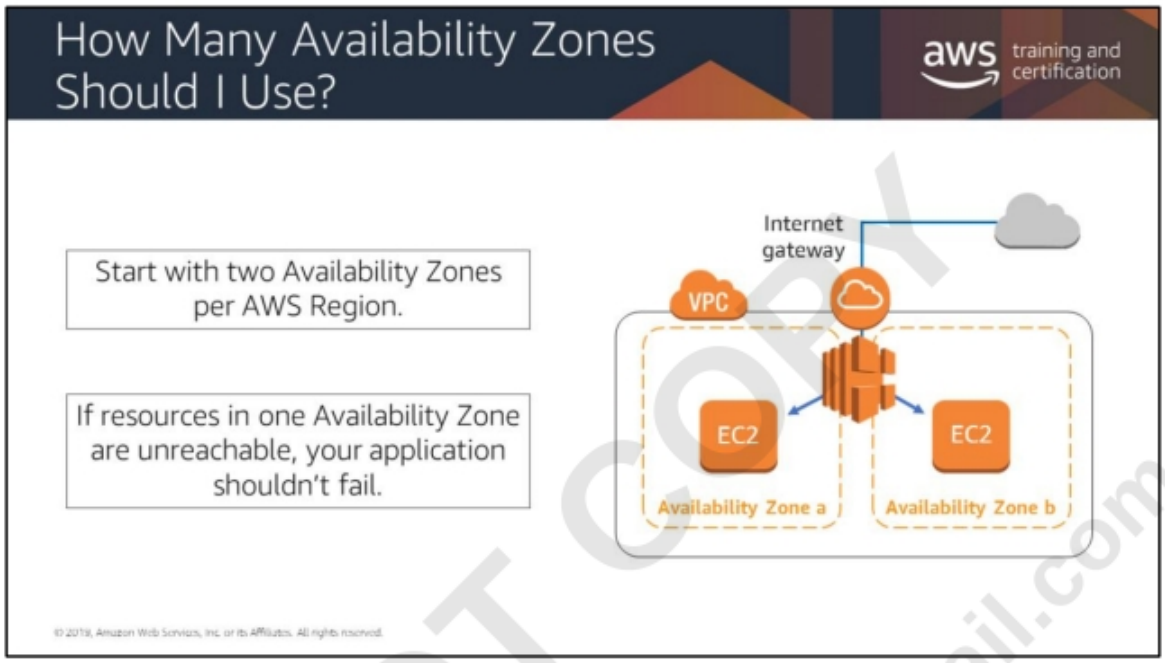
aws training and certification

Your application can recover from a failure or roll over to a secondary source within an acceptable amount of degraded performance time.

Percent of Uptime	Max Downtime per Year	Equivalent Downtime per Day
90%	36.5 days	2.4 hrs
99%	3.65 days	14 min
99.9%	8.76 hrs	86 sec
99.99%	52.6 min	8.6 sec
99.999%	5.25 min	.86 sec





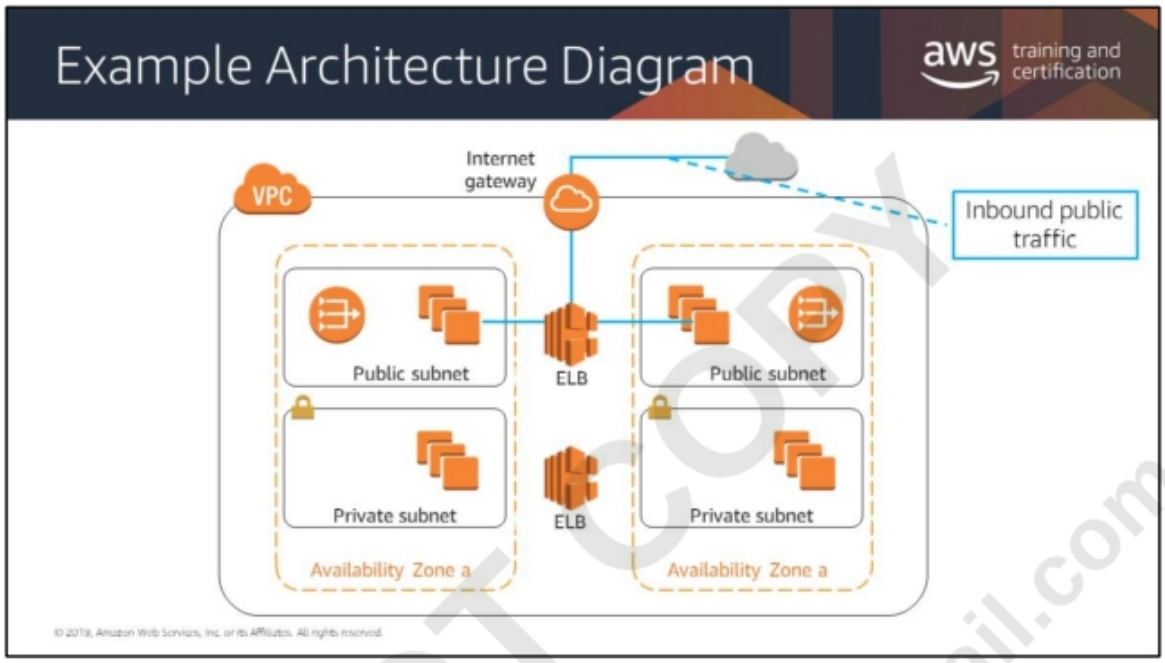


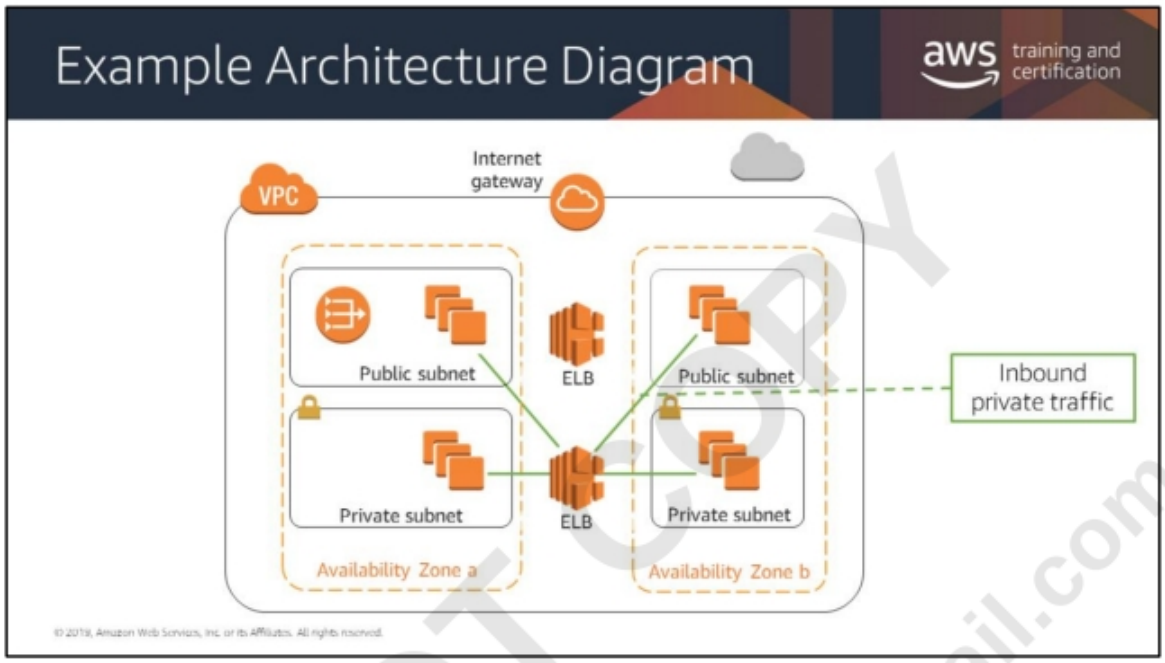
Most applications can be designed to support two Availability Zones. They may not benefit from the use of more Availability Zones than that, if you use data sources that only support primary/secondary failover. Availability Zones are spread out physically, so you won't receive much benefit from duplicating your resources in three or more Availability Zones in one AWS Region.

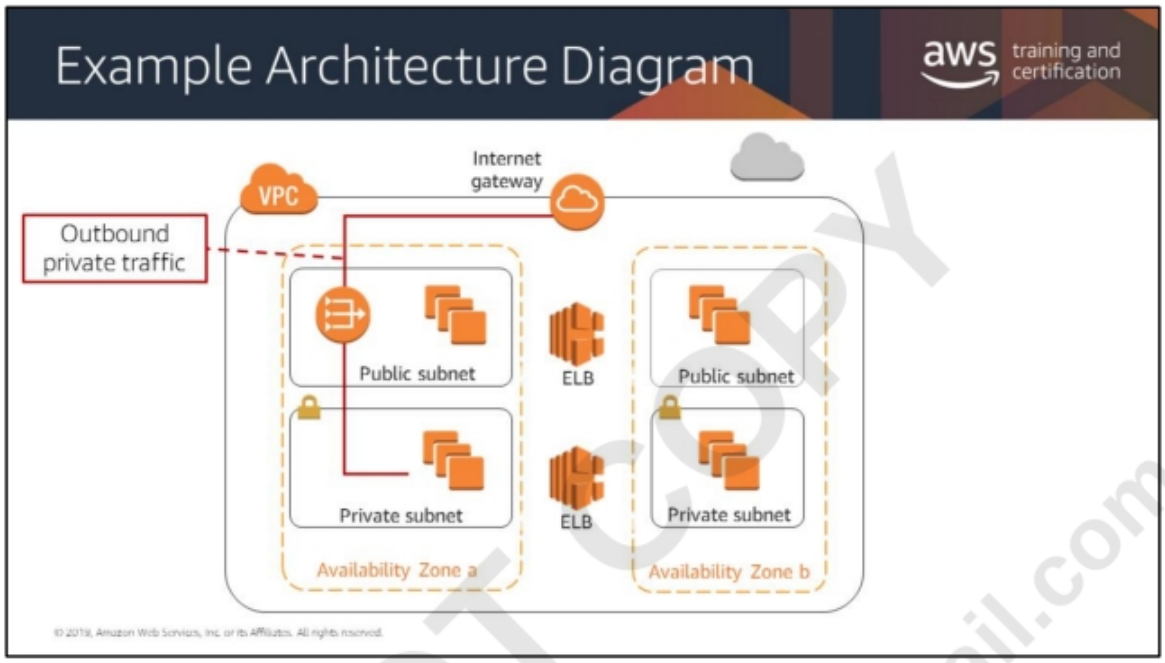
For heavy Amazon EC2 Spot Instance usage or data sources that go beyond active/passive, such as Amazon DynamoDB, there may be a benefit to using more than two Availability Zones.

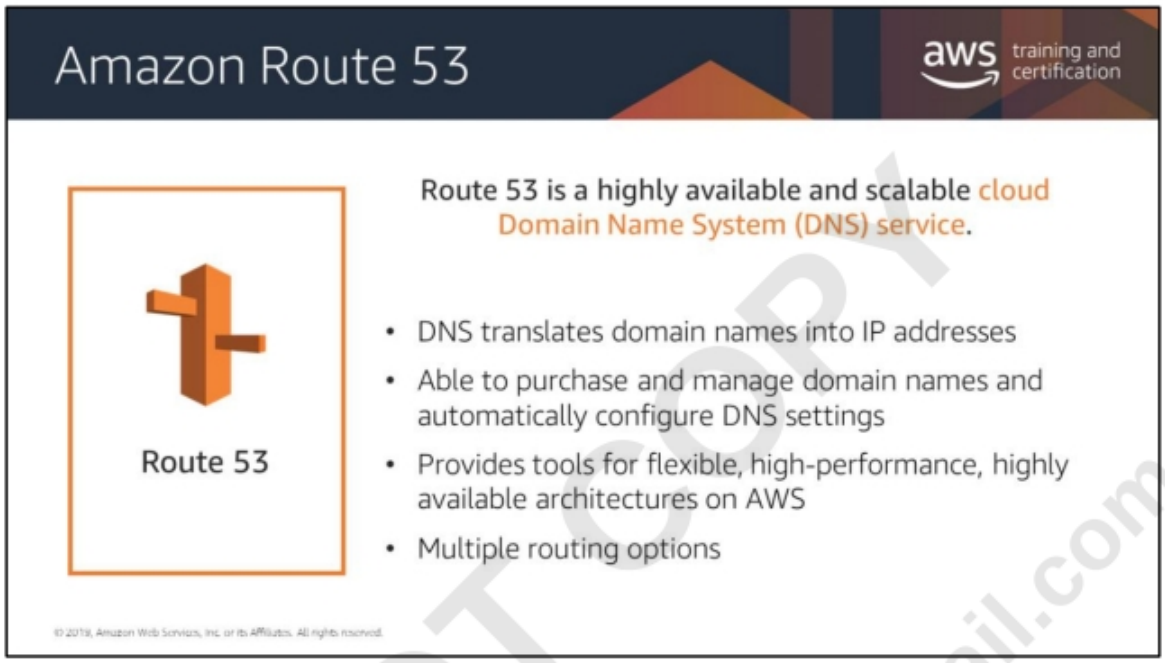
In this basic pattern, the two web servers (Amazon EC2) are positioned behind an ELB load balancer, which distributes traffic between them. If one of the servers becomes unavailable, the load balancer recognizes this. It stops distributing traffic to the unhealthy instance. That way, if there's a problem in one of the Availability Zones where a component resides, your application is still available.

You can further increase the availability of your infrastructure using other methods, which will be covered in a later module.

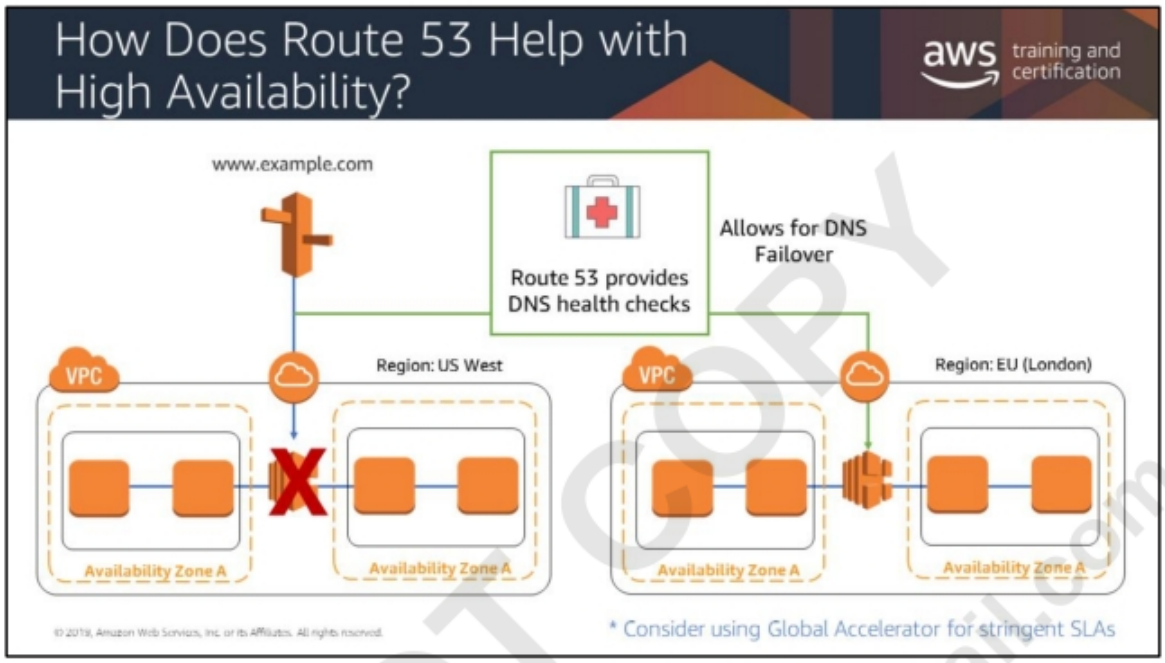








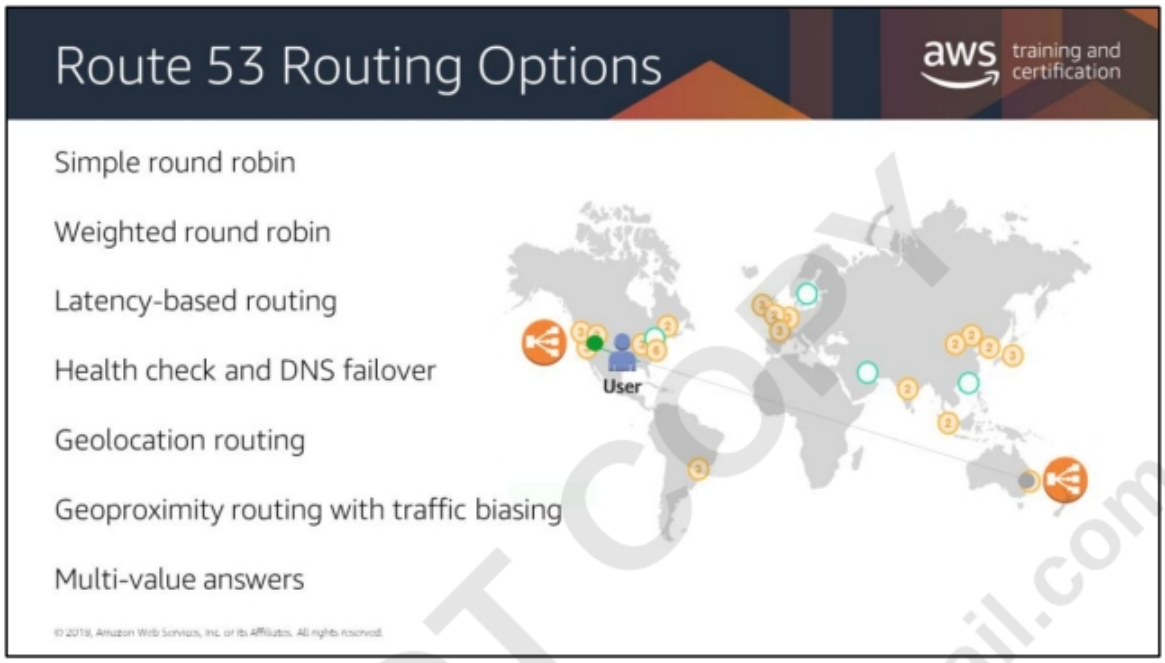
Amazon Route 53 provides a Domain Name System (DNS), domain name registration, and health-checking web services. The service was designed to give developers and businesses a reliable and cost-effective way to route end users to internet applications by translating names like *example.com* into the numeric IP addresses, such as *192.0.2.1*, that computers use to connect to each other. You can combine your DNS with health-checking services to route traffic to healthy endpoints or to independently monitor and/or alarm on endpoints. You can also purchase and manage domain names such as *example.com* and automatically configure DNS settings for your domains. Route 53 effectively connects user requests to infrastructure running in AWS—such as Amazon EC2 instances, ELB load balancers, or Amazon S3 buckets—and can also be used to route users to infrastructure outside of AWS.



If you have stringent SLAs or your application demands the fastest possible failover; please consider adding global accelerator into your architecture.

Global Accelerator provides your network with greater resiliency by removing the role of DNS in failover. It can protect your users and applications from caching issues and allows nearly instantaneous redirection of traffic to healthy endpoints. Additionally, when you add new endpoints into your architecture, they can receive traffic immediately without waiting for DNS propagation.

Global accelerator uses Static IP addresses that are Anycast from multiple AWS edge locations giving us a fixed entry point address that enables ingress of user traffic at an edge location closest to them



Simple routing (round robin) distributes the number of of requests as evenly as possible between all participating servers

Weighted round robin allows you to assign weights to resource record sets in order to specify the frequency with which different responses are served. You may want to use this capability to do A/B testing, sending a small portion of traffic to a server on which you've made a software change. For instance, suppose you have two record sets associated with one DNS name: one with weight 3 and one with weight 1. In this case, 75 percent of the time, Amazon Route 53 will return the record set with weight 3. Twenty-five percent of the time, Amazon Route 53 will return the record set with weight 1. Weights can be any number between 0 and 255.

Latency-based routing (LBR) helps you improve your application's performance for a global audience. LBR works by routing your customers to the AWS endpoint (e.g., Amazon EC2 instances, Elastic IP addresses, or load balancers) that provides the fastest experience based on actual performance measurements of the different AWS regions where your application is running.

Amazon Route 53 health checks monitor the health and performance of your web applications, web servers, and other resources. Each health check that you create can monitor one of the following:

- The health of a specified resource, such as a web server
- The status of other health checks
- The status of an Amazon CloudWatch alarm

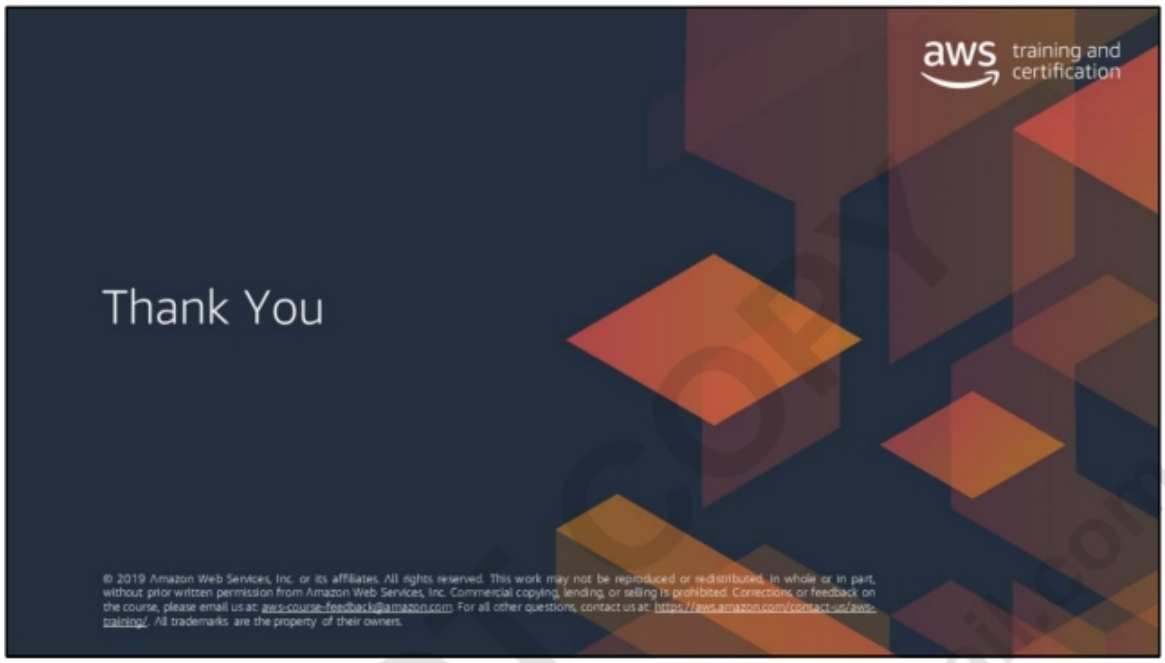
After you create a health check, you can get the status of the health check, get notifications when the status changes, and configure DNS failover.

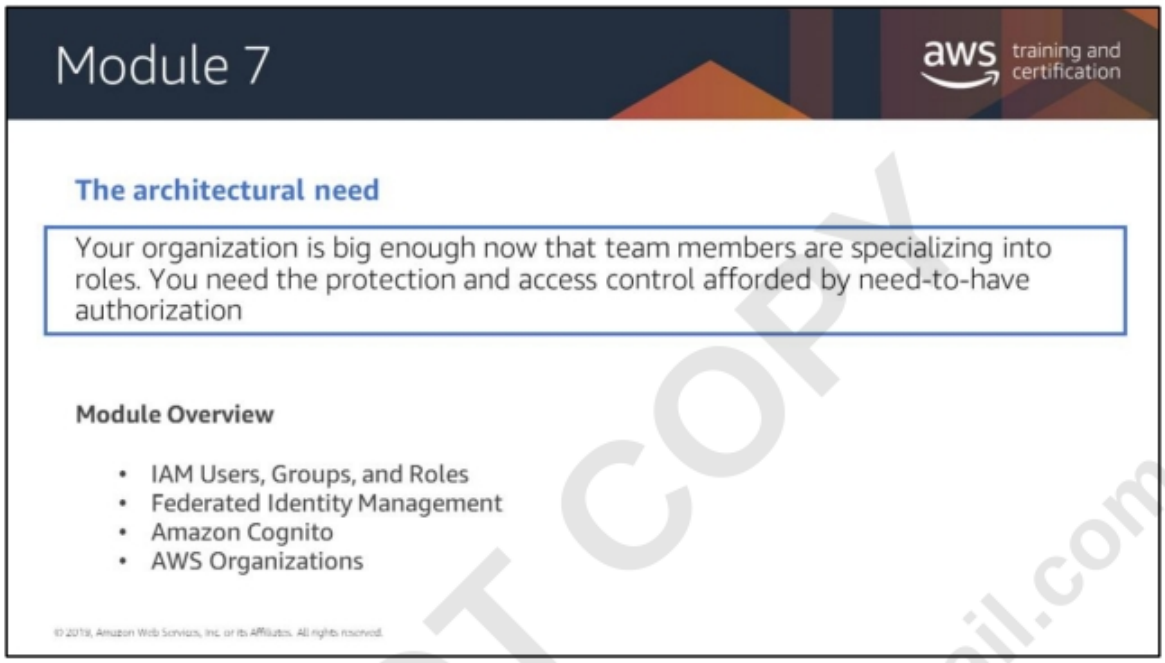
Geolocation routing lets you choose the resources that serve your traffic based on the geographic location of your users (the origin of DNS queries). When you use geolocation routing, you can localize your content and present some or all of your website in the language of your users. You can also use geolocation routing to restrict distribution of content to only the locations in which you have distribution rights. You can also balance load across endpoints in a predictable, easy-to-manage way, so each end-user location is consistently routed to the same endpoint.

With **DNS failover,** Amazon Route 53 can help detect an outage of your website and redirect your end users to alternate locations where your application is operating properly. When you enable this feature, Amazon Route 53 health-checking agents will monitor each location/endpoint of your application to determine its availability. You can take advantage of this feature to increase the availability of your customer-facing application.

Geoproximity routing lets you route traffic based on the physical distance between your users and your resources if you're using Route 53 traffic flow. You can also route more or less traffic to each resource by specifying a positive or negative bias. When you create a traffic flow policy, you can specify either an AWS Region (if you're using AWS resources) or the latitude and longitude for each endpoint.

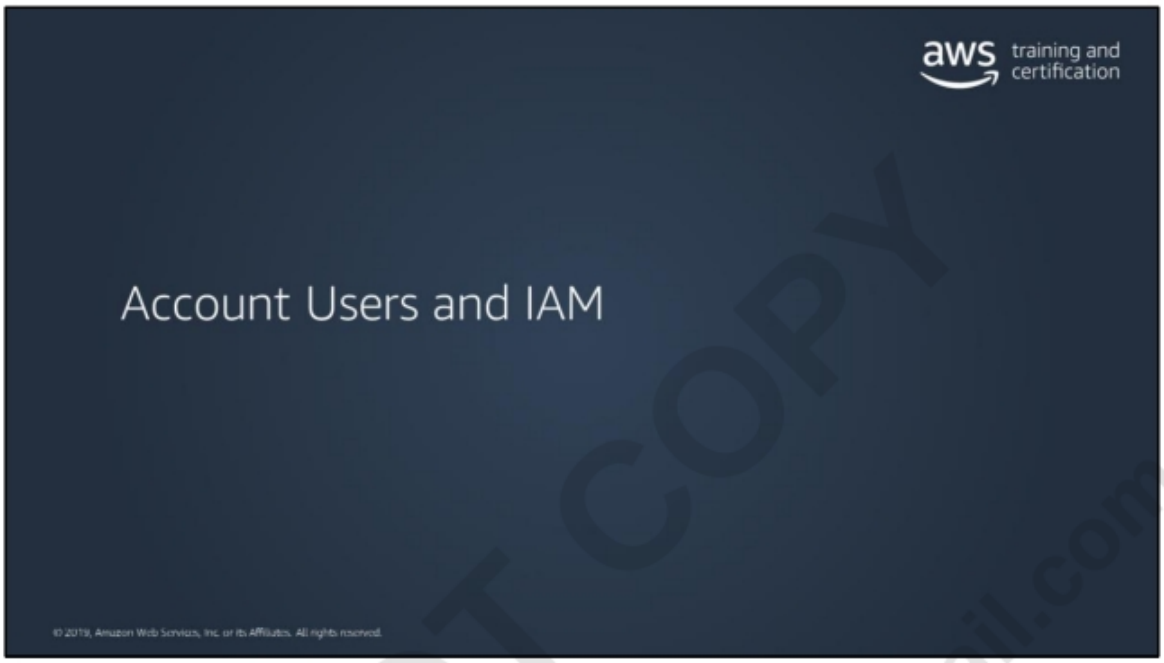
With **multi-value answers**, if you want to route traffic approximately randomly to multiple resources, such as web servers, you can create one multi-value answer record for each resource and, optionally, associate an Amazon Route 53 health check with each record. For example, suppose you manage an HTTP web service with 12 web servers that each have their own IP address. No one web server could handle all of the traffic, but if you create a dozen multi-value answer records, Amazon Route 53 responds to DNS queries with up to eight healthy records in response to each DNS query. Amazon Route 53 gives different answers to different DNS resolvers. If a web server becomes unavailable after a resolver caches a response, client software can try another IP address in the response.

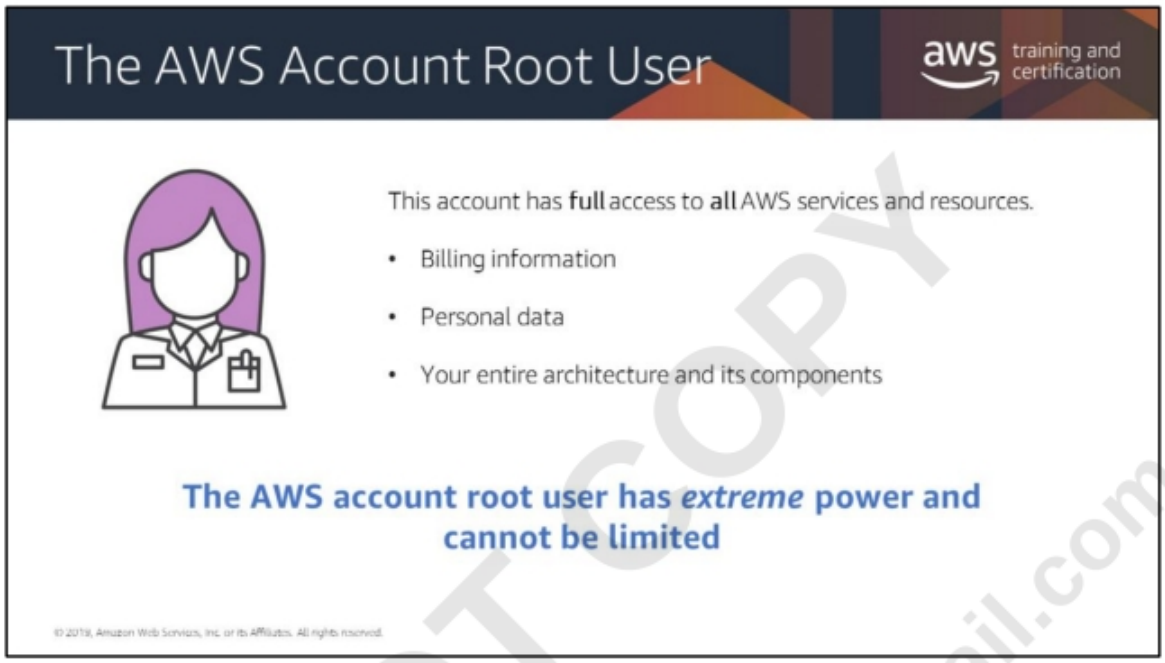




- Some users need access to the AWS Management Console, others only need to use AWS Command Line Interface (AWS CLI).
- Different environments (Dev/Test/Production) have different sets of access requirements.
- On-premises authentication is managed with federation with SSO.
- The security operations 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.

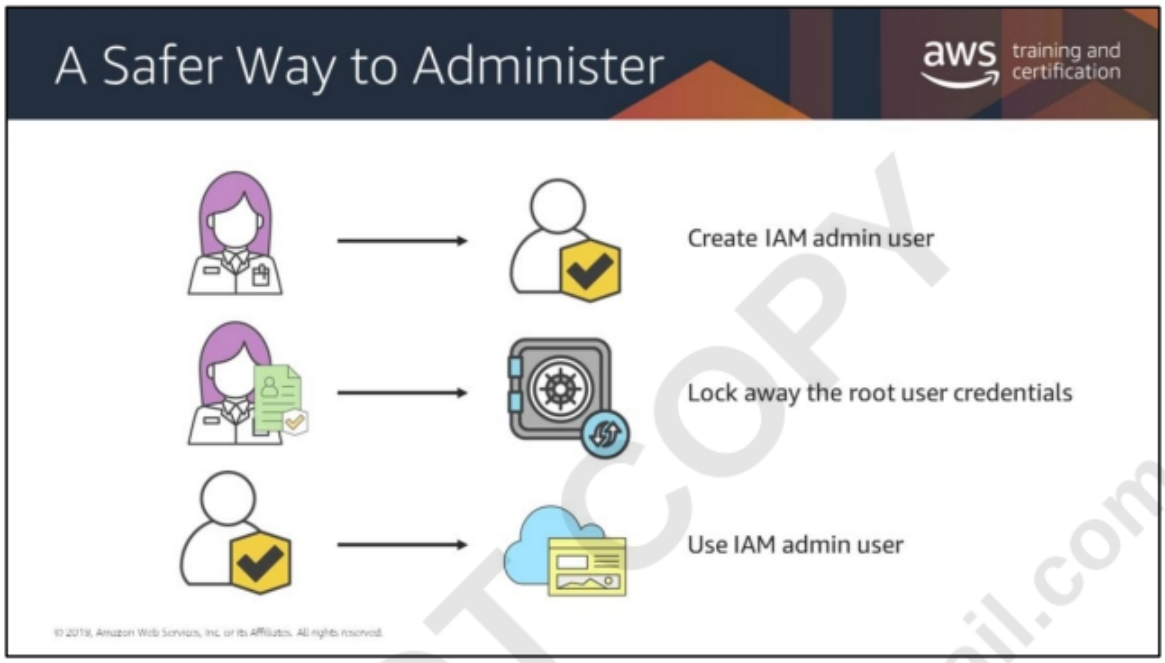
/





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.

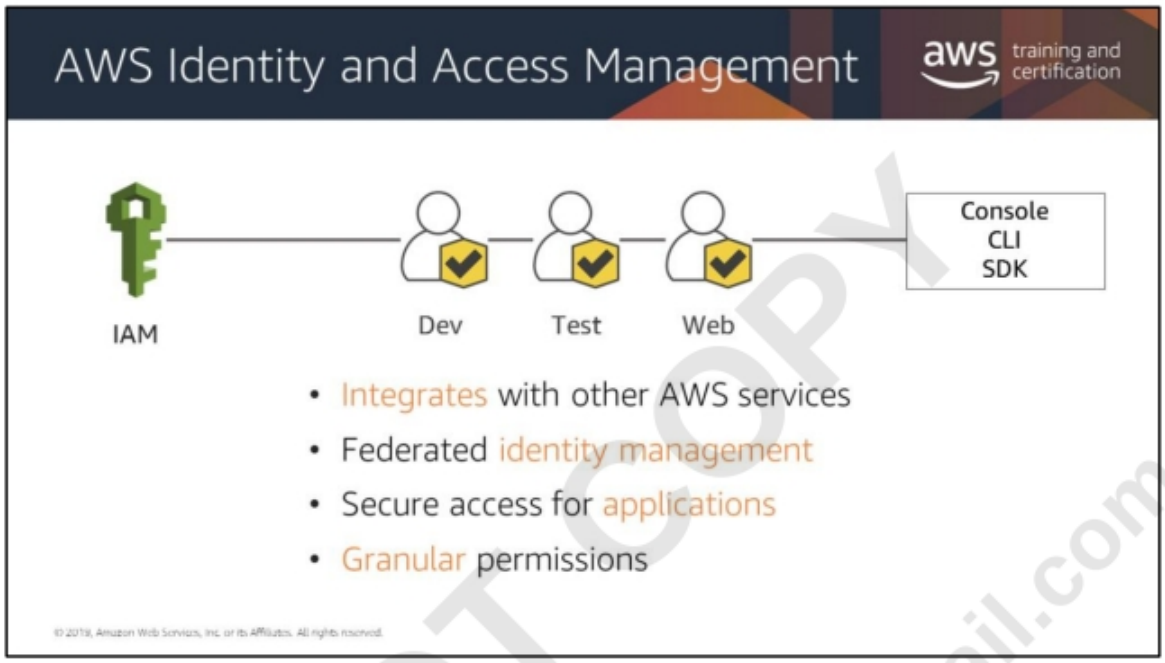
/

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.

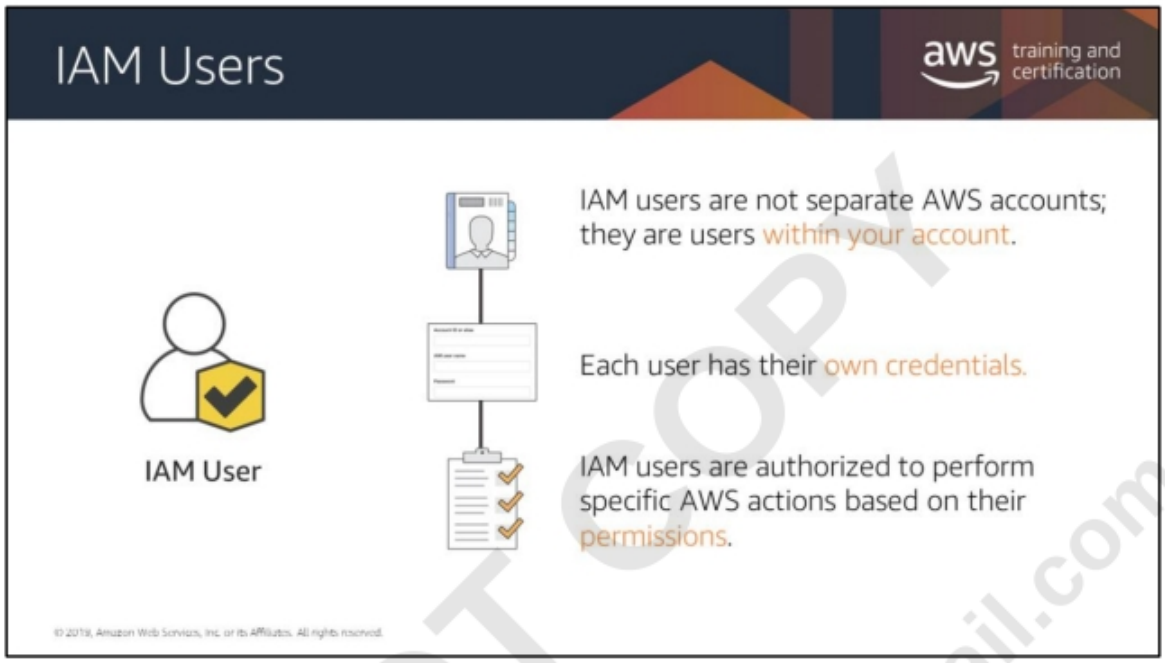
IAM can be accessed from the console, the AWS CLI, the AWS SDK, and a secure API endpoint.

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

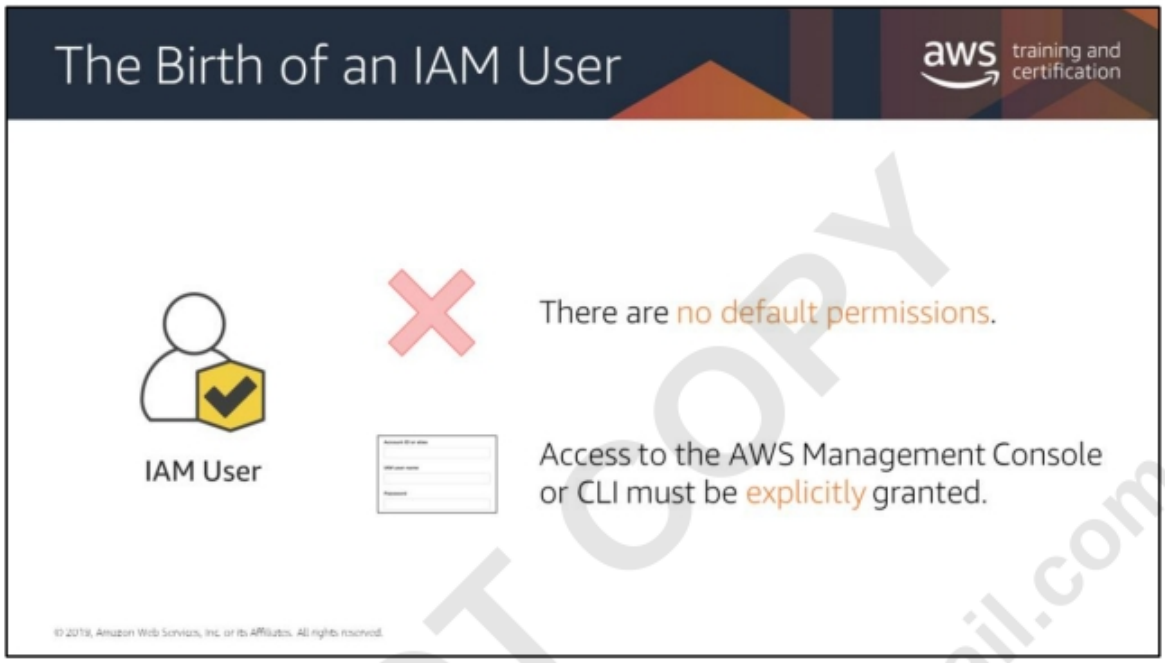


IAM users are not separate accounts; they are principals within your account. Each IAM user can have its own password for access to the console. You can also create an individual access key for each user so that the user can make programmatic requests to work with resources in your account. It is possible 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.

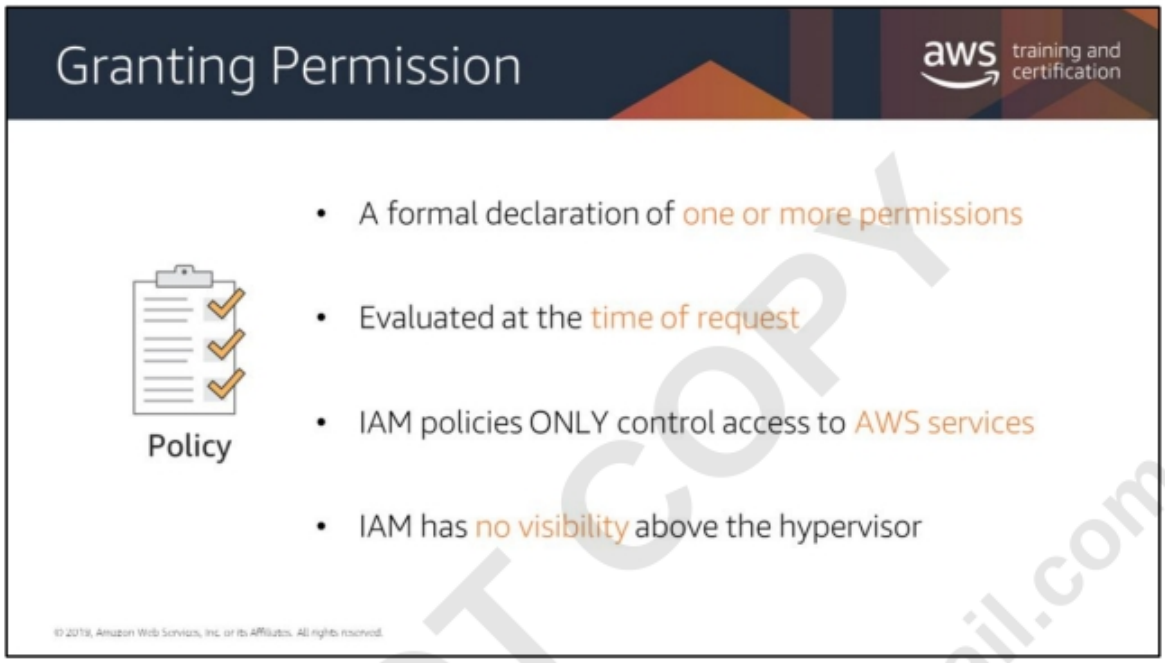
Newly created IAM users have no default credentials to use to authenticate themselves and access AWS resources. You first need to assign security credentials to them for authentication and then attach permissions that authorize them to perform any AWS actions or to access any AWS resources. The credentials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

/



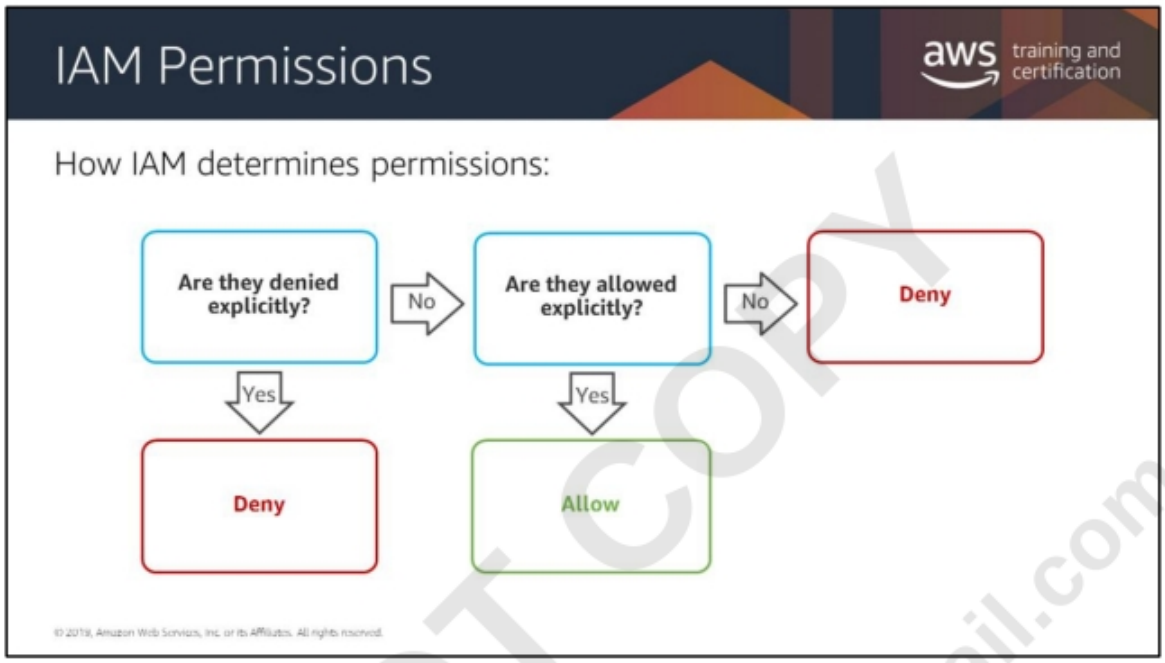
There are no default permissions for IAM principals. AWS does not recommend giving everyone admin rights. We urge you to follow the least-privilege principle.



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.

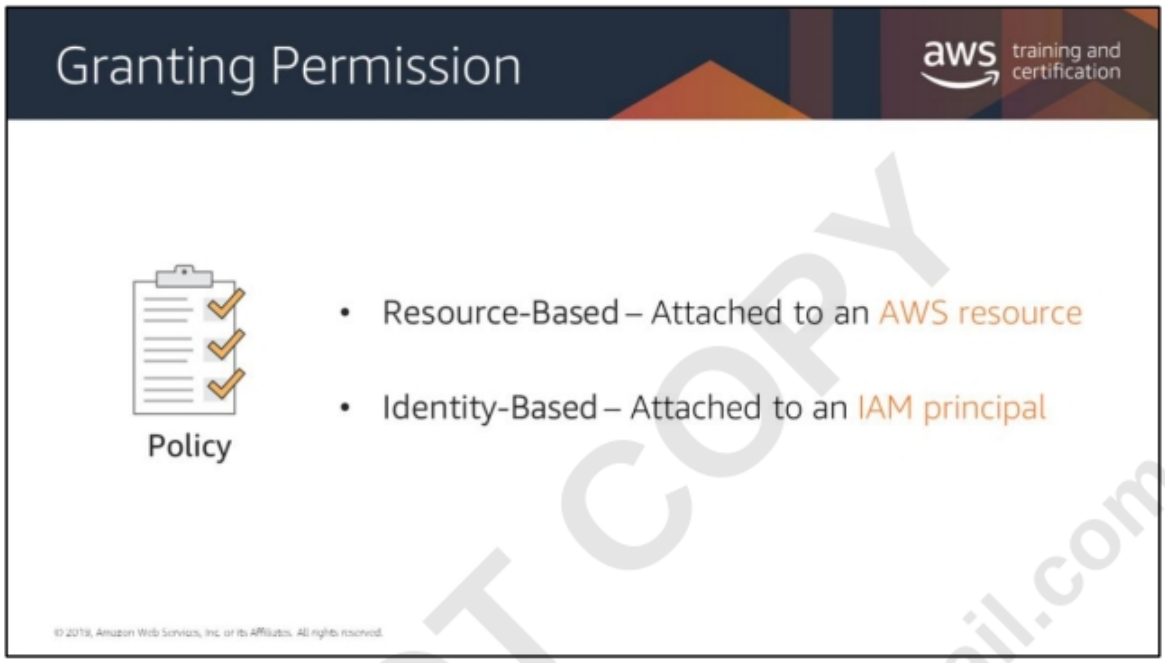
If you want OS support, use LDAP, SAML, or other identity management system.



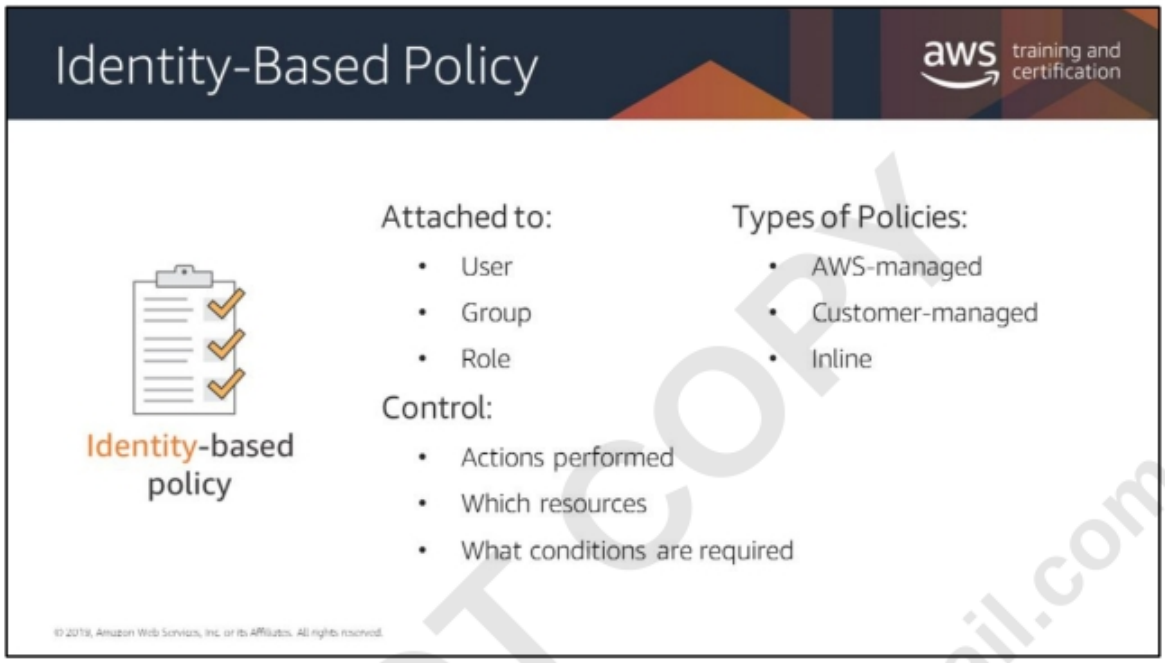
Policies give you the opportunity to fine-tune privileges granted to IAM users, groups, and roles. Because policies are stored in JSON format, they can be used in conjunction with a version control system. It's a good idea to define least-privilege access to each user, group, or role. Then, you can customize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



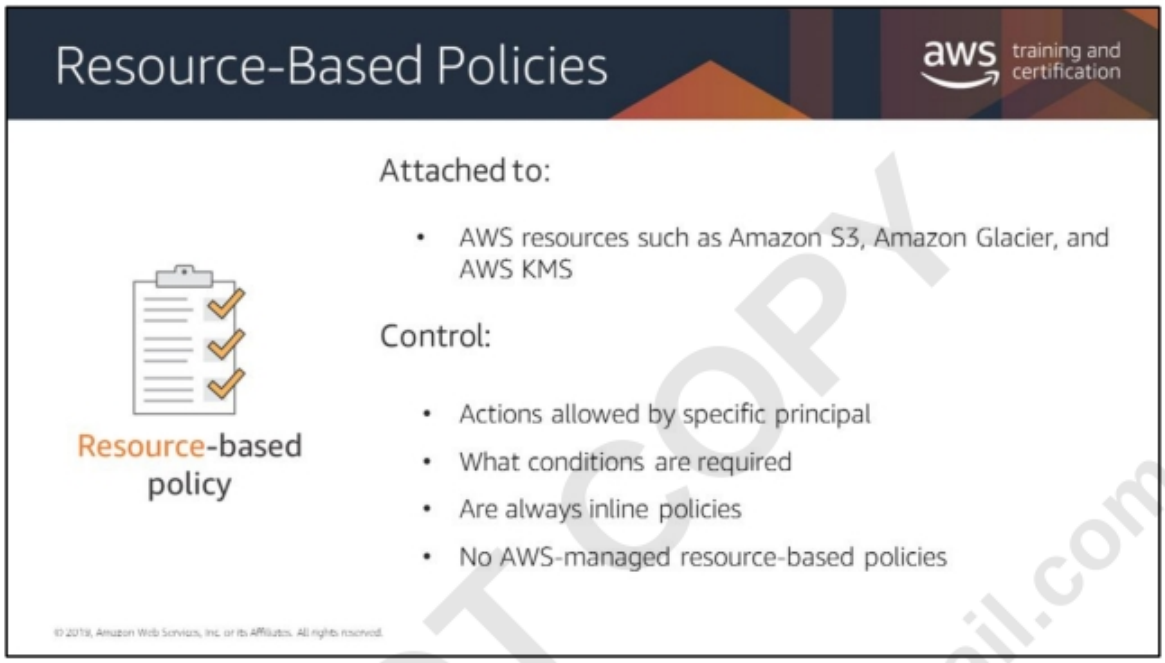
Identity-Based Policies
Identity-based policies are permission policies that you can attach to a principal (or identity), such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions.

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.

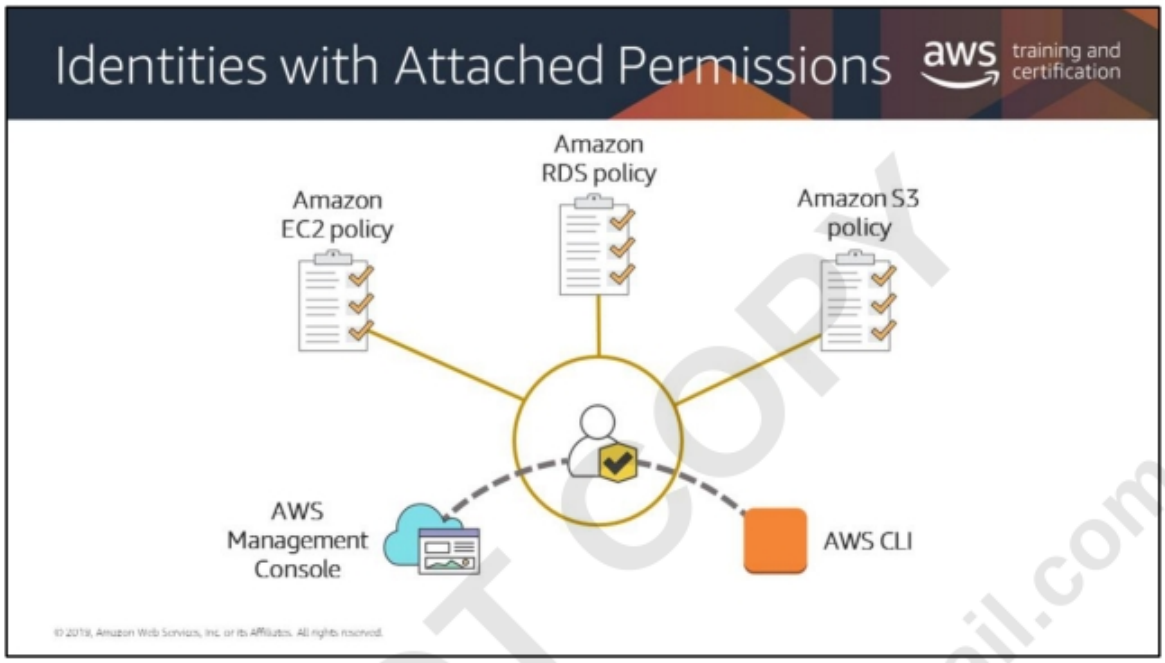


Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket.

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.

/

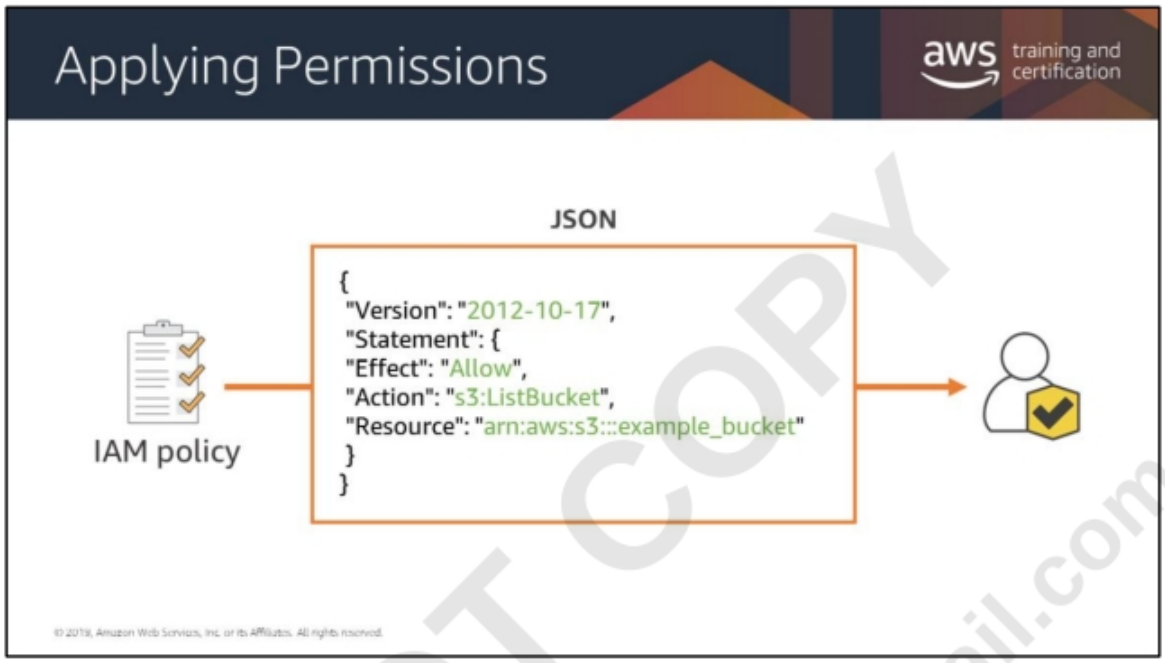


An IAM user is really just an identity (principal) with associated permissions.

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

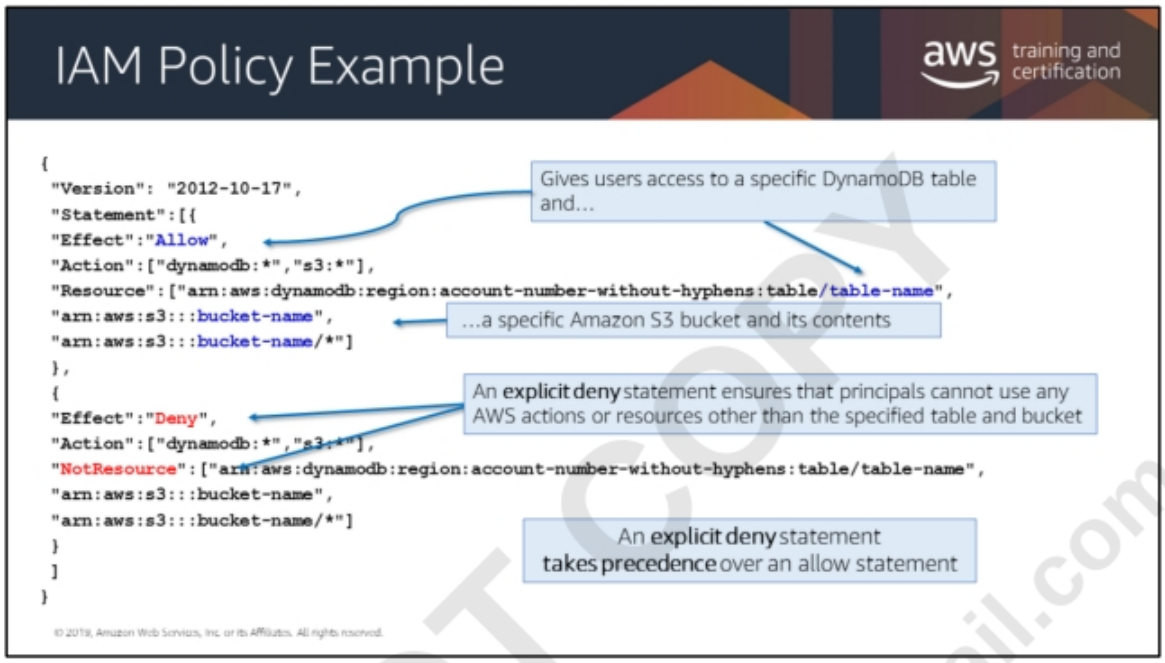
An application might have its own identity in your account and its own set of permissions to access AWS services. This is similar to how, in modern operating systems, processes have their own identities and permissions. If an application or even an EC2 instance has permissions to access something like 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.

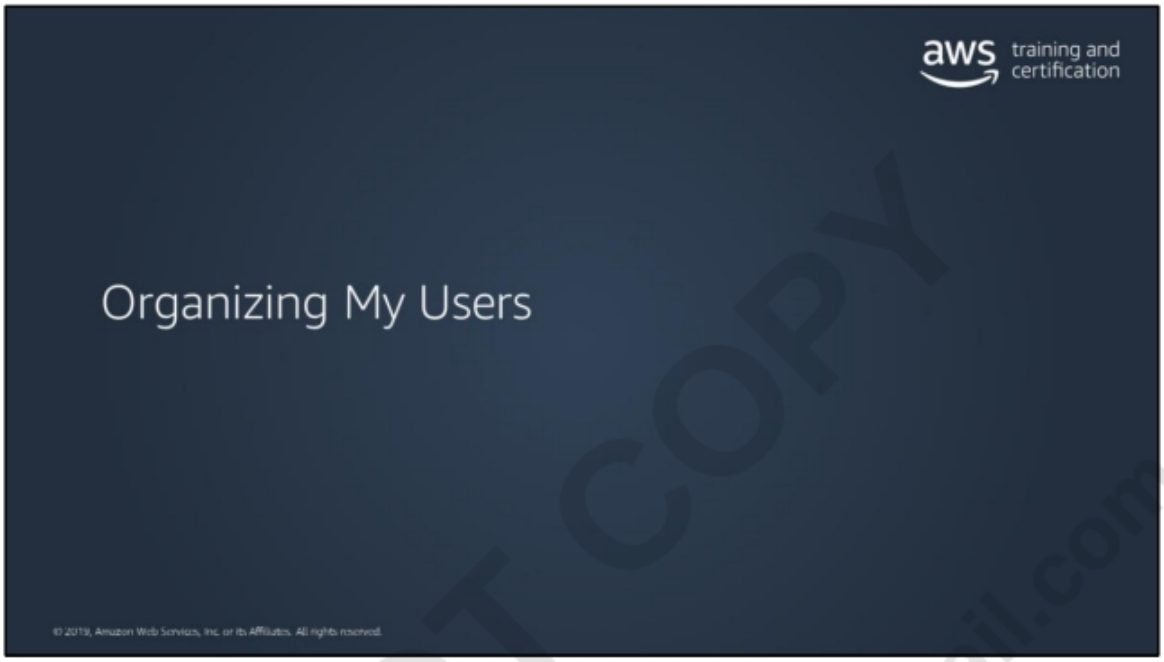
In the Deny section, NotResource can be confusing if you've never seen it before. NotResource is an advanced policy element that explicitly matches everything except the specified list of resources.

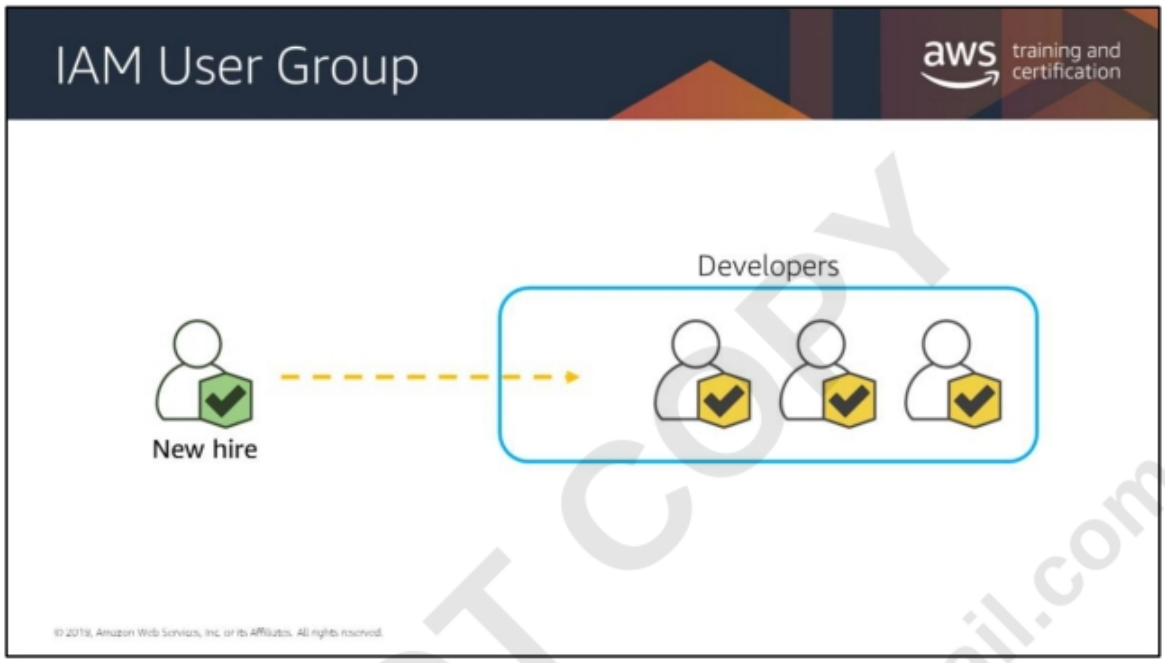
Here, NotResource means explicitly deny *everything other than* what's listed here. If you were to make a change to the top half of the policy, you'd have to make a corresponding change to the deny statement.

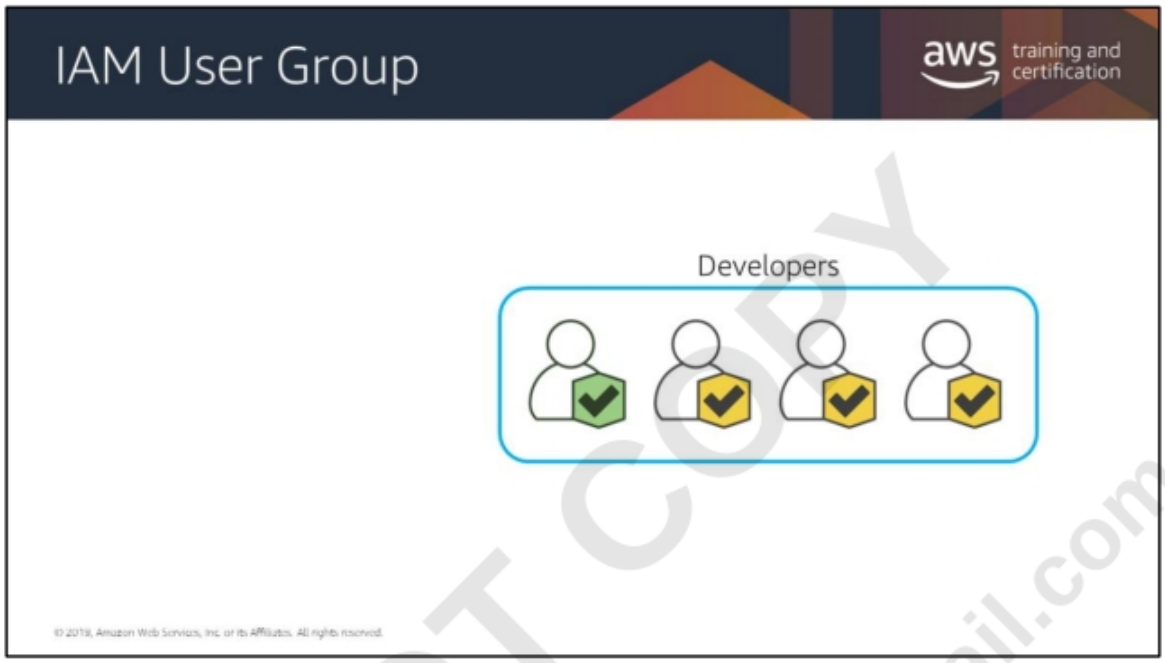
Why do this if there's an implicit deny? For some, it's important to lock down access to prevent inadvertent the granting of permission. (It's easy to assume malicious intent, but it may also be a user who has good intentions but poor understanding.) Be aware that this creates added complexity. That's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

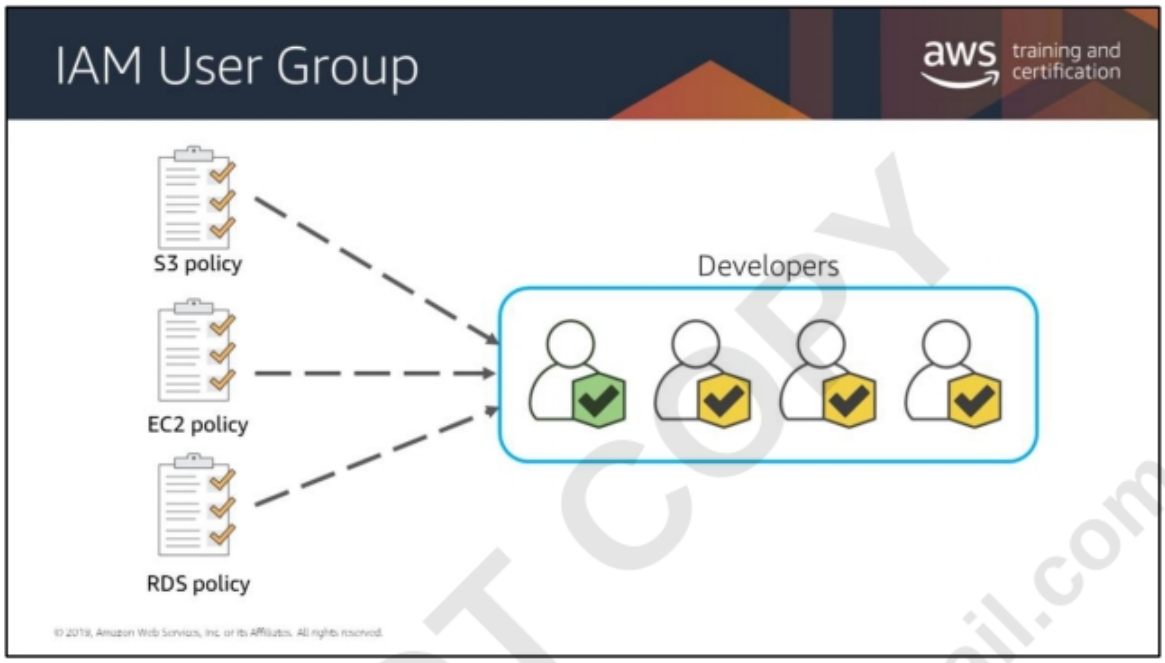
Using NotResource can result in a shorter policy by listing only a few resources that should not match, rather than including a long list of resources that will match. When using NotResource, keep in mind that resources specified in this element are the *only* resources that are limited.

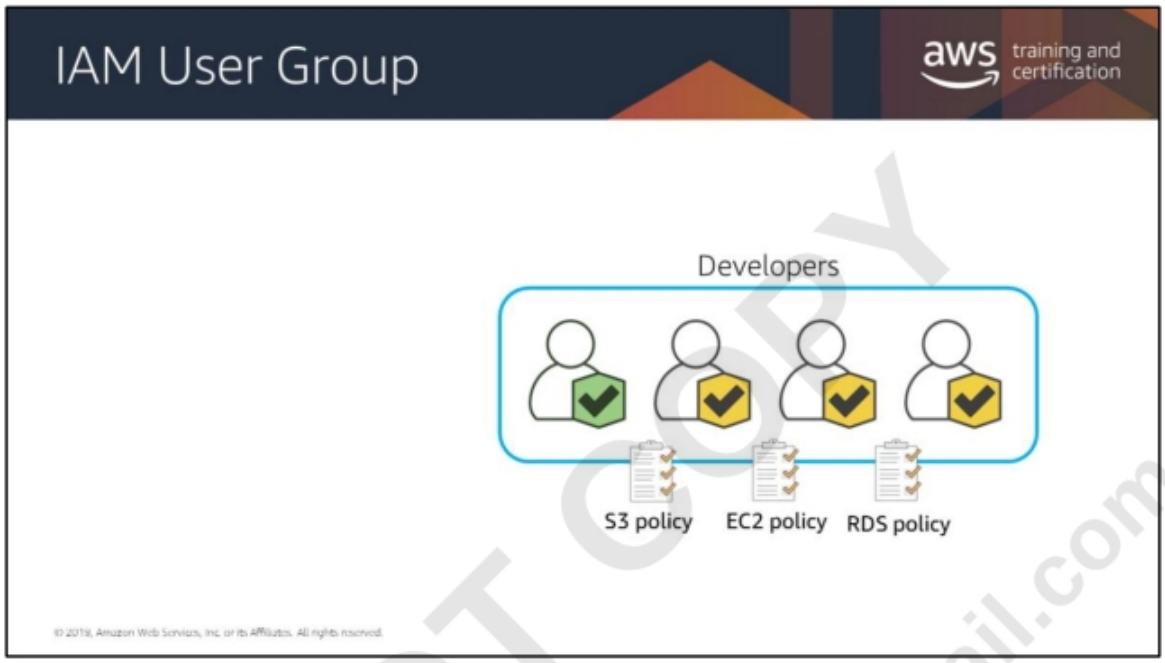
/

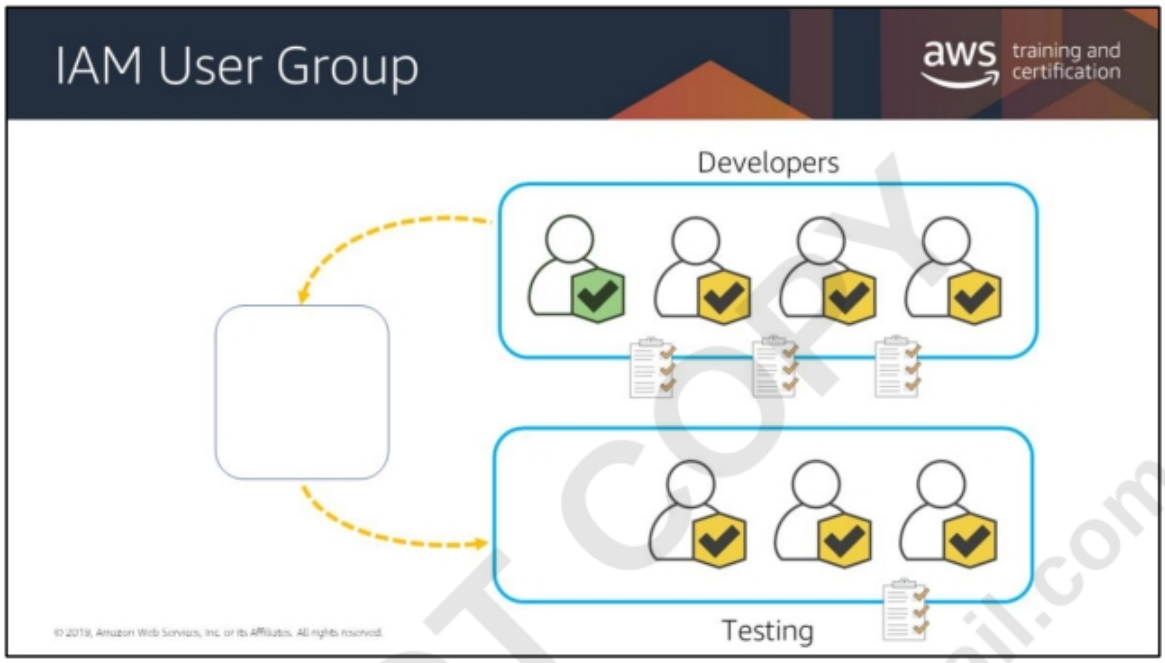


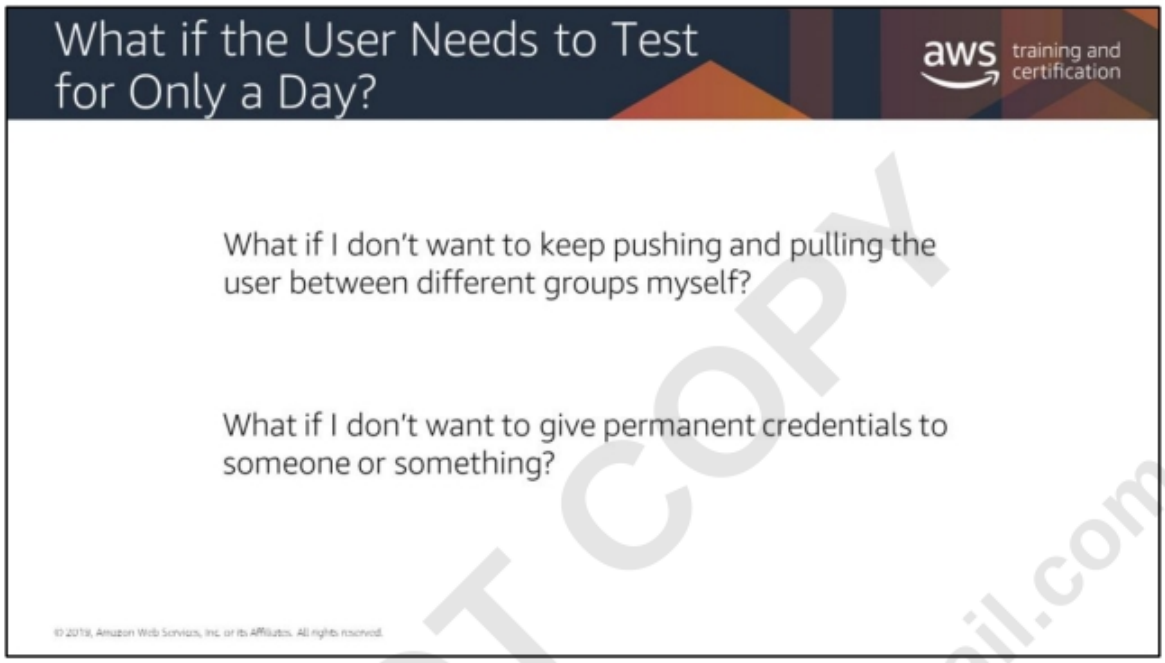


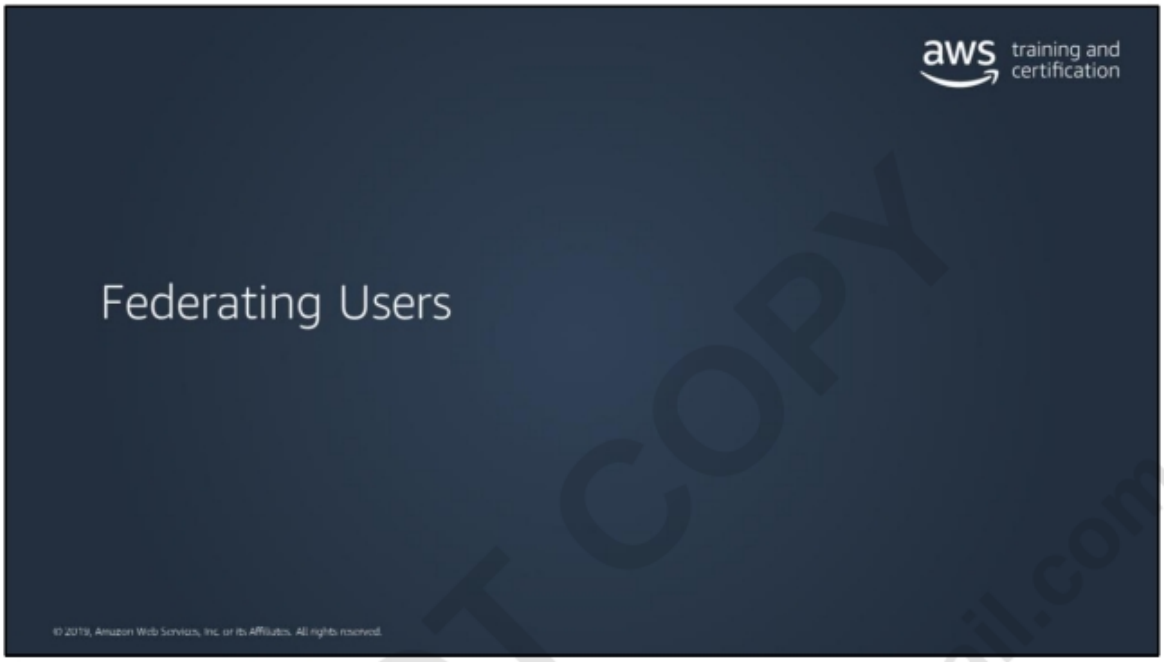


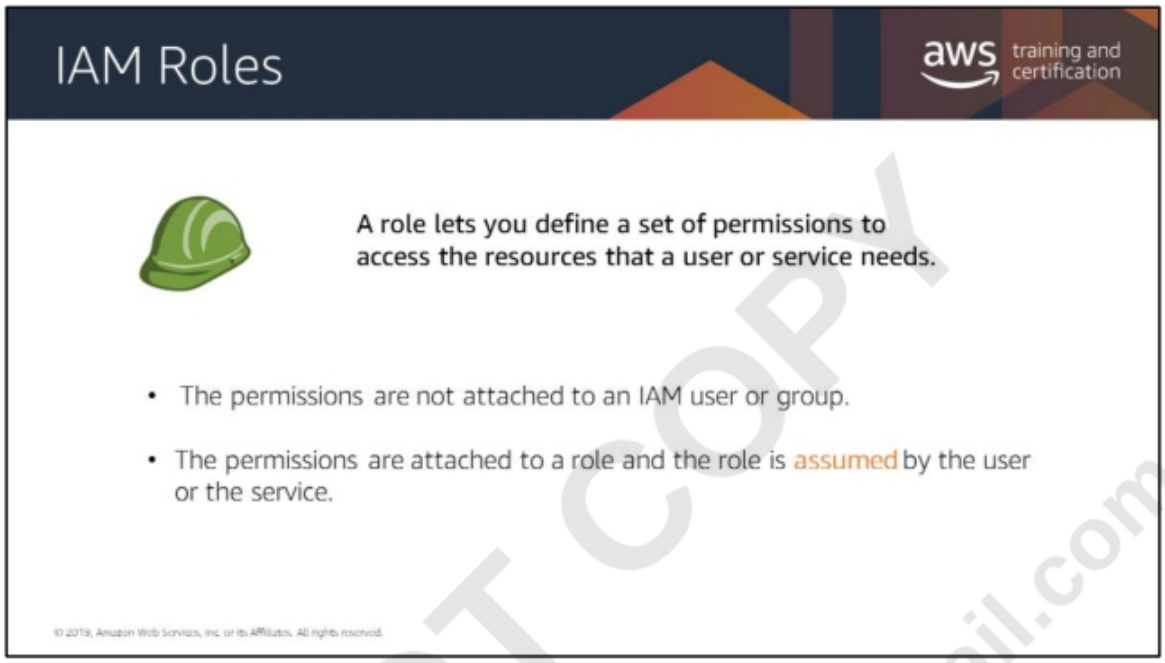












A *role* lets you define a set of permissions to access the resources that a user or service needs, but the permissions are not attached to an IAM user or group. The permissions are attached to a role and the role is *assumed* by the user or the service.

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.

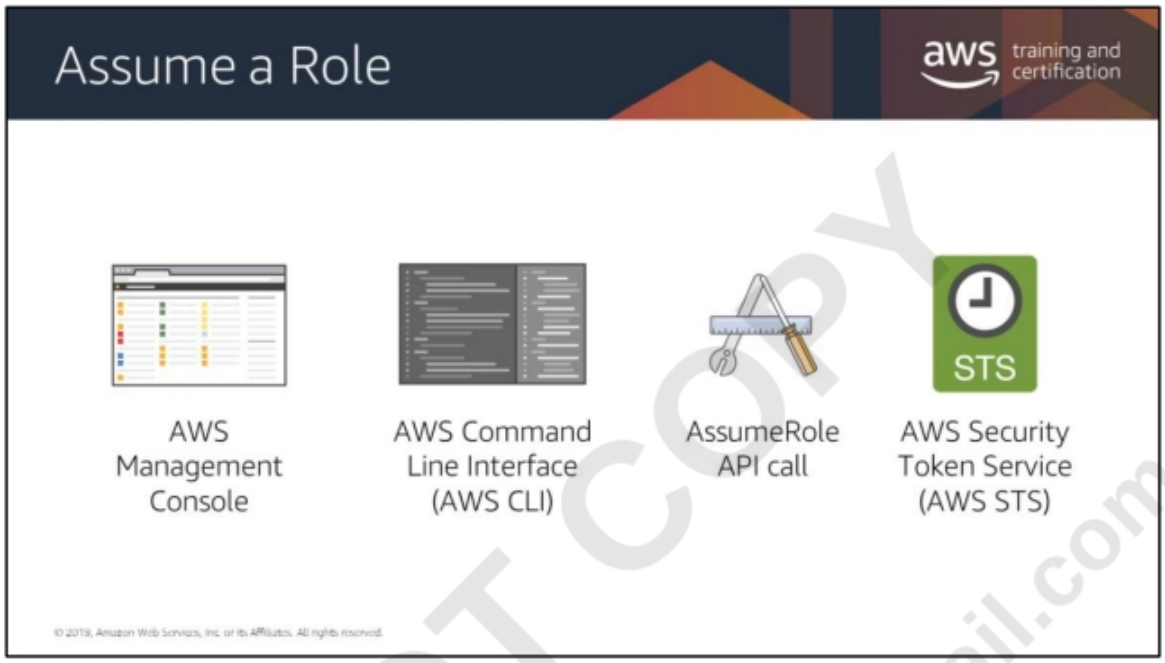
/

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.

/