Managing identities and access in the Amazon Public Cloud

Aristide Bouix MSc
specialized in IAM at KPMG Cyber and currently works as a cyber specialist at Catawiki.
a.bouix@catawiki.nl

Public Cloud is often associated with innovation and new capabilities for faster product delivery. However, aside from these fancy Cloud capabilities which allow to easily enable Machine Learning in applications or broadcasting over a satellite network, the truth is that the applications and the data they contain remain an API call away from being breached. Compared to the Microsoft Cloud, the Amazon Cloud isn't directly compatible with on-premise Active Directories and relies on its own attribute-based identity engine called AWS IAM. This publication tackles the different ways of managing access on the AWS platform with associated tooling and best practices as well as how to implement them successfully.

INTRODUCTION

For decades, the first line of security controls of the traditional IT Data Center was the network access through physical and then virtual Local Area Network isolated by firewalls, which allowed IT personal to jump from one network to another. Once this first line of defense was passed, users were confronted with the application interface for authentication.

In that sense, the Data Center environment has been built around network segregation and isolation before adopting a unified user identification approach. Based on this aspect, Public Cloud environments are an entirely different paradigm as they are, by definition, a shared pool of resources logically isolated per customer. The term Public Cloud literally means any worldwide user can access this pool of shared resources and use it. As a result, all of these resources should be Internet-facing.

To prevent the inadvertent exposure of confidential data and services, Public Cloud providers offer three main layers of protection:

- Layer 1: The Cloud Default configuration
- Layer 2: Identity and Access Management permissions
- Layer 3: Data encryption (At rest/in-transit)

Layer 1 and 3 are vast topics that span over the approximately two hundred AWS services, and they won't be covered in the current publication. The second layer is highly dependent on the Cloud Platform itself as each provider was free to develop and provide its own vision of what a modern authentication and authorization service should be. The rest of this Compact article will solely focus on the Amazon Web Services Cloud Platform IAM specificities.

AWS IAM THE BASICS

In contrast to Azure, where a strong distinction is made between Azure Active Directory, part of the Tenant, and other Azure resources, part of a Subscription, AWS IAM is a service in which access to API is managed the same way as for any other service (see Figure 1). While this approach provides an excellent granularity on the platform configurability, it also makes a Role-Based Access Control strategy for a specific set of resources more challenging than on Azure, where resources are literally encapsulated in a logical abstraction called Resource Group.

Because of these specificities, AWS earlier recommended that its customers divide their resources into as many AWS Accounts as possible according to their maturity, the most basic approach being having a minimum of three Accounts for the development, testing, and production environments. Later, subdivisions can be created at the department level and then at the application level. By default, an AWS Account is a fully closed and isolated environment with no access to resources outside of it. For the rest of this publication, we will refer to an AWS Account with a capital letter to prevent confusion with any other user account type.

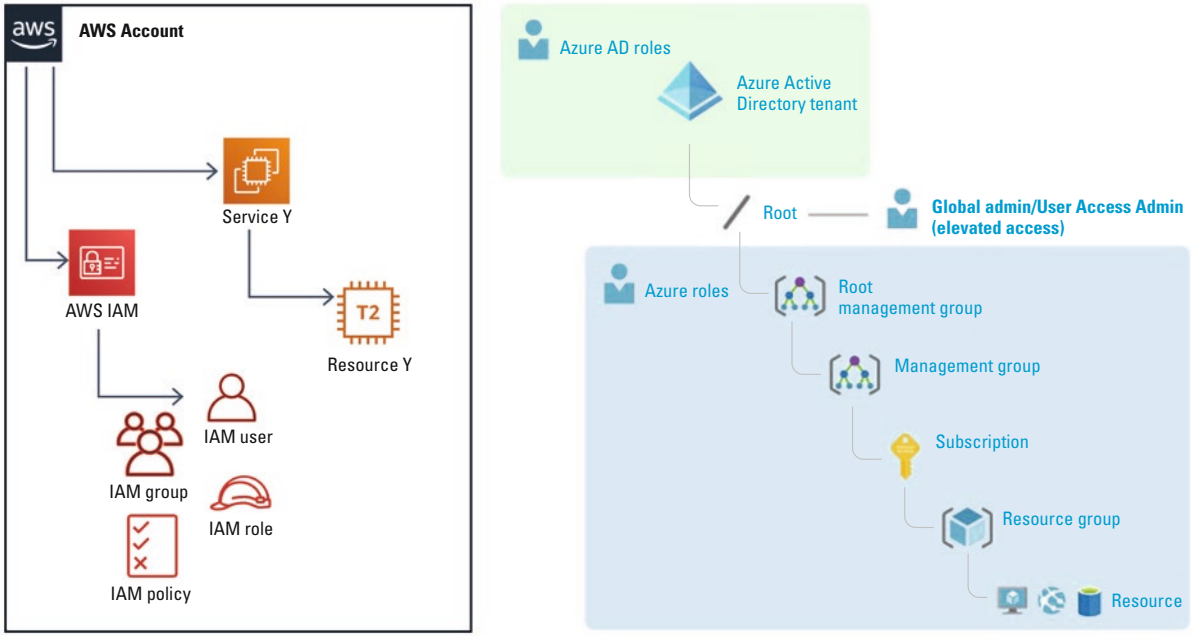
When you create a new AWS Account, AWS initially provides a user account called Root, which isn't bound to IAM restrictions. The Root account is set to facilitate the onboarding to AWS. However, as this account isn't identity-based and cannot be restricted, the first security rule from AWS is to set a strong password as well as an MFA to this account, create a first user with administrator permissions, backup Root account credentials in a safe and redounded manner and then only use them for rare actions that cannot be done with a standard IAM access ([AWS21]).

Similar to the Cloud environment itself, the AWS IAM engine is a public Representational State Transfer (RESTful) native service. It has a flat-file structure instead of an organizational directory and is compatible with HTTP(S) based authentication protocols such as Security Assertion Markup Language (SAML), OpenID, or OAuth. There are four main categories of resources that can be created through the AWS IAM service *users*, *groups*, *roles*, and *policies*. In the following section, we will discuss each of these resources to provide an extensive overview.

Users are identities created inside of the AWS IAM service in a specific Account. They can be made for a natural user such as a developer and sysadmin or for a technical user such as an external application (service account).

A Group isn't an identity but a way to attach policies to multiple users. This is quite handy if you wish to enforce Role-Based Access Control (RBAC) and give a standard baseline set of permission to each user category. For instance, all company developers would be members of the developer Group and all system administrators would be members of the sysadmin Group.

Figure 1. Logical organization in AWS vs Azure.



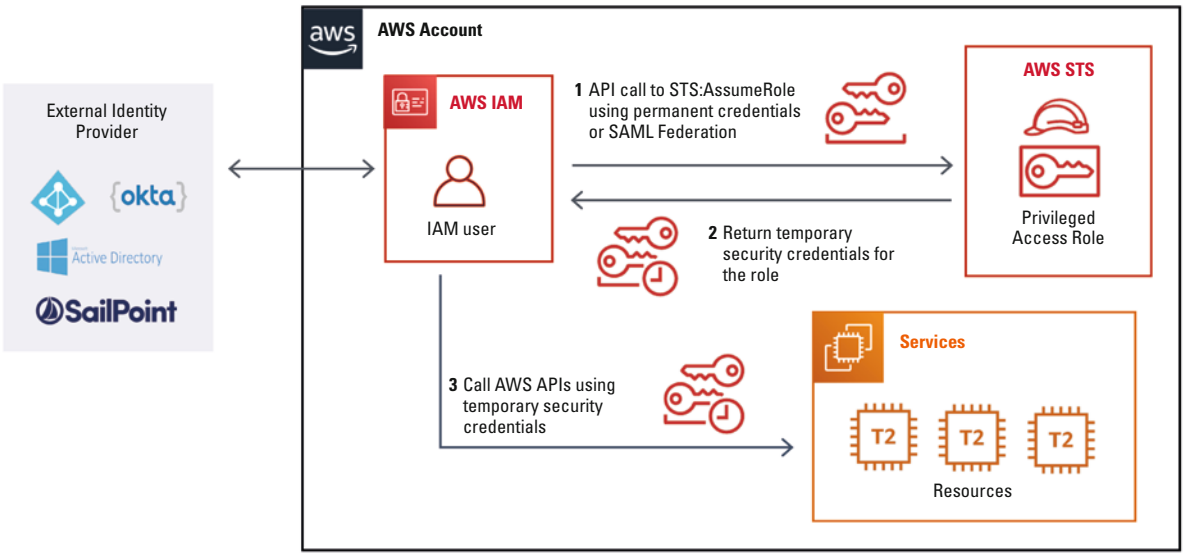


Figure 2. Interaction between roles and the STS service.

The inconvenient part of relying on IAM user identity in a specific account is that you need to provide each user with a set of permanent credentials that can inadvertently be leaked and are difficult to audit. For these reasons, Amazon created another type of identity resource called roles. A role is an AWS identity that can be assumed by a user or resource defined inside of AWS or by an external identity provider using federation (SAML). Unlike IAM user resources, IAM roles rely on the Security Token Service (STS) to obtain credentials with a validity period that can be adjusted between 1 and 12 hours. The interaction between a role and STS is shown in Figure 2.

We will now discuss the last but not least element from the AWS IAM ecosystem: the AWS policies. As discussed previously, these are flat JavaScript Object Notation (JSON) files with specific fields allowing or denying actions on the AWS API of selected AWS services with potential conditions and resource restrictions. There are three primary types of policies:

- Trust policies are attached to a role to define which Principal (AWS account, identity, or service) can assume it.
- Identity-based permission policies are attached to an identity (user or role) to define which API calls are allowed or denied on which resources.

Figure 3. Example of an IAM policy applied to an S3 bucket.

```
{
    "Version": "2012-10-17",          // Only two existing versions, use the latest
    "Id": "some-unique-id",   // (Optional) - Id required for a few AWS services
    "Statement":
    {
        "Sid": "1",     // (Optional) - Id for each individual Statement [Not API exposed]
        "Effect": "Allow",     // Allow or Deny
        "Principal": {"AWS": "arn:aws:iam::111222333444:user/colonel"},  // Who can use this policy, could be a whole AWS account //
        "Action": [
                "s3:PutObject",
                "s3:Get*"                  // What API are locked or unlocked by this policy
            ],
        "Resource": "arn:aws:s3:::kfc-bucket/*",     // Which specific AWS resource or group of resources this access applies
        "Condition": {
                    "DateGreaterThan": {  // Condition for this policy to take effect, here it is from 1rst of January 2022
                                "aws:CurrentTime": "2022-01-01T00:00:00Z"
                                }
                }
    }
}
```

- Resource-based policies are a combination of Trust and Permission policies that enable a principal to perform certain operations on a resource. (For instance, a Simple Storage Service [S3] web storage container or a Key Management Service [KMS] encryption key.)

Figure 3 shows an example of S3 resource-based policies with all supported attribute fields.

In every new AWS Account, AWS provides a maintained list of AWS-managed permission policies to swiftly roll out minimal access control to services and new features. While these policies are a great start to implement a RBAC strategy, they tend to be overly permissive (as they are not resource specific) and shouldn't be trusted entirely. To give a concrete example, in the past, some of these policies were successfully hijacked by security researchers as the starting point for privilege escalation ([Witt19]).

In this context, it is highly recommended to create customer-managed policies that give access to specific AWS services or resources (e.g. EC2, Lambda, ECS), especially for a service account for which activities can be scoped more easily.

SCALING AT THE ORGANIZATION LEVEL

In the previous section, we've learned about the logic behind Amazon's Identity Access Management engine and its components. In this section, we will assemble them into an IAM strategic roadmap.

As we said at the beginning, the best possible isolation in AWS is between different Accounts. However, a remaining issue is that it is then more challenging to enforce a similar degree of compliance in decentralized managed environments. To remediate it, AWS released the AWS Organization in February 2017, initially to enable more straightforward consolidated billing and management between accounts. The service has been progressively extended to support Service Control Policies, which are IAM policies at the account level, backup and tagging policies, centralized Firewall, and cross-account Infrastructure As Code service (CloudFormation Stack-Sets).

It is not recommended to enforce top-down policies using the AWS Organization master account, as it would introduce a new single point of failure if one of the account administrators gets compromised, ruining all the gains of a multi-account AWS setup. Nevertheless, the AWS Organization remains a convenient way to provide a minimum security baseline on a set of AWS Accounts, for instance, by enabling and redirecting all activity logs in a restricted sub-account and preventing IAM identity resources to modify it. For larger environments, we could even suggest having several AWS Organizations, which would enable a higher level of flexibility and isolation between teams working on different products.

To establish a multi-account Organization, you need to define and implement what AWS calls a Landing-Zone. The main characteristics of a Landing-Zone are to unify user authentication and permission provisioning between accounts using AWS SSO and to set an account vending machine service to provision new AWS Accounts in self-service. AWS SSO works similarly to IAM but scales on multiple AWS Accounts to provide a seamless experience. It can be federated to Active Directory and other Identity providers through the Security Assertion Markup Language (SAML) protocol and results in a much more effective user management practice cross-account. Typically, the AWS SSO service has configured this in the AWS Organization master account.

The original landing zone blueprint was released by AWS in 2018 ([AWS18]). There are a few ways to implement it, the most convenient is to have it tailored to your need by AWS Consulting Services or by one of their privileged partners.

Another option could be to deliver it yourself. While deploying such an environment requires a good knowledge of AWS and your corporate needs, Gruntwork published a step by step Terraform guide that constitutes a solid foundation ([Grun20]).

The last option is to use the relatively new AWS Control Tower service, which was released in June 2019 and simplifies the whole process. However, Control Tower currently presents several limits. It is a very clickOps service with little API support. It doesn't support Guard Duty and offers very few options to configure initial resources inside a new Account, such as Virtual Private Clouds (VPC). For all these reasons, this last option is recommended for mid-size organizations with a tight budget and/or engineering capabilities.

A minimal setup would contain at least five AWS Accounts: Development, Testing, Production, Shared Services, Logging and Security.

According to where your user identities are defined, you should also update the password policy to make it fit your organization's requirement as well as to generalize the use of MFA tokens. AWS supports the U2F – standard, although only one device can be registered per user –, virtual One-Time Password (e.g. Google Authenticator, Authy), and a few hardware OTPs.

Before closing this section, please note that while the process to provision new AWS accounts is relatively

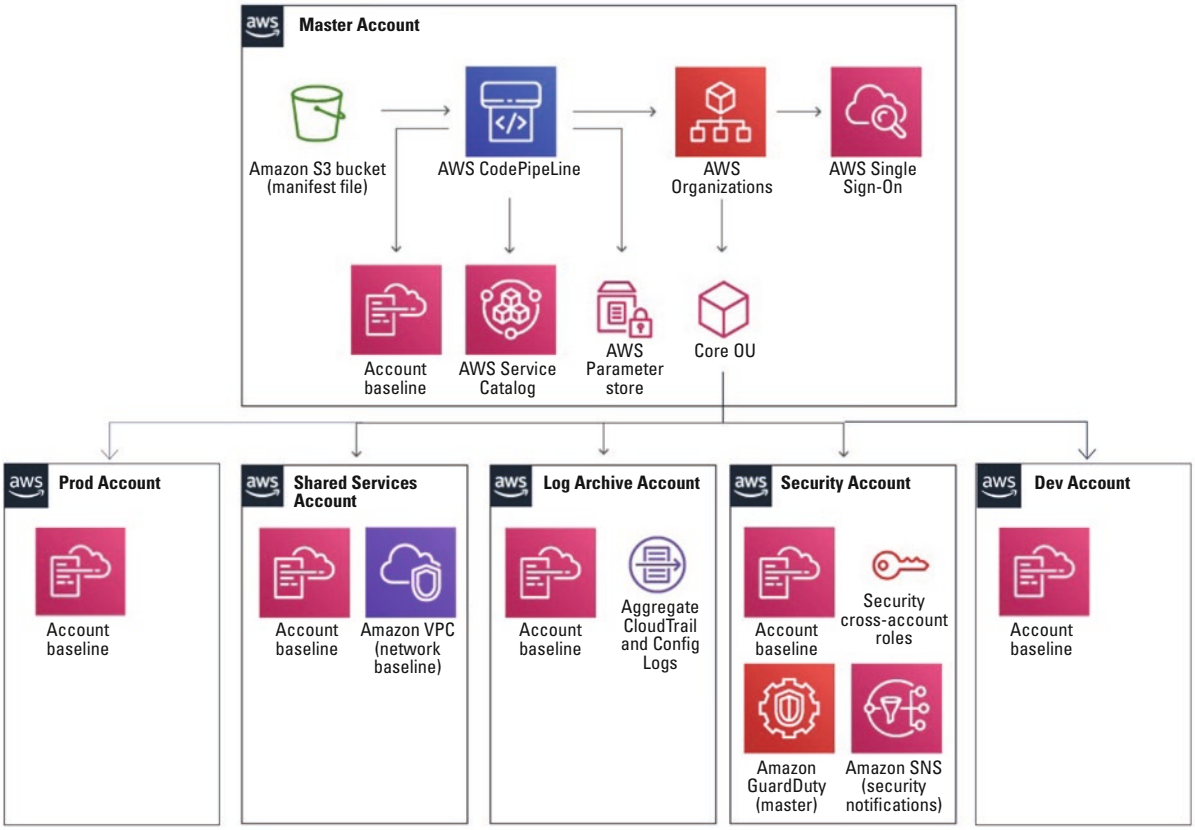


Figure 4. Overview of an AWS multi-account organization's structure.

straightforward, the steps to delete them is quite slow and manual ([Mcka19]). A proper organizational process is therefore needed to request them.

OPTIMIZED TOWARD ZERO-TRUST AND LEAST PRIVILEGE

By now, you should have a multi-Account AWS infrastructure federated to your identity provider of choice. RBAC IAM roles are present in each Account, so end-users can assume them seamlessly through the AWS SSO service.

There are still a few crucial elements that may be configured to harden your deployment. In our opinion, the first one is to secure the static and permanent user credentials for API access.

While these credentials are very handy to improve your Cloud Engineering crew's work efficiency, MFA doesn't apply to them by default, and it may be frightening to imagine that any hacker browsing the download folder of one of the engineers could potentially compromise the production environment with little data available to differentiate the attacker from the normal user. Even though AWS will directly send an email to your account contact email when they detect leaked credentials, a careful attacker with sufficient privileges can easily obfuscate their activities on the platform ([Boui18]).

To mitigate this risk, it is best practice to attach a specific policy to all users that force them to set an MFA and to enter it before using any API ([AWS20]). By default, the session token returned by the Session Token Service (STS) has a validity of one hour, this isn't usually a concern, but when using an MFA, it means your engineers will have to re-enter it every 60 minutes. Therefore, it is a good idea to extend this duration to at least 4 hours, the maximum being 36, to cover half a working day. Unfortunately, this feature cannot be easily deployed with Universal 2nd Factor (U2F) MFAs, as AWS' Command Line Interface (CLI) does not support these MFAs natively at the moment. For U2F devices supporting the generation of One Time Passwords (OTP) such as the Yubikey, Awsu ([Kreu19]) can alternatively be used as a mapper to get a similar result.

Now, all your users need to authenticate with an MFA before performing a change. It is also possible to explicitly request them to re-enter their MFA to perform actions on certain resources using resource-based policies. For instance, to prevent a user from deleting objects in a specific S3 bucket such as the one used to store security log data.

It is always a good thing to have users strictly authenticated, but how about authorizations? We will address this point by proposing 3 levels of maturity and a few well-maintained opensource solutions to enforce them:

- **Level 1:** Rely on AWS managed policies, which unfortunately contain many wildcards meaning they are not resource-restricted. It means that, by default, these permissions apply on all corresponding resources in an AWS Account.
- **Level 2:** Audit users' API access with a tool, such as Cloudmapper IAM reports, and use another tool, such as Cloud Sentry, to craft scope-restricted managed policies based on the activity reports.
- **Level 3:** Audit policies with Cloudsplaining to verify they don't allow privilege escalation or data exfiltration and then use IAMCTL to confirm that all policies have been correctly replicated across all AWS Accounts.

However, the case of resource-based roles and credentials relied on by external service account-like applications, is still to be addressed. Similarly, like for users, you can view the services that are accessed in Cloudmapper IAM reports. For further insights into specific accessed resources, you can also enable the AWS X-ray recording service in custom applications code using the AWS Software Deployment Kit (SDK). While X-Ray is primarily intended for application performance optimization, it is a handy tool to view which API calls are used on which specific resources.

For company-wide policies such as enforcing MFA, the AWS Organization service using Service Control policies (SCP) is a better choice. However, it is advised to restrict the number of such policies as the associated error messages for access denial don't link back to the corresponding policy, which can be a source of frustration and escalation for Cloud infrastructure and operational teams.

There is a last area of control that we haven't discussed, which is access restriction at the workload or Resource Group level using tags. Tag-based access control is a relatively new feature announced during Re:Inforce 2019 ([John19]). It introduces conditions on a tag in IAM policies so a user could only perform actions on resources with the corresponding tags. While this sounds like a notable improvement feature, its current implementation has many limitations. Tags have limited support across AWS services, and not all services supporting tagging support tag policies. Furthermore, as the feature is rolling out, several security flaws are brought to light ([Mcka20]). It is therefore too early to recommend tag-based access control for a production environment.

CONCLUSION

Hopefully this publication has given you some insight into AWS's primary resources to manage identities and access and maximize the value you can get out of them.

You probably now know how to set up your multi-Account strategy and potentially federate them to your usual identity providers. And, last but not least, you probably know how to optimize IAM across your Accounts for fine-tuned user and service authentication and permission.

Unfortunately, we cannot dive into the detail of all those components' rollout in a single publication. Nevertheless, the previously enumerated knowledge should be more than sufficient to implement your usual IAM compliance requirements and beyond.

References

[AWS18] AWS (2018). AWS Landing Zone. Retrieved from: https://aws.amazon.com/solutions/implementations/aws:-landing-zone/

[AWS20] AWS (2020). How can I enforce MFA authentication for IAM users that use the CLI? Knowledge Center. Retrieved from: https://aws.amazon.com/premiumsupport/knowledge-center/mfa-iam-user-aws-cli

[AWS21] AWS (2021). General Reference Documentation. AWS account root user credentials and IAM user credentials. Retrieved from: https://docs.aws.amazon.com/general/latest/gr/root-vs-iam.html#aws_tasks-that-require-root

[Boui18] Bouix, A. (2018, 24 September). Penetration Testing on AWS. Retrieved from: https://aristidebouix.cloud/en/2018/09/penetration-testing-on-aws/index.html/

[Grun20] Gruntwork (2020). How to configure a production-grade AWS account structure using Gruntwork AWS Landing Zone. Retrieved from: https://gruntwork.io/guides/foundations/how-to-configure-production-grade-aws-account-structure/

[John19] Johnson, B. (2019). RE:INFORCE SDD350-R: Scale permissions management in AWS with attribute-based access control. Retrieved from: https://www.slideshare.net/AmazonWebServices/scale-permissions-management-in-aws-with-attributebased-access-control-sdd350r-aws-reinforce-2019

[Kreu19] Kreuzwerker GmbH (2019). AWSU's Github repository. Retrieved from: https://github.com/kreuzwerker/awsu

[Mcka19] Mckay, I. (2019, 26 June). Automating AWS Account Deletion. Retrieved from: https://onecloudplease.com/blog/automating-aws-account-deletion

[Mcka20] Mckay, I. (2020, 29 September). Security September: Still Early Days for ABAC. Retrieved from: https://onecloudplease.com/blog/security-september-still-early-days-for-abac

[Witt19] Wittig, M. (2019, 11 June). AWS SSM is a trojan horse: fix it now! Retrieved from: https://cloudonaut.io/aws-ssm-is-a-trojan-horse-fix-it-now/

About the author

Aristide Bouix MSc works as Core Security Engineer at Catawiki B.V. Fond of Cloud, Open Source and new technologies, he leads the implementation of security controls on Catawiki's Auction platform.