CloudCheckr

Mastering Amazon Identity and Access Management (IAM)

A Beginner's Guide to IAM Policies and Best Practices

Abstract

Amazon Identity and Access Management (IAM) is a powerful tool for delegating permissions, securing information, and auditing user access and sessions on AWS environment. Extensive knowledge of IAM is imperative or secure deployment in the cloud—whether you're just starting out with Amazon Web Services (AWS) or a seasoned veteran with several complex AWS deployments. This guide provides AWS professionals with a detailed overview of the Amazon IAM service, with particular focus on policies and best practices for effective and secure cloud deployment.

Overview of IAM Features

IAM is, without a doubt, the most crucial service in the AWS technology stack. Although it's not used to provision new cloud components (virtual server instances, databases, storage, or networking of any kind), every company that has any kind of deployment in AWS public cloud, no matter the size, should know the ins and outs of IAM. Even if your company only uses AWS to offload data to S3 or Glacier as a backup disaster recovery solution, you should apply Amazon's best practices for IAM in order to secure your investment and protect it from both external and internal threats.

As you may already know, IAM is used to configure your users and groups, give them access through roles and policies, and manage their credentials (passwords or access keys). In addition, with IAM, you can set up federated authentication, in order to utilize other authentication sources as a database to read user data (for example, your on-premises Active Directory or LDAP environment).

You can access IAM with your browser by logging into Amazon Management Console or you can configure IAM settings with the AWS Command Line Interface from the terminal environment in your operating system, and as well with the programming language of your choice utilizing AWS SDK or HTTP API Calls (requests need to be digitally signed with user's credentials).

When you log in to IAM via the console, an overview dashboard summarizes the total number of users, groups, roles, policies, and identity providers. From this start page, you can customize the login link for your AWS environment, as well as review the Security Status IAM checklist, a reminder to enable multi-factor authentication, organize users into groups, implement password policy and rotate your access keys.

Users and Groups

The account that you create when signing up for AWS is your **root** account. It's highly recommended that you protect this account and not use it for accessing AWS services, because root account has unrestricted access to every resource in your environment. For this reason, you need to create other accounts, and give them specific permissions.

When you create a new user, you decide whether the user will have **AWS Management Console** and/or **Programmatic access**. The first option allows a user to access the console through a web browser (and would require generating a user password as well), whereas programmatic access allows you to generate a key pair (an **access key** and a **secret access key**), which can be used to create API calls utilizing AWS SDK. Each key pair is unique, should be kept secure, and never stored on any AWS service directly (hard coded in EC2 instances or stored in RDS/DynamoDB tables).

Groups can be utilized to tie together users with the same permissions. When you add a user to a particular group, the user inherits all the policies assigned to that group.

IAM Policies

Amazon is a strong advocate of the "least privilege" rule, so by default, users and groups are not assigned any permissions when created. If you assign console access to a user, without attaching policies directly to that user or his group, the user will be able to log in to the AWS Management console, and nothing more. You can attach predefined, AWS-managed IAM policies, which are organized based on AWS services, or you can attach a custom policy you created to suit your needs.

In essence, IAM policies are JSON text files. Policies consist of one or more statements, and every statement in turn describes one set of permissions. Inside a single statement, there are three required parts:

- **Effect: Allow** or **Deny** values only, meaning that the policy would give access to the attached entity or completely deny it.
- **Action:** The action of the AWS service to be allowed or denied. Every AWS service has its own group of actions. For example, if you're allowing a user to control EC2 instances, maybe you would like to only allow the user to start and stop the instances, but not create new ones. Or with S3 buckets, you might want to allow the user to view and download files, but not upload new ones. You can define multiple **actions** under this block, however they need to be tied to the same service.
- **Resource:** Where the **effect** and **action** will be applied. For example, you would like to **allow** a user full access to S3 service, but only for specific buckets. In that case, you would define your S3 buckets in the **Resource** block inside a policy statement.

Let's review a sample policy that allows the user full access to all S3 buckets:

```
{
 "Version": "2012-10-17",
 "Statement": {
 "Effect": "Allow",
 "Action": "s3:*",
 "Resource": "*"
 }
}
```

Above the actual policy statement, it's necessary to define the **Version** element, which specifies the policy language version (in this example, 2012-10-17). This date should appear in all of your policies until the policy language version changes.

Star (*), as in glob syntax, is used to replace all supported values. In this policy, it means that it will allow all actions on all S3 buckets created under your account. If you would like to specify only certain S3 buckets in your AWS account or even folders under those S3 buckets, you need to use **AWS Resource Name** (ARN) format.

ARNs uniquely identify AWS resources and are defined in the

```
arn:partition:service:region:account:resource format.
```

- **arn:** is mandatory at the beginning of the definition.
- **Partition** is used to define a partition where your resource is located. In most cases, you'll just use **aws** here, but sometimes you might need to change partition (for example, if you're deploying resources in China or US Government partitions).
- **Service** and **Region** identify the **service** and **region** under which you're organizing your resources.
- **Account** is only used if you're trying to reach a resource in one of your cross-accounts. For example, if you're writing a policy that will allow users in your development AWS account to read files from an S3 bucket tied to a production account.
- **Resource** is actually the resource you're trying to reach. With S3, that would be the name of a bucket or folders inside a bucket.

For example, an ARN for accessing everything in the **Test** folder under the **MyBucket** S3 bucket would be defined as follows:

```
arn:aws:s3:::MyBucket/test/*
```

ARNs also support wildcards, so if you would like to allow your users to access all **test** S3 buckets, you can use the `arn:aws:s3:::test*` to define your ARN. Note that if you would like to target any other service, you would need to change the **Service** part of the ARN. The **Region** definition is only needed if you're targeting a region-based service (for example, you're enabling access to EC2 instances inside one specific region).

Once you have finished writing your custom IAM policy, use IAM Policy Validator to check for syntax errors. If all is in order, attach the policy to an entity (user, group or a role) to start using the service with the required permissions.

IAM Roles

Besides users and groups, policies can also be attached to Roles, which are typically used when you want to assign a certain permission to an AWS service, or for cross-account access or access through an identity provider. Since there is no actual login process tied with roles, key pairs used through them are dynamically generated and maintained by AWS, so there is no security risk when using roles.

The most common scenario where you would use roles is with EC2 instances. EC2 instances can have only one role, but you can attach multiple policies to an EC2 role. With an attached EC2 role, you can change the interaction of an EC2 instance with other AWS services on-the-fly through policies tied to the role itself.predefined, AWS-managed IAM policies, which are organized based on AWS services, or you can attach a custom policy you created to suit your needs.

Identity Federation

If you already have a directory service in your on-premises environment, you can use it in the Amazon cloud as well, with the help of identity federation providers. Your federated users will use their current credentials, and permissions in the AWS cloud will be implemented using roles. AWS supports SAML 2.0 (required for LDAP or Active Directory) and OpenID Connect (OIDC) as identity providers, which means you can also authenticate Google or Facebook users.

Best Practices for IAM

Once AWS customers start to use fine-grained IAM policies attached to users, groups or roles, they soon realize how complex IAM is and how much power it gives to the user. They also discover the extent to which it can become a security liability. Misuse

of IAM can severely damage your cloud infrastructure, so be sure to follow IAM best practices as best you can.

- **Always assign permissions using the least privilege rule.** Whenever you create a new policy or modify an existing one, make sure it allows least permissions to an entity. Even if it means that you will have to go back and forth to tailor and test the needed access rights—don't give higher permissions than truly necessary. Even if your AWS environment is maintained by a group of senior IT engineers, not all of them need to have full administrator access.
- **Use roles as much as you can.** Never store credentials in EC2 instances, and always utilize roles instead of key pairs. Even if your developers or sysadmins need API access, it's smart to set up a bastion EC2 host, with required EC2 roles and audit access through that protected jump server.
- **Always use complex passwords.** Implement a complex password policy for your IAM users, and utilize the same approach for any passwords you define in your AWS cloud. Try not to use dictionary words, include small and capital letters, numbers and special characters, and keep your passwords long. Utilize a protected password safe to store all the data, including service accounts.
- **Rotate credentials on a regular basis.** If you need to provide key access, or you have multiple users utilizing the AWS Management console, rotate all given credentials on a schedule. Force users to change their sensitive login data every couple months, and always disable expired credentials.
- **Protect root account.** The root account should only be used while signing up and delegating access for the first time. Access to the root account should be kept safe and protected with multifactor authentication, which should be considered for all privileged accounts (with access to production services, for example). Don't create access keys for a root account, set up another account instead.
- **Monitor your IAM usage.** There are several AWS services that can help you with monitoring your IAM users and what they do when they access your cloud infrastructure. As soon as you can, enable AWS CloudTrail and AWS Config. CloudTrail will log all AWS API calls tied to your account, and will store them inside an S3 bucket. AWS Config serves as chronological log of all configuration changes in your AWS environment. It's also smart to set up custom alarms in AWS CloudWatch to get notifications on the changes and metrics you define. In addition, third-party tools can be leveraged to audit your IAM permissions and user actions.

Conclusion

Following best practices for IAM is not the only thing you need to do to secure your AWS environment, but it's a good place to start. Once you have all your access locked up, we recommend you refer to AWS Security resources for information on how to secure other AWS services in your AWS cloud. While you can never be 100% protected in today's dynamic IT landscape, adopting tools and best practices to implement and enforce security controls can ensure cloud infrastructure remains secure.

Need CloudCheckr for your organization?
Learn more at www.cloudcheckr.com

CloudCheckr

About CloudCheckr

We deliver total visibility—across multiple public clouds and hybrid workloads—making the most complex cloud infrastructures easy to manage. From government agencies to large enterprise and managed service providers, CloudCheckr customers deploy our SaaS-based platform to secure, manage, and govern the most sensitive environments in the world. Our industry-leading products include CloudCheckr CMx, CMx High Security, CMx Federal, and CloudCheckr Finance Manager for cost management, billing & invoicing, cloud security, compliance, inventory & utilization, and cloud automation.

CLOUDCHECKR.COM

© CloudCheckr. All Rights Reserved. Updated December 23, 2020