**IAM policies**

An IAM policy is a document which by itself does nothing. Only when it's attached to something does it take effect. Now it could be attached to an identity, in which case it's known as an **identity policy** or it could be attached to a resource such as an S3 bucket, in which case it's known as a **resource policy**.

IAM is all about identities. We will be focusing on Identity Policies.

A policy document is a list of statements:   
  
{ “version”: “2012-10-17”,  
 “Statement”: [{…},{…},{…]]

}

An IAM policy or a policy document, is just a JSON document. We can tell it's a JSON document because it starts and ends with curly braces. It includes a version number, so this is AWS's way of allowing the format for these policy documents to be updated. The current version is 2012-10-17. So you're going to see every single policy document will contain this date and then the remainder of a policy document is essentially **a list of individual statements**. Now all the statements are is themselves a JSON document, so it starts and ends with a curly brace.

{

“Sid”: “SpecificTable”,

“Effect”: “Allow”,

“Action”:

[ “dynamodb: BatchGet\*”,  
 “dynamodb: DescribeStream”,  
 “dynamodb: DescribeTable”,  
 “dynamodb: Get\*”, “dynamodb:Query”,  
 “dynamodb: Scan”, “dynamodb: BatchWrite\*”,  
 “dynamodb: CreateTable”,  
 “dynamodb: Delete\*”, “dynamodb: Update\*”,  
 “dynamodb: PutItem”  
 ],  
“Resource”: “arn: aws : dynamodb : \*: \*: table/CatPics”  
}

This is an example of a single JSON statement, a single statement in an IAM policy. All of them share a couple of key components. First, we got the SID, which is a unique identifier for this particular statement. So SID stands for statement ID. **The effect is what this statement actually does. So does it allow something, or does it deny something?** In this case, it's an allow. Now what it's allowing or denying is whatever the statement matches and the way it determines what it matches is it looks for what actions the identity is attempting to perform. So what operations on AWS as well as those actions, it looks at the resource component so together it's looking for a combination of one or more actions and a combination of one or more resources. So in this particular case, this statement allows any of these operations to occur on a particular resource. So it's a DynamoDB table, and it's the cat pics table. Now the way that resources are referenced is by using the standard AR informer or Amazon resource name and I've already discussed that in a previous lesson. The list of actions can either be specific actions so DynamoDB: describe stream, DynamoDB: described table, or they can include wild cards so you can have dynamodb:get and then a star to indicate anything after the star. So essentially you've got your actions you've got your resources, and together they describe what operation on what resource this statement targets, and the effect is either allow or deny. Now there are other components to a statement, such as conditions, but that's a fairly advanced topic.   
**A statement either allows or denies whatever operation that statement is attempting to match against and we determine that by looking at the actions and the resources.** So let's switch across to a real example and this example is going to be an iron policy that I'm going to use and demonstrate in this lesson and the next lesson.

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "BucketListing",

"Effect": "Allow",

"Action": [

"s3:ListAllMyBuckets",

"s3:GetBucketLocation"

],

"Resource": "arn:aws:s3:::\*"

},

{

"Sid": "ListObjectsInBucket",

"Effect": "Allow",

"Action": "s3:ListBucket",

"Resource": [

"arn:aws:s3:::ac-catpics-XXX",

"arn:aws:s3:::ac-dogpics-XXX”

]

},

{

"Sid": "AllObjectActions",

"Effect": "Allow",

"Action": "s3:\*Object",

"Resource": [

" arn:aws:s3:::ac-dogpics-XXX/\*"

]

}

]

}

So this IAM policy is a single JSON document. So that's the entire document, starting at the curly bracket and ending at the curly bracket. We've got the version component that I talked about a second ago, so that indicates the format this entire document and then we got this statement part, which is a list of individual statements. So everything between these curly braces. So this component that's a statement, that component that's a statement, and this final component that's a statement. They're each individual, and they each allow or deny a different thing. In this example, the first statement is allowing whatever identity this is attached to to perform, list all my buckets, and get bucket location, and the identity to which this is attached is allowed to do that on any S3 resource. This, in effect, allows us to do a bucket listing. The second statement focuses on two specific S3 buckets ac-catpics-xxx ac-dogpics-xxx and this allows the S3 list bucket operation to be performed on either of these two resources. That essentially allows you to list the objects in either of those buckets. The last statement this allows S3:\* objects. That's a wild card, and that means anything before object. So get object, put object, delete object, and so on. Anything that could be before object that's represented by this wild card, and it allows the action or that wild card collection of actions to occur on any resources inside the dog pics bucket. So essentially any identity that this is associated with will be able to list all of the buckets inside the S3 account. It'll be able to list objects in any of these two S3 buckets, and it will be able to interact with objects inside the dog pics bucket.

Now to demonstrate this, first I want to create the S3 buckets that this refers to. Created 2 s3 buckets.

Policy documents could be directly added to identities so users, groups, and roles and these are known **as inline policies**. Alternatively, they could be created as their own unique entity, which is known as a **managed policy**. Now managed policies are defined inside the policy section of the IAM console. There are two types of managed policies, ones that AWS manage, and they're called logically enough, **AWS managed policies.** They require very little admin overhead but of course they lack flexibility because you're relying on AWS to define these policies for you so they're not going to fit your exact business needs. Examples of AWS managed policies are **Amazon chime read only, Access Amazon API Gateway Administrator, Administrator Access,** which gives admin functionality of to an IAM user, and there's many more and then you've also got **customer managed policies**.

Now customer manage policies is the name suggests a created and managed by the customer. That's what we're going to create and with customer manage policies you trade a little bit of the admin overhead for flexibility, so you need to admin these customer manage policies, but you get complete flexibility. Now when you go to create a customer managed policy, you can do so using the visual editor so you're able to select from a list of services populated by AWS. Once you've selected that service, you can define actions and resources and then any conditions which we'll talk about later in the course. We've already got the policy ready so we'll click on the JSON tab. I'm going to switch back over to my code editor and copy all of this into my clipboard. Once I've done that, I'm going to paste it into the JSON editor. Now there are some placeholders that I'll need to replace, so I'm going to change all of the XXXs to 333 which is a suffix I chose when I created my S3 buckets. Now that I've done that, this policy will reference the buckets that I've just created. So anything I attach this policy to will be able to list the buckets in the account. It'll be able to list the objects in these two specific buckets and it will be able to interact with the dog pics bucket. I'm going to go ahead and click Review Policy. Assuming there's no errors, I'm going to name this policy cats and dogs. Now this is a managed policy we'll be able to reuse it for the things. So it's important that if you do in this in production, you use something suitable for both the name and the description because this is only an example, I'll use cats and dogs in both and hit Create Policy. **Now the benefit of using these managed policies is that you can create the policy once and attach it to many users or groups or roles or a combination of any of those. You build it once, and you use it many times but of course, that does reduce the flexibility because it means you can't make a user by user or a group by group customization. So architecturally, the best practice is that you use these managed policies to control the base level permissions for your organization, and then, for any customization any exceptions, you can use inline policies.** That's either for allowing extra permissions or adding explicit denies which prevent people from accessing resources. It's important to understand that from a best practice perspective use these managed policies for general overall permissions, and then you customize it using inline policies.

It's really important to understand how permissions are evaluated inside AWS. When you create any IAM policies, whether that's inline or managed, you're assigning them to a particular user, may be a group, or maybe a role. What happens is that whenever an identity attempts to access AWS, AWS and specifically, IAM looks at all of the policies that could apply to that user accessing that resource. Now that could include some resource policies, which we haven't talked about yet. For now, focus on the identity policies. So when a user let's say you accesses AWS, your user might have in line policies, or it might have associative managed policies. We haven't covered groups yet, but you could be in a group which itself has inline policies or managed policies. Basically all of the policies that can apply to you. So all of your user policies and all of your group policies, they're all merged down into a single set of policies, a single set of permissions that you have for accessing that resource and then, once all of the policies that apply are combined down AWS evaluate them. If there isn't a matching statement so if there isn't a statement that applies to your identity accessing, for example, a given S3 bucket. If there isn't that, if that doesn't exist, there is an implicit default deny. Remember that term "**implicit default deny."** If an IAM user doesn't have any associated allow. If it isn't allowed to access something, it does not have access by default, IAM users have no access to anything. If a request is explicitly denied. So if there's a deny statement that applies, if I deny you access to an S3 bucket then that overrides everything. You can have an explicit allow, and if you've got an allow then you are allowed access to that resource, unless you've got an explicit deny and the way I like to remember this is deny allow deny. **Explicit denies overrule everything**. If you're denied access to an S3 bucket that's it, game over. You don't have any access. **If there isn't an explicit denying and you're allowed** then congratulations, you get access to the cat and dog pics. If neither of those apply, there's the default deny. Now only policies which were attached have any impact. **When evaluating policies, all the applicable policies are merged** down all identities, so users, groups, and roles all of those that apply, and any resource policies are all combined and evaluated in one go. **Managed policies allow the same policy to impact many identities.** They allow reuse its best practice to use that for common permissions. **Inline policies allow exceptions to be applied to identities.** **AWS manage policies they're low overhead, but they lack flexibility. Customer managed policies are flexible but they require additional administration** and then **inline and managed policies can apply to users, groups, and roles.** There is a couple of things I want you take away from this lesson. It's to understand the absence of any applicable policies. There are no access rights, default deny, implicit default deny. If there is ever an explicit deny, that's it denied. If there's an explicit allow that overrides an implicit deny. The denying policy on its own without it being attached to anything has no effect.