CLOUDBERRY LAB IS NOW
MSP360
#1 MSP BACKUP

E-BOOK

Mastering AWS IAM for Amazon S3

E-BOOK

Mastering AWS IAM for Amazon S3

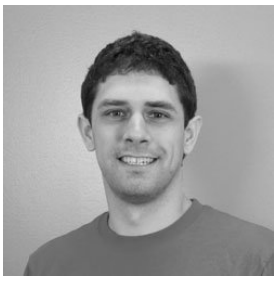
Intro

To use Amazon S3 effectively, you need to be aware of the security mechanisms provided by AWS to control your S3 resources. This e-book will help you to master these mechanisms.

Further on, we will discuss the three access control tools provided by AWS to manage your S3 resources, take a deeper look at writing policies in AWS Identity and Access Management (IAM) and finally cover the authentication and identity in AWS IAM.

We will start, however, from taking the very first steps in AWS security - secure the account and give basic advice on how to safely use Amazon Web Services.

About author



Alex DeBrie

A software developer with a background in data engineering and cloud infrastructure management. He has deep experience with AWS and is a proponent of infrastructure as code. In his free time, he likes running and spending time with his wife and four kids

AWS IAM Quick Start

In Amazon Web Services you have one root account with access to all services, as well as the ability to set up user accounts with granular permissions to necessary services. These users are created using the AWS Identity and Access Management (IAM) service. This service gives you control over **all** aspects of **all** services inside AWS.

Here is an overview of the best practices when working with AWS.

Enable MFA on Your Root Account and Never Share Credentials

Once you sign into the root account, you can do anything in AWS: sign up for services, set preferences, create users and storage buckets, and so on. You can do destructive things like delete virtual machines and data in Amazon S3. Therefore, it's imperative you protect your root account.

Use a strong password and enable multi-factor authentication (MFA). AWS has several MFA options to choose from, so select the one based that best suits your internal policies and preferences.

If you need to provide a subset of access to administrators to manage services in the AWS web console or provide access to someone on another team, create users who have limited access to the needed services.

Create Users and Groups

A User is the entity in AWS IAM that you create to allow other people or applications access to services in your account. You create the user, then assign specific permissions to access the required services. If you have lots of users with access needs, you can create groups, assign the appropriate permissions, and then add users to the group and they will automatically receive all of the permissions of that group.

Access Keys

Access Keys are a set of credentials that you use to communicate your identity and permissions between AWS services and third-party applications. AWS will then authenticate the software and allow it to upload and download data to and from Amazon S3.

> **Your access keys consists of the access key and the secret key. In AWS, access keys do not exist before you create them. Access keys will be visible after creation, but a secret key will only be shown once. Therefore, if you forget or lose your secret key, you need to create another access key/secret key pair.**

Don't Create Root Access Keys

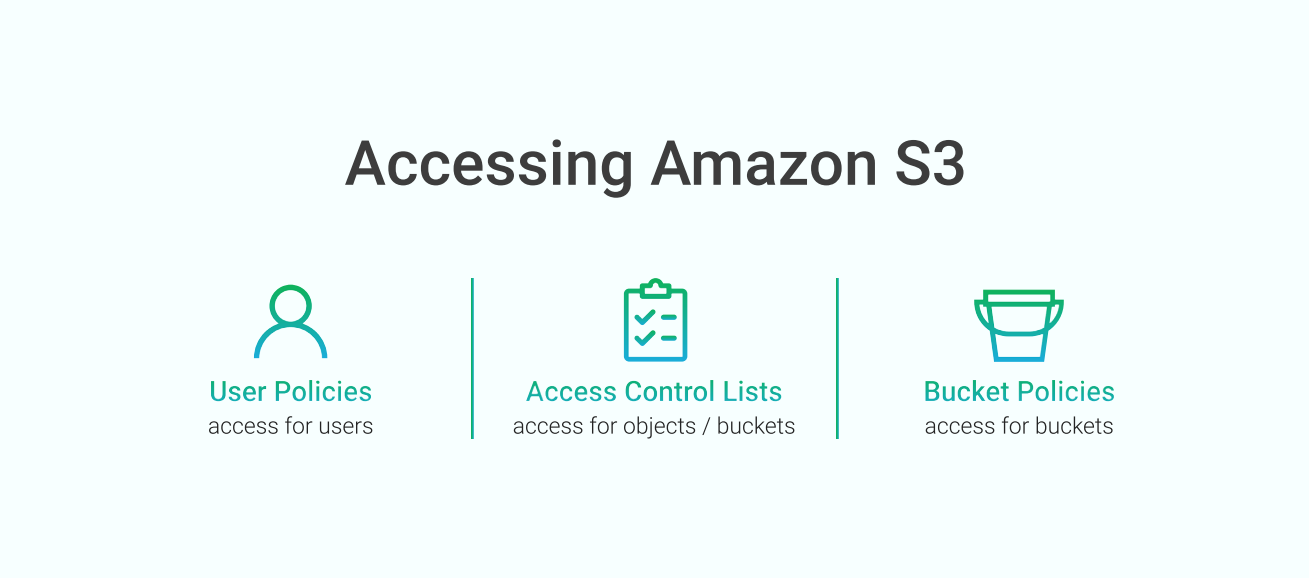
Root access keys are a set of credentials that provide access to all services and features of AWS. It's a best practice not to create these keys at all, even if doing so seems like a fast solution.

Don't Give Public Access to Data

By default, AWS does not allow public access to your data in Amazon S3, meaning it is invisible from the Internet. You can enable public access manually, but I don't recommend doing so. Data breaches happen when someone grants public access to your cloud storage. If you need to share data with a third party, create a dedicated IAM user with limited access.

That's all with the basic safety rules inside your Amazon Web Services account. Further we will discuss in-depth techniques of working with Amazon S3 access tools. We will start from the general overview of these tools.

Part 1 - Introduction to Amazon S3 Access Tools



AWS has three different options for controlling access to your S3 buckets and objects. Each option is tailored for different circumstances. The three options are:

- User Policies: Use the AWS IAM policy syntax to grant access to IAM users in your account;
- Bucket Policies: Use the AWS IAM policy syntax to manage access for a particular S3 bucket;
- Access Control Lists (ACLs): Use XML syntax to grant access to specific S3 buckets or objects.

One useful distinction between the access control types is whether they are attached to a *user* or to a *resource*. User policies are attached to a particular IAM user to indicate whether that user can access various S3 buckets and objects. In contrast, bucket policies and ACLs are attached to the resource itself - either an S3 bucket or an S3 object - to control access.

In the sections that follow, we'll review each of the three types of access control methods and describe when you should use each one.

Access Tool 1 - IAM Policies

The most common way to manage access to your S3 resources is via IAM policies. IAM is AWS's comprehensive tool for managing identity and access control across all of its services. IAM allows you to set fine-grained access rules on your S3 resources for a particular user, from the wide ability to read, write, and destroy all S3 resources to a narrow ability to read a single S3 object.

Generally, you should prefer using IAM policies to manage access to your S3 bucket. User policies fit naturally with the broader AWS access management ecosystem, so you can manage all of your access policies in a central place. The tooling and capabilities around IAM are mature and can be used to identify human users or AWS resources - such as EC2 instances or Lambda functions - that have access to your end resources.

EXAMPLE

IAM user policies are written in JavaScript Object Notation (JSON) and have a consistent structure across all AWS services. An IAM user policy will look similar to the following:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":"arn:aws:s3:::mybucket/*"
      }
   ]
}
```

This policy would allow any user that is associated with it to read any object (via the "s3:GetObject" action) in the S3 bucket called "mybucket".

Parts 2 and 3 of this e-book will take a deeper look at writing S3 user policies and managing IAM identities, including creating IAM policies with MSP360 Explorer.

MSP360 Explorer provides a user interface allowing to access, move and manage files across your local storage and all classes of the Amazon S3. Cloud file management software by MSP360 is available in two versions: Freeware and PRO.

Download

Access Tool 2 - Bucket Policies

The second mechanism to control S3 access is via bucket policies. Bucket policies are similar to IAM user policies. They're written in the same JSON syntax and can be used to provide granular permissions on S3 resources.

The main difference from IAM user policies is that bucket policies are attached to an S3 resource directly rather than to an IAM user.

There are three occasions where you may want to use a bucket policy rather than a user policy. First, if the only AWS service you're using is S3, you may find it easier to manage permissions directly within S3 via bucket policies. This mostly comes down to personal preference. However, if you're doing more advanced AWS usage, we would recommend using IAM for all access control where possible for the reasons mentioned in the previous section - more mature toolset and a centralized place for all identity management.

The second occasion where you may want to use bucket policies is for allowing access from a different AWS account. While you can delegate IAM permissions across AWS accounts, it can be complex. Bucket policies are slightly easier to allow access to a different AWS account or even a particular IAM user within a separate AWS account.

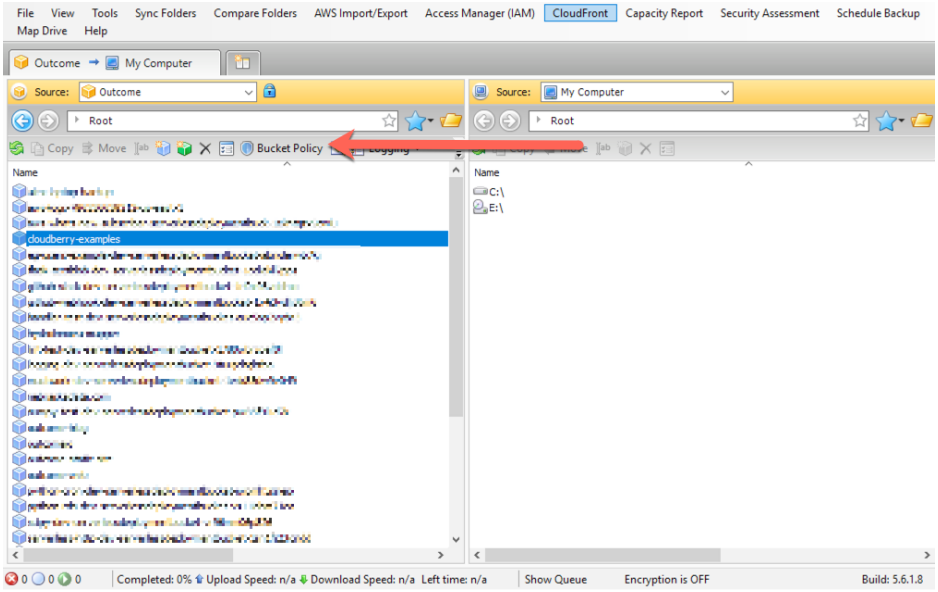
Finally, you might need to use bucket policies for allowing access to S3 resources based on something other than AWS IAM identity. For example, you could limit S3 access to requests from particular IP addresses. You could also limit access to media assets in your S3 bucket to requests from a specific referrer so as to only allow your website to display images and video.

While bucket policies may be helpful in specific circumstances, it is best to use IAM user policies where possible. IAM user policies can do almost everything that S3 bucket policies can do, plus IAM provides a centralized location for all of your AWS access control. It can be difficult to debug unexpected user access when you are spread across both IAM user policies and S3 bucket policies.

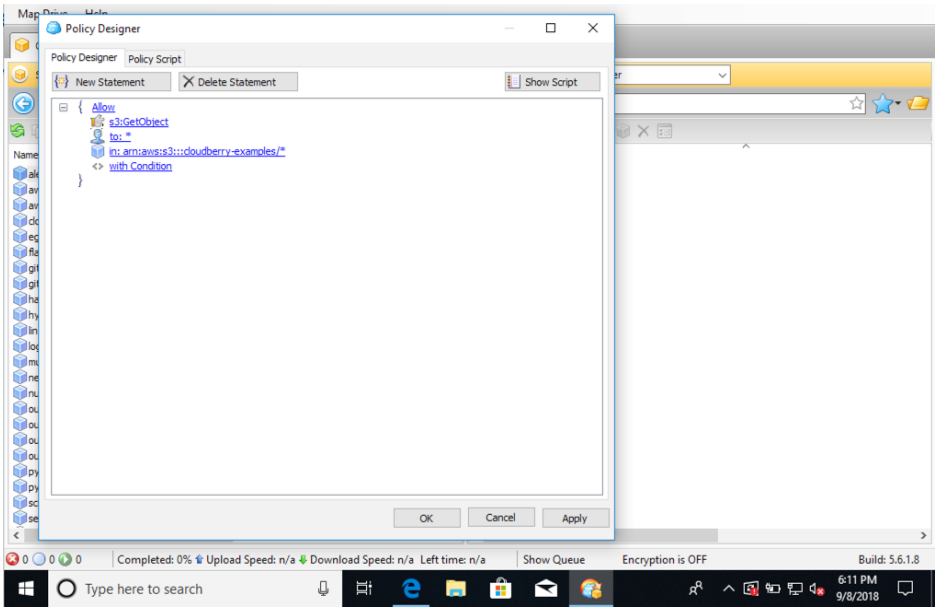
Setting Bucket Policies with MSP360 Explorer

You can use MSP360 Explorer to easily manage your S3 bucket policies.

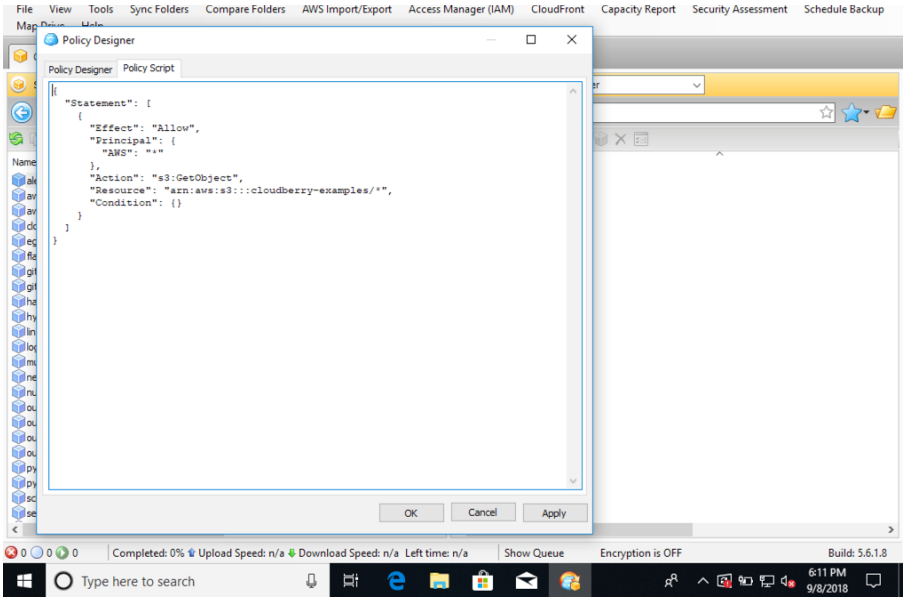
First, when looking at your buckets in MSP360 Explorer, select the bucket to which you wish to add a bucket policy. Then, click the "Bucket Policy" button in the toolbar, as shown below.



This will open a wizard to help you write a bucket policy. You can use the wizard to construct your policy by specifying which permissions you want to apply, the principal to whom they will apply, the resource to which they will apply, and the conditions that will apply. A completed statement will look as follows:



You can check the syntax that will be created from the wizard by clicking the “Show Script” button. It will display your bucket policy statement as follows.



For more on writing bucket policy statements, check out Part 2 of this e-book in which we do a deep dive on the syntax of writing IAM and bucket policies to provide access to your S3 resources.

Access Tool 3 - ACL Policies

The final mechanism to control S3 access is using access control lists (ACLs). ACLs are similar to bucket policies in that they are attached directly to an S3 resource, either a bucket or an object.

> **ACLs are more of a legacy feature and generally should be avoided. IAM user policies and bucket policies should be used whenever possible.**

That said, there are a few situations where ACLs may be used to control S3 access. First, if you want to enable Server Access Logging on your S3 bucket, you will need to provide a bucket-level ACL that allows AWS's Log Delivery group to write to a particular S3 bucket.

A second situation when you may want to use ACLs is if you want to provide cross-account access on an object-level basis, rather than on a bucket-level basis. As discussed in the previous section, it can be difficult to manage cross-account access via IAM policies. Bucket policies can assist with this but they can only be applied to a bucket. While it's possible to apply rules to prefixes within a bucket, it can still be difficult to provide granular access control that you need. ACLs let you attach access control rules to S3 objects directly, giving you more flexibility.

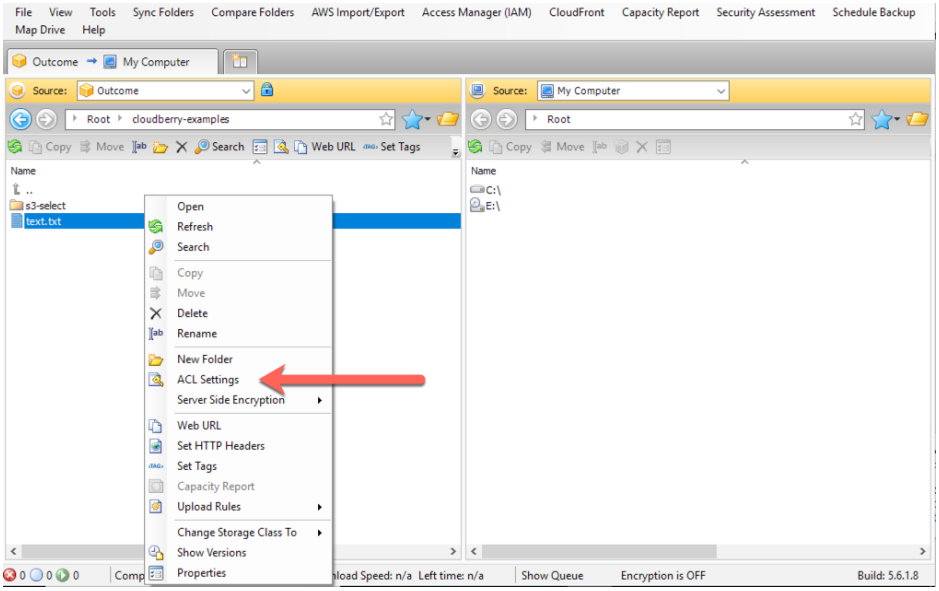
There is one final note about ACLs that often trips up users. Each ACL rule you create must include a "Grantee" that specifies to whom the ACL applies. The Grantee may be either a specific AWS account or one of three predefined groups of users provided by AWS. The three predefined groups are the "Log Delivery group" (used for the Server Access Logging described above), the "All Users" group (indicating that the ACL applies to all requests to the given S3 resource), and the "Authenticated Users" group.

The "Authenticated Users" group is a common source of confusion with ACLs. Many AWS users believe this means "any authenticated IAM user within my AWS account." However, the group actually includes all users in any AWS account. Essentially, this gives access to the given S3 resource for anyone that is making a signed, authenticated request for your S3 resource. There are very few reasons to use the "Authenticated Users" group in ACL policies.

Managing ACLs with MSP360 Explorer

You can use MSP360 Explorer to set and update your ACLs on S3. Like bucket policies, ACLs can be attached to S3 buckets. Unlike bucket policies, ACLs can also be attached to individual S3 objects. In this section, you will learn how to use MSP360 Explorer to set ACLs on an S3 object. The experience is similar to set ACLs on an S3 bucket.

First, navigate to the S3 object for which you wish to manage its ACL. After selecting it, right click and choose the "ACL Settings" menu item as shown in the following image.



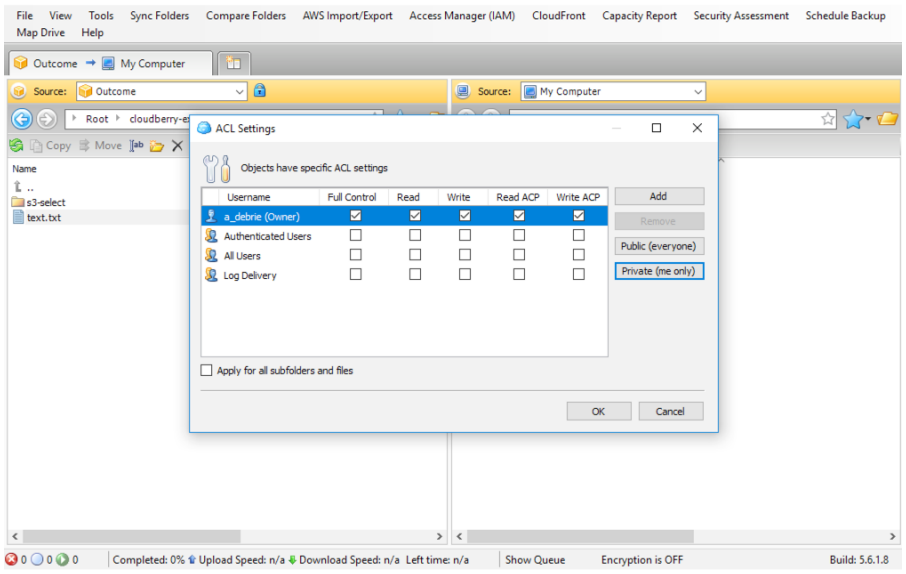
This will open a wizard to manage your ACL policies for the S3 object. In setting an ACL, you need to think about two questions:

- **To whom** you wish to grant permissions
- **Which** permissions you wish to grant

In choosing the Grantee ("To whom" you will grant permissions), you can specify a particular AWS account, either by its root email address or by its account ID, or you can choose one of the predefined groups from AWS. Please read the preceding section on ACLs to know more about the predefined groups.

When choosing permissions, you can choose to provide read and write permissions on the object itself. You can also provide read and write permissions on the ACLs for the object. Finally, you can choose to give "Full Control", which gives the ability for the Grantee to read and write both the object and the ACLs on the object.

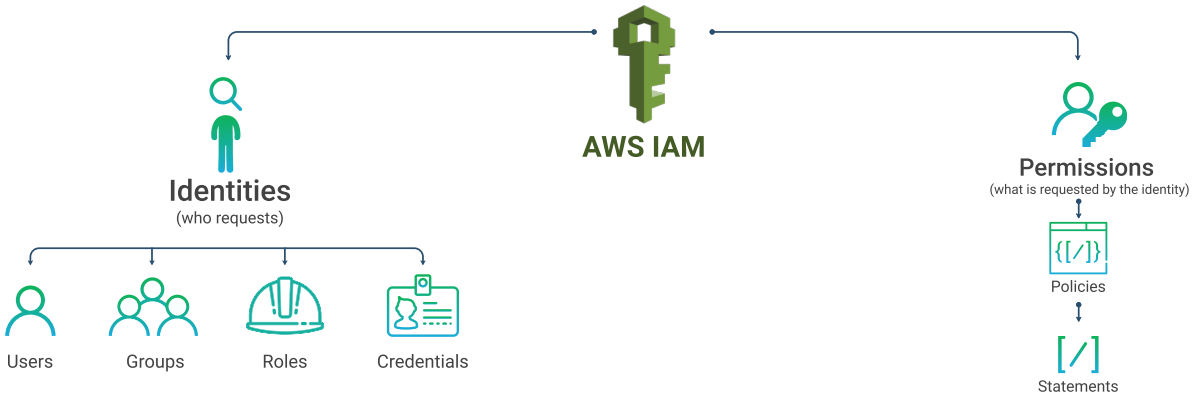
In the screenshot below, you can see how to set ACLs with MSP360 Explorer.



Further reading If you are an MSP, check out our guide to providing Backup-as-a-Service with MSP360 and AWS

Part 2 - Mastering IAM Policies

In this part, we'll take a deeper look at writing IAM user policies to control access to S3 resources. When thinking about IAM, there are two broad categories to consider: ***identities*** and ***permissions.***



Identities refer to the various mechanisms that AWS provides to identify who is requesting a particular AWS action, for authenticating that person or entity, and for organizing similar entities into groups. This topic includes IAM users and groups, credentials, and roles. We will cover more about best practices for managing IAM identities in Part 3 of this e-book.

Permissions refer to what a particular identity is allowed to do in the AWS account. Permissions are managed by writing identity-based policies, which are collections of statements. Writing identity-based policies within IAM is the main focus of this part. Identity-based policies are used mostly within AWS Identity and Access Management (IAM), so I will refer to them as IAM policies throughout this part. However, the same syntax in IAM policies is also used for S3 bucket policies that were discussed in Part 1.

Further in Part 2 we'll cover:

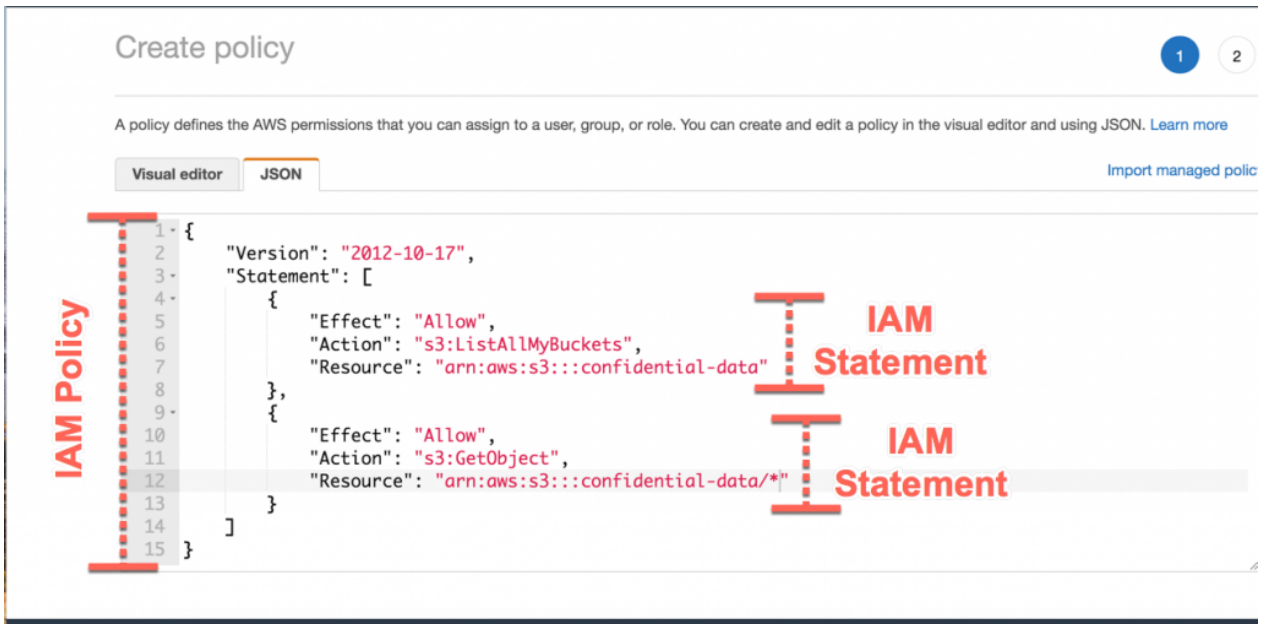
- The basics of IAM policies and statements, including a breakdown of the important elements;
- An example walkthrough of an IAM policy with multiple statements;
- Best practices for writing and managing IAM statements.

AWS IAM Policies and Statements

IAM is an AWS service for managing both authentication and authorization in determining who can access which resources in your AWS account. At the core of IAM's authorization system is an *IAM policy*.

Let's take a look at the example below of an IAM policy being created in the AWS console.

EXAMPLE



The entire document from lines 1-15 is the IAM policy. An IAM policy is a JSON document with an optional "Version" key plus a "Statement" key. The value of the "Statement" key is an array of IAM statements.

There are two IAM statements in this example – one is from lines 4-8, and the other is from lines 9-13. There is no limit on the number of IAM statements in an IAM policy, though there are limits on the number of characters in an IAM policy. The number of characters varies from 2,048 characters to 10,240 characters depending on the type of policy.

There are three critical elements to an IAM statement. They are:

- Action
- Resource
- Effect

We'll take a look at each of these separately.

IAM Statement Elements: Action

Action describes the API actions that the statement affects. An action is in the form of ***"<resource>:<verb>"***, where the resource is a particular AWS service (e.g., s3, ec2, or dynamodb) and the verb is a particular action to take within that service (e.g. GetObject or CreateBucket). There are 51 different actions within S3 alone. When writing IAM statements for S3 access, some of the popular actions you will use are:

- **s3:GetObject: For reading an object on S3;**
- **s3:PutObject: For writing an object to S3;**
- **s3:ListBucket: For reading the objects in an S3 bucket;**
- **s3:ListAllMyBuckets: For listing all of the S3 buckets in your AWS account.**

When writing actions, you can use an asterisk to act as a wildcard on your action. You can use the wildcard at the beginning, middle, or end of your action, or as the entire action if you want to write a statement about all actions. For example, each of the following are valid actions:

- **s3:Get*:** This would affect all S3 actions that start with "Get", such as "GetObject", "GetBucketPolicy", and "GetBucketLocation"
- **s3:*:** This would affect all actions within the S3 service.*: This would affect all API actions across AWS, including all actions in S3 but also all actions in EC2, SQS, DynamoDB, etc.

You should be careful when using wildcards in your Actions to avoid unintentionally granting broad access. Check out the "Be Careful with Wildcards" piece in the Best Practices section below on page 18.

IAM Statement Elements: Resource

The second critical part of an IAM statement is the Resource. The resource element describes the AWS resources to which the statement applies.

Every AWS resource is given an ***Amazon Resource Name***, or ARN. This ARN uniquely identifies a single AWS resource across all AWS accounts. An ARN is structured into 6 parts, each separated by a colon. As you move from left to right, each part gets more specific about the particular resource it identifies.

The general structure is:

EXAMPLE

arn:partition:service:region:account-id:resource

Let's look at these pieces individually:

1. ***arn:*** The first part is always "arn", regardless of the resource type.
2. ***partition:*** The second part describes the partition of AWS you're using. This is usually "aws", but it may be different if you're using special regions of AWS, such as GovCloud ("aws-us-gov") or the China region ("aws-cn").
3. ***service:*** The third part is the service you're using within AWS, such as "s3", "ec2", or "sqs".
4. ***region:*** The fourth part is the AWS region you're using if applicable. As of September 2018, AWS has 18 regions around the world, such as "us-east-1" for the Northern Virginia region in the United States, or "eu-central-1" for the Frankfurt, Germany region. For S3 resources, region is not included in the ARN.
5. ***account id:*** The fifth part is the account ID of the AWS account that owns the resource. For S3 resources, account id is not included in the ARN.
6. ***resource:*** The sixth and final part of the ARN is the resource name. This could be the name of an S3 bucket, an EC2 instance id, or an SQS queue name. Depending on the AWS service, the resource name may include a path to further identify a resource.

EXAMPLE

Below are examples of ARNs for an S3 bucket, an EC2 instance, and an SQS queue, respectively.

- arn:aws:s3:::my_bucket
- arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0
- arn:aws:sqs:us-east-1:123456789012:queue1

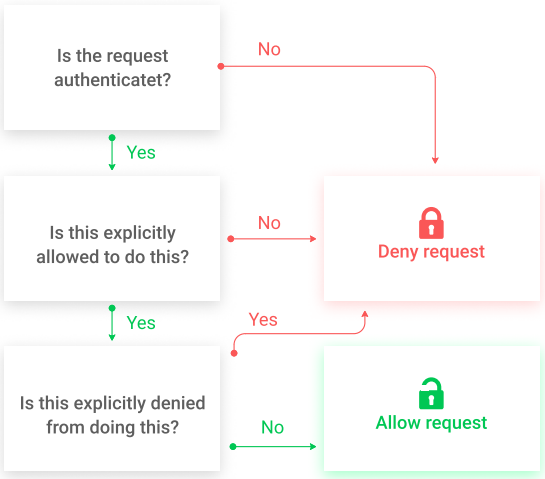
Notice how the S3 bucket does not have a value for the region or account id. This is because S3 bucket names are globally unique and thus do not need region or account identifiers.

One important note about S3 ARNs - there's a key distinction between bucket resources and object resources. If you want to specify an S3 object, you must include a "/" in the resource name to indicate the bucket and the object. For example, if you have a bucket named "my_bucket" with a single object named "logs.txt", the resource for the bucket would be **"arn:aws:s3:::my_bucket"**, while the resource for the object would be **"arn:aws:s3:::my_bucket/logs.txt"**.

When writing an IAM statement, you must specify one or more resources to which the statement will apply. This allows you to narrowly tailor your IAM statements - one statement could provide broader permissions to a wide range of resources, while another statement could grant narrow permissions to one or two protected resources.

IAM Statement Elements: Effect

The final element required element of an IAM statement is the effect. Before looking into the effect, let's review the authorization flow in IAM.



First, all API requests are denied by default. If you make a request that is not authenticated, it will be denied. An API request may be allowed if there is a specific statement that allows the API request for the requesting user. Finally, a request will be denied if there is a specific statement that denies the API request for the requesting user, even in the face of a statement that purports to allow the request.

With this authorization flow in mind, let's return to the "Effect" portion of an IAM statement. The value for the "Effect" property can be "Allow" or "Deny". A value of "Allow" will give the attached identity the ability to make the API request on the requested resources in the statement. A value of "Deny" will strictly deny the API request on the resources in the statement.

In the next section, we'll put all of the elements together to show you how to write multiple IAM statements in an IAM policy.

Putting it all together: Writing IAM policies

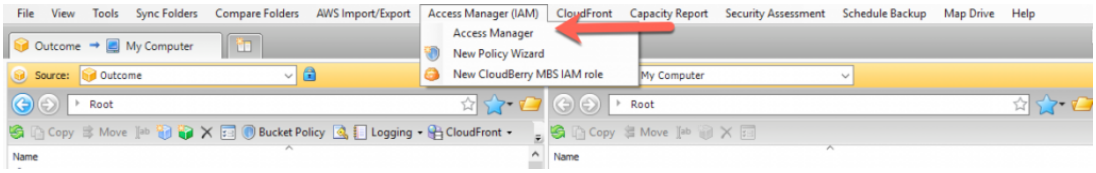
Now that we know the basics of IAM statements, let's write our first IAM policy.

EXAMPLE

Imagine we have the following scenario. We own an S3 bucket called "MSP360-examples". We want to allow one of our users to list all of the objects in the bucket. We also want the user to generally be able to read and write freely to the bucket. However, there is a prefix of "secrets/" in the bucket that contains sensitive information. We do not want the user to be able to read or write in the "secrets/" prefix.

You can use the AWS console, the AWS command-line interface, or numerous infrastructure-as-code tools to manage your IAM permissions. We are going to use the Access Manager built into the MSP360 Explorer as it has an easy wizard for generating IAM statements.

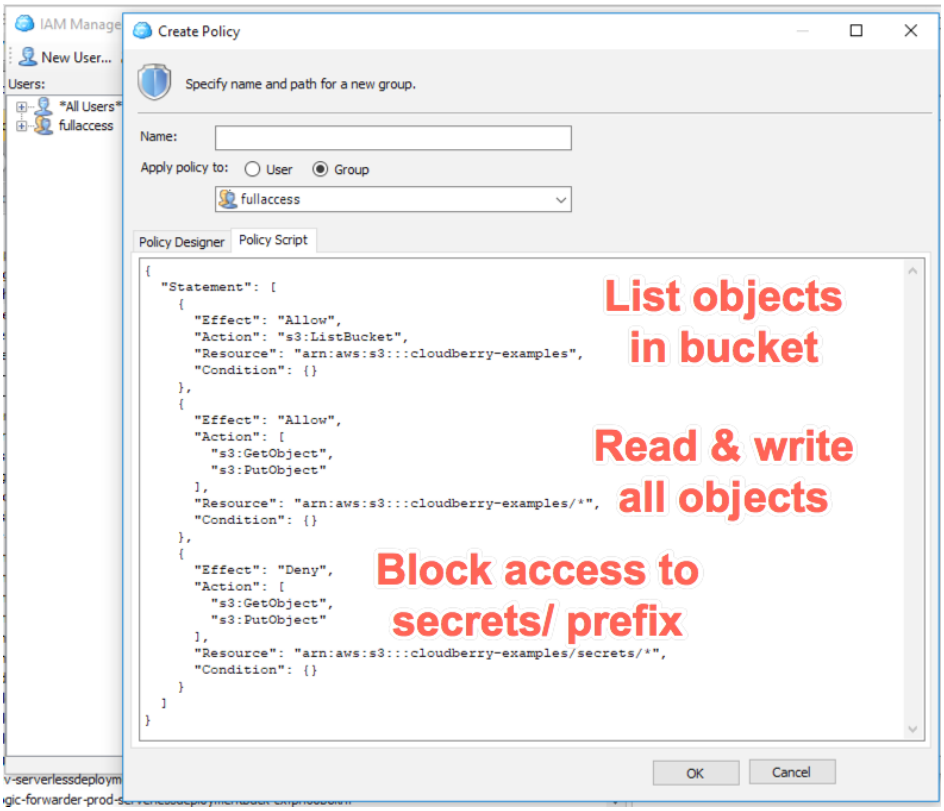
To manage IAM within MSP360 Explorer, click the "Access Manager" button in the toolbar.



As we break down our needs, we can think about three statements for our policy:

- A statement to *allow* listing the objects in a bucket;
- A statement to *allow* read & write access in the bucket;
- A statement to *deny* read and write access to objects that start with "secrets/*".

We can see these three statements in the following policy in MSP360's Access Manager:



The text version of the IAM statement is as follows:

```
{
  “Statement”: [
    {
      “Effect”: “Allow”,
      “Action”: “s3:ListBucket”,
      “Resource”: “arn:aws:s3:::MSP360-examples”,
      “Condition”: {}
    },
    {
      “Effect”: “Allow”,
      “Action”: [
        “s3:GetObject”,
        “s3:PutObject”
      ],
      “Resource”: “arn:aws:s3:::MSP360-examples/*”,
      “Condition”: {}
    },
    {
      “Effect”: “Deny”,
      “Action”: [
        “s3:GetObject”,
        “s3:PutObject”
      ],
      “Resource”: “arn:aws:s3:::MSP360-examples/secrets/*”,
      “Condition”: {}
    }
  ]
}
```

In the first statement, we *allow* access to the “s3:ListBucket” *action* on our “MSP360-examples” bucket *resource*. In the second statement, we *allow* access for the “s3:GetObject” and “s3:PutObject” *actions* on all objects in the “MSP360-examples” bucket. Notice how we indicated that it applied to the objects in the bucket by including a “/” separator at the end of the resource name, then using an asterisk to indicate it’s a wildcard.

Finally, the last statement uses the “Deny” effect to explicitly deny “s3:GetObject” and “s3:PutObject” actions on all objects in the “secrets/” prefix in our bucket. Note that this explicit “Deny” overrides the permissions granted in the previous statement.

In this example, you can see how you can write multiple statements in a single IAM policy. You also can see how an explicit Deny of a particular action can override a separate statement that Allows the same action.

S3 IAM Permission Best Practices

Now that we know the basics of IAM policies and statements, let's cover a few of the best practices.

Before we dive into specific recommendations, the first and most important practice is to follow the Principle of Least Privilege. This is a general best practice across information security in which a particular user should be given only the privileges which are required to perform its given function. When applied to S3 policies, this means don't give a user read and write permissions when only read permissions are needed, and don't give access to an entire bucket if only a specific prefix is needed.

To put it short - always give minimal required permissions.

With this general principle in mind, here are a few additional tips.

#1 Be Careful with Wildcards

You can use an asterisk - "*" - in the Action and Resource properties in your IAM statements. These asterisks may seem like an easy shortcut to quickly write your IAM statements, but they can easily create security holes.

Wildcard characters, such as * and ? can be used to provide the given action to all resources or items

You should avoid using wildcards in the Action property at all. It may be tempting to use "s3:*" to provide full access to S3, but it's rare that a user will actually need all of those permissions. There are only 51 total actions within S3, and only a few of them are relevant to most users. It's worth the extra time to be explicit about the actions you want to allow.

In the *Resources* property, the story is more nuanced. If you are granting permissions on multiple objects in an S3 bucket, you will likely need to use a wildcard to specify a particular prefix that a user is allowed to access. It will be infeasible to list each of the specific objects that a user can access and would require frequent updates as new objects are added or deleted. For this reason, it is acceptable to use wildcards for specifying prefixes in S3 bucket resources. You should avoid using wildcards at any other part of the ARN, such as the bucket name.

EXAMPLE

For example, using "arn:aws:s3:::MSP360-examples/uploads/*" to provide access to all objects under the "uploads/" prefix is acceptable. Using "arn:aws:s3:::*" to provide access to all S3 buckets in your account should be avoided.

#2 Split Bucket Statements from Object Statements

Each S3 action applies to either bucket resources or object resources.

For example, the "s3:ListBucket" action applies to buckets only and allows a user to list all of the objects in the bucket. The "s3:GetObject" action applies to objects only and allows a user to read the content of an object and its associated metadata. AWS's list of S3 Actions does a nice job of showing whether an action applies to buckets or objects.

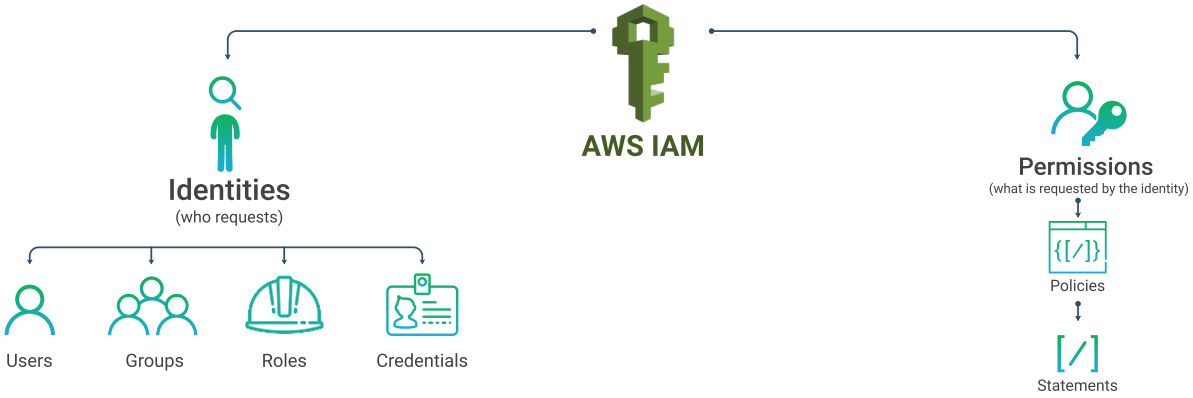
> **When writing IAM policies, keep your bucket statements separate from your object statements. It will keep your statements narrowly tailored and will be easier to reason about the effects of making changes.**

#3 Use AWS Managed Policies

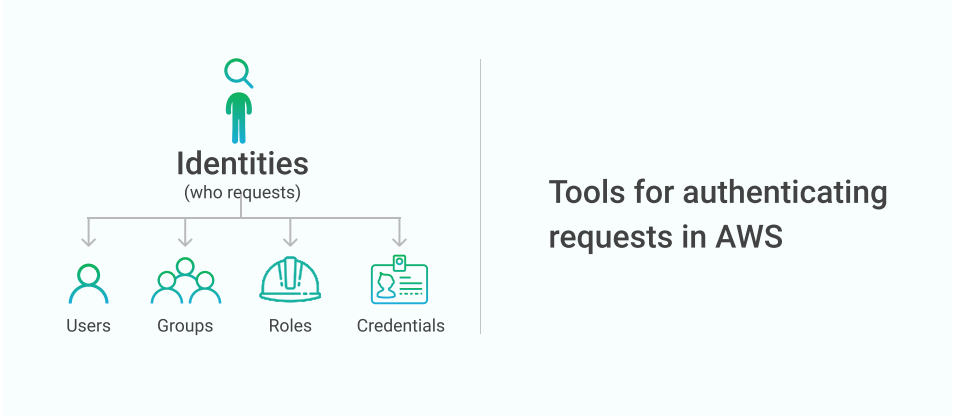
AWS provides AWS Managed Policies which are policies created by AWS to fill a specific use case. Often you will need to provide some S3 access as part of using other AWS services, such as Amazon Redshift for data warehousing, Amazon SageMaker for machine learning, or AWS Lambda for event-driven compute. Using AWS managed policies with these services can help you get started quickly while still using least-privilege permissions.

Part 3 - Mastering Amazon S3 Identities

In Part 2, we noted that IAM has two broad categories: ***identities*** and ***permissions***. Permissions describe which actions a particular identity may or may not perform. These permissions are managed in IAM policies, and AWS allows you to write fine-grained policies to lock down access to your resources.



The second half of IAM is ***identities***, which are the tools used to authenticate requests within the AWS ecosystem. This topic includes user and group management, authentication credentials, and delegation. IAM identities are the focus of Part 3.



In this Part, we'll cover:

- How AWS handles authentication on API requests;
- IAM users, groups, and roles, and the differences between them;
- Best practices with IAM identities.

Authenticating with Access Keys

When making a request to Amazon S3, your request is either anonymous or authenticated. If your request is anonymous, this means you have not included any credentials used to identify yourself as a user. For your request to succeed, you will need the relevant permissions to be granted via an S3 bucket policy or an ACL.

More commonly, you will make authenticated requests to your S3 resources. Authenticated requests use access keys to identify the requesting user. Access keys are similar to a username and password - there is an *access key ID*, which is comparable to your username, and a secret *access key*, which is similar to your password. AWS uses a request-signing method using the AWS Signature Version 4 algorithm, or "Sigv4" for short. The details of Sigv4 is outside the scope of this e-book, but you can check out AWS's materials on authenticating with Sigv4.

AWS IAM Users and Groups

In the previous section, we learned that AWS uses access keys to authenticate the identity of the entity making a request. In this section, we'll learn about one way to provision access keys by creating an IAM user.

An IAM user is an entity in IAM that should map to an actual human user. When creating an IAM user, you can provision a password that allows the user to sign in to the AWS console. You can also create up to two access key pairs for the IAM user. These access key pairs are most commonly used with something like the AWS CLI for interacting with AWS resources in your terminal or with GUI tools like MSP360 Explorer that help you manage your AWS resources.

By default, an IAM user has no permissions. You can grant an IAM user the ability to access and manipulate resources by attaching *policies* to the user. We covered writing IAM policies in the previous Part 2. Be sure to check the "Use Managed Policies Over Inline Policies" portion of the Best Practices section below **(page 23)** for additional advice on IAM policies.

Often you will have multiple IAM users that need the same IAM permissions. To make this easier, you can create an IAM group. An IAM group is a collection of IAM users. You can attach policies to an IAM group just as you would to an individual IAM user, and the policies will apply to all users in the group.

AWS IAM Roles

An IAM role is similar to an IAM user, with a few key differences.

- First, it is not tied to a particular identity, such as a human user. Rather, an IAM role is intended to be assumed by someone or something (such as an EC2 instance) that has permission to use the role. The person or service that assumes the role will temporarily have all of the IAM permissions of the assumed role.
- Secondly, an IAM role is temporarily assumed, thus has no permanent credentials associated with it. There's no password to sign into the AWS console as an IAM role, and there are no permanent access key pairs for the IAM role. AWS will automatically generate temporary access keys for any person or entity that assumes a role. These temporary credentials will be valid for a specified period of time and will have all permissions assigned to the role.

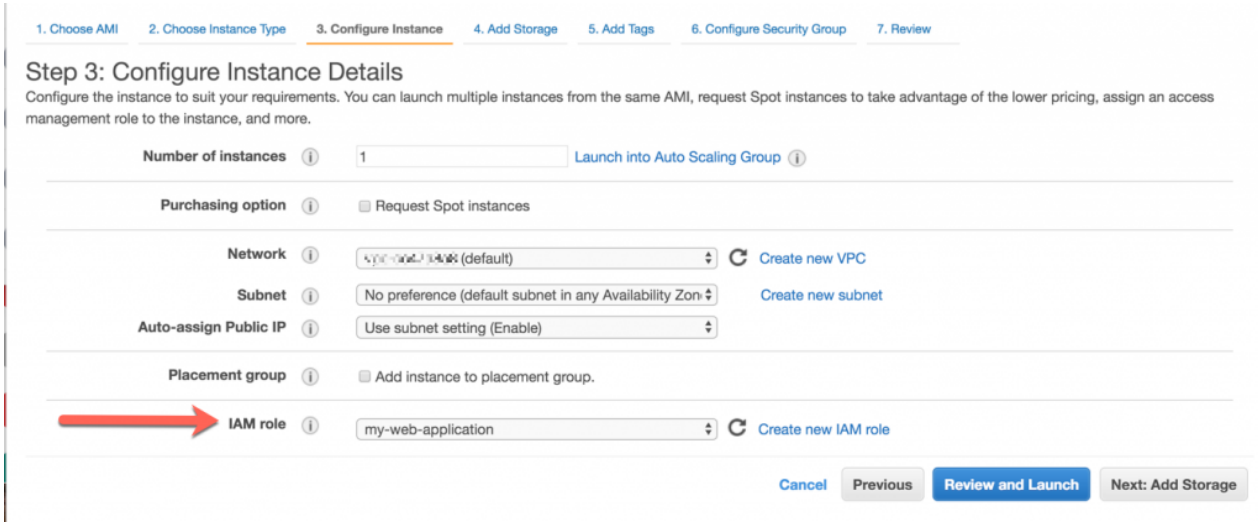
IAM roles can be unintuitive at first, so let's walk through two examples of where IAM roles prove invaluable.

#1 IAM Permissions for an EC2 Instance

EXAMPLE

First, imagine that you have a user-facing web application that is running on EC2 instances. Your application will save and display images from Amazon S3 and thus will need the proper IAM permissions to interact with your S3 resources. Without IAM roles, you would need to create an IAM user and generate an access key pair. Then you would need to make sure the access key pair is available to your web application on the EC2 instance. If you're automatically creating and destroying EC2 instances as your application scales up and down, you'll need to determine a secure way to store your credentials that your new instance can access on boot. Finally, you'll need to manage the rotation of those access keys without causing downtime to your users.

With IAM roles, this process is much easier. You can associate your EC2 instance with an IAM role upon creation, as shown in the screenshot below.



Your EC2 instance will have access to temporary access credentials for making requests to AWS services. These credentials will have the permissions given to the IAM role you delegated to the EC2 instance. You no longer need to worry about access key management or key rotation on your instances - it's all automatic.

To retrieve these credentials, your EC2 instance will query the instance metadata service. The instance metadata service is an HTTP-accessible service provided by AWS that provides important metadata that is specific to your EC2 instance. The instance metadata service is accessible from each EC2 instance at http://169.254.169.254, and it includes information such as the AMI used to launch your EC2 instance, the public and private hostname of your instance, and, if associated with an IAM role, a pair of temporary access keys for your instance to use for all AWS API requests.

#2 IAM Roles in the AWS Console

IAM roles also can be helpful in the AWS console.

EXAMPLE 1

- To provide different levels of access to a particular IAM user in the AWS console

If you want to force an IAM user to go through an additional level of security before performing important actions. For example, deleting an S3 bucket may require a user in the AWS console to assume a higher role that requires multi-factor authentication. This can ensure that S3 deletion by a bad actor on a browser that was left open.

EXAMPLE 2

- To allow an IAM user to access multiple AWS accounts in the AWS console.

If your organization handles multiple AWS accounts, such as different accounts for development, staging, and production. You allow an IAM user in one account to assume an IAM role in another account. This allows your developers to manage one set of authentication credentials while still maintaining separation between AWS accounts.

Amazon S3 IAM Identity Best Practices

#1 Prefer Roles Over Users

IAM roles simplify many aspects of credential management. By always using temporary credentials, you don't need to worry as much about key rotation and having your credentials fall into the wrong hands. Further, having AWS manage access credentials for you means you don't need to build systems to encrypt and decrypt credentials when they need to be shared across multiple machines.

You can use IAM roles in conjunction with MSP360 Explorer. For details, check out our blog post on assuming an IAM role via MSP360 Explorer.

#2 Don't Share Access Keys

When it is necessary to use IAM users rather than roles, do not share your user's access keys with multiple people. Each person that needs AWS console or programmatic access via the AWS CLI or language-specific SDKs should have their own IAM user with their own credentials.

There are two reasons for this.

- Sharing access keys increases the attack vector for someone to steal the keys. The keys are now stored in more locations that could be left vulnerable.
- It makes it more difficult to monitor usage in your AWS account. If a single pair of access keys is shared across an entire department, it's impossible to find out who used the keys for nefarious purposes in the event of an internal breach.

#3 Rotate IAM User Credentials Regularly

For IAM users with AWS console passwords and/or access keys, you should rotate these credentials regularly. Rotating passwords and access keys will limit the amount of time an exposed credential is available for attack. It also helps you create a nice rhythm of rotating your credentials in the event you need to disable your credentials in an emergency situation, such as during an attack.

Note that if you're using IAM roles, you don't need to worry about rotating access keys. AWS will automatically provide you with temporary access keys when you're using a role and will manage the rotation of these keys.

#4 Use Managed Policies over Inline Policies

When writing IAM policies to grant permissions to an IAM user, you can create managed or inline policies. Inline policies are policies that are attached directly to an IAM user, group, or role. Managed policies are created separately and may be attached to one or more IAM users, groups, or roles.

When possible, you should prefer using managed policies rather than inline policies. Managed policies provide a few benefits.

- AWS provides versioning of managed policies. Whenever you update an existing managed policy, AWS is actually creating a new version of the policy. AWS will retain up to five versions of a managed policy. This makes it easier to rollback to a previous version if you make a mistake as well as to audit any changes by checking the differences between the two versions.
- Managed policies are more reusable. You will often have multiple users or roles that need similar permissions across services. If you add these to a user or role via an inline policy, you would need to update each user or role whenever you need a new permission. This can lead to configuration drift across users and roles and make it more difficult to audit what each user has. By using managed policies, you can update these policies in one place and have it apply to all relevant users and roles.

#5 Monitor IAM User and Role Usage

AWS provides a few different services that allow you to monitor the actual API requests made by particular users in your systems. There are two that are relevant for Amazon S3.

First, AWS CloudTrail is a service that tracks certain high-level API calls across many different AWS services. Generally, CloudTrail works to log API actions that occur at the control layer - deleting an S3 bucket, creating a new DynamoDB table, etc. It usually does not log actions that occur at the data layer - reading an S3 object, inserting an item into a DynamoDB table.

The CloudTrail integration for Amazon S3 is more expansive. By default, CloudTrail will log all bucket-related actions in Amazon S3. However, you can also choose to enable object-level actions, including all object reads and writes. You will be charged for the additional volume, but it can be a great way to get granular data on your S3 access patterns.

The second way to monitor is to use Amazon S3 Server Access Logging. With Server Access Logging, AWS will monitor all requests to your S3 bucket. It will periodically make dump files of the requests. Each request log will include information such as the time, the object requested, the requesting user (if authenticated), the object size, and more. Server Access Logging can be cheaper than using CloudTrail for data layer events, particularly if your Amazon S3 resources are accessed quite frequently.

Conclusion

Now you should have an understanding of the various tools at your disposal and how to avoid some serious mistakes. Security is a complex issue that is never finished, so don't stop learning. Set up the proper processes to allow your team to move quickly while staying secure.

About MSP360

Established in 2011 by a group of experienced IT professionals, MSP360™ provides cloud-based backup and file management services to small and mid-sized businesses (SMBs). MSP360's offerings include powerful, easy-to-use backup management capabilities and military-grade encryption using customer-controlled keys.

Customers can choose to store their backup data with more than 20 online storage providers, including Amazon S3 and Amazon Glacier. MSP360 also collaborates with thousands of VARs and MSPs to provide them with turnkey, white-la bel data protection services. It has been an Amazon Web Services Advanced Technology Partner since 2012. MSP360 has also achieved Storage Competency Partner status in the AWS Partner Network. For more information, visit www.msp360.com. Follow us on Twitter at @msp360.

About AWS

AWS offers over 90 fully featured services for compute, storage, databases, analytics, mobile, Internet of Things (IoT) and enterprise applications from 42 Availability Zones (AZs) across 16 geographic regions in the U.S., Australia, Brazil, Canada, China, Germany, India, Ireland, Japan, Korea, Singapore, and the UK.

AWS services are trusted by millions of active customers around the world monthly – including the fastest growing startups, largest enterprises, and leading government agencies – to power their infrastructure, make them more agile, and lower costs. To learn more about AWS, visit aws.amazon.com