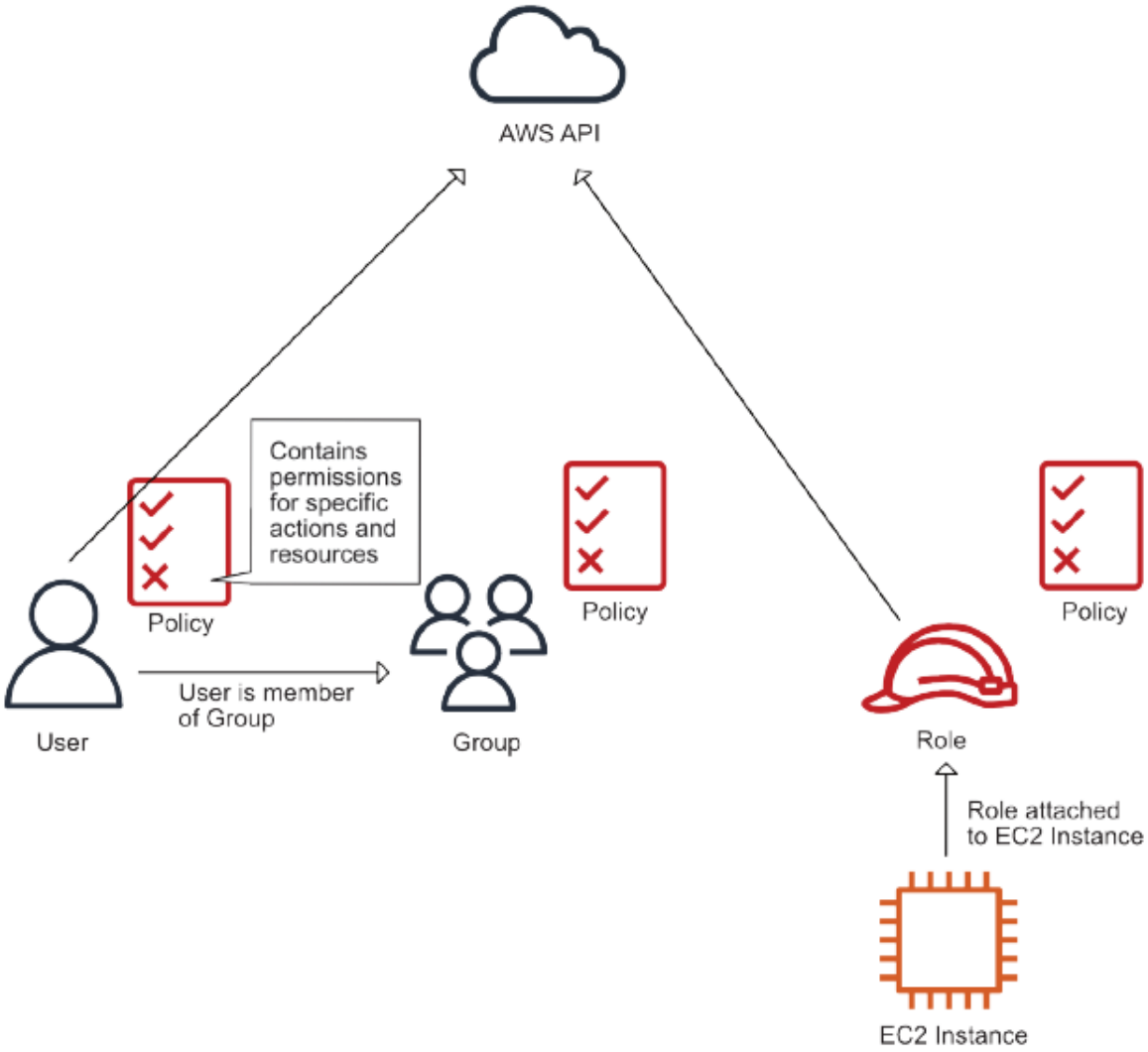
IAM Identity Policy

Defining permissions with an IAM identity policy

An IAM identity policy is used to define the permissions for a user, group, or role.



By default, users and roles can't do anything. You have to create an identity policy stating what actions they're allowed to perform. IAM users and IAM roles use identity policies for authorization. Let's look at identity policies first.

By attaching one or multiple IAM identity policies to an IAM user or role, you are granting permissions to manage AWS resources. Identity policies are defined in JSON and contain one or more statements. A statement can either allow or deny specific actions on specific resources. The wildcard character * can be used to create more generic statements.

The following identity policy has one statement that allows every action for the EC2 service, for all resources:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": "ec2:*",
"Resource": "*"
}]
}
```

1. Specifies 2012-10-17 to lock down the version
2. This statement allows access to actions and resources.
3. Any action offered by the EC2 service (wildcard *)
4. On any resource

If you have multiple statements that apply to the same action, Deny overrides Allow. The following identity policy allows all EC2 actions except terminating EC2 instances:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": "ec2:*",
"Resource": "*"
}, {
"Effect": "Deny",
"Action": "ec2:TerminateInstances",
"Resource": "*"
}]
}
```

- Action is denied
- Terminating EC2 instances

The following identity policy denies all EC2 actions. The ec2:TerminateInstances statement

isn’t crucial, because Deny overrides Allow. When you deny an action, you can’t allow that action with another statement:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Deny",
"Action": "ec2:*",
"Resource": "*"
}, {
"Effect": "Allow",
"Action": "ec2:TerminateInstances",
"Resource": "*"
}]
}
```

- Denies every EC2 action
- Allow isn’t crucial; Deny overrides Allow.

So far, the Resource part has been set to * to apply to every resource. Resources in AWS have an Amazon Resource Name (ARN); figure 5.7 shows the ARN of an EC2 instance. Figure 5.7 Components of an Amazon Resource Name (ARN) identifying an EC2 instance.

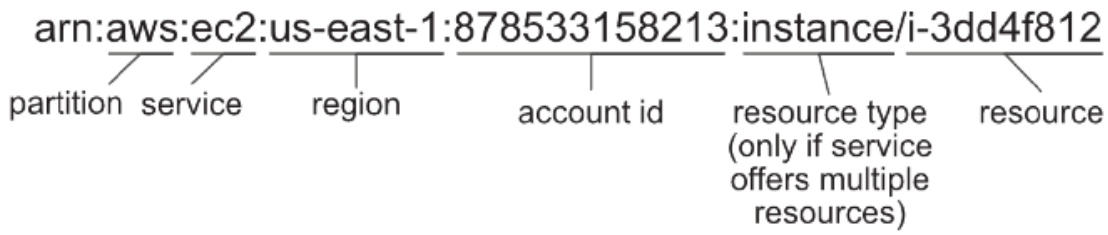


Figure 5.7 Components of an Amazon Resource Name (ARN) identifying an EC2 instance

To find out the account ID, you can use the CLI:

aws sts get-caller-identity --query "Account" --output text

111111111111

Account ID has always 12 digits.

If you know your account ID, you can use ARNs to allow access to specific resources of a service:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": "ec2:TerminateInstances",
"Resource":
[CA] "arn:aws:ec2:us-east-1:111111111111:instance/i-0b5c991e026104db9"
}]
}
```

There are two types of identity policies:

1. **Managed policy—**If you want to create identity policies that can be reused in your account, a managed policy is what you're looking for. There are two types of managed policies:
    - **AWS managed policy—**An identity policy maintained by AWS. There are identity policies that grant admin rights, read-only rights, and so on.
    - **Customer managed—**An identity policy maintained by you. It could be an identity policy that represents the roles in your organization, for example.
2. **Inline policy—**An identity policy that belongs to a certain IAM role, user, or group. An inline identity policy can't exist without the IAM role, user, or group that it belongs to.

With CloudFormation, it's easy to maintain inline identity policies; that's why we use inline identity policies most of the time in this book. One exception is the mycli user: this user has the AWS managed policy AdministratorAccess attached.

IAM Components

- **Users for authentication, and groups to organize users**
- **Authenticating AWS resources with roles**

Users for authentication, and groups to organize users

A user can authenticate using either a username and password, or access keys. When you log in to the Management Console, you're authenticating with your username and password. When you use the CLI from your computer, you use access keys to authenticate as the mycli user.

You're using the AWS account root user at the moment to log in to the Management Console. You should create an IAM user instead, for two reasons.

- Creating IAM users allows you to set up a unique user for every person who needs to access your AWS account.
- You can grant access only to the resources each user needs, allowing you to follow the least privilege principle.

To make things easier if you want to add users in the future, you'll first create a group for all users with administrator access. Groups can't be used to authenticate, but they centralize authorization. So, if you want to stop your admin users from terminating EC2 instances, you only need to change the identity policy for the group instead of changing it for all admin users. A user can be a member of zero, one, or multiple groups.

Creating IAM users allows you to set up a unique user for every person who needs to access your AWS account.

Sign in to the AWS Management Console:

Go to the AWS Management Console at https://console.aws.amazon.com/ and sign in using your root account credentials.

Access the IAM Console:

Once logged in, navigate to the IAM console by typing "IAM" in the search bar or selecting "IAM" from the list of services under "Security, Identity, & Compliance."

Navigate to "Users":

In the IAM console, select "Users" from the left-hand navigation pane. This will display a list of existing IAM users in your account.

Click "Add user":

To create a new IAM user, click the "Add user" button at the top of the page.

Enter User Details:

In the "Set user details" section, enter the username for the new IAM user. You can also choose whether the user will have programmatic access (access key ID and secret access key) and/or AWS Management Console access (password).

Set Permissions:

In the "Set permissions" section, you can choose how to grant permissions to the IAM user. You have several options:

Attach existing policies directly: Attach existing managed policies to the user. This allows you to grant permissions based on pre-defined policies.

Add user to group: If you have created IAM groups with specific permissions, you can add the user to one or more groups to inherit those permissions.

Copy permissions from existing user: If you have an existing user with similar permissions, you can choose to copy the permissions from that user.

Add Tags (Optional):

You can optionally add tags to the IAM user for better organization and management. Tags are key-value pairs that can be used for various purposes such as cost allocation, access control, etc.

Review and Create:

Review the user details, permissions, and tags, and click "Create user" to create the IAM user.

Access Credentials:

After creating the IAM user, you'll be provided with the user's access key ID, secret access key, and optionally, a password if you enabled console access. Make sure to securely store these credentials as they are required for programmatic access to AWS services.

Authenticating AWS resources with roles

There are various use cases where an EC2 instance needs to access or manage AWS resources. For example, an EC2 instance might need to:

- Back up data to the object store S3.
- Terminate itself after a job has been completed.
- Change the configuration of the private network environment in the cloud.

To be able to access the AWS API, an EC2 instance needs to authenticate itself. You could create an IAM user with access keys and store the access keys on an EC2 instance for authentication. But doing so is a hassle and violates security best practices, especially if you want to rotate the access keys regularly.

Instead of using an IAM user for authentication, you should use an IAM role whenever you need to authenticate AWS resources like EC2 instances. When using an IAM role, your access keys are injected into your EC2 instance automatically.

If an IAM role is attached to an EC2 instance, all identity policies attached to those roles are evaluated to determine whether the request is allowed. By default, no role is attached to an EC2 instance and therefore the EC2 instance is not allowed to make any calls to the AWS API.

The following example will show you how to use an IAM role for an EC2 instance to back up the data into S3.

To enable EC2 instances to access full S3 buckets with full permissions, you can achieve this by attaching an IAM role with appropriate permissions to the EC2 instance. Here's a step-by-step guide:

Create an IAM Policy:

Create a custom IAM policy that grants full access to the S3 service. You can create this policy in the IAM console under "Policies" > "Create policy" > "JSON". Below is an example of a policy that grants full access to all S3 resources:

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

Customize the policy as needed, considering the principle of least privilege.

Create an IAM Role:

Create an IAM role in the IAM console and attach the policy created in the previous step to this role. This role will be assumed by the EC2 instances, granting them the permissions defined in the policy.

Attach the IAM Role to the EC2 Instance:

When launching a new EC2 instance or modifying an existing one, specify the IAM role created in the previous step under "IAM role" in the instance configuration. If you're modifying an existing instance, you can attach the IAM role by stopping the instance, modifying its configuration, and then restarting it.

Accessing S3 from the EC2 Instance:

Once the IAM role is attached to the EC2 instance, any AWS SDK, CLI, or other tools running on the instance will automatically use the permissions granted by the IAM role when accessing S3 buckets. You don't need to provide any additional credentials or keys explicitly.

AWS Lambda

Automating operational tasks with AWS Lambda : Executing your code with AWS Lambda

- What is Serverless?
- Running your code on AWS Lambda
- Comparing AWS Lambda with Virtual Machines (Amazon EC2)

What is Serverless?

Serverless computing, also known as Function as a Service (FaaS), is a cloud computing model that abstracts the underlying infrastructure from developers, allowing them to focus solely on writing and deploying code in the form of functions. In a serverless architecture, developers write code that is triggered by events, and the cloud provider manages the execution of these functions, automatically scaling them as needed. Here's a detailed explanation of serverless computing:

Event-Driven Architecture:

Serverless computing revolves around an event-driven architecture. Functions are triggered by various events such as HTTP requests, database changes, file uploads, scheduled events, or messages from queues or streams.

When an event occurs, the corresponding function is invoked to handle the event. This event-driven approach enables developers to build applications in a highly modular and scalable manner.

No Server Management:

In traditional server-based architectures, developers are responsible for provisioning, managing, and scaling servers to handle application workloads. In contrast, serverless computing abstracts away the infrastructure layer entirely.

Developers do not need to worry about server provisioning, operating system maintenance, or infrastructure scaling. The cloud provider handles these tasks automatically, allowing developers to focus solely on writing code.

Pay-Per-Use Billing:

Serverless platforms typically follow a pay-per-use billing model, where users are charged based on the actual compute resources consumed by their functions, rather than for the provisioned capacity.

Billing is based on factors such as the number of function invocations, the duration of each invocation, and additional resources utilized (e.g., memory, storage, network).

This model offers cost-efficiency by eliminating the need to pay for idle resources and allowing users to scale resources dynamically based on demand.

Scalability and Elasticity:

Serverless platforms are inherently scalable and elastic. Functions can automatically scale up or down in response to changes in workload, ensuring optimal performance and resource utilization.

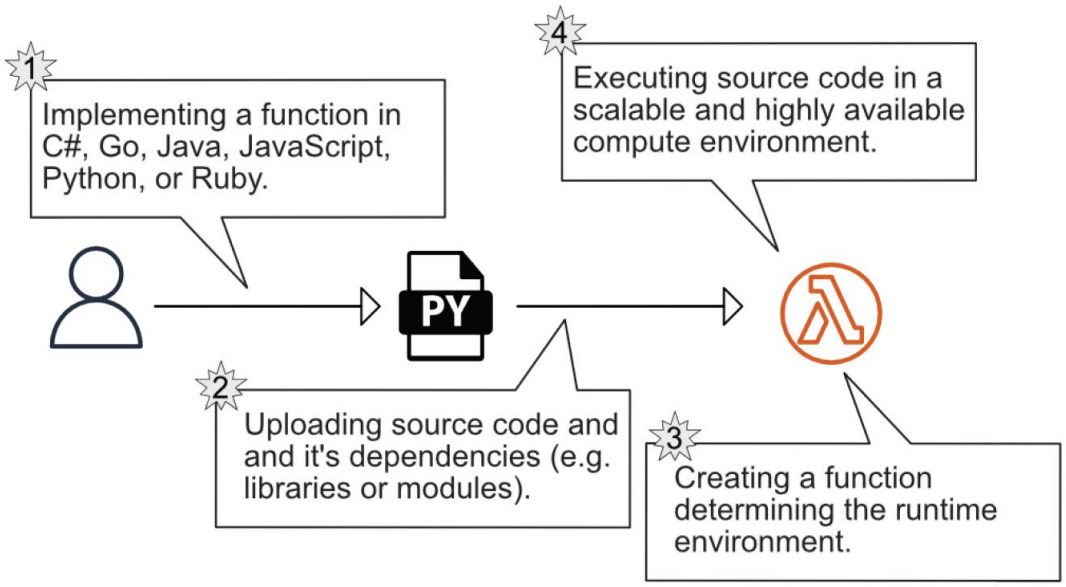
This elasticity enables applications to handle sudden spikes in traffic without manual intervention. Functions are instantiated and scaled dynamically to accommodate varying workloads, offering high availability and responsiveness.

Running your code on AWS Lambda

AWS Lambbda is the basic building block of the serverless platform provided by AWS. The first step in the process is to run your code on Lambda instead of on your own server.

As shown in figure, to execute your code with AWS Lambda, follow these steps:

1. Write the code.
2. Upload your code and its dependencies (such as libraries or modules).
3. Create a function determining the runtime environment and configuration.
4. Invoke the function to execute your code in the cloud.



Currently, AWS Lambda offers runtime environments for the following languages:

- C#/.NET Core
- Go
- Java
- JavaScript/Node.js
- Python
- Ruby

Besides that, you can bring your own runtime by using a cutom runtime. In theory, a custom runtime supports any programming language. You need to bring your own container image and follow conventions for initializing the function and processing tasks. Doing so adds extra complexity, so we recommend to go with one of the available runtimes.

Next, we will compare AWS Lambda with EC2 virtual machines.

Comparing AWS Lambda with virtual machines (Amazon EC2)

What is the difference between using AWS Lambda and virtual machines?

First, there is the granularity of virtualization. Virtual machines provide a full operating system for running one or multiple applications. In contrast, AWS Lambda offers an execution environment for a single function, a small part of an application.

Furthermore, Amazon EC2 offers virtual machines as a service, but you are responsible for operating them in a secure, scalable and highly available way. Doing so requires you to put a substantial amount of effort into maintenance. By contrast, when building with Lambda, AWS manages the underlying infrastructure for you and provides a production-ready infrastructure.

Beyond that, AWS Lambda is billed per execution, and not per second like when a virtual machine is running. You don't have to pay for unused resources that are waiting for requests or tasks. For example, running a script to check the health of a website every 5 minutes on a virtual machine would cost you about $3.71 USD. Executing the same health check with AWS Lambda will cost about $0.04 USD.

Table 6.1 AWS Lambda compared to Amazon EC2

	AWS Lambda	Amazon EC2
Granularity of virtualization	Small piece of code (a function)	An entire operating system
Scalability	Scales automatically. A throttling limit prevents you from creating unwanted charges accidentally and can be increased by AWS support if needed.	As you will learn in chapter 17, using an Auto Scaling Group allows you to scale the number of EC2 instances serving requests automatically. But configuring and monitoring the scaling activities is your responsibility.
High availability	Fault tolerant by default. The computing infrastructure spans multiple machines and data centers.	Virtual machines are not highly available by default. Nevertheless, as you will learn in chapter 13 it is possible to set up a highly available infrastructure based on EC2 instances as well.
Maintenance effort	Almost zero. You need only to configure your function.	You are responsible for maintaining all layers between your virtual machine's operating system and your application's runtime environment.
Deployment effort	Almost zero due to a well-defined API	Rolling out your application to a fleet of virtual machines is a challenge that requires tools and know-how.
Pricing model	Pay per request as well as execution time and allocated memory	Pay for operating hours of the virtual machines, billed per second.

Securing your AWS Account using IAM

- **Why IAM user preferred over root user**
- **Securing your AWS account's IAM user and AWS account's root user**

Why IAM user preferred over root user

IAM (Identity and Access Management) users are preferred over the root user for several reasons:

1) **Granular Permissions:** IAM users allow you to grant only the necessary permissions to individuals or systems. This means you can limit access to specific resources or actions, reducing the risk of accidental or intentional misuse.
2) **Auditing and Accountability:** IAM users are tied to specific individuals or entities, providing better accountability and traceability of actions performed within the system. This facilitates auditing and compliance efforts.
3) **Security Best Practices:** Granting root access to users gives them unrestricted control over all resources and services, which increases the risk of unauthorized access or malicious activity. IAM users, on the other hand, adhere to the principle of least privilege, limiting potential damage in case of a security breach.
4) **Easier Management:** Managing access and permissions for IAM users can be done centrally through the IAM service, making it easier to add, remove, or modify user access as needed. This is more efficient and scalable compared to managing access through the root account.
5) **Protecting the Root Account:** The root account has full access to all resources and services in an AWS account, making it a high-value target for attackers. By minimizing the use of the root account and instead relying on IAM users with limited permissions, you reduce the risk of unauthorized access to critical resources.

Securing your AWS account's IAM user and AWS account's root user

Securing your AWS account

- Securing your AWS account is critical.
- If someone gets access to your AWS account, they can steal your data, use resources at your expense, or delete all your data.
- As figure 5.4 shows, an AWS account is a basket for all the resources you own: EC2 instances, CloudFormation stacks, IAM users, and so on.
- Each AWS account comes with a root user granted unrestricted access to all resources.
- So far, you've used the AWS account root user to log into the Management Console.
- We can create an additional user to log into the Management Console, to avoid using the AWS account root user at all.
- Doing so allows you to manage multiple users, each restricted to the resources that are necessary for their roles.

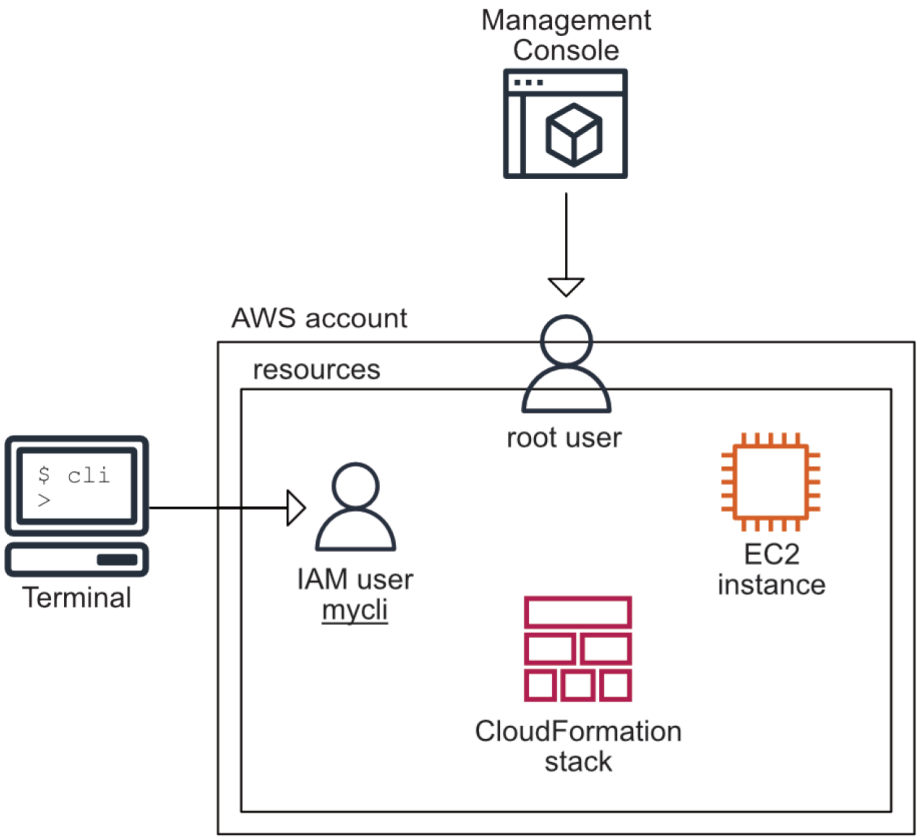


Figure 5.4 An AWS account contains all the AWS resources and comes with an AWS account root user by default.

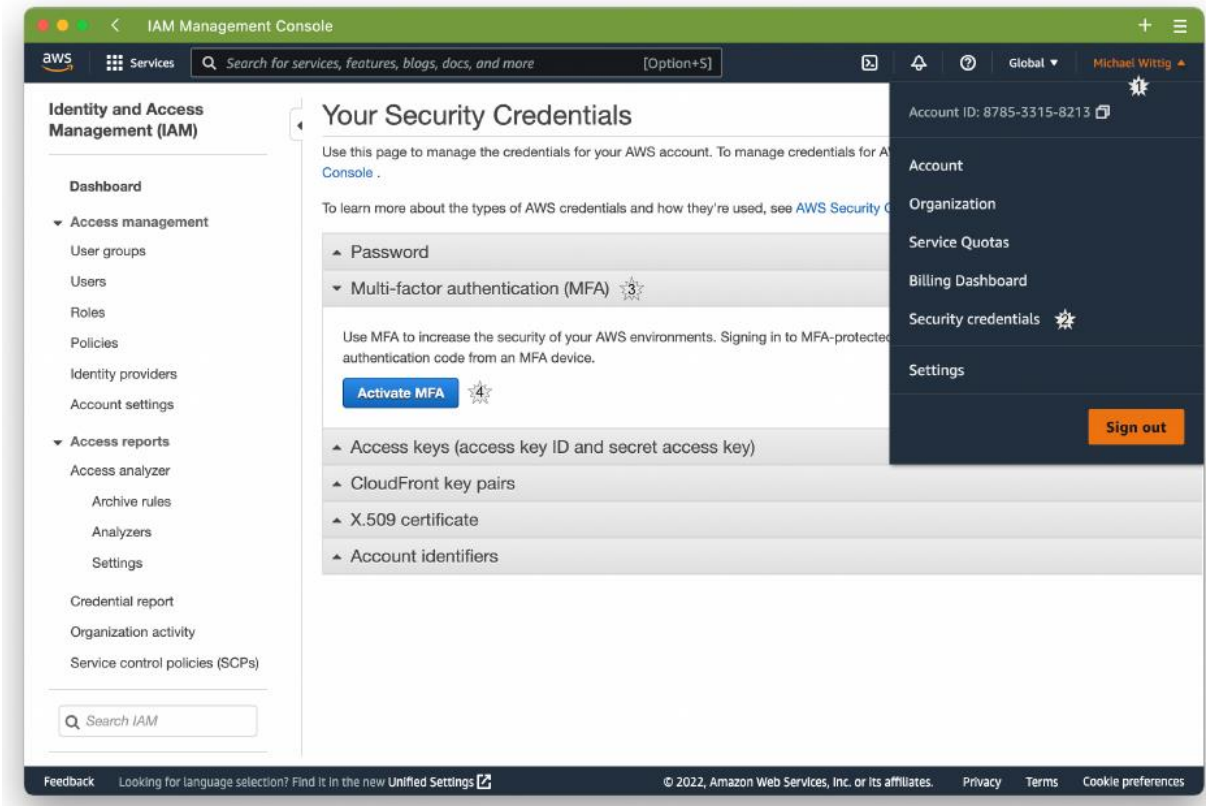
- To access your AWS account, an attacker must be able to authenticate to your account.
- There are three ways to do so: using the AWS account root user, using an IAM user, or authenticating as an AWS resource like an EC2 instance.
- To authenticate as a AWS account root user or IAM user, the attacker needs the username and password or the access keys.
- To authenticate as an AWS resource like an EC2 instance, the attacker needs access to the machine to communicate with the instance metadata service (IMDS).
- To protect yourself from an attacker stealing or cracking your passwords or access keys, you will enable multi-factor authentication (MFA) for your AWS account root user, to add an additional layer of security to the authentication process.

Securing your AWS account's root user

To enable MFA for the AWS account root user of your AWS account follow these steps:

1) Click your name in the navigation bar at the top right of the Management Console.
2) Select Security credentials.
3) Install an MFA app on your smartphone, one that supports the TOTP standard (such as Google Authenticator).
4) Expand the Multi-factor authentication (MFA) section.
5) Click Activate MFA.
6) Select Virtual MFA device and proceed with the next step.
7) Follow the instructions. Use the MFA app on your smartphone to scan the QR code that is displayed.

If you're using your smartphone as a virtual MFA device, it's a good idea not to log in to the Management Console from your smartphone or to store the AWS account root user's password on the phone. Keep the MFA token separate from your password. YubiKeys and hardware MFA tokens are also supported.


IAM Management Console
aws
Services
Search for services, features, blogs, docs, and more
[Option+S]
Global
Michael Wittig
Identity and Access Management (IAM)
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access analyzer
Archive rules
Analyzers
Settings
Credential report
Organization activity
Service control policies (SCPs)
Search IAM
Your Security Credentials
Use this page to manage the credentials for your AWS account. To manage credentials for A
Console .
To learn more about the types of AWS credentials and how they're used, see AWS Security
Password
Multi-factor authentication (MFA)
3
Use MFA to increase the security of your AWS environments. Signing in to MFA-protecte
authentication code from an MFA device.
Activate MFA
4
Access keys (access key ID and secret access key)
CloudFront key pairs
X.509 certificate
Account identifiers
Account ID: 8785-3315-8213
Account
Organization
Service Quotas
Billing Dashboard
Security credentials
2
Settings
Sign out
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

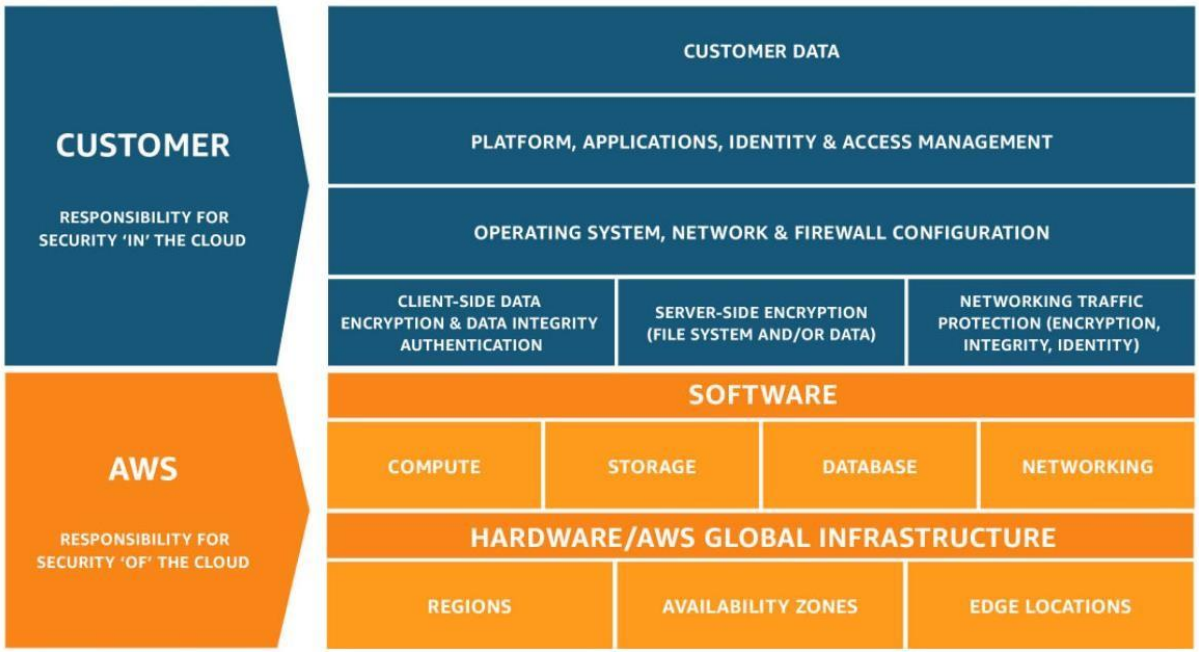
Introduction to IAM

Topics Covered:

1. **Who's responsible for security?**
2. **AWS Identity and Access Management (IAM)**
3. **Comparison of IAM user and root user.**

Who's responsible for security?

Security is a shared responsibility between AWS and the customer. AWS is responsible for 'Security of the Cloud' and the customer is responsible for 'Security in the Cloud'.



The customers are responsible for the secure usage of the services that they select. The customer's role in shared responsibility model varies based on the services used.

Amazon is responsible for the security of AWS infrastructure, which includes – hardware, software, networking, and physical facilities that host AWS services.

AWS is responsible for the following :

- Protecting the network through automated monitoring systems and robust internet access, to prevent Distributed Denial of Service (DDoS) attacks.
- Performing background checks on employees who have access to sensitive areas.
- Decommissioning storage devices by physically destroying them after end of life.
- Ensuring the physical and environmental security of data centers, including fire protection and security staff.

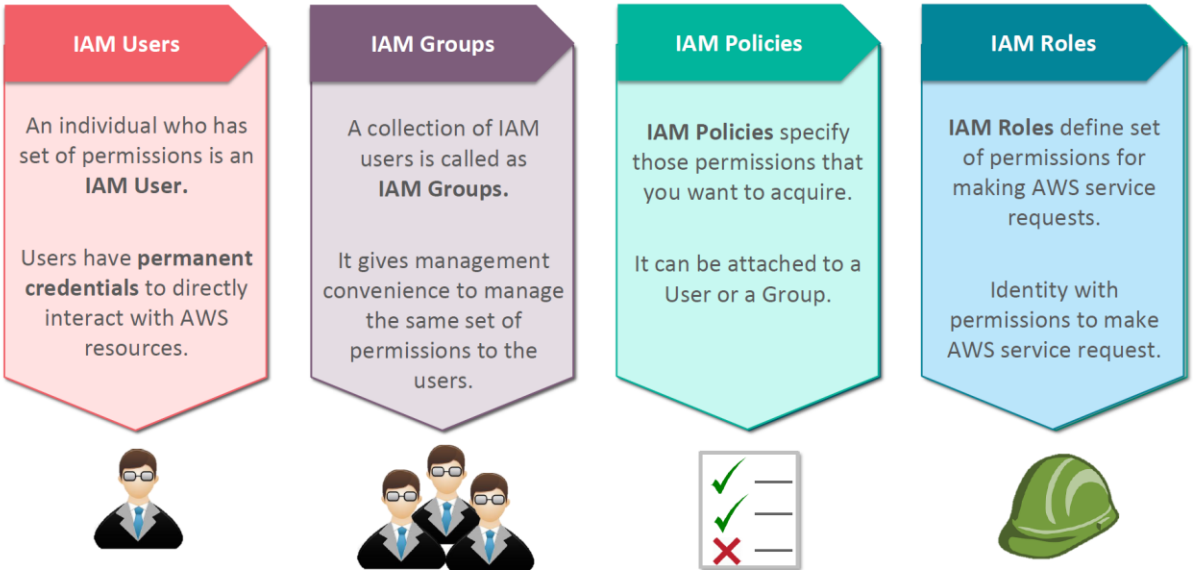
What are your responsibilities?

- Configuring access management that restricts access to AWS resources like S3 and EC2 to a minimum, using AWS IAM.
- Encrypting network traffic to prevent attackers from reading or manipulating data (for example, using HTTPS).
- Configuring a firewall for your virtual network that controls incoming and outgoing traffic with security groups and NACLs.
- Encrypting data at rest. For example, enable data encryption for your database or other storage systems.
- Managing patches for the OS and additional software on virtual machines.

AWS Identity and Access Management (IAM)

IAM is an AWS service that sets the permissions to allocate the right resources to the right person at the right time.

- IAM allows access to compute, storage, database, and application services
- IAM deals with four terms such as Users, Groups, Policies, and Roles
- IAM creates and manages AWS users and groups. Also, it provides permissions to allow and deny access to AWS resources
- It controls both centralized and fine grained-API resources plus management console
- An IAM user is used to authenticate people or workloads running outside of AWS.
- An IAM group is a collection of IAM users with the same permissions.
- An IAM role is used to authenticate AWS resources, for example an EC2 instance.
- An IAM identity policy is used to define the permissions for a user, group, or role.



Comparison of IAM user and root user.

	Root User	**IAM User**
Purpose	The root user is the initial user created when an AWS account is established.	IAM users are created to represent individuals or entities
Access Level	The root user has unrestricted access to all resources and services within the AWS account	The access level of an IAM user is determined by the permissions policies attached to that user.
Security	The root user possesses the highest level of access and is a potential security risk	IAM users provide better security compared to the root user
Can have a password	Always	Yes
Can belong to a group	No	Yes

Docker Image to host a simple Web Page

Deploy a custom Nginx Docker Image and Push it to Docker Hub

What is Docker?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

What is a Docker Container Image?

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application for instance code, runtime, system tools, system libraries, and settings.

What is ECR?

Amazon Elastic Container Registry (Amazon ECR) is an AWS-managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM.

Objectives:

1. Create your own image using Nginx. Include a customized index.html file that can be viewed from a local browser.

2. Deploy your container with port 8080 open

3. Save your container data to the AWS Elastic Container Registry (ECR), and to your Docker Hub account.

Preconditions:

1. AWS Free Tier Account

2. DockerHub Account

3. Basic Knowledge of HTML

4. IDE of your chouse (I will be using Cloud9)

Step 1: Create the Docker image

Before we get started with creating the Docker Image we will create a Directory to keep all the necessary files and resources in one place, making it easier to manage, maintain, and update the image.

The directory will also help Docker to find and use the required files during the image-building process. Use the mkdir command to create the directory

cd to move into the directory

mkdir directory name

cd directory name

Step 2: Pull the Docker image

For us to deploy our website we need to pull the latest Nginx Image from Docker. We can use the following command to accomplish that.

docker pull nginx

Step 3: Verify that the NGINX image was pulled successfully

We can confirm from the Status line that the Nginx image is the latest and that the pull was successful. We can also use the docker images command to confirm that our pull was successful.

docker images

Step 4: Run the container

Use the following command to create a container with the image we created earlier using the following command.

docker run -d -p 80:80 nginx

Step 5: Check if the container was successfully created.

With the following command, we can verify that our container was successfully created.

docker ps -a

Step 6: Verify that the container is running

Take the Public IPv4 Address of the Instance your Cloud9 is running on and paste it into a web browser. Make sure to add :8080 at the end.

<public_ip>:8080