**What is the Cloud?**

In the past, companies and organizations hosted and maintained hardware such as compute, storage, and networking equipment in their own data centers. They needed to allocate entire infrastructure departments to take care of them, resulting in a costly operation that made some workloads and experimentation impossible.

As internet usage became more widespread, the demand for compute, storage, and networking equipment increased. For some companies and organizations, the cost of maintaining a large physical presence was unsustainable. To solve this problem, cloud computing was created.

Cloud computing is the on-demand delivery of IT resources over the internet with pay-as-you-go pricing. You no longer have to manage and maintain your own hardware in your own data centers. Companies like AWS own and maintain these data centers and provide virtualized data center technologies and services to users over the internet.

To help differentiate between running workloads on-premises versus in the cloud, consider the scenario where your developers need to deploy a new feature on your application. Before they deploy, the team wants to test the feature in a separate quality assurance (QA) environment that has the exact same configurations as production.

If you run your application on-premises, creating this additional environment requires you to buy and install hardware, connect the necessary cabling, provision power, install operating systems, and more. All of these tasks can be time-consuming and take days to perform. Meanwhile, the new product feature’s time-to-market is increasing and your developers are waiting for this environment.

If you ran your application in the cloud, you can replicate the entire environment as often as needed in a matter of minutes or even seconds. Instead of physically installing hardware and connecting cabling, you can logically manage your physical infrastructure over the internet.

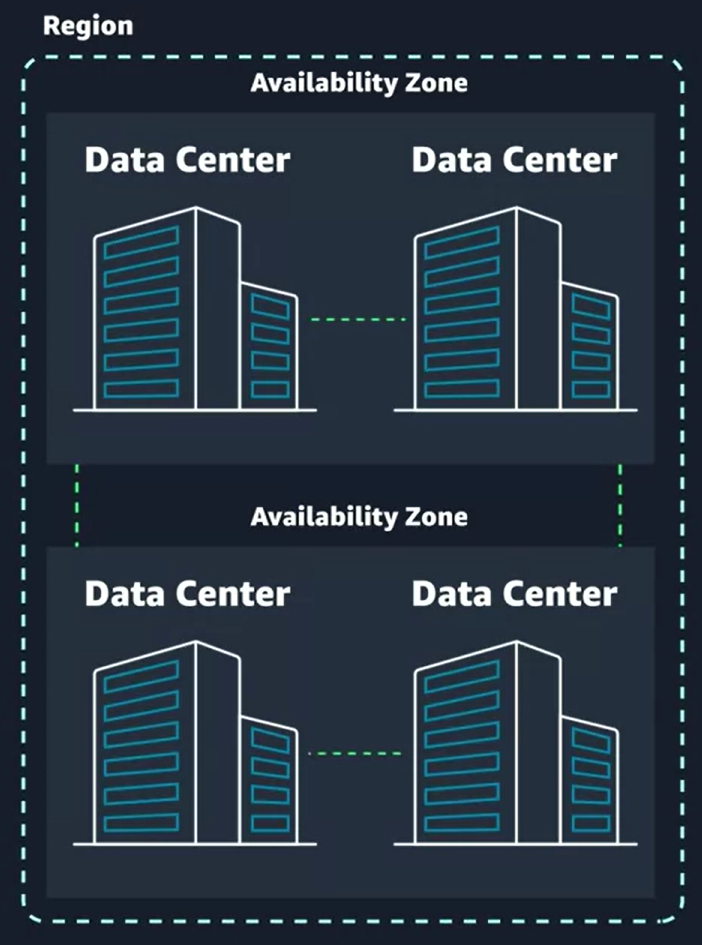
Using cloud computing not only saves you time from the set-up perspective, but it also removes the undifferentiated heavy lifting. If you look at any application, you’ll see that some of the aspects of it are very important to your business, like the code. However, there are other aspects that are no different than any other application you might make: for instance, the compute the code runs on. By removing repetitive common tasks that don’t differentiate your business, like installing virtual machines, or storing backups, you can focus on what is strategically unique to your business and let AWS handle the tasks that are time consuming and don’t separate you from your competitors.

So where does AWS fit into all of this? Well AWS simply just provides cloud computing services. Those IT resources mentioned in the cloud computing definition are AWS services in this case. We’ll need to use these AWS services to architect a scalable, highly available, and cost-effective infrastructure to host our corporate directory application. This way we can get our corporate directory app out into the world quickly, without having to manage any heavy-duty physical hardware. There are the six main advantages to running your workloads on AWS.

# The Six Benefits of Cloud Computing

* **Pay as you go.** Instead of investing in data centers and hardware before you know how you are going to use them, you pay only when you use computing resources, and pay only for how much you use.
* **Benefit from massive economies of scale.** By using cloud computing, you can achieve a lower cost than you can get on your own. Because usage from hundreds of thousands of customers is aggregated in the cloud, AWS can achieve higher economies of scale, which translates into lower pay as-you-go prices.
* **Stop guessing capacity.** Eliminate guessing on your infrastructure capacity needs. When you make a capacity decision prior to deploying an application, you often end up either sitting on expensive idle resources or dealing with limited capacity. With cloud computing, these problems go away. You can access as much or as little capacity as you need and scale up and down as required with only a few minutes notice.
* **Increase speed and agility.** IT resources are only a click away, which means that you reduce the time to make those resources available to your developers from weeks to just minutes. This results in a dramatic increase in agility for the organization since the cost and time it takes to experiment and develop is significantly lower.
* **Stop spending money running and maintaining data centers.** Focus on projects that differentiate your business, not the infrastructure. Cloud computing lets you focus on your customers, rather than on the heavy lifting of racking, stacking, and powering physical infrastructure. This is often referred to as undifferentiated heavy lifting.
* **Go global in minutes.** Easily deploy your application in multiple Regions around the world with just a few clicks. This means you can provide lower latency and a better experience for your customers at a minimal cost.

# AWS Global Infrastructure

****

# Choose the right AWS Region

Consider four main aspects when deciding which AWS Region to host your applications and workloads: latency, price, service availability, and compliance.

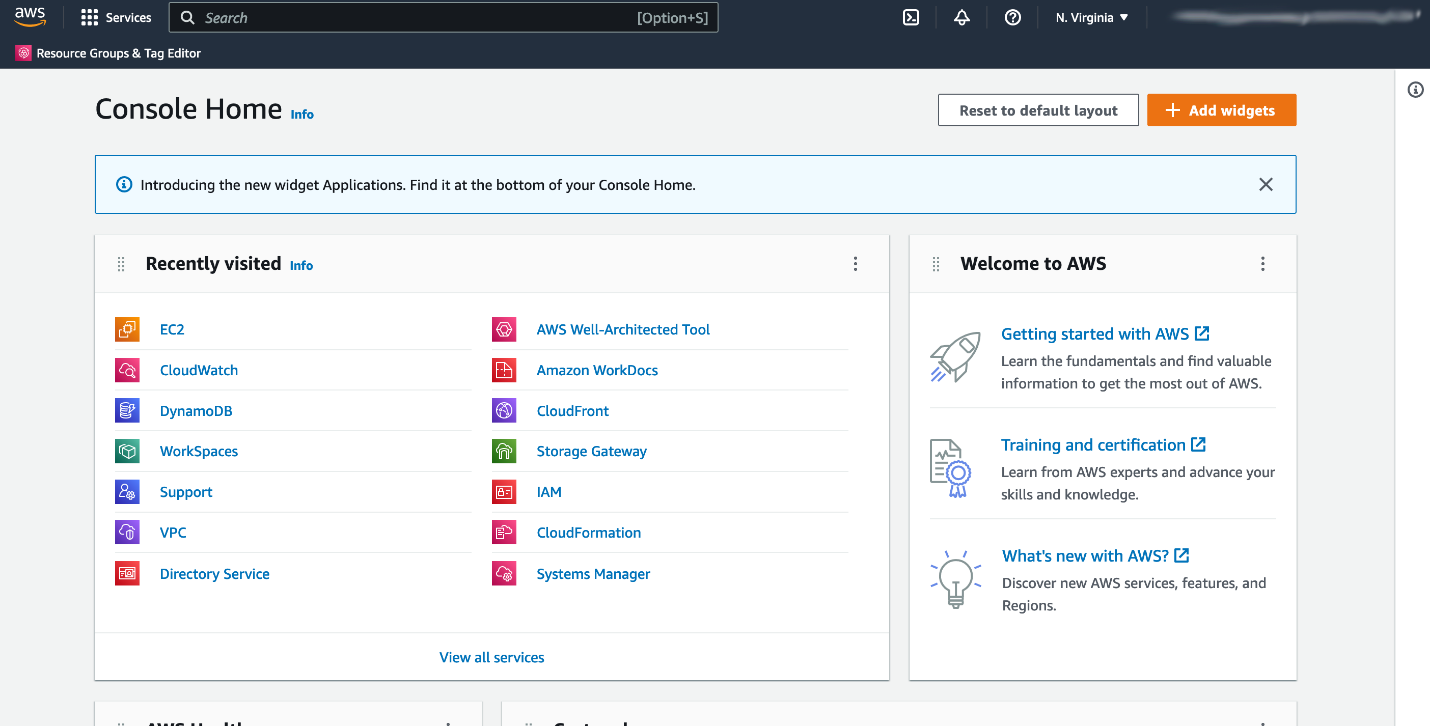
* **Latency.** If your application is sensitive to latency, choose a Region that is close to your user base. This helps prevent long wait times for your customers. Synchronous applications such as gaming, telephony, WebSocket and IoT are significantly affected by higher latency, but even asynchronous workloads, such as ecommerce applications, can suffer from an impact on user connectivity.
* **Price.** Due to the local economy and the physical nature of operating data centers, prices may vary from one Region to another. The pricing in a Region can be impacted by internet connectivity, prices of imported pieces of equipment, customs, real estate, and more. Instead of charging a flat rate worldwide, AWS charges based on the financial factors specific to the location.
* **Service availability.** Some services may not be available in some Regions. The AWS documentation provides a table containing the Regions and the available services within each one.
* **Data compliance.** Enterprise companies often need to comply with regulations that require customer data to be stored in a specific geographic territory. If applicable, you should choose a Region that meets your compliance requirements.

# Interacting with AWS

Every action you make in AWS is an API call that is authenticated and authorized. In AWS, you can make API calls to services and resources through the AWS Management Console, the AWS Command Line Interface (CLI), or the AWS Software Development Kits (SDKs).

**THE AWS MANAGEMENT CONSOLE**

One way to manage cloud resources is through the web-based console, where you log in and click on the desired service. This can be the easiest way to create and manage resources when you’re first begin working with the cloud. Below is a screenshot that shows the landing page when you first log into the AWS Management Console.



The services are placed in categories, such as compute, database, storage and security, identity and compliance. On the upper right corner is the Region selector. If you click it and change the Region, you will make requests to the services in the chosen Region. The URL changes, too. Changing the Region directs the browser to make requests to a whole different AWS Region, represented by a different subdomain.

**THE AWS COMMAND LINE INTERFACE (CLI)**

Consider the scenario where you run tens of servers on AWS for your application’s frontend. You want to run a report to collect data from all of these servers. You need to do this programmatically every day because the server details may change. Instead of manually logging into the AWS Management Console and copying/pasting information, you can schedule an AWS Command Line Interface (CLI) script with an API call to pull this data for you. The AWS CLI is a unified tool to manage AWS services. With just one tool to download and configure, you control multiple AWS services from the command line and automate them with scripts. The AWS CLI is open-source, and there are installers available for Windows, Linux, and Mac OS. Here is an example of running an API call against a service using the AWS CLI:

You get this response:

{

"Reservations": [

{"Groups": [],

"Instances": [

{"AmiLaunchIndex": 0,

and so on.

**AWS SOFTWARE DEVELOPMENT KITS (SDKS)**

API calls to AWS can also be performed by executing code with programming languages. You can do this by using AWS Software Development Kits (SDKs). SDKs are open-source and maintained by AWS for the most popular programming languages, such as C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python, and Ruby. Developers commonly use AWS SDKs to integrate their application source code with AWS services. Let’s say the frontend of the application runs in Python and every time it receives a cat photo, it uploads that photo to a storage service. This action can be achieved from within the source code by using the AWS SDK for Python.

Here is an example of code you can implement to work with AWS resources using the Python AWS SDK.

import boto3

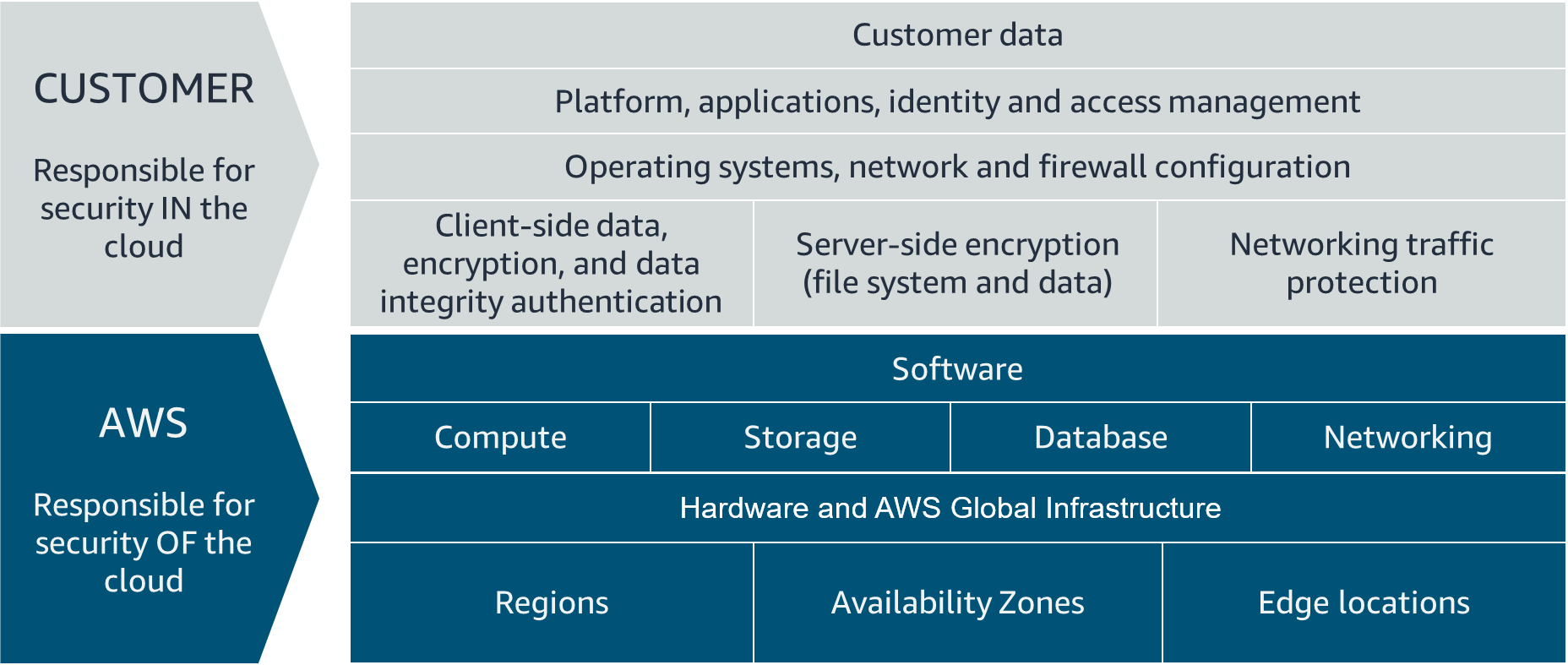
ec2 = boto3.client('ec2')

response = ec2.describe\_instances()

print(response)

# Security and the AWS Shared Responsibility Model

When you begin working with the AWS Cloud, managing security and compliance is a shared responsibility between AWS and you. To depict this shared responsibility, AWS created the shared responsibility model. This distinction of responsibility is commonly referred to as security of the cloud, versus security in the cloud.



Graphical user interface

Description automatically generated

## WHAT IS AWS RESPONSIBLE FOR?

AWS is responsible for security of the cloud. This means AWS is required to protect and secure the infrastructure that runs all the services offered in the AWS Cloud. AWS is responsible for:

* Protecting and securing AWS Regions, Availability Zones, and data centers, down to the physical security of the buildings
* Managing the hardware, software, and networking components that run AWS services, such as the physical server, host operating systems, virtualization layers, and AWS networking components

The level of responsibility AWS has depended on the service. AWS classifies services into three different categories. The following table provides information about each, as well as the AWS responsibility.

| **Category** | **Examples of AWS Services in the Category** | **AWS Responsibility** |
| --- | --- | --- |
| Infrastructure services | Compute services, such as Amazon Elastic Compute Cloud (Amazon EC2) | AWS manages the underlying infrastructure and foundation services. |
| Container services | Services that require less management from the customer, such as Amazon Relational Database Service (Amazon RDS) | AWS manages the underlying infrastructure and foundation services, operating system, and application platform. |
| Abstracted services | Services that require very little management from the customer, such as Amazon Simple Storage Service (Amazon S3) | AWS operates the infrastructure layer, operating system, and platforms, as well as server-side encryption and data protection. |

### Note

Container services refer to AWS abstracting application containers behind the scenes, not Docker container services. This enables AWS to move the responsibility of managing that platform away from customers.

## WHAT IS THE CUSTOMER RESPONSIBLE FOR?

You’re responsible for security in the cloud. When using any AWS service, you’re responsible for properly configuring the service and your applications, as well as ensuring your data is secure.The level of responsibility you have depends on the AWS service. Some services require you to perform all the necessary security configuration and management tasks, while other more abstracted services require you to only manage the data and control access to your resources. Using the three categories of AWS services, you can determine your level of responsibility for each AWS service you use.

| **Category** | **AWS Responsibility** | **Customer Responsibility** |
| --- | --- | --- |
| Infrastructure services | AWS manages the infrastructure and foundation services. | You control the operating system and application platform, as well as encrypting, protecting, and managing customer data. |
| Container services | AWS manages the infrastructure and foundation services, operating system, and application platform. | You are responsible for customer data, encrypting that data, and protecting it through network firewalls and backups. |
| Abstracted services | AWS operates the infrastructure layer, operating system, and platforms, as well as server-side encryption and data protection. | You are responsible for managing customer data and protecting it through client-side encryption. |

Due to the varying level of effort, it’s important to consider which AWS service you use and review the level of responsibility required to secure the service. It’s also important to review how the shared security model aligns with the security standards in your IT environment, as well as any applicable laws and regulations. It’s important to note that you maintain complete control of your data and are responsible for managing the security related to your content. Here are some examples of your responsibilities in context.

* Choosing a Region for AWS resources in accordance with data sovereignty regulations
* Implementing data protection mechanisms, such as encryption and managing backups
* Using access control to limit who has access to your data and AWS resources

# Protect the AWS Root User

## What’s the Big Deal About Auth?

When you’re configuring access to any account, two terms come up frequently: **authentication and authorization**. Though these terms may seem basic, you need to understand them to properly configure access management on AWS. It’s important to keep this mind as you progress in this course. Let’s define both terms.

## Understand Authentication

When you create your AWS account, you use a combination of an email address and a password to verify your identity. If the user types in the correct email and password, the system assumes the user is allowed to enter and grants them access. This is the process of authentication.Authentication ensures that the user is who they say they are. Usernames and passwords are the most common types of authentication, but you may also work with other forms, such as token-based authentication or biometric data like a fingerprint. Authentication simply answers the question, “Are you who you say you are?”

## Understand Authorization

Once you’re inside your AWS account, you might be curious about what actions you can take. This is where authorization comes in. Authorization is the process of giving users permission to access AWS resources and services. Authorization determines whether the user can perform an action—whether it be to read, edit, delete, or create resources. Authorization answers the question, “What actions can you perform?”

## What Is the AWS Root User?

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS root user and is accessed by signing in with the email address and password that you used to create the account.

## Understand the AWS Root User Credentials

The AWS root user has two sets of credentials associated with it. One set of credentials is the email address and password used to create the account. This allows you to access the AWS Management Console. The second set of credentials is called access keys, which allow you to make programmatic requests from the [AWS Command Line Interface (AWS CLI) or AWS API](https://trailhead.salesforce.com/content/learn/modules/aws-cloud-technical-professionals). Access keys consist of two parts:

* An access key ID, for example, A2lAl5EXAMPLE
* A secret access key, for example, wJalrFE/KbEKxE

Similar to a username and password combination, you need both the access key ID and secret access key to authenticate your requests via the AWS CLI or AWS API. Access keys should be managed with the same security as an email address and password.

## Follow Best Practices When Working with the AWS Root User

Keep in mind that the root user has complete access to all AWS services and resources in your account, as well as your billing and personal information. Due to this, securely lock away the credentials associated with the root user and do not use the root user for everyday tasks. To ensure the safety of the root user:

* Choose a strong password for the root user.
* Never share your root user password or access keys with anyone.
* Disable or delete the access keys associated with the root user.
* Do not use the root user for administrative tasks or everyday tasks.

When is it OK to use the AWS root user? There are some tasks where it makes sense to use the AWS root user. Check out the links at the end of this section to read about them.

## Delete Your Keys to Stay Safe

If you don't already have an access key for your AWS account root user, don't create one unless you absolutely need to. If you do have an access key for your AWS account root user and want to delete the keys:

1. Go to the [My Security Credentials page](https://console.aws.amazon.com/iam/home?#security_credential) in the AWS Management Console and sign in with the root user’s email address and password.
2. Open the Access keys section.
3. Under Actions, click **Delete**.
4. Click **Yes**.

## The Case for Multi-Factor Authentication

When you create an AWS account and first log in to that account, you use single-factor authentication. Single-factor authentication is the simplest and most common form of authentication. It only requires one authentication method. In this case, you use a username and password to authenticate as the AWS root user. Other forms of single-factor authentication include a security pin or a security token.However, sometimes a user’s password is easy to guess.

For example, your coworker Bob’s password, IloveCats222, might be easy for someone who knows Bob personally to guess, because it’s a combination of information that is easy to remember and describes certain things about Bob (1. Bob loves cats, and 2. Bob’s birthday is February 22).

If a bad actor guessed or cracked Bob’s password through social engineering, bots, or scripts, Bob might lose control of his account. Unfortunately, this is a common scenario that users of any website often face.

This is why using MFA has become so important in preventing unwanted account access. MFA requires two or more authentication methods to verify an identity, pulling from three different categories of information.

* Something you know, such as a username and password, or pin number
* Something you have, such as a one-time passcode from a hardware device or mobile app
* Something you are, such as fingerprint or face scanning technology

Using a combination of this information enables systems to provide a layered approach to account access. Even though the first method of authentication, Bob’s password, was cracked by a malicious user, it’s very unlikely that a second method of authentication, such as a fingerprint, would also be cracked. This extra layer of security is needed when protecting your most sacred accounts, which is why it’s important to enable MFA on your AWS root user.

## Use MFA on AWS

If you enable MFA on your root user, you are required to present a piece of identifying information from both the something you know category and the something you have category. The first piece of identifying information the user enters is an email and password combination. The second piece of information is a temporary numeric code provided by an MFA device.Enabling MFA adds an additional layer of security because it requires users to use a supported MFA mechanism in addition to their regular sign-in credentials. It’s best practice to enable MFA on the root user.

## Review Supported MFA Devices

|  |  |  |
| --- | --- | --- |
| Device | Description | Supported Devices |
| Virtual MFA | A software app that runs on a phone or other device that provides a one-time passcode. Keep in mind that these applications can run on unsecured mobile devices, and because of that, may not provide the same level of security as hardware or U2F devices. | Authy, Duo Mobile, LastPass Authenticator, Microsoft Authenticator, Google Authenticator |
| Hardware | A hardware device, generally a key fob or display card device that generates a one-time six-digit numeric code | Key fob, display card |
| U2F | A hardware device that you plug into a USB port on your computer | YubiKey |

AWS supports a variety of MFA mechanisms, such as virtual MFA devices, hardware devices, and Universal 2nd Factor (U2F) security keys. For instructions on how to set up each method, check out the Resources section.

# Introduction to AWS Identity and Access Management

## WHAT IS IAM?

IAM is a web service that enables you to manage access to your AWS account and resources. It also provides a centralized view of who and what are allowed inside your AWS account (authentication), and who and what have permissions to use and work with your AWS resources (authorization).With IAM, you can share access to an AWS account and resources without having to share your set of access keys or password. You can also provide granular access to those working in your account, so that people and services only have permissions to the resources they need. For example, to provide a user of your AWS account with read-only access to a particular AWS service, you can granularly select which actions and which resources in that service they can access.

## GET TO KNOW THE IAM FEATURES

To help control access and manage identities within your AWS account, IAM offers many features to ensure security.

* IAM is global and not specific to any one Region. This means you can see and use your IAM configurations from any Region in the AWS Management Console.
* IAM is integrated with many AWS services [by default](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-that-work-with-iam.html).
* You can establish password policies in IAM to specify complexity requirements and mandatory rotation periods for users.
* IAM supports MFA.
* IAM supports identity federation, which allows users who already have passwords elsewhere—for example, in your corporate network or with an internet identity provider—to get temporary access to your AWS account.
* Any AWS customer can use IAM; the service is offered at no additional charge.

## WHAT IS AN IAM USER?

An IAM user represents a person or service that interacts with AWS. You define the user within your AWS account. And any activity done by that user is billed to your account. Once you create a user, that user can sign in to gain access to the AWS resources inside your account.You can also add more users to your account as needed. For example, for your cat photo application, you could create individual users in your AWS account that correspond to the people who are working on your application. Each person should have their own login credentials. Providing users with their own login credentials prevents sharing of credentials.

## IAM USER CREDENTIALS

An IAM user consists of a name and a set of credentials. When creating a user, you can choose to provide the user:

* Access to the AWS Management Console
* Programmatic access to the AWS Command Line Interface (AWS CLI) and AWS Application Programming Interface (AWS API)

To access the AWS Management Console, provide the users with a user name and password. For programmatic access, AWS generates a set of access keys that can be used with the AWS CLI and AWS API. IAM user credentials are considered permanent, in that they stay with the user until there’s a forced rotation by admins.When you create an IAM user, you have the option to grant permissions directly at the user level.This can seem like a good idea if you have only one or a few users. However, as the number of users helping you build your solutions on AWS increases, it becomes more complicated to keep up with permissions. For example, if you have 3,000 users in your AWS account, administering access becomes challenging, and it’s impossible to get a top-level view of who can perform what actions on which resources.If only there were a way to group IAM users and attach permissions at the group level instead. Guess what: There is!

## WHAT IS AN IAM GROUP?

An IAM group is a collection of users. All users in the group inherit the permissions assigned to the group. This makes it easy to give permissions to multiple users at once. It’s a more convenient and scalable way of managing permissions for users in your AWS account. This is why using IAM groups is a best practice.If you have a an application that you’re trying to build and have multiple users in one account working on the application, you might decide to organize these users by job function. You might want IAM groups organized by developers, security, and admins. You would then place all of your IAM users in the respective group for their job function.This provides a better view to see who has what permissions within your organization and an easier way to scale as new people join, leave, and change roles in your organization.Consider the following examples.

* A new developer joins your AWS account to help with your application. You simply create a new user and add them to the developer group, without having to think about which permissions they need.
* A developer changes jobs and becomes a security engineer. Instead of editing the user’s permissions directly, you can instead remove them from the old group and add them to the new group that already has the correct level of access.

Keep in mind the following features of groups.

* Groups can have many users.
* Users can belong to many groups.
* Groups cannot belong to groups.

The root user can perform all actions on all resources inside an AWS account by default. This is in contrast to creating new IAM users, new groups, or new roles. New IAM identities can perform no actions inside your AWS account by default until you explicitly grant them permission.The way you grant permissions in IAM is by using IAM policies.

## WHAT IS AN IAM POLICY?

## IAM POLICY EXAMPLES

Most policies are stored in AWS as JSON documents with several policy elements. Take a look at the following example of what providing admin access through an IAM identity-based policy looks like.

**{**

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

**"Statement": [{ "Effect": "Allow",**

**"Action": "\*",**

**"Resource": "\*"**

**}]**

**}**

In this policy, there are four major JSON elements: Version, Effect, Action, and Resource.

* The **Version** element defines the version of the policy language. It specifies the language syntax rules that are needed by AWS to process a policy. To use all the available policy features, include "Version": "2012-10-17" before the "Statement" element in all your policies.
* The **Effect** element specifies whether the statement will allow or deny access. In this policy, the Effect is "Allow", which means you’re providing access to a particular resource.
* The **Action** element describes the type of action that should be allowed or denied. In the above policy, the action is "\*". This is called a wildcard, and it is used to symbolize every action inside your AWS account.
* The **Resource** element specifies the object or objects that the policy statement covers. In the policy example above, the resource is also the wildcard "\*". This represents all resources inside your AWS console.

Putting all this information together, you have a policy that **allows** you to perform **all actions** on **all resources** inside your AWS account. This is what we refer to as an administrator policy.

Let’s look at another example of a more granular IAM policy.

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

**"Statement": [{**

**"Effect": "Allow",**

**"Action": [**

**"iam: ChangePassword",**

**"iam: GetUser"**

**]**

**"Resource":**

**"arn:aws:iam::123456789012:user/${aws:username}"**

**}]**

**}**

After looking at the JSON, you can see that this policy **allows** the IAM user to **change their own IAM password** (iam:ChangePassword) and **get information about their own user** (iam:GetUser). It only permits them to access their own credentials because the resource restricts access with the variable substitution ${aws:username}.

| **Element** | **Description** | **Required** | **Example** |
| --- | --- | --- | --- |
| Effect | Specifies whether the statement results in an allow or an explicit deny | ✔ | "Effect": "Deny" |
| Action | Describes the specific actions that will be allowed or denied | ✔ | "Action": "iam:CreateUser" |
| Resource | Specifies the object or objects that the statement covers | ✔ | "Resource": "arn:aws:iam::account-ID-without-hyphens:user/Bob" |

## UNDERSTAND POLICY STRUCTURE

When creating a policy, it is required to have each of the following elements inside a policy statement.

# Role Based Access in AWS

Throughout these last few lessons, there have been sprinklings of IAM best practices. It’s helpful to have a summary of some of the most important IAM best practices you need to be familiar with before building out solutions on AWS.

## LOCK DOWN THE AWS ROOT USER



The root user is an all-powerful and all-knowing identity within your AWS account. If a malicious user were to gain control of root-user credentials, they would be able to access every resource within your account, including personal and billing information. To lock down the root user:

* Don’t share the credentials associated with the root user.
* Consider deleting the root user access keys.
* Enable MFA on the root account.

## FOLLOW THE PRINCIPLE OF LEAST PRIVILEGE



Least privilege is a standard security principle that advises you to grant only the necessary permissions to do a particular job and nothing more. To implement least privilege for access control, start with the minimum set of permissions in an IAM policy and then grant additional permissions as necessary for a user, group, or role.

## USE IAM APPROPRIATELY

IAM is used to secure access to your AWS account and resources. It simply provides a way to create and manage users, groups, and roles to access resources within a single AWS account. IAM is not used for website authentication and authorization, such as providing users of a website with sign-in and sign-up functionality. IAM also does not support security controls for protecting operating systems and networks.

## USE IAM ROLES WHEN POSSIBLE



Maintaining roles is easier than maintaining users. When you assume a role, IAM dynamically provides temporary credentials that expire after a defined period of time, between 15 minutes and 36 hours. Users, on the other hand, have long-term credentials in the form of user name and password combinations or a set of access keys.User access keys only expire when you or the admin of your account rotates these keys. User login credentials expire if you have applied a password policy to your account that forces users to rotate their passwords.

## CONSIDER USING AN IDENTITY PROVIDER

If you decide to make your cat photo application into a business and begin to have more than a handful of people working on it, consider managing employee identity information through an identity provider (IdP). Using an IdP, whether it be an AWS service such as AWS IAM Identity Center (Successor to AWS Single Sign-On) or a third-party identity provider, provides you a single source of truth for all identities in your organization.You no longer have to create separate IAM users in AWS. You can instead use IAM roles to provide permissions to identities that are federated from your IdP.For example, you have an employee, Martha, that has access to multiple AWS accounts. Instead of creating and managing multiple IAM users named Martha in each of those AWS accounts, you can manage Martha in your company’s IdP. If Martha moves within the company or leaves the company, Martha can be updated in the IdP, rather than in every AWS account you have.

## CONSIDER AWS IAM IDENTITY CENTER



If you have an organization that spans many employees and multiple AWS accounts, you may want your employees to sign in with a single credential.AWS IAM Identity Center is an IdP that lets your users sign in to a user portal with a single set of credentials. It then provides them access to all their assigned accounts and applications in one central location.AWS IAM Identity Center is similar to IAM, in that it offers a directory where you can create users, organize them in groups, and set permissions across those groups, and grant access to AWS resources. However, AWS IAM Identity Center has some advantages over IAM. For example, if you’re using a third-party IdP, you can sync your users and groups to AWS IAM Identity Center.This removes the burden of having to re-create users that already exist elsewhere, and it enables you to manage those users from your IdP. More importantly, AWS IAM Identity Center separates the duties between your IdP and AWS, ensuring that your cloud access management is not inside or dependent on your IdP.

# Default Amazon Machine Image (AMI) for Amazon EC2

As of March 15, 2023 the default Amazon Machine Image (AMI) for Amazon EC2 has been updated to use the Amazon Linux 2023 AMI. In the demonstrations for this course, we use the Amazon Linux 2 AMI. If you are following along with the videos please be aware that if you use the new Amazon Linux 2023 AMI with the user data the way it appears in the videos the script will not run properly and the application will not launch. We are in the process of updating the course to reflect this change.

In the meantime, there are a few ways to work around this issue. You can either use the Amazon Linux 2 AMI with the user data as shown in the demonstrations and this will resolve the issue, or you can use an updated version of the user data script which I will include in this message.

To recap, we have a new default AMI for EC2 instances called the Amazon Linux 2023 AMI. The videos show us using Amazon Linux 2. Because of changes between these two AMIs the user data script shown in the videos will not run properly on Amazon Linux 2023 based instances. You can either choose Amazon Linux 2 as the AMI when launching the instance, and use the original user data script or you can use the Amazon Linux 2023 AMI and use the updated user data script.

## **Amazon Linux 2 user data script:**

#!/bin/bash -ex

wget <https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-GCNv2/FlaskApp.zip>

unzip FlaskApp.zip

cd FlaskApp/

yum -y install python3 mysql

pip3 install -r requirements.txt

amazon-linux-extras install epel

yum -y install stress

export PHOTOS\_BUCKET=${SUB\_PHOTOS\_BUCKET}

export AWS\_DEFAULT\_REGION=<INSERT REGION HERE>

export DYNAMO\_MODE=on

FLASK\_APP=application.py /usr/local/bin/flask run --host=0.0.0.0 --port=80

## **Amazon Linux 2023 user data script:**

#!/bin/bash -ex

wget <https://aws-tc-largeobjects.s3-us-west-2.amazonaws.com/DEV-AWS-MO-GCNv2/FlaskApp.zip>

unzip FlaskApp.zip

cd FlaskApp/

yum -y install python3-pip

pip install -r requirements.txt

yum -y install stress

export PHOTOS\_BUCKET=${SUB\_PHOTOS\_BUCKET}

export AWS\_DEFAULT\_REGION=<INSERT REGION HERE>

export DYNAMO\_MODE=on

FLASK\_APP=application.py /usr/local/bin/flask run --host=0.0.0.0 --port=80

When using the user data scripts, remember to replace the <INSERT REGION HERE> with whatever AWS region you are operating in, and ensure you remove both brackets as well.

# Compute as a Service on AWS

**Understanding Servers**

The first building block you need to host an application is a server. Servers often times can handle Hypertext Transfer Protocol (HTTP) requests and send responses to clients following the client-server model, though any API based communication also falls under this model. A client being a person or computer that seconds a request, and a server handling the requests is a computer, or collection of computers, connected to the internet serving websites to internet users. These servers power your application by providing CPU, memory, and networking capacity to process users’ requests and transform them into responses. For context, common HTTP servers include:

* Windows options, such as Internet Information Services (IIS).
* Linux options, such as Apache HTTP Web Server, Nginx, and Apache Tomcat.

To run an HTTP server on AWS, you need to find a service that provides compute power in the AWS Management Console. You can log into the console and view the complete list of AWS compute services.



**Choose the Right Compute Option**

If you’re responsible for setting up servers on AWS to run your infrastructure, you have many compute options. You need to know which service to use for which use case. At a fundamental level, there are three types of compute options: virtual machines, container services, and serverless. If you’re coming to AWS with prior infrastructure knowledge, a virtual machine can often be the easiest compute option in AWS to understand. This is because a virtual machine emulates a physical server and allows you to install an HTTP server to run your applications. To run these virtual machines, you install a hypervisor on a host machine. This hypervisor provisions the resources to create and run your virtual machines. In AWS, these virtual machines are called Amazon Elastic Compute Cloud or Amazon EC2. Behind the scenes, AWS operates and manages the host machines and the hypervisor layer. AWS also installs the virtual machine operating system, called the guest operating system. Some AWS compute services use Amazon EC2 or use virtualization concepts under the hood, therefore it is best to understand this service first before moving on to container services and serverless compute.

# Introduction to Amazon Elastic Compute Cloud

**What Is Amazon EC2?**

****

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. It allows you to provision virtual servers called EC2 instances. Although AWS uses the phrase “web service” to describe it, it doesn’t mean that you are limited to running just web servers on your EC2 instances. You can create and manage these instances through the AWS Management Console, the AWS Command Line Interface (CLI), AWS Software Development Kits (SDKs), or through automation tools and infrastructure orchestration services.In order to create an EC2 instance, you need to define:

* Hardware specifications, like CPU, memory, network, and storage.
* Logical configurations, like networking location, firewall rules, authentication, and the operating system of your choice.

When launching an EC2 instance, the first setting you configure is which operating system you want by selecting an Amazon Machine Image (AMI).

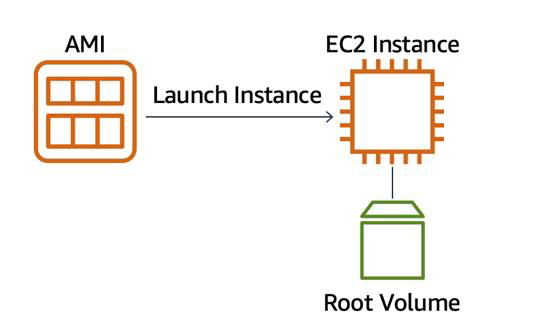
**What Is an AMI?**

In the traditional infrastructure world, the process of spinning up a server consists of installing an operating system from installation disks, installation drives, or installation wizards over the network. In the AWS Cloud, this operating system installation is no longer your responsibility, and is instead built into the AMI that you choose.Not only does an AMI let you configure which operating system you want, you can also select storage mappings, the architecture type (such as 32-bit, 64-bit, or 64-bit ARM), and additional software installed.

**What Is the Relationship Between AMIs and EC2 Instances?**

EC2 instances are live instantiations of what is defined in an AMI, much like a cake is a live instantiation of a cake recipe. If you are familiar with software development, you can also see this kind of relationship between a Class and an Object.

A Class is something you model and define, while an object is something you interact with. In this case, the AMI is how you model and define your instance, while the EC2 instance is the entity you interact with, where you can install your web server, and serve your content to users. When you launch a new instance, AWS allocates a virtual machine that runs on a hypervisor. Then the AMI you selected is copied to the root device volume, which contains the image used to boot the volume. In the end, you get a server you can connect to and install packages and any additional software. In this case, you install a web server along with the properly configured source code of your employee directory app.

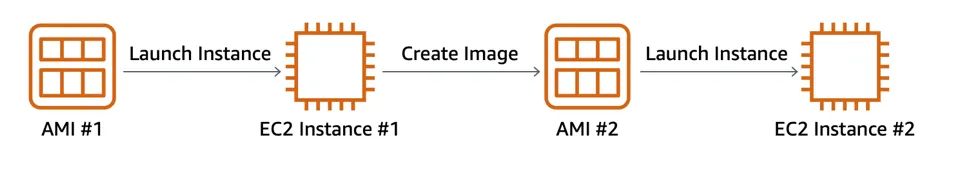


One advantage of using AMIs is that they are reusable.

You might choose a Linux-based AMI and configure the HTTP server, application packages, and any additional software you may need to run your application.

If you wanted to create a second EC2 instance with the same configurations, how can you easily do that? One option is to go through the entire instance creation and configuration process and try to match your settings to the first instance. However, this is time consuming and leaves room for human error.

The second, better option, is to create an AMI from your running instance and use this AMI to start a new instance. This way, your new instance will have all the same configurations as your current instance, because the configurations set in the AMIs are the same.

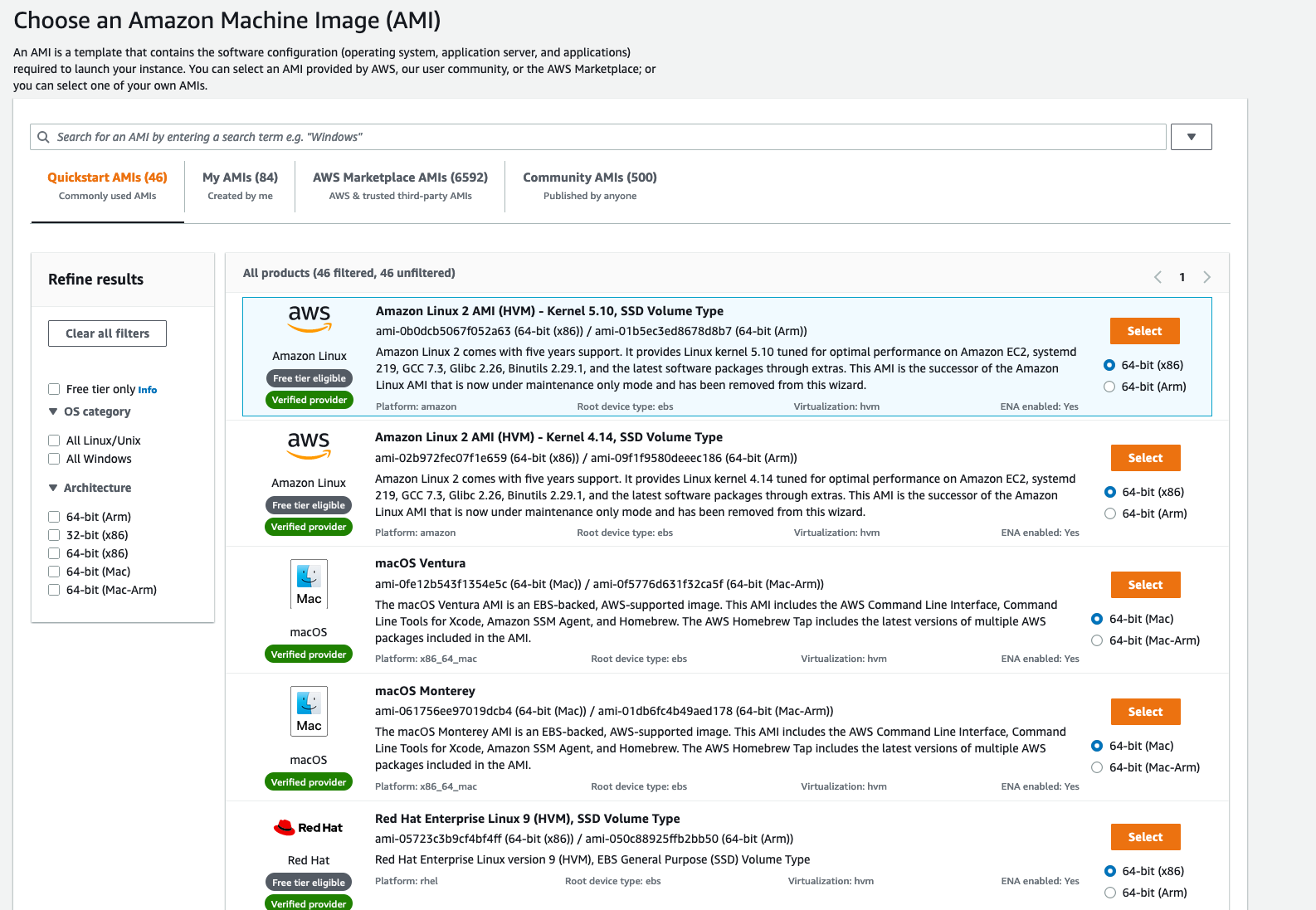


**Where Can You Find AMIs?**

You can select an AMI from the following categories.

* Quick Start AMIs that are premade by AWS and allow you to get started quickly.
* AWS Marketplace AMIs that provide popular open source and commercial software from third-party vendors.
* My AMIs that are created from your EC2 instances.
* Community AMIs that are provided by the AWS user community.
* Build your own custom image with EC2 Image Builder.

Each AMI in the AWS Management Console has an AMI ID, which is prefixed by “ami-”, followed by a random hash of numbers and letters. These IDs are unique to each AWS region.



# Amazon EC2 Instance Lifecycle

Now that you know how to select an operating system for your EC2 instance, it’s time to choose other configurations to create your EC2 instance, such as the instance type, network, and storage. For an application like the employee directory application, you need instances with enough capacity to run web servers and process incoming customer requests. Your instance sizing will depend on both the demands of your application and the anticipated size of your user base. Forecasting server capacity for an on-premises application requires difficult decisions involving significant up-front capital spending, while changes to the allocation of your cloud-based services can be made with a simple API call. Because of AWS’s pay-as-you-go model, you can match your infrastructure capacity to your application’s demand, instead of the other way around.

Diagram

Description automatically generated

Diagram

Description automatically generated

**What Makes Up an EC2 Instance?**

EC2 instances are a combination of virtual processors (vCPUs), memory, network, and in some cases, instance storage and graphics processing units (GPUs). When you create an EC2 instance, you need to choose how much you need of each of these components.

AWS offers a variety of instances that differ based on performance. Some instances provide you with more capacity and others provide less. To get an overview of the capacity details for a particular instance, you should look at the instance type. Instance types consist of a prefix identifying the type of workloads they’re optimized for, followed by a size. For example, the instance type c5.large can be broken down into the following elements.

* **c5** determines the instance family and generation number. Here, the instance belongs to the fifth generation of instances in an instance family that’s optimized for generic computation.
* **large**, which determines the amount of instance capacity.

**What Are Instance Families?**

| **Instance Family** | **Description** | **Use Cases** |
| --- | --- | --- |
| General purpose | Provides a balance of compute, memory, and networking resources, and can be used for a variety of workloads. | Scale-out workloads such as web servers, containerized microservices, caching fleets, distributed data stores, and development environments. |
| Compute optimized | Ideal for compute-bound applications that benefit from high-performance processors. | High-performance web servers, scientific modeling, batch processing, distributed analytics, high-performance computing (HPC), machine/deep learning, ad serving, highly scalable multiplayer gaming. |
| Memory optimized | Designed to deliver fast performance for workloads that process large data sets in memory. | Memory-intensive applications such as high-performance databases, distributed web-scale in-memory caches, mid-size in-memory databases, real-time big-data analytics, and other enterprise applications. |
| Accelerated computing | Use hardware accelerators or co-processors to perform functions such as floating-point number calculations, graphics processing, or data pattern matching more efficiently than is possible with conventional CPUs. | 3D visualizations, graphics-intensive remote workstations, 3D rendering, application streaming, video encoding, and other server-side graphics workloads. |
| Storage optimized | Designed for workloads that require high, sequential read and write access to large data sets on local storage. They are optimized to deliver tens of thousands of low-latency random I/O operations per second (IOPS) to applications that replicate their data across different instances. | NoSQL databases, such as Cassandra, MongoDB, and Redis, in-memory databases, scale-out transactional databases, data warehousing, Elasticsearch, and analytics. |

**Where Does Your EC2 Instance Live?**

By default, your EC2 instances are placed in a network called the default Amazon Virtual Private Cloud (VPC). This network was created so that you can easily get started with Amazon EC2 without having to learn how to create and configure a VPC. Any resource you put inside the default VPC will be public and accessible by the internet, so you shouldn’t place any customer data or private information inside of it. Once you get more comfortable with networking on AWS, you should change this default setting to choose your own custom VPCs and restrict access with additional routing and connectivity mechanisms.

**Architect for High Availability**

Inside this network, your instance resides in an Availability Zone of your choice. AWS services that are scoped at the Availability Zone level must be architected with high availability in mind. While EC2 instances are typically reliable, two is better than one, and three is better than two. Specifying the instance size gives you an advantage when designing your architecture because you can use more smaller instances instead of a few larger ones. If your frontend only has a single instance and that instance fails, your application goes down. On the other hand, if your workload is distributed across 10 instances and one fails, you lose only 10 percent of your fleet and your application availability is hardly affected. When architecting any application for high availability, consider using at least two EC2 instances in two separate Availability Zones.

**Explore the EC2 Instance Lifecycle**

An EC2 instance transitions between different states from the moment you create it all the way through to its termination.

When you launch an instance, it enters the pending state (1). When the instance is pending, billing has not started. At this stage, the instance is preparing to enter the running state. Pending is where AWS performs all actions needed to set up an instance, such as copying the AMI content to the root device and allocating the necessary networking components. When your instance is running (2), it's ready to use. This is also the stage where billing begins. As soon as an instance is running, you are then able to take other actions on the instance, such as reboot, terminate, stop, and stop-hibernate. When you reboot an instance (3), it’s different than performing a stop action and then a start action. Rebooting an instance is equivalent to rebooting an operating system. The instance remains on the same host computer and maintains its public and private IP address, and any data on its instance store. It typically takes a few minutes for the reboot to complete. When you stop and start an instance (4), your instance may be placed on a new underlying physical server. Therefore, you lose any data on the instance store that were on the previous host computer. When you stop an instance, the instance gets a new public IP address but maintains the same private IP address. When you terminate an instance (5), the instance store are erased, and you lose both the public IP address and private IP address of the machine. Termination of an instance means you can no longer access the machine.

**What Is the Difference Between Stop and Stop-Hibernate?**

When you stop your instance, it enters the stopping state, and then the stopped state. AWS does not charge usage or data transfer fees for your instance after you stop it, but storage for any Amazon EBS volumes is still charged. While your instance is in the stopped state, you can modify some attributes, like the instance type. When you stop your instance, the data stored in memory (RAM) is lost. When you stop-hibernate your instance, AWS signals the operating system to perform hibernation (suspend-to-disk), which saves the contents from the instance memory (RAM) to the Amazon EBS root volume. Consider a scenario where you build a standard three tier application, where you have web servers, application servers and database servers. Turns out, the application you built becomes extremely popular. To relieve some stress on the database that supports your application, you want to implement a custom backend layer that caches database information in memory (RAM). You decide to run this custom backend caching solution on Amazon EC2. In this scenario, the stop-hibernate feature would be instrumental in persisting storage. It would prevent you from having to manually create scripts to save this RAM data before shutting down the server.

**What Makes Up the Pricing?**

To understand EC2 pricing, let’s decouple the instance price from other services attached to it, such as storage and networking costs. In this unit we refer to the instance cost as the cost associated with the instance in terms of specifications and not the total blended cost of running an instance. Once an instance is launched in your AWS account, the billing usually accrues on a per-second basis. For simplicity of calculation, prices are stated per-hour. For example, if you have an instance running for 5 minutes and 38 seconds during a given month, you only pay for 338 seconds of utilization at the end of the month. One exception to this pricing convention may be third-party AMIs purchased from the AWS Marketplace, which may have a minimum billing of 1 hour. For more details, check out the resources section of this unit.

**What Are the EC2 Pricing Options?**

One of the ways to reduce costs with Amazon EC2 is to choose the right pricing option for the way your applications run. There are three main purchasing options for EC2 instances: on-demand, reserved, and spot instances.

**Pay As You Go with On-Demand Instances**

With On-Demand instances, you pay for compute capacity with no long-term commitments. Billing begins whenever the instance is running, and billing stops when the instance is in a stopped or terminated state. The price per second for a running On-Demand instance is fixed. For applications that require servers to be running all the time, you are less likely to benefit from the On-Demand pricing model, simply because there is no situation where you will need to turn servers off. For example, you might want the web server hosting the frontend of your corporate directory application to be running 24/7 so that users can access the website at any time. Even if there are no users connected to your website, you don’t want to shut down the servers supporting the site in case of potential user activity. In the case when servers cannot be stopped, consider using a Reserved Instance to save on costs.

**Reserve Capacity with Reserved Instances (RIs)**

RIs provide you with a significant discount compared to On-Demand instance pricing. RIs provide a discounted hourly rate and an optional capacity reservation for EC2 instances. You can choose between three payment options: All Upfront, Partial Upfront, or No Upfront. You can select either a 1-year or 3-year term for each of these options. Depending on which option you choose, you are discounted differently.

* All Upfront offers a higher discount than Partial Upfront instances.
* Partial Upfront instances offer a higher discount than No Upfront.
* No Upfront offers a higher discount than On-Demand.

*On-Demand* and *No Upfront* are similar since both do not require any upfront payment. However, there is a major difference. When you choose an On-Demand instance, you stop paying for the instance when you stop or terminate the instance. When you stop an RI, you still pay for it because you committed to a 1-year or 3-year term. Reserved Instances are associated with an instance type and an Availability Zone depending on how you reserve it. The discount applied by a Reserved Instance purchase is not directly associated with a specific instance ID, but with an instance type.

**Save on Costs with Spot Instances**

Another way of paying for EC2 instances is by using Spot Instances. Amazon EC2 Spot Instances allow you to take advantage of unused EC2 capacity in the AWS Cloud. They are available at up to a 90% discount compared to On-Demand prices. With Spot Instances, you set a limit on how much you would like to pay for the instance hour. This is compared against the current Spot price that AWS determines. If the amount you pay is more than the current Spot price and there is capacity, then you will receive an instance. While they are very promising from the billing perspective, there are some architectural considerations you will need to consider in order to use them effectively. One consideration is that your spot instance may be interrupted. For example, if AWS determines that capacity is no longer available for a particular spot instance or if the Spot price exceeds how much you are willing to pay, AWS will give you a 2-minute warning before it interrupts your instance. That means any application or workload that runs on a Spot instance must be able to be interrupted. Because of this unique consideration, inherently fault-tolerant workloads are typically good candidates to use with Spot instances. These include big data, containerized workloads, continuous integration/continuous delivery (CI/CD), web servers, high-performance computing (HPC), image and media rendering, or other test and development workloads.

# Container Services on AWS

AWS offers a broad spectrum of compute offerings that give you the flexibility to choose the right tool for the right job. The three main categories of compute are virtual machines, containers, and serverless. There is no one-size-fits-all service because it depends on your needs. The key is to understand what each option has to offer in order to build a more appropriate cloud architecture for your use case. In this unit, you learn about containers and how to run them on AWS. Containers can host a variety of different workloads, including web applications, lift and shift migrations, distributed applications, and streamlining of development, test, and production environments.

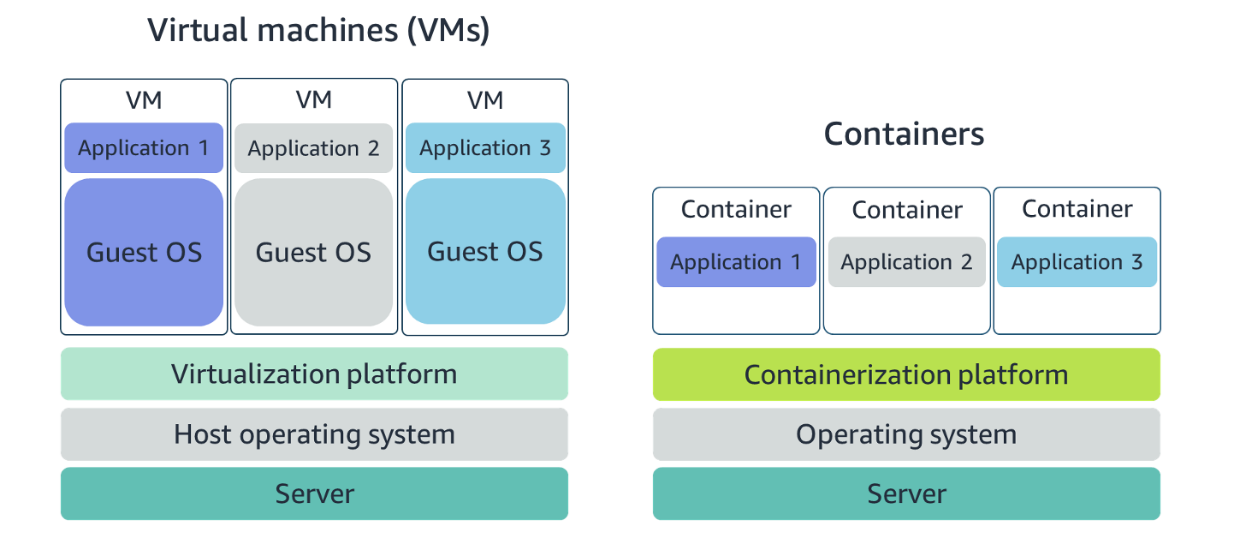
**WHAT IS A CONTAINER?**

While containers are often referred to as a new technology, the idea started in the 1970s with certain Linux kernels having the ability to separate their processes through isolation. At the time, this was configured manually, making operations complex. With the evolution of the open-source software community, containers evolved. Today, containers are used as a solution to problems of traditional compute, including the issue of getting software to run reliably when it moves from one compute environment to another. A container is a standardized unit that packages up your code and all of its dependencies. This package is designed to run reliably on any platform, because the container creates its own independent environment. This makes it easy to carry workloads from one place to another, such as from development to production or from on-premises to the cloud.

**WHAT IS DOCKER?**

When you hear the word container, you may associate it with Docker. Docker is a popular container runtime that simplifies the management of the entire operating system stack needed for container isolation, including networking and storage. Docker makes it easy to create, package, deploy, and run containers.

**WHAT IS THE DIFFERENCE BETWEEN CONTAINERS AND VMS?**

****

Containers share the same operating system and kernel as the host they exist on, whereas virtual machines contain their operating system. Since each virtual machine has to maintain a copy of an operating system, there’s a degree of wasted space. A container is more lightweight. They spin up quicker, almost instantly. This difference in startup time becomes instrumental when designing applications that need to scale quickly during input/output (I/O) bursts. While containers can provide speed, virtual machines offer you the full strength of an operating system and offer more resources, like package installation, a dedicated kernel, and more.

**ORCHESTRATE CONTAINERS**

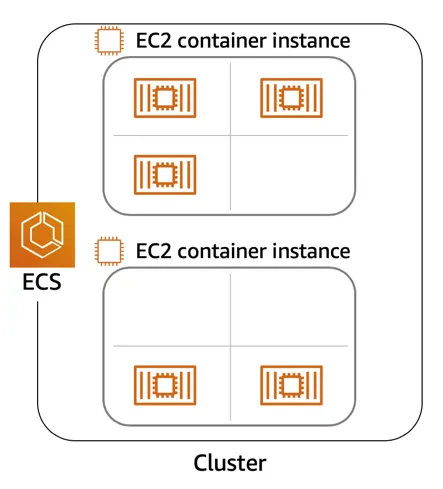
In AWS, containers run on EC2 instances. For example, you may have a large instance and run a few containers on that instance. While running one instance is easy to manage, it lacks high availability and scalability. Most companies and organizations run many containers on many EC2 instances across several Availability Zones. If you’re trying to manage your compute at a large scale, you need to know:

* How to place your containers on your instances.
* What happens if your container fails.
* What happens if your instance fails.
* How to monitor deployments of your containers.

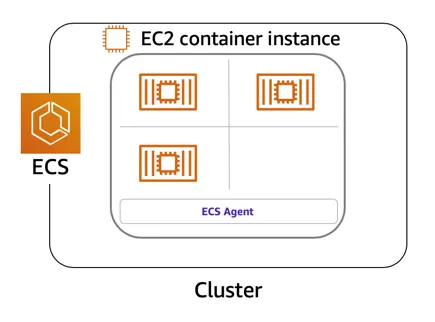
This coordination is handled by a container orchestration service. AWS offers two container orchestration services: Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS).

**MANAGE CONTAINERS WITH AMAZON ELASTIC CONTAINER SERVICE (AMAZON ECS)**

Amazon ECS is an end-to-end container orchestration service that allows you to quickly spin up new containers and manage them across a cluster of EC2 instances.



To run and manage your containers, you need to install the Amazon ECS Container Agent on your EC2 instances. This agent is open source and responsible for communicating back to the Amazon ECS service about cluster management details. You can run this agent on both Linux and Windows AMIs. An instance with the container agent installed is often called a *container instance.*



Once the Amazon ECS container instances are up and running, you can perform actions that include, but are not limited to, launching and stopping containers, getting cluster state, scaling in and out, scheduling the placement of containers across your cluster, assigning permissions, and meeting availability requirements.

To prepare your application to run on Amazon ECS, you create a task definition. The task definition is a text file, in JSON format, that describes one or more containers. A task definition is similar to a blueprint that describes the resources you need to run that container, such as CPU, memory, ports, images, storage, and networking information.

Here is a simple task definition that you can use for your corporate director application. In this example, the runs on the Nginx web server.

{

"family": "webserver",

"containerDefinitions": [ {

"name": "web",

"image": "nginx",

"memory": "100",

"cpu": "99"

} ],

"requiresCompatibilities": [ "FARGATE" ],

"networkMode": "awsvpc",

"memory": "512",

"cpu": "256"

}

**USE KUBERNETES WITH AMAZON ELASTIC KUBERNETES SERVICE (AMAZON EKS)**

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services. By bringing software development and operations together by design, Kubernetes created a rapidly growing ecosystem that is very popular and well established in the market. If you already use Kubernetes, you can use Amazon EKS to orchestrate these workloads in the AWS Cloud. Amazon EKS is conceptually similar to Amazon ECS, but there are some differences.

* An EC2 instance with the ECS Agent installed and configured is called a container instance. In Amazon EKS, it is called a worker node.
* An ECS Container is called a task. In the Amazon EKS ecosystem, it is called a pod.
* While Amazon ECS runs on AWS native technology, Amazon EKS runs on top of Kubernetes.

If you have containers running on Kubernetes and want an advanced orchestration solution that can provide simplicity, high availability, and fine-grained control over your infrastructure, Amazon EKS is the tool for you.

# Serverless and AWS Lambda

**REMOVE THE UNDIFFERENTIATED HEAVY LIFTING**

If you run your code on Amazon EC2, AWS is responsible for the physical hardware and you are responsible for the logical controls, such as guest operating system, security and patching, networking, security, and scaling. If you run your code in containers on Amazon ECS and Amazon EKS, AWS is responsible for more of the container management, such as deploying containers across EC2 instances and managing the container cluster. However, when running ECS and EKS on EC2, you are still responsible for maintaining the underlying EC2 instances. If you want to deploy your workloads and applications without having to manage any EC2 instances, you can do that on AWS with *serverless* compute.

**GO SERVERLESS**

Every definition of serverless mentions four aspects.

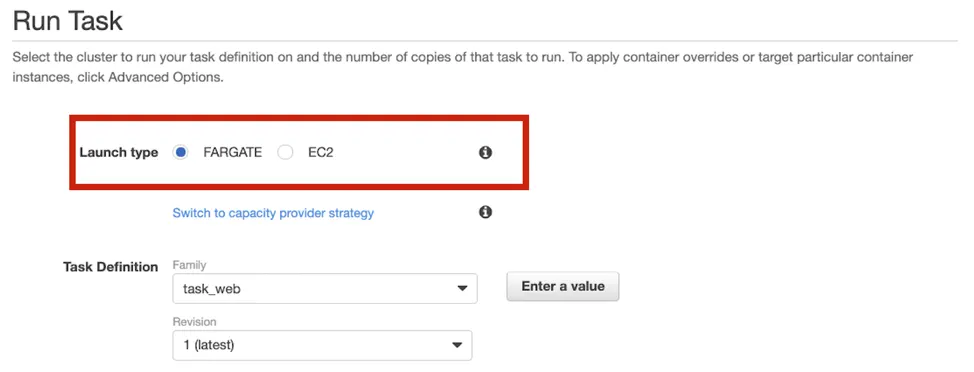
* No servers to provision or manage.
* Scales with usage.
* You never pay for idle resources.
* Availability and fault tolerance are built-in.

With serverless, spend time on the things that differentiate your application, rather than spending time on ensuring availability, scaling, and managing servers. AWS has several serverless compute options, including AWS Fargate and AWS Lambda.

**EXPLORE SERVERLESS CONTAINERS WITH AWS FARGATE**

Amazon ECS and Amazon EKS enable you to run your containers in two modes.

* Amazon EC2 mode
* AWS Fargate mode



AWS Fargate is a purpose-built serverless compute engine for containers. Fargate scales and manages the infrastructure, allowing developers to work on what they do best: application development.It achieves this by allocating the right amount of compute, eliminating the need to choose and handle EC2 Instances and cluster capacity and scaling. Fargate supports both Amazon ECS and Amazon EKS architecture and provides workload isolation and improved security by design.

AWS Fargate abstracts the EC2 instance so you’re not required to manage it. However, with AWS Fargate, you can use all the same ECS primitives, APIs, and AWS integrations. It natively integrates with AWS Identity and Access Management (IAM) and Amazon Virtual Private Cloud (VPC). Having native integration with Amazon VPC allows you to launch Fargate containers inside your network and control connectivity to your applications.

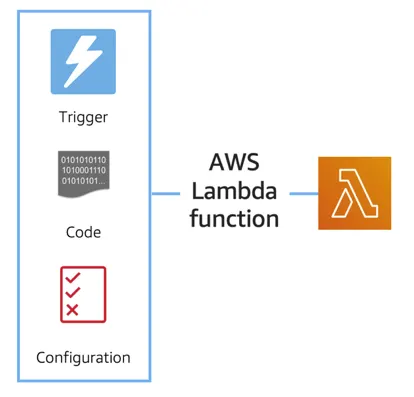
**RUN YOUR CODE ON AWS LAMBDA**

If you want to deploy your workloads and applications without having to manage any EC2 instances or containers, you can use AWS Lambda. AWS Lambda lets you run code without provisioning or managing servers or containers. You can run code for virtually any type of application or backend service, including data processing, real-time stream processing, machine learning, WebSockets, IoT backends, mobile backends, and web apps, like your corporate directory app!

AWS Lambda requires zero administration from the user. You upload your source code and Lambda takes care of everything required to run and scale your code with high availability. There are no servers to manage, bringing you continuous scaling with subsecond metering and consistent performance.

**HOW LAMBDA WORKS**

There are three primary components of a Lambda function: the trigger, code, and configuration. The code is source code, that describes what the Lambda function should run. This code can be authored in three ways.



* You create the code from scratch.
* You use a blueprint that AWS provides.
* You use same code from the AWS Serverless Application Repository, a resource that contains sample applications, such as “hello world” code, Amazon Alexa Skill sample code, image resizing code, video encoding, and more.

When you create your Lambda function, you specify the runtime you want your code to run in. There are built-in runtimes such as Python, Node.js, Ruby, Go, Java, .NET Core, or you can implement your Lambda functions to run on a custom runtime. The configuration of a Lambda function consists of information that describes how the function should run. In the configuration, you specify network placement, environment variables, memory, invocation type, permission sets, and other configurations. To dive deeper into these configurations, check out the resources section of this unit. Triggers describe when the Lambda function should run.

A trigger integrates your Lambda function with other AWS services, enabling you to run your Lambda function in response to certain API calls that occur in your AWS account. This makes you quicker to respond to events in your console without having to perform manual actions. All you need is the what, how, and when of a Lambda function to have functional compute capacity that runs only when you need it to.Amazon’s CTO, Werner Vogels, says, “No server is easier to manage than no server.” This quote summarizes the convenience you can have when running serverless solutions, like AWS Fargate and AWS Lambda.

In the next unit, you apply all the information you’ve learned about Amazon EC2, Amazon ECS and Amazon EKS, and AWS Fargate and learn the use cases for each service.

**AWS Lambda function handler**

The AWS Lambda function handler is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method. When the handler exits or returns a response, it becomes available to handle another event. You can use the following general syntax when creating a function handler in Python:

def handler\_name(event, context): ... return some\_value

**NAMING**

The Lambda function *handler* name specified at the time you create a Lambda function is derived from the following:the name of the file in which the Lambda handler function is locatedthe name of the Python handler functionA function handler can be any name; however, the default on the Lambda console is lambda\_function.lambda\_handler. This name reflects the function name as lambda\_handler, and the file where the handler code is stored in lambda\_function.py.If you choose a different name for your function handler on the Lambda console, you must update the name on the **Runtime settings** pane.

**BILLING GRANULARITY**

AWS Lambda lets you run code without provisioning or managing servers, and you pay only for what you use. You are charged for the number of times your code is triggered (requests) and for the time your code executes, rounded up to the nearest 1ms (duration). AWS rounds up duration to the nearest millisecond with no minimum execution time. With this pricing, it can be very cost effective to run functions whose execution time is very low, such as functions with durations under 100ms or low latency APIs.

**SOURCE CODE**

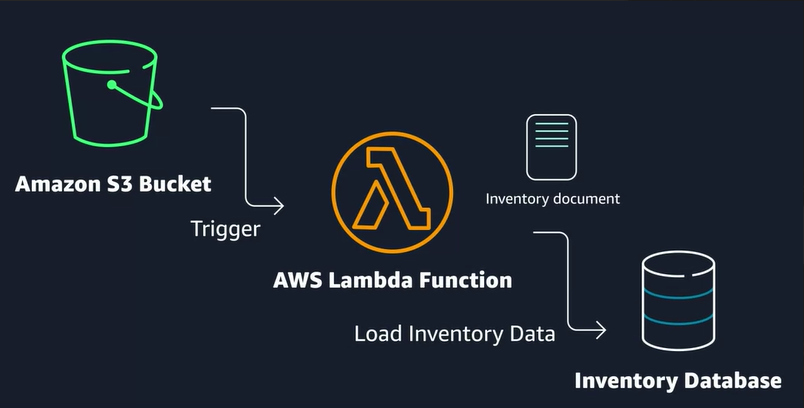
This video used a small amount of sample code illustrating a pattern for lazily generating assets using AWS Lambda and Amazon S3. If you’re looking to deploy a service to resize images to production, consider using the new release  [Serverless Image Handler](https://aws.amazon.com/solutions/implementations/serverless-image-handler/) which is a robust solution to handle image manipulation and can be deployed via an AWS CloudFormation template.

You can find a tutorial on creating the AWS Lambda function as well as the code used in the AWS Lambda demo here: see [AWS News Blog](https://aws.amazon.com/blogs/compute/resize-images-on-the-fly-with-amazon-s3-aws-lambda-and-amazon-api-gateway/).

**Used Scenarios**

1. Consider a scenario where you are a developer who is tasked with creating a new feature for a web application being hosted on EC2. The web application is an online store. And right now, all the items being sold in the store are loaded into a database manually behind the scenes. By manually, I mean there is a person who adds a new row to a database for each new item to be sold in the store. This process takes a long time, isn't very scalable, and is prone to error. You are tasked with automating the process of getting the new item information loaded into the inventory database. The goal is to have a person upload an inventory spreadsheet into Amazon S3, the object storage service, then have a process automatically load the data into the inventory database. What compute would you use to host the processing logic to load the items from the file into the database?

**AWS Lambda** is the correct answer for this one. There are a few reasons. First of all, to address your concern on cost, AWS Lambda only charges you for the compute you consume when the code is actually running. And code is only run in response to triggers or a direct invitation. So here's my suggestion. You know that the goal is to have someone upload an inventory document to S3, which should kick off the process of updating the database. You also learned that AWS Lambda has triggers that run your Lambda functions code. AWS Lambda integrates with many AWS services to act as triggers, and Amazon S3 is one of them. So my suggestion would be to create an AWS Lambda function; configure a PutEvent as the trigger from Amazon S3; then when the inventory is uploaded, Amazon S3 will trigger the Lambda function to run and the code in the function will parse the inventory document and add each item to the database.



2.Let's say you have an application currently hosted in your on-premises data center, which needs to be migrated to AWS. It's currently running on Linux servers in the data center, and you want to minimize the amount of refactoring needed to migrate to AWS. It's important that this workload is elastic and can support varying demand. What compute option would you choose?

AWS Lambda could work, but you can't just upload the same code you would run on Amazon EC2 into a Lambda function. There would have to be a decent amount of refactoring in order to take advantage of that service. Same idea with any of the AWS container services, like ECS or EKS. Again, you'd have some amount of rework required to migrate to containers. Therefore, **Amazon EC2** is the best option for this migration.

3.Imagine a scenario where you are planning to write a brand-new application using a microservices or service-oriented design. And you want to architect the application where it can scale up or down quickly, and you want to lower the risk of deploying new changes to production. Which AWS compute service would you use?

The answer is either **ECS** or **EKS** for this one because using containers makes it easier to support microservice or service-oriented designs. Containers boot up quickly, so scaling is quicker than EC2 instances, and the use of containers helps with code portability. Meaning, if I write the code on my laptop and run it in a container, test it in QA in a container, I can then expect the same container to behave the same way once deployed to production, thus reducing the risk of deployments causing errors because of environmental issues.

Graphical user interface, application

Description automatically generated

# Networking on AWS

**WHAT IS NETWORKING?**

Networking is how you connect computers around the world and allow them to communicate with one another. In this trail, you’ve already seen a few examples of networking. One is the AWS global infrastructure. AWS has created a network of resources using data centers, Availability Zones, and Regions.

**KNOW THE NETWORKING BASICS**

Think about sending a letter. When sending a letter, there are three pieces of information you need.

* The payload or letter inside the envelope.
* The address of the sender in the From section.
* The address of the recipient in the To section.

Let’s go further. Each address must contain information such as:

* Name of sender and recipient
* Street
* City
* State or province
* Zip, area, or postal code
* Country

You need all parts of an address to ensure that your letter gets to its destination. Without the correct address, postal workers are not able to properly deliver the message. In the digital world, computers handle the delivery of messages in a similar way. This is called routing.

**WHAT ARE IP ADDRESSES?**

In order to properly route your messages to a location, you need an address. Just like each home has a mail address, each computer has an IP address. However, instead of using the combination of street, city, state, zip code, and country, the IP address uses a combination of bits, 0s and 1s.

Here is an example of a 32-bit address in binary format:

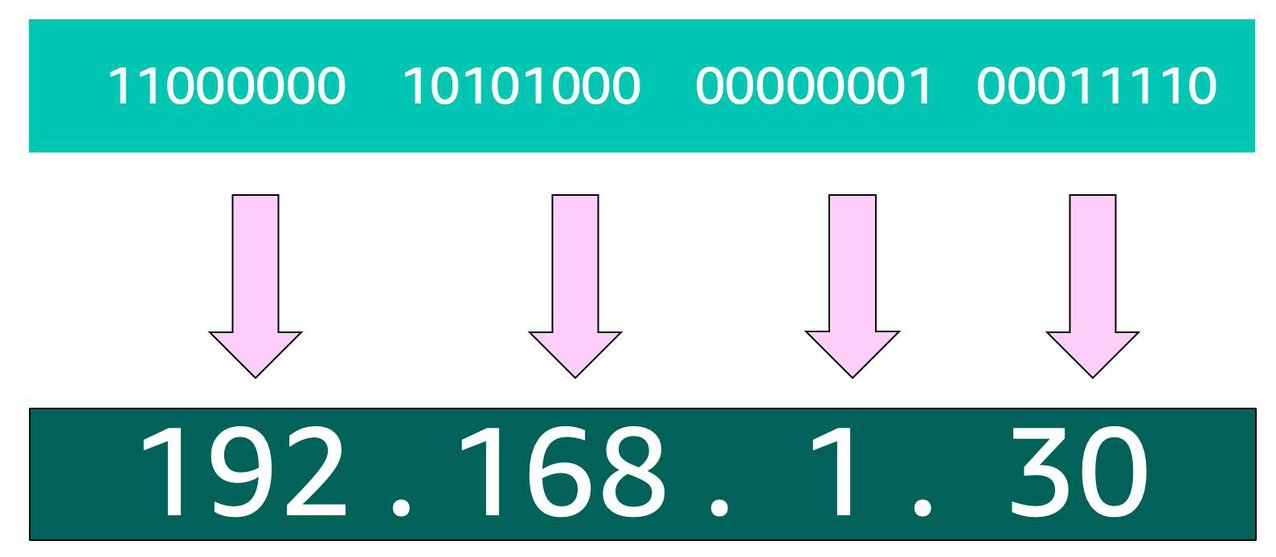


It’s called 32-bit because you have 32 digits. Feel free to count!

**WHAT IS IPV4 NOTATION?**

Typically, you don’t see an IP address in this binary format. Instead, it’s converted into decimal format and noted as an Ipv4 address.

In the diagram below, the 32 bits are grouped into groups of 8 bits, also called octets. Each of these groups is converted into decimal format separated by a period.



In the end, this is what is called an Ipv4 address. This is important to know when trying to communicate to a single computer. But remember, you’re working with a network. This is where CIDR Notation comes in.

**USE CIDR NOTATION**

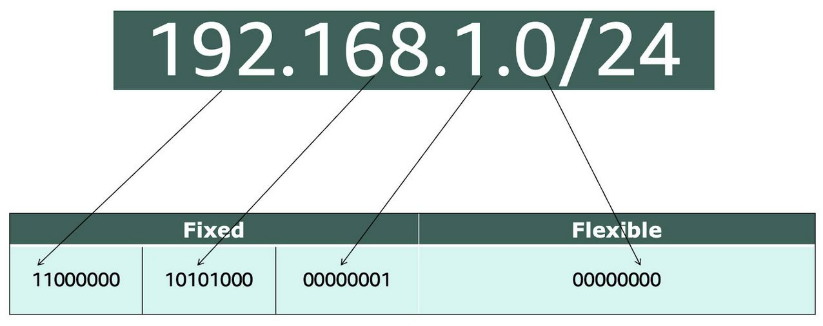
192.168.1.30 is a single IP address. If you wanted to express IP addresses between the range of 192.168.1.0 and 192.168.1.255, how can you do that?

One way is by using Classless Inter-Domain Routing (CIDR) notation. CIDR notation is a compressed way of specifying a range of IP addresses. Specifying a range determines how many IP addresses are available to you.

CIDR notation looks like this:



It begins with a starting IP address and is separated by a forward slash (the “/” character) followed by a number. The number at the end specifies how many of the bits of the IP address are fixed. In this example, the first 24 bits of the IP address are fixed. The rest are flexible.



32 total bits subtracted by 24 fixed bits leaves 8 flexible bits. Each of these flexible bits can be either 0 or 1, because they are binary. That means you have two choices for each of the 8 bits, providing 256 IP addresses in that IP range.

The higher the number after the /, the smaller the number of IP addresses in your network. For example, a range of 192.168.1.0/24 is smaller than 192.168.1.0/16.

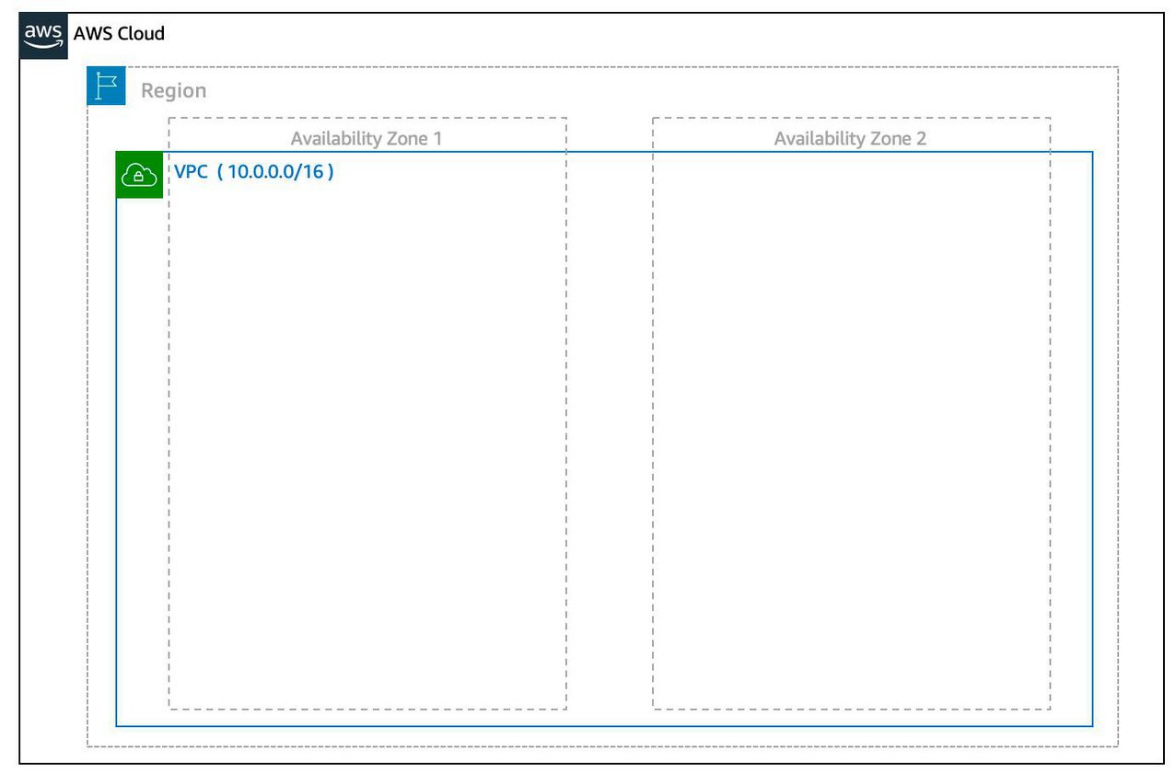
When working with networks in the AWS Cloud, you choose your network size by using CIDR notation. In AWS, the smallest IP range you can have is /28, which provides you 16 IP addresses. The largest IP range you can have is a /16, which provides you with 65,536 IP addresses.

# Introduction to Amazon VPC

A VPC is an isolated network you create in the AWS cloud, similar to a traditional network in a data center. When you create a VPC, you need to choose three main things.

1. The name of your VPC.
2. A Region for your VPC to live in. Each VPC spans multiple Availability Zones within the Region you choose.
3. A IP range for your VPC in CIDR notation. This determines the size of your network. Each VPC can have up to four /16 IP ranges.

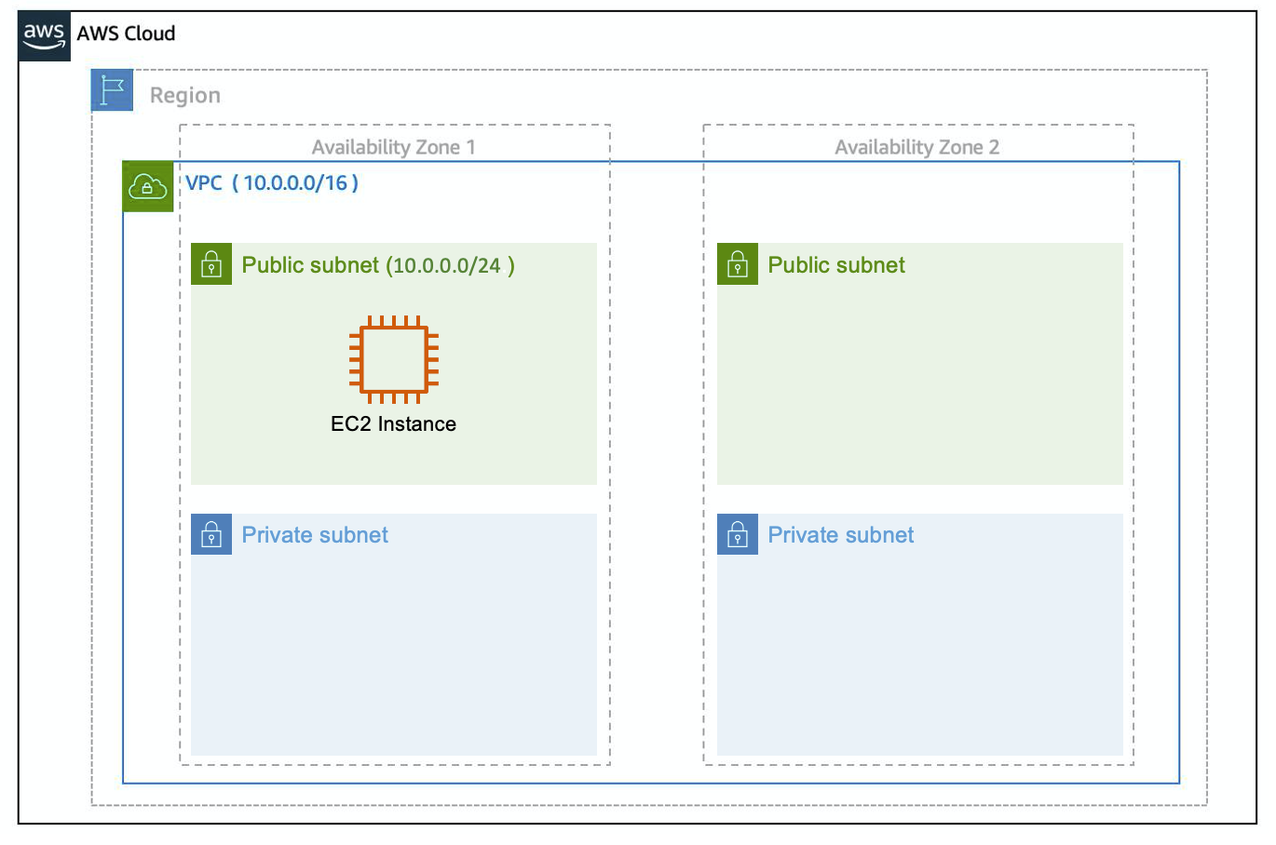
Using this information, AWS will provision a network and IP addresses for that network.



**Create a Subnet** After you create your VPC, you need to create subnets inside of this network. Think of subnets as smaller networks inside your base network—or virtual area networks (VLANs) in a traditional, on-premises network. In an on-premises network, the typical use case for subnets is to isolate or optimize network traffic. In AWS, subnets are used for high availability and providing different connectivity options for your resources. When you create a subnet, you need to choose three settings.

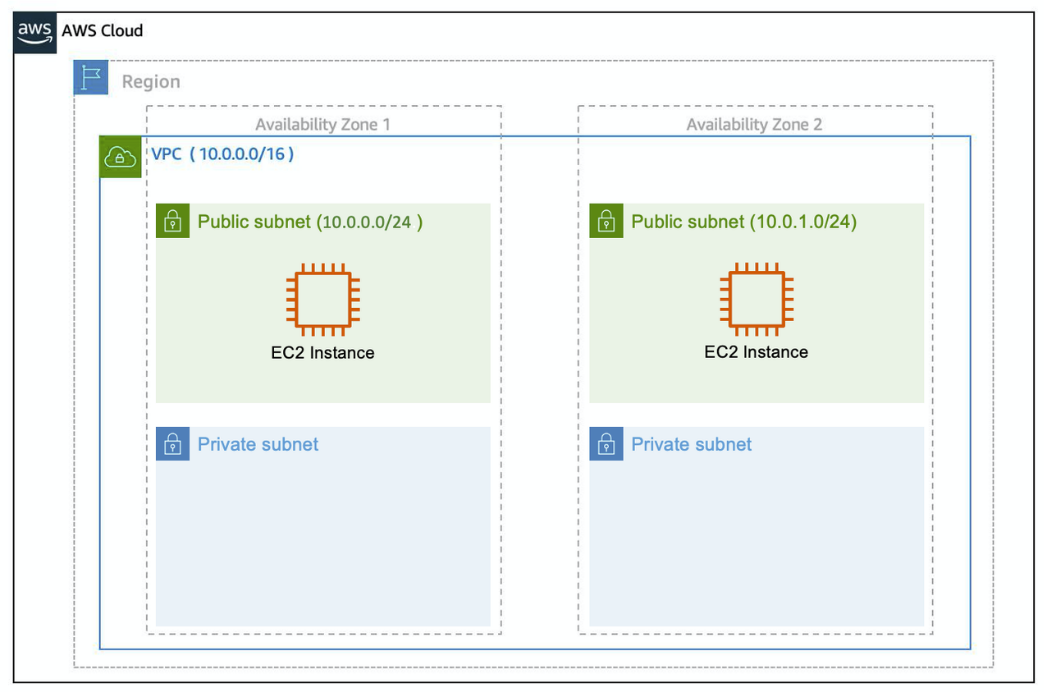
1. The VPC you want your subnet to live in, in this case VPC (10.0.0.0/16).
2. The Availability Zone you want your subnet to live in, in this case AZ1.
3. A CIDR block for your subnet, which must be a subset of the VPC CIDR block, in this case 10.0.0.0/24.

When you launch an EC2 instance, you launch it inside a subnet, which will be located inside the Availability Zone you choose.



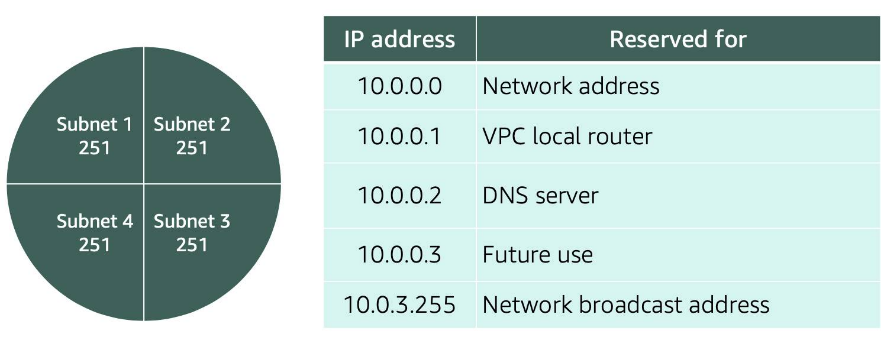
**High Availability with A VPC** When you create your subnets, keep high availability in mind. In order to maintain redundancy and fault tolerance, create at least two subnets configured in two different Availability Zones.

As you learned earlier in the trail, it’s important to consider that “everything fails all the time.” In this case, if one of these AZs fail, you still have your resources in another AZ available as backup.



**Reserved IPs** For AWS to configure your VPC appropriately, AWS reserves five IP addresses in each subnet. These IP addresses are used for routing, Domain Name System (DNS), and network management.

For example, consider a VPC with the IP range 10.0.0.0/22. The VPC includes 1,024 total IP addresses. This is divided into four equal-sized subnets, each with a /24 IP range with 256 IP addresses. Out of each of those IP ranges, there are only 251 IP addresses that can be used because AWS reserves five.



Since AWS reserves these five IP addresses, it can impact how you design your network. A common starting place for those who are new to the cloud is to create a VPC with a IP range of /16 and create subnets with a IP range of /24. This provides a large amount of IP addresses to work with at both the VPC and subnet level.

**Gateways**

**Internet Gateway**

To enable internet connectivity for your VPC, you need to create an internet gateway. Think of this gateway as similar to a modem. Just as a modem connects your computer to the internet, the internet gateway connects your VPC to the internet. Unlike your modem at home, which sometimes goes down or offline, an internet gateway is highly available and scalable. After you create an internet gateway, you then need to attach it to your VPC.

**Virtual Private Gateway**

A virtual private gateway allows you to connect your AWS VPC to another private network. Once you create and attach a VGW to a VPC, the gateway acts as anchor on the AWS side of the connection. On the other side of the connection, you’ll need to connect a customer gateway to the other private network. A customer gateway device is a physical device or software application on your side of the connection. Once you have both gateways, you can then establish an encrypted VPN connection between the two sides.

# Amazon VPC Routing and Security

**The Main Route Table**

When you create a VPC, AWS creates a route table called the main route table. A route table contains a set of rules, called routes, that are used to determine where network traffic is directed. AWS assumes that when you create a new VPC with subnets, you want traffic to flow between them. Therefore, the default configuration of the main route table is to allow traffic between all subnets in the local network. Below is an example of a main route table:



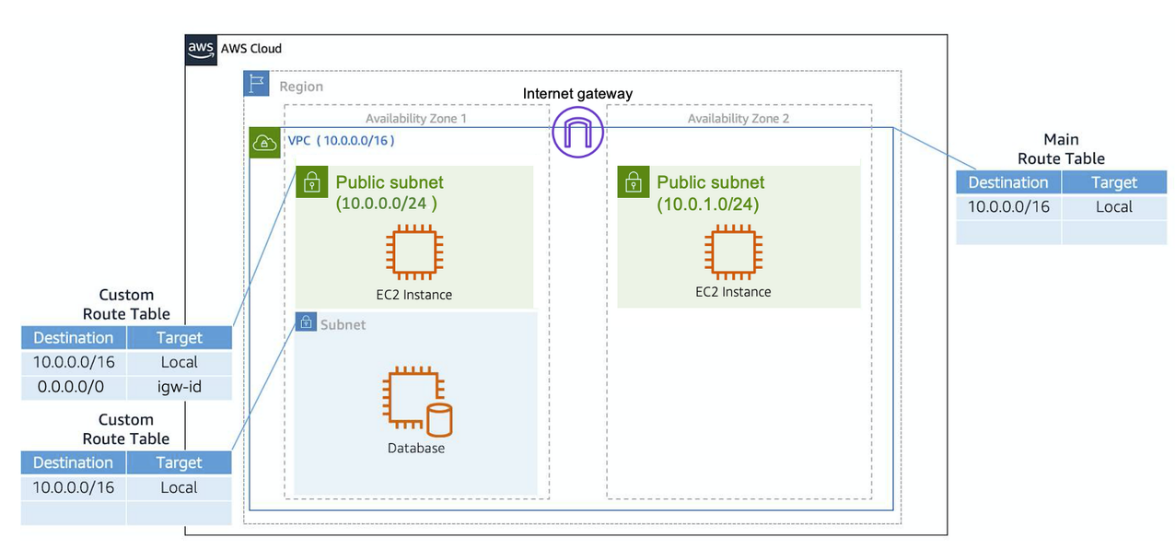
There are two main parts to this route table.

* The destination, which is a range of IP addresses where you want your traffic to go. In the example of sending a letter, you need a destination to route the letter to the appropriate place. The same is true for routing traffic. In this case, the destination is the IP range of our VPC network.
* The target, which is the connection through which to send the traffic. In this case, the traffic is routed through the local VPC network.

**Custom Route Tables**

While the main route table controls the routing for your VPC, you may want to be more granular about how you route your traffic for specific subnets. For example, your application may consist of a frontend and a database. You can create separate subnets for these resources and provide different routes for each of them.

If you associate a custom route table with a subnet, the subnet will use it instead of the main route table. By default, each custom route table you create will have the local route already inside it, allowing communication to flow between all resources and subnets inside the VPC.



**Secure Your Subnets with Network ACLs**

Think of a network ACL as a firewall at the subnet level. A network ACL enables you to control what kind of traffic is allowed to enter or leave your subnet. You can configure this by setting up rules that define what you want to filter. Here’s an example.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inbound** |  |  |  |  |  |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow/Deny** |
| 100 | All IPv4 traffic | All | All | 0.0.0.0/0 | ALLOW |
| \* | All IPv4 traffic | All | All | 0.0.0.0/0 | DENY |
| **Outbound** |  |  |  |  |  |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow/Deny** |
| 100 | All IPv4 traffic | All | All | 0.0.0.0/0 | ALLOW |
| \* | All IPv4 traffic | All | All | 0.0.0.0/0 | DENY |

The default network ACL, shown in the table above, allows all traffic in and out of your subnet. To allow data to flow freely to your subnet, this is a good starting place. However, you may want to restrict data at the subnet level. For example, if you have a web application, you might restrict your network to allow HTTPS traffic and remote desktop protocol (RDP) traffic to your web servers.

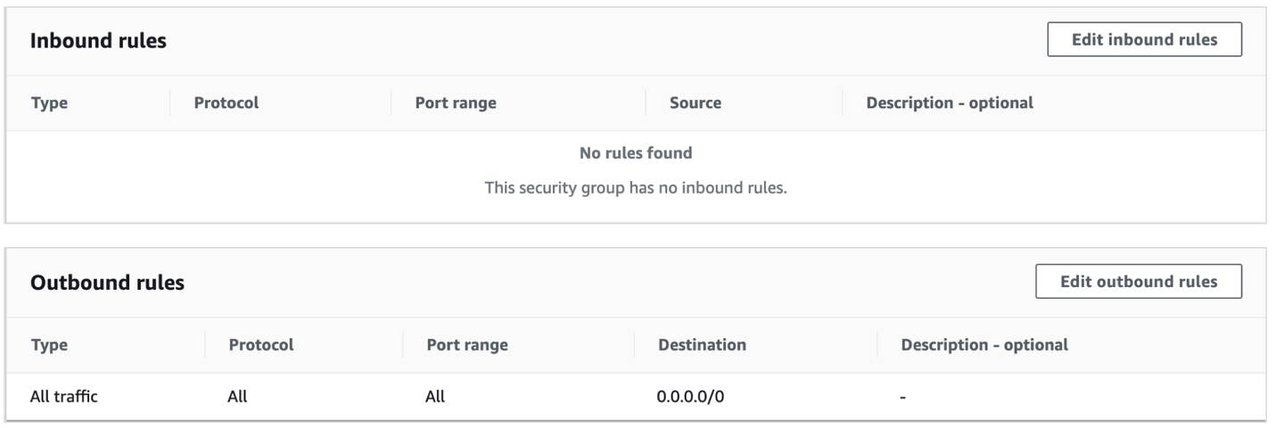
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inbound** |  |  |  |  |  |
| **Rule #** | **Source IP** | **Protocol** | **Port** | **Allow/Deny** | **Comments** |
| 100 | All IPv4 traffic | TCP | 443 | ALLOW | Allows inbound HTTPS traffic from anywhere |
| 130 | 192.0.2.0/24 | TCP | 3389 | ALLOW | Allows inbound RDP traffic to the web servers from your home network’s public IP address range (over the internet gateway) |
| \* | All IPv4 traffic | All | All | DENY | Denies all inbound traffic not already handled by a preceding rule (not modifiable) |
| **Outbound** |  |  |  |  |  |
| **Rule #** | **Destination IP** | **Protocol** | **Port** | **Allow/Deny** | **Comments** |
| 120 | 0.0.0.0/0 | TCP | 1025-65535 | ALLOW | Allows outbound responses to clients on the internet (serving people visiting the web servers in the subnet) |
| \* | 0.0.0.0/0 | All | All | DENY | Denies all outbound traffic not already handled by a preceding rule (not modifiable) |

Notice that in the network ACL example above, you allow inbound 443 and outbound range 1025-65535. That’s because HTTP uses port 443 to initiate a connection and will respond to an ephemeral port. Network ACL’s are considered stateless, so you need to include both the inbound and outbound ports used for the protocol. If you don’t include the outbound range, your server would respond but the traffic would never leave the subnet.

Since network ACLs are configured by default to allow incoming and outgoing traffic, you don’t need to change their initial settings unless you need additional security layers.

**Secure Your EC2 Instances with Security Groups**

The next layer of security is for your EC2 Instances. Here, you can create a firewall called a security group. The default configuration of a security group blocks all inbound traffic and allows all outbound traffic.

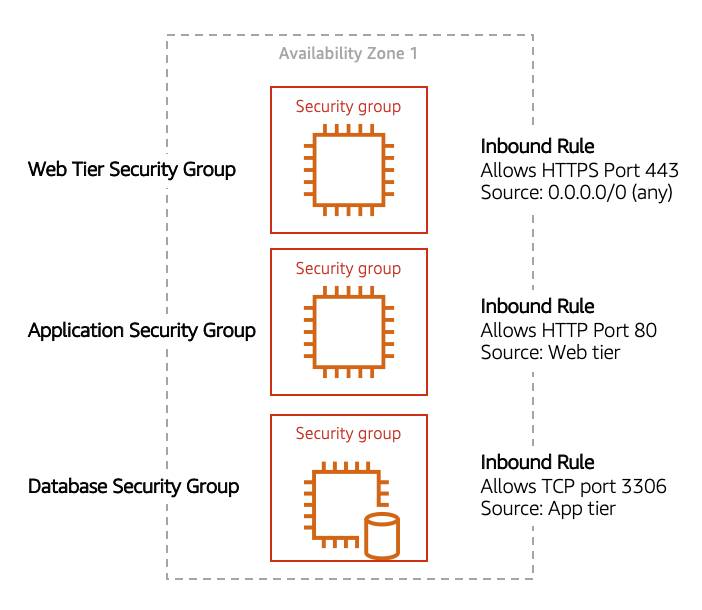


You might be wondering: “Wouldn’t this block all EC2 instances from receiving the response of any customer requests?” Well, security groups are stateful, meaning they will remember if a connection is originally initiated by the EC2 instance or from the outside and temporarily allow traffic to respond without having to modify the inbound rules.

If you want your EC2 instance to accept traffic from the internet, you’ll need to open up inbound ports. If you have a web server, you may need to accept HTTP and HTTPS requests to allow that type of traffic in through your security group. You can create an inbound rule that will allow port 80 (HTTP) and port 443 (HTTPS) as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Inbound rules** |  |  |  |
| **Type** | **Protocol** | **Port Range** | **Source** |
| HTTP (80) | TCP (6) | 80 | 0.0.0.0/0 |
| HTTP (80) | TCP (6) | 80 | ::/0 |
| HTTPS (443) | TCP (6) | 443 | 0.0.0.0/0 |
| HTTPS (443) | TCP (6) | 443 | ::/0 |

You learned in a previous unit that subnets can be used to segregate traffic between computers in your network. Security groups can be used to do the same thing. A common design pattern is organizing your resources into different groups and creating security groups for each to control network communication between them.



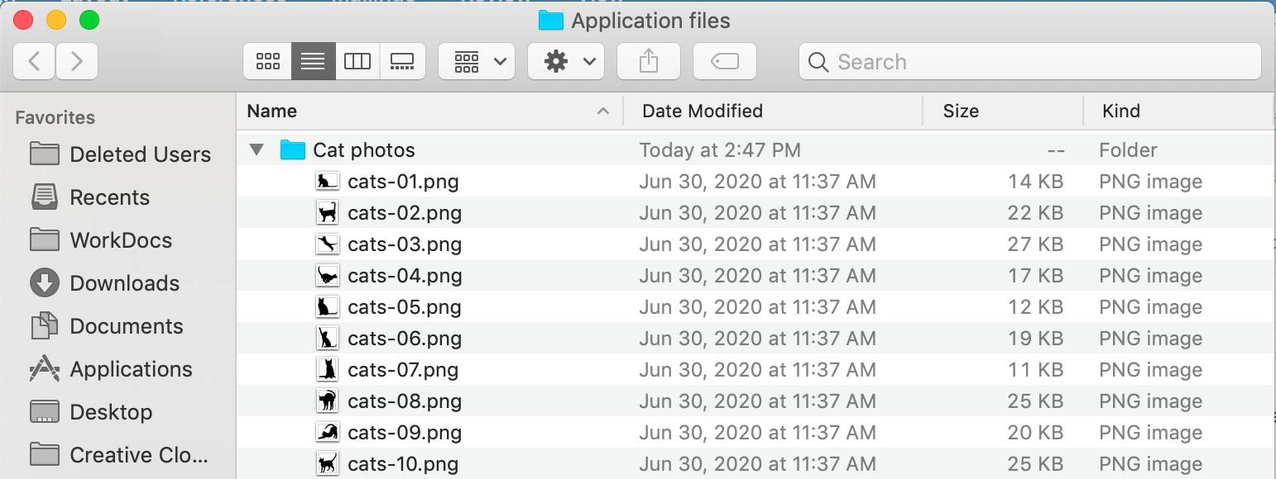
This example allows you to define three tiers and isolate each tier with the security group rules you define. In this case, you only allow internet traffic to the web tier over HTTPS, Web Tier to Application Tier over HTTP, and Application tier to Database tier over MySQL. This is different from traditional on-premises environments, in which you isolate groups of resources via VLAN configuration. In AWS, security groups allow you to achieve the same isolation without tying it to your network.

# Storage Types on AWS

AWS storage services are grouped into three different categories: block storage, file storage, and object storage.

**File Storage**

You may be familiar with file storage if you’ve interacted with file storage systems like Windows File Explorer or Finder on MacOS. You place your files in a tree-like hierarchy that consists of folders and subfolders. For example, if you have hundreds of cat photos on your laptop, you may want to create a folder called Cat photos and place those images inside that folder to organize them. Since you know these images will be used in an application, you may want to place the cat photos folder inside another folder called Application files.



Each file has metadata such as file name, file size, and the date the file was created. The file also has a path, for example, computer/Application\_files/Cat\_photos/cats-03.png. When you need to retrieve a file, your system can use the path to find it in the file hierarchy.

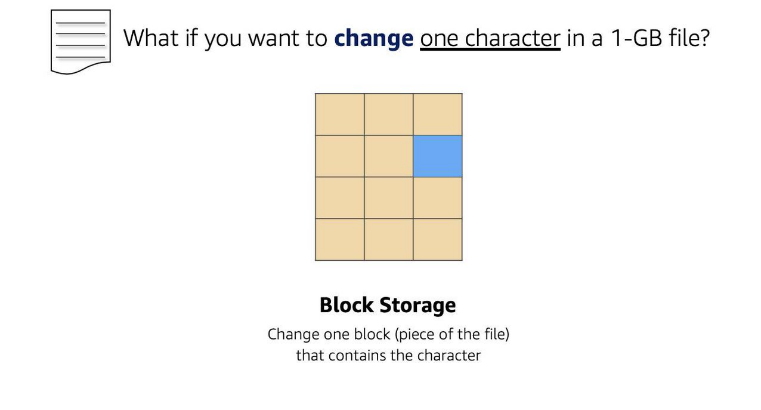
File storage is ideal when you require centralized access to files that need to be easily shared and managed by multiple host computers. Typically, this storage is mounted onto multiple hosts and requires file locking and integration with existing file system communication protocols. Common use cases for file storage include:

* Large content repositories
* Development environments
* User home directories

**Block Storage**

While file storage treats files as a singular unit, block storage splits files into fixed-size chunks of data called blocks that have their own addresses. Since each block is addressable, blocks can be retrieved efficiently.

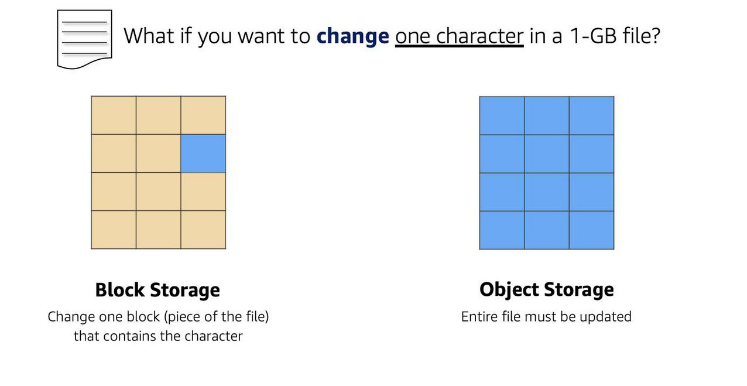
When data is requested, these addresses are used by the storage system to organize the blocks in the correct order to form a complete file to present back to the requestor. Outside of the address, there is no additional metadata associated with each block. So, when you want to change a character in a file, you just change the block, or the piece of the file, that contains the character. This ease of access is why block storage solutions are fast and use less bandwidth.



**Object Storage**

Objects, much like files, are also treated as a single unit of data when stored. However, unlike file storage, these objects are stored in a flat structure instead of a hierarchy. Each object is a file with a unique identifier. This identifier, along with any additional metadata, is bundled with the data and stored.

Changing just one character in an object is more difficult than with block storage. When you want to change one character in a file, the entire file must be updated.



With object storage, you can store almost any type of data, and there is no limit to the number of objects stored, making it easy to scale. Object storage is generally useful when storing large data sets, unstructured files like media assets, and static assets, such as photos.

**Relate Back to Traditional Storage Systems**

If you’ve worked with storage on-premises, you may already be familiar with block, file, and object storage. Consider the following technologies and how they relate to systems you may have seen before.

* Block storage in the cloud is analogous to **direct-attached storage (DAS) or a storage area network (SAN).**
* File storage systems are often supported with a **network attached storage (NAS)** server.

Adding more storage in a traditional data center environment is a more rigid process, as you need to purchase, install, and configure these storage solutions. With cloud computing, the process is more flexible. You can create, delete, and modify storage solutions all within a matter of minutes.

# Amazon EC2 Instance Storage and Amazon Elastic Block Store

**Amazon EC2 Instance Store**

Amazon EC2 Instance Store provides temporary block-level storage for your instance. This storage is located on disks that are physically attached to the host computer. This ties the lifecycle of your data to the lifecycle of your EC2 instance. If you delete your instance, the instance store is deleted as well. Due to this, instance store is considered ephemeral storage. Read more about it in the [AWS documentation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html) .

Instance store is ideal if you are hosting applications that replicate data to other EC2 instances, such as Hadoop clusters. For these cluster-based workloads, having the speed of locally attached volumes and the resiliency of replicated data helps you achieve data distribution at high performance. It’s also ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content.

**Amazon Elastic Block Storage (Amazon EBS)** As the name implies, Amazon EBS is a block-level storage device that you can attach to an Amazon EC2 instance. These storage devices are called Amazon EBS volumes. EBS volumes are essentially drives of a user-configured size attached to an EC2 instance, similar to how you might attach an external drive to your laptop.

EBS volumes act similarly to external drives in more than one way.

* Most Amazon EBS volumes can only be connected with one computer at a time. Most EBS volumes have a one-to-one relationship with EC2 instances, so they cannot be shared by or attached to multiple instances at one time. Note: Recently, AWS announced the Amazon EBS multi-attach feature that enables volumes to be attached to multiple EC2 instances at one time. This feature is not available for all instance types and all instances must be in the same Availability Zone. Read more about this scenario in the [EBS documentation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html) .
* You can detach an EBS volume from one EC2 instance and attach it to another EC2 instance in the same Availability Zone, to access the data on it.
* The external drive is separate from the computer. That means, if an accident happens and the computer goes down, you still have your data on your external drive. The same is true for EBS volumes.
* You’re limited to the size of the external drive, since it has a fixed limit to how scalable it can be. For example, you may have a 2 TB external drive and that means you can only have 2 TB of content on there. This relates to EBS as well, since volumes also have a max limitation of how much content you can store on the volume.

**Scale Amazon EBS Volumes**

You can scale Amazon EBS volumes in two ways.

1. Increase the volume size, as long as it doesn’t increase above the maximum size limit. For EBS volumes, the maximum amount of storage you can have is 16 TB. That means if you provision a 5 TB EBS volume, you can choose to increase the size of your volume until you get to 16 TB.
2. Attach multiple volumes to a single Amazon EC2 instance. EC2 has a one-to-many relationship with EBS volumes. You can add these additional volumes during or after EC2 instance creation to provide more storage capacity for your hosts.

**Amazon EBS Use Cases**

Amazon EBS is useful when you need to retrieve data quickly and have data persist long-term. Volumes are commonly used in the following scenarios.

* **Operating systems**: Boot/root volumes to store an operating system. The root device for an instance launched from an Amazon Machine Image (AMI) is typically an Amazon EBS volume. These are commonly referred to as EBS-backed AMIs.
* **Databases**: A storage layer for databases running on Amazon EC2 that rely on transactional reads and writes.
* **Enterprise applications**: Amazon EBS provides reliable block storage to run business-critical applications.
* **Throughput-intensive applications**: Applications that perform long, continuous reads and writes.

**Amazon EBS Volume Types**

|  | **EBS Provisioned IOPS SSD** | **EBS General Purpose SSD** | **Throughput Optimized HDD** | **Cold HDD** |
| --- | --- | --- | --- | --- |
| **Description** | Highest performance SSD designed for latency-sensitive transactional workloads | General purpose SSD that balances price and performance for a wide variety of transactional workloads | Low-cost HDD designed for frequently accessed, throughput intensive workloads | Lowest cost HDD designed for less frequently accessed workloads |
| **Use Cases** | I/O-intensive NoSQL and relational databases | Boot volumes, low-latency interactive apps, development, and test | Big data, data warehouses, log processing | Colder data requiring fewer scans per day |
| **Volume Size** | 4 GB-16 TB | 1 GB-16 TB | 500 GB-16 TB | 500 GB-16 TB |
| **Max IOPS/Volume** | 64,000 | 16,000 | 500 | 250 |
| **Max Throughput/Volume** | 1,000 MB/s | 250 MB/s | 500 MB/s | 250 MB/s |

There are two main categories of Amazon EBS volumes: **solid-state drives (SSDs)** and **hard-disk drives (HDDs)**. SSDs provide strong performance for random input/output (I/O), while HDDs provide strong performance for sequential I/O. AWS offers two types of each. The following chart can help you decide which EBS volume is the right option for your workload.

**Benefits of Using Amazon EBS**

Here are the following benefits of using Amazon EBS (in case you need a quick cheat sheet).

* **High availability**: When you create an EBS volume, it is automatically replicated within its Availability Zone to prevent data loss from single points of failure.
* **Data persistence**: The storage persists even when your instance doesn’t.
* **Data encryption**: All EBS volumes support encryption.
* **Flexibility**: EBS volumes support on-the-fly changes. You can modify volume type, volume size, and input/output operations per second (IOPS) capacity without stopping your instance.
* **Backups**: Amazon EBS provides you the ability to create backups of any EBS volume.

**EBS Snapshots**

Errors happen. One of those errors is not backing up data, and then, inevitably losing that data. To prevent this from happening to you, you should back up your data—even in AWS. Since your EBS volumes consist of the data from your Amazon EC2 instance, you’ll want to take backups of these volumes, called snapshots.

EBS snapshots are incremental backups that only save the blocks on the volume that have changed after your most recent snapshot. For example, if you have 10 GB of data on a volume, and only 2 GB of data have been modified since your last snapshot, only the 2 GB that have been changed are written to Amazon Simple Storage Service (Amazon S3).

When you take a snapshot of any of your EBS volumes, these backups are stored redundantly in multiple Availability Zones using Amazon S3. This aspect of storing the backup in Amazon S3 will be handled by AWS, so you won’t need to interact with Amazon S3 to work with your EBS snapshots. You simply manage them in the EBS console (which is part of the EC2 console).

EBS snapshots can be used to create multiple new volumes, whether they’re in the same Availability Zone or a different one. When you create a new volume from a snapshot, it’s an exact copy of the original volume at the time the snapshot was taken.