**IAM roles**

So roles are specific type of identity available inside

IAM users were best used when you could visualize the entity or entities who would make use of the IAM user. It could be a single human. It could be an application, or it could be a service account used by backend systems, and I also mentioned the hard limit of 5,000 IAM users, so you can only ever have 5,000 IAM users in any one single AWS account. Those are two critical limitations of IAM users and one of the benefits of IAM roles.

Roles are not logged into they're assumed. So an entity is allowed to assume the role and when it assumes the role it gains the permissions that that role has.

Now earlier in the course I compared this to a fire drill or a fire exercise within an office environment. You might know somebody who is a fire marshal who's one of the people that puts on a nice glowing hat and a nice glowing vest and helps people evacuate a building in the event of a fire. This person doesn't normally have the authority to boss around senior management or the CEO, but in his or her role as a fire marshal, perhaps he or she does. So what that person is doing is assuming the role of being a fire marshal and by doing that, gaining additional power or additional permissions in the business to be able to do things. The fire marshal is a role. Multiple people might become that role. They might assume that role. It might be Bob one week, and it might be Jane the following week. The fire marshal is the role and Bob and Jane are the identities. So bring that back to AWS. You might have multiple identities in your account or other accounts, or even non AWS identities, and they might assume a role to gain the permissions of that role.

I'm going to create a role that can be utilized by an AWS service and I want to keep it simple, and it's going to be an EC2 role. So I'll select EC2, click next permissions, and then in the filter box I'm going to type S3 because I want to filter to this down to S3 and select Amazon S3 read only access. Now this policy provides read only access to S3, as the name suggests. So it just allows you to do get star on list star on any resource. So that's important to understand. It's a managed policy. So I'm going to apply this to the role. I'll accept the rest of the defaults and I'll just call this for now test role. I'll hit Create Role to create it and then we can investigate what this actually does. I have created a role that could be assumed by EC2. So let me rewind a little bit. With a role, you define entities which can assume that role. Now this is controlled by a component of that role called the trust policy. So the trust policy defines who can assume that role. So who could become the fire marshal in the physical example, there must be a list of people who are qualified to be a fire marshal, a list of people who were allowed to become fire marshals, and with this role example, there are a list of entities or identities or services that can assume this role. That's declared in the trust relationship of the role. So let's have a look at that. In this particular case, I'm going to show the policy document. So this is the trust policy of a role. Every role has two policies: a trust policy and the permissions policy. The permissions policy is just a permissions policy. It's just an IAM policy that gives that role permissions on things. It's the same as an IAM policy on a user or an IAM policy on a group. Nothing special about that. What's special about the trust policy is it controls who can assume the role. In this case, we're allowing EC2 to assume this role. EC2 is a service in AWS that provides virtual machines. You can make a virtual machine using EC2 and this role allows an EC2 instance to assume that role. Now, why might you do that?

When something assumes a role, it gets access to whatever the permissions policy says that it does. That's critical to understand. So it might be an IAM user. It might be somebody in a different AWS account. It might be an AWS service. **One of the key reasons roles get used is to give AWS services permissions on your behalf.** So let's quickly step through what happens when somebody assumes a role. **They make an assume role API call. This is just asking AWS let me assume a role.** If they're on the trust policy so if this service or this user or somebody in this remote AWS account is allowed to assume the role, then they actually assume that role and **when they assume the role, a service called STS, which is the security token service that gets involved.** What **STS does is generate what's known as a temporary security credentials.** These are like access keys that we talked about in a previous lesson, but they are **time limited**. So if STS determines that somebody can assume a role it generates temporary security credentials and those temporary security credentials are basically time limited access keys that get the permissions based on the permissions policy of the role. So another identity, this might be an IAM user. It might be a user in a different AWS account. It might be an AWS service. They assume a role if they can assume that role then STS looks at what the role's permissions policy is and it generates temporary credentials that give whatever identity assumed that role access to the AWS account that that role is in based on the permissions policy.

.So let's run through that. I want to talk about a few patterns and antipatterns. They're the technical terms for things which are ideally suited for roles, so patterns are things that roles are ideally suited for and antipatterns are inversely things that roles are not suited for. I'll start off with antipatterns. **Roles can't be used when you need something to log in to**. **So role usage is based on an identity that already exists, so you can't use a role to log in to the AWS console.** **Roles don't have usernames and passwords.** **Roles don't have any access keys.** So if we step through what a role has, **there are no long term credentials. There are no access keys, no usernames, and passwords. You cannot log into this role. You have to assume it**. The only way a role has any use is if you already have an identity.

Example 1. Let's say you need extra rights for something. Let's say you have a group of support engineers who are supporting an architecture that you've designed as a solution architect. Now you want to give the support engineers some fairly basic access rights relatively low access rights but you know that occasionally something serious happens an application crashes and you want to give the option to use something like a break glass style process. So you want to give the support engineers the ability if something really bad happens to gain some extra permissions, and you could use a role for that. So you can have a role that granted emergency super user or system administration permissions and then allow these support engineers to assume that role only if they needed to.

Example 2. might be service access. So let's say you've got other services inside AWS. Until this lesson, if you wanted to grant an AWS service access to something else, you might have struggled to think of how this is possible. Well, roles could be used to grant services access to things. You might give an EC2 instance access to a role so it can access an S3 bucket without needing to use regular credentials. You might give a role to a Lambda function to allow it to execute and interact with other AWS services without giving it access keys.

Example 3. let's say you're merging with another company. Your current company has 2,000 employees, and their company also has 2,000. You want all of their employees to be able to access your AWS account while the merger is taking place. Well you could create 2,000 brand new IAM users. This would still strictly be below the 5,000 IAM users for a single account, but it would be a lot of work on your behalf. One option is that you could make a role in your AWS account and set the trust policy on that role to the account of the company that's merging with yours. Then the people in that remote AWS account could assume the role in your account to access resources in your account, so your admin overhead is only creating the role and by adding the trust policy and saying that you trust this remote account, you're giving them all of the administrative overhead. So you're allowing a remote account, say Company B, to assume the role in your account. The remote account is responsible for then allowing their individual IAM users to assume your role. They can assume that role and get access to your account but you're the one who has had to create that one single role and just that one trust policy that's called **delegation**. It's a way that you can say that you trust another account.

Example 4. is that you might have multiple accounts in your company, say, 10 different AWS accounts. How do you allow staff to access each one of those accounts? You could create everyone 10 IAM users one in each AWS account. You could do that, but a better option will be to create a single IAM role in each of the accounts. Let the users log in to one single AWS account and then assume the role in the remote account. Doing that, you only need to manage a single IAM account.

So there is a way if you manage multiple AWS accounts, to access other AWS accounts by just using the single IAM user that you have and that's generally something that's used within AWS organizations, which is a multiaccount management product inside AWS.

Example 5. is that maybe you're working on a mobile application, maybe your developer expects 500,000 users of the application. The developer has told you that she wants to store high scores in a DynamoDB table. Now, 500,000 is far too large number to give every user of the application their own IAM to use. You can't do it. You got that 5,000 user limit but what you could do is allow the application to assume a single role and that role have the permissions to store the data. This is a process called **Web Identity Federation**, because what you can do is create a role and say that you allow any Twitter user, any Facebook user, any Google user to assume that role. In doing so, you could potentially allow millions of users of your application to assume that role and access high scores stored in DynamoDB and that's the method that's used for a lot of mobile applications to access AWS resources.

Exam Hints:

* IAM roles have no long term credentials.
* To get access to a role, you'd use STS assume role so that STS assume role is done by another identity. That's IAM users, AWS services, external accounts, and web identities.
* Temporary security credentials are generated by STS.
* The trust policy controls which identities can assume the role.
* The permissions policy defines what permissions are provided by STS.
* Temporary credentials expire. They are time limited and once they expire, you can't use them anymore, and you need to refresh them.
* Some example scenarios that I have talked about the company merger, the AWS service access, the break style extra access, the cross-account access, and even Web Identity Federation.