AWS Compute

The first building block you need to host an application is a server. Servers often 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 sends a request, and a server handling the requests is a computer, or collection of computers, connected to the internet serving websites to internet users.

To run an HTTP server on AWS, you need to find a service that provides compute power in the AWS Management Console.

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.

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?

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?

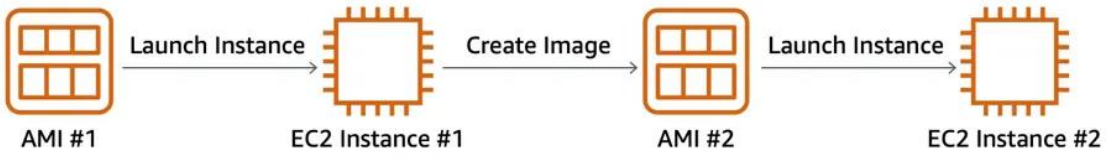
EC2 instances are live instantiations of what is defined in an AMI, much like a cake is a live instantiation of a cake recipe. If you are familiar with software development, you can also see this kind of relationship between a Class and an Object.

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.

Note:

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.



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.

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.

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.

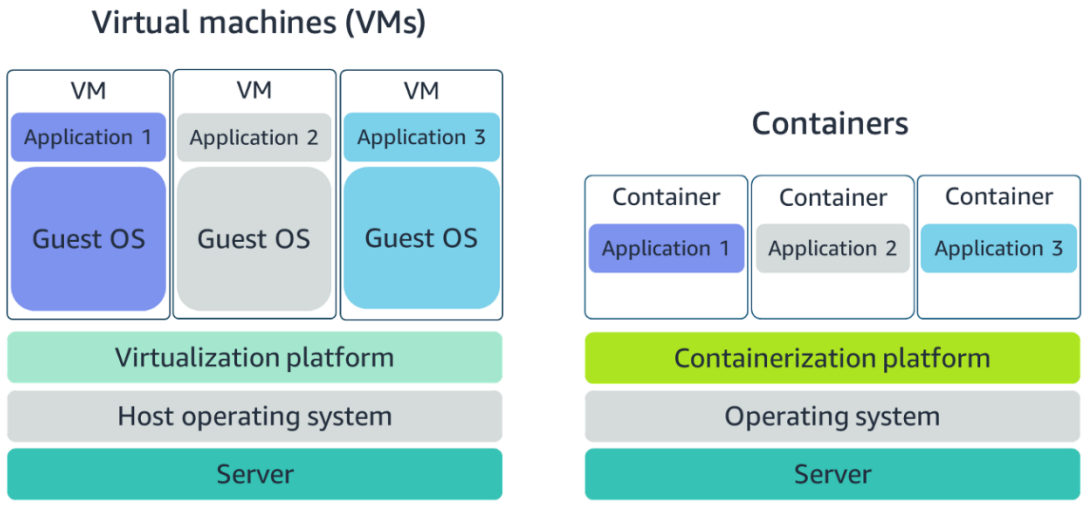
On-Demand and *No Upfront* are similar since both do not require any upfront payment. However, there is a major difference. When you choose an On-Demand instance, you stop paying for the instance when you stop or terminate the instance. When you stop an RI, you still pay for it because you committed to a 1-year or 3-year term.

WHAT IS A CONTAINER?

A container is a standardized unit that packages up your code and all 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.

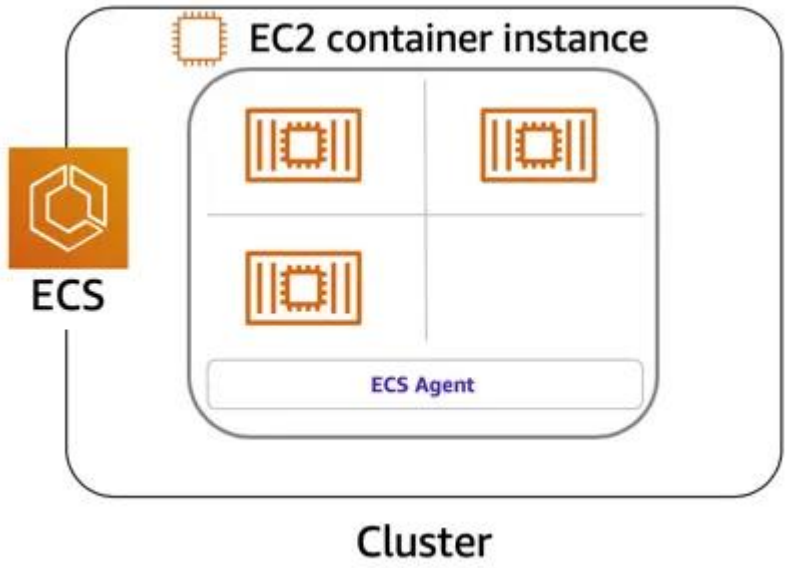
WHAT IS THE DIFFERENCE BETWEEN CONTAINERS AND VMS?

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 must 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.



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*.

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 like 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)

Amazon EKS is conceptually like 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.

Serverless and AWS Lambda

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.

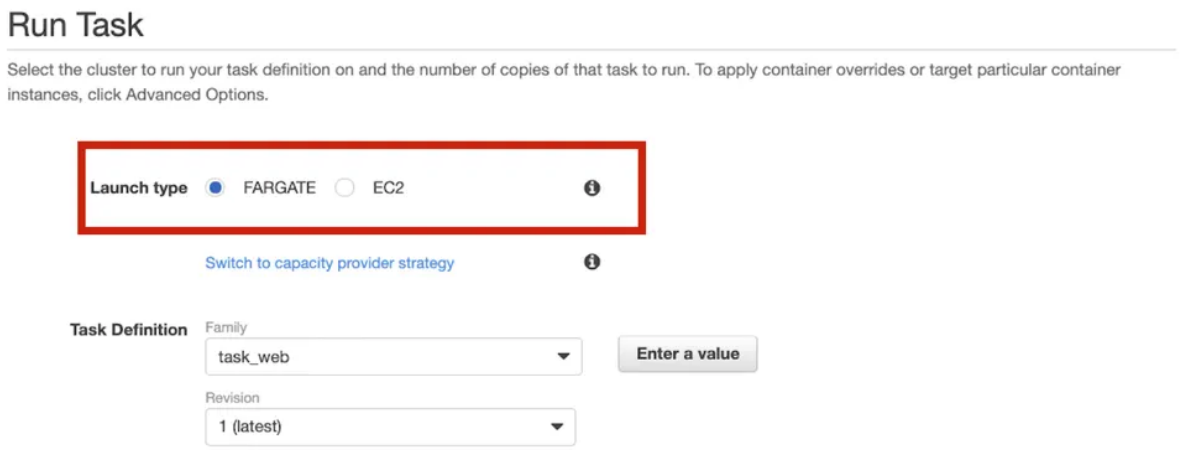
- No servers to provision or manage.
- Scales with usage.
- You never pay for idle resources.
- Availability and fault tolerance are built in.

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



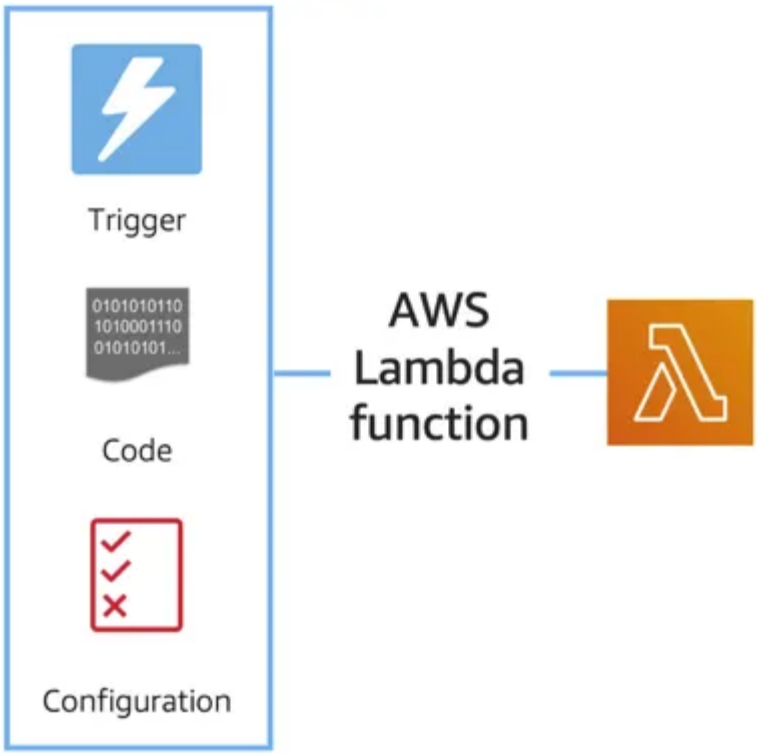
AWS Fargate is a purpose-built serverless compute engine for containers. Fargate scales and manages the infrastructure, allowing developers to work on what they do best: application development. It achieves this by allocating the right amount of compute, eliminating the need to choose and handle EC2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

RUN YOUR CODE ON AWS LAMBDA

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

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.

All you need is what, how, and when of a Lambda function to have functional compute capacity that runs only when you need it to.

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

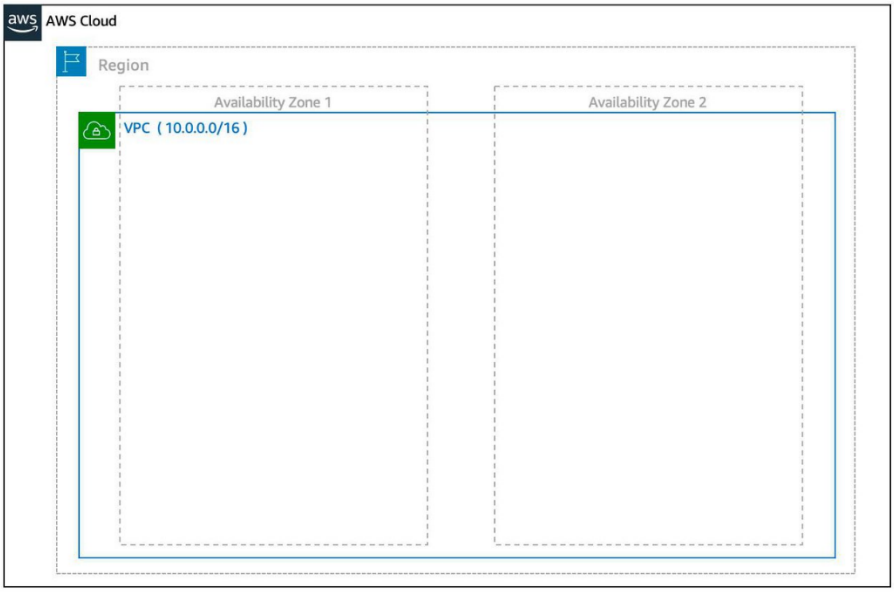
AWS NETWORKING

Introduction to Amazon VPC

A VPC is an isolated network you create in the AWS cloud, like a traditional network in a data centre. 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.

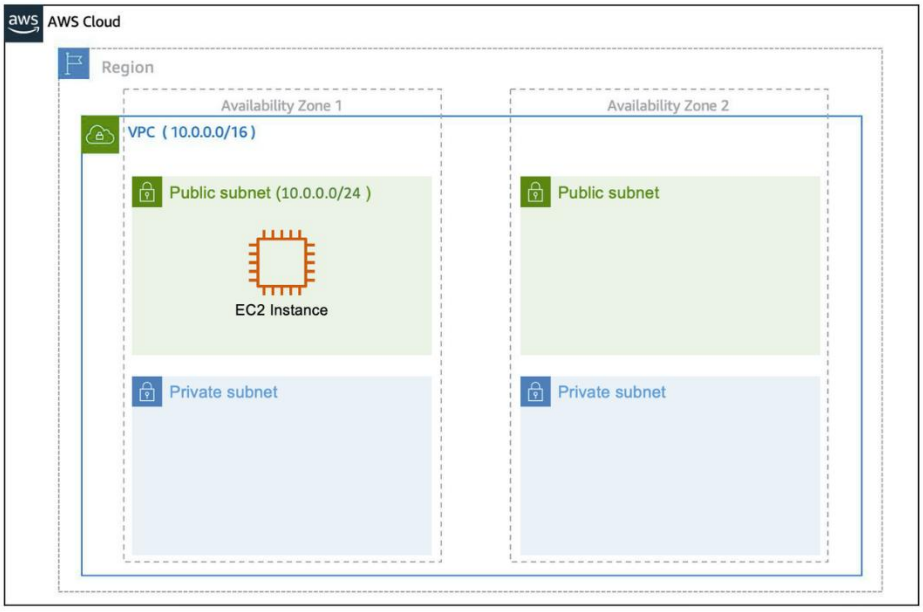
Using this information, AWS will provision a network and IP addresses for that network.



Create a Subnet: After you create your VPC, you need to create subnets inside of this network. 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.

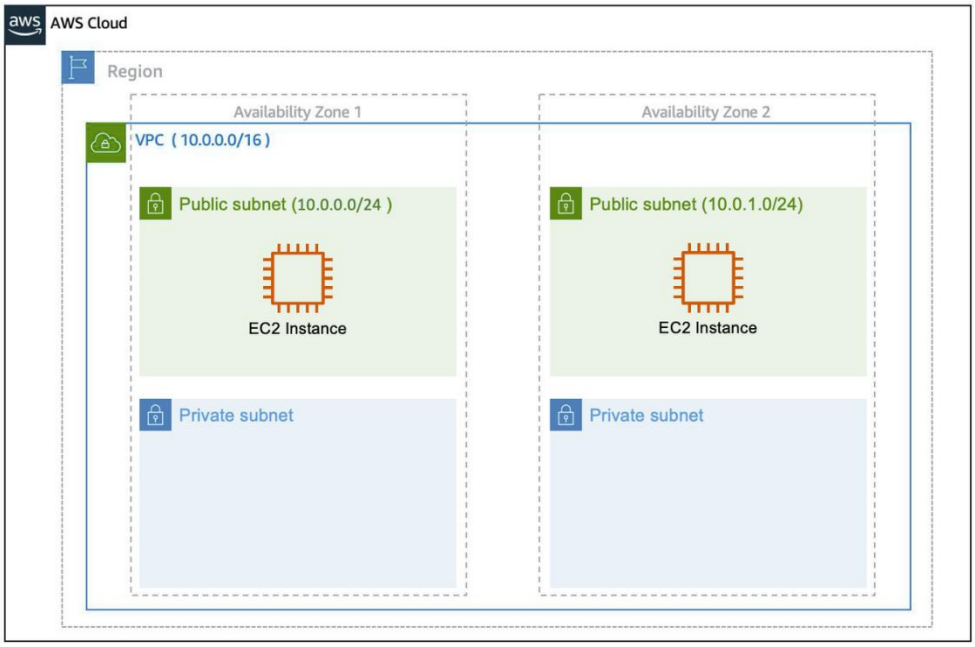
1. The VPC you want your subnet to live in, in this case VPC (10.0.0.0/16).
2. The Availability Zone you want your subnet to live in, in this case AZ1.
3. A CIDR block for your subnet, which must be a subset of the VPC CIDR block, in this case 10.0.0.0/24.

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. 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.

Gateways

Internet Gateway

To enable internet connectivity for your VPC, you need to create an internet gateway. Think of this gateway like 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.

Destination	Target	Status	Propagated
10.2.0.0/16	local	active	No

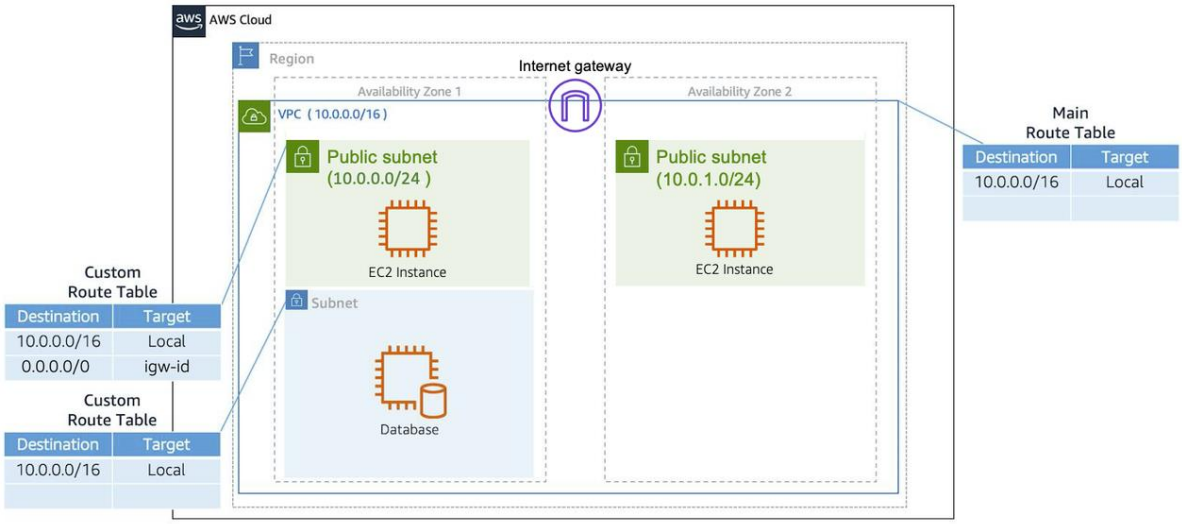
There are two main parts to this route table.

- 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.

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.

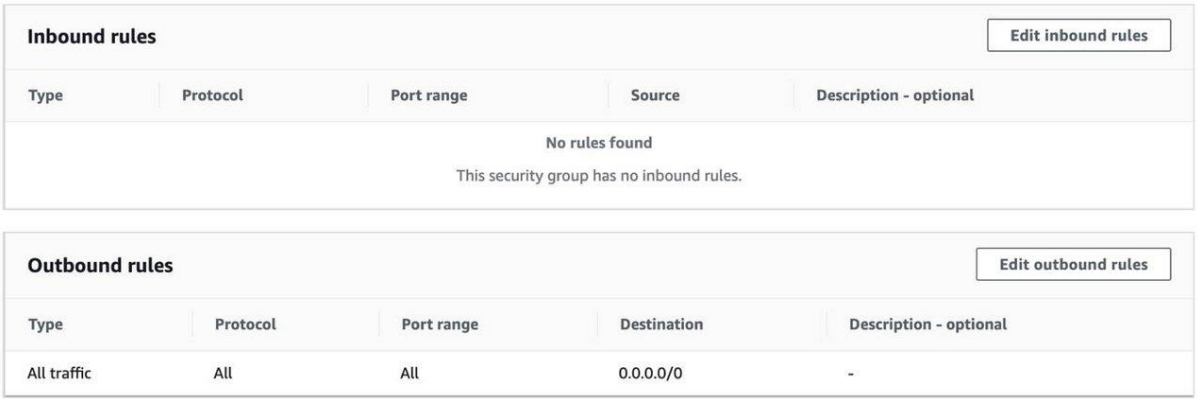
Inbound					
Rule #	**Type**	**Protocol**	**Port Range**	**Source**	**Allow/Deny**
100	All IPv4 traffic	All	All	0.0.0.0/0	ALLOW
*	All IPv4 traffic	All	All	0.0.0.0/0	DENY
Outbound					
Rule #	**Type**	**Protocol**	**Port Range**	**Source**	**Allow/Deny**
100	All IPv4 traffic	All	All	0.0.0.0/0	ALLOW
*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

Network ACLs 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.

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 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

Common network troubleshooting steps for Amazon VPC

you will see the Employee Directory Application being launched onto an Amazon EC2 instance that is placed in a public subnet in an Amazon VPC. There are multiple networking configurations that play into whether an instance is accessible to the internet or not.

Below we will run down a list of configurations you should check if you ever have a public EC2 instance with a web application that is not loading as expected.

1. Internet gateway

Ensure that an Internet Gateway (IGW) is attached to your VPC. Without the internet gateway, no traffic will be allowed in or out of the VPC.

2. Route tables

Check the route table associated with the subnet of your EC2 instance. Ensure there is a route with a destination of 0.0.0.0/0 that points to the Internet Gateway. This route allows outbound traffic to the internet.

3. Security groups

Review the security group settings attached to your EC2 instance. Make sure there are inbound rules allowing HTTP (port 80) and/or HTTPS (port 443) traffic from the internet (0.0.0.0/0). Also, verify that outbound rules allow traffic to leave the instance.

4. Network Access Control Lists

Check the NACLs associated with your subnet. Ensure that they allow inbound and outbound traffic for HTTP and HTTPS. Unlike security groups, NACLs are stateless, so you must explicitly allow both inbound and outbound rules.

5. Public IP address

Ensure your EC2 instance has a public IP address assigned. You can check this in the EC2 console under the instance details. If it does not have a public IP, relaunch the instance and ensure you have the configuration for assigning a public IP address set correctly.

6. HTTP vs HTTPS

Confirm that your application is accessible via the correct protocol. If your application is configured for HTTPS, ensure SSL/TLS certificates are correctly installed and configured. Also, check if the web browser is trying to connect via the wrong protocol (HTTP instead of HTTPS or vice versa). For this course, the application is operating via HTTP, double check that your browser is not trying to connect via HTTPS. You can do this by selecting the address bar in the browser and making sure the address starts with http and not https.

7. User data script

If your instance uses a user data script to configure the application on launch, verify that the script has run successfully. Check the instance logs (/var/log/cloud-init.log or /var/log/cloud-init-output.log) for any errors that may have occurred during the execution of the user data script.

8. Permissions

Verify the permissions and roles attached to your EC2 instance. Ensure the instance has the necessary IAM roles and policies to access any required AWS services, such as S3, DynamoDB, or RDS.

9. Personal network permissions

Ensure that your personal or corporate network does not have restrictions blocking access to the public IP address of your EC2 instance. Some networks might have firewalls or proxy settings that could block outbound traffic to certain IP ranges or ports.

10. Application

Ensure that your application code is correctly deployed and running. Check the application's logs to diagnose any runtime errors. Also, make sure the web server (e.g., Apache, Nginx) is installed and running.

STORAGE 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.

File storage is ideal when you require centralized access to files that need to be easily shared and managed by multiple host computers.

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.

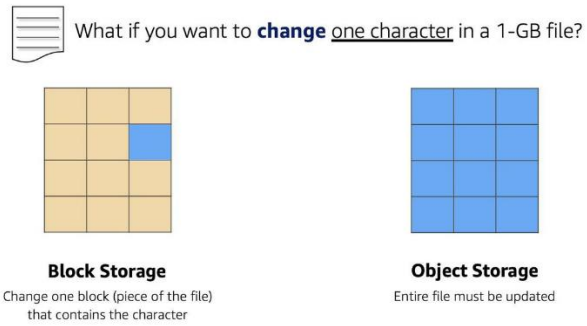
When data is requested, these addresses are used by the storage system to organize the blocks in the correct order to form a complete file to present back to the requestor.

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.

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.



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.

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, like 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 relate to 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.*
- 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.

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 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

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).

Object Storage with Amazon S3

WHAT IS AMAZON S3?

Amazon S3 is an *object storage service*. Object storage stores data in a flat structure, using unique identifiers to look up objects when requested.

UNDERSTAND AMAZON S3 CONCEPTS

In Amazon S3, you must store your objects in containers called **buckets**. You can't upload any object, not even a single photo, to S3 without creating a bucket first. When you create a bucket, you choose, at the very minimum**, two things: the bucket name and the AWS Region you want the bucket to reside in.**

AWS uses this name as part of the object identifier. In S3, each object is identified using a URL, which looks like this:

After the http://, you see the bucket name. In this example, the bucket is named *doc*. Then, the identifier uses the service name, *s3* and specifies the service provider *amazonaws*. After that, you have an implied folder inside the bucket called *2006-03-01* and the object inside the folder that is named *AmazonS3.html*. The object name is often referred to as *the key name*.

S3 USE CASES

Amazon S3 is one of the most widely used storage services, with far more use cases than could fit on one screen. The following list summarizes some of the most common ways you can use Amazon S3.

- **Backup and storage:** S3 is a natural place to back up files because it is highly redundant. As mentioned in the last unit, AWS stores your EBS snapshots in S3 to take advantage of its high availability.

- **Media hosting:** Because you can store unlimited objects, and each individual object can be up to 5 TBs, S3 is an ideal location to host video, photo, or music uploads.
- **Software delivery:** You can use S3 to host your software applications that customers can download.
- **Data lakes:** S3 is an optimal foundation for a data lake because of its virtually unlimited scalability. You can increase storage from gigabytes to petabytes of content, paying only for what you use.
- **Static websites:** You can configure your bucket to host a static website of HTML, CSS, and client-side scripts.
- **Static content:** Because of the limitless scaling, the support for large files, and the fact that you access any object over the web at any time, S3 is the perfect place to store static content.

CHOOSE THE RIGHT CONNECTIVITY OPTION FOR YOUR RESOURCES

Everything in Amazon S3 is private by default. This means that all S3 resources, such as buckets, folders, and objects can only be viewed by the user or AWS account that created that resource. Amazon S3 resources are all private and protected to begin with.

If you decide that you want everyone on the internet to see your photos, you can choose to make your buckets, folders, and objects public. Keep in mind that a public resource means that everyone on the internet can see it. Most of the time, you don't want your permissions to be all or nothing. Typically, you want to be more granular about the way you provide access to your resources.

To be more specific about who can do what with your S3 resources, Amazon S3 provides two main access management features**: IAM policies and S3 bucket policies**.

UNDERSTAND IAM POLICIES

Previously, you learned about creating and using IAM policies, and now you get to apply this to Amazon S3. When IAM policies are attached to IAM users, groups, and roles, the policies define which actions they can perform. IAM policies are not tied to any one AWS service and can be used to define access to nearly any AWS action**. You should use IAM policies for private buckets when:**

- You have many buckets with different permission requirements. Instead of defining many different S3 bucket policies, you can use IAM policies instead.
- You want all policies to be in a centralized location.

UNDERSTAND S3 BUCKET POLICIES

S3 bucket policies are like IAM policies, in that they are both defined using the same policy language in a JSON format. The difference is IAM policies are attached to users, groups, and roles, whereas S3 bucket policies are only attached to **buckets**. S3 bucket policies specify what actions are allowed or denied on the bucket.

For example, if you have a bucket called employeebucket, you can attach an S3 bucket policy to it that allows another AWS account to put objects in that bucket.

Or if you wanted to allow anonymous viewers to read the objects in employeebucket, then you can apply a policy to that bucket that allows anyone to read objects in the bucket using *"Effect":Allow* on the *"Action:["s3:GetObject"]"*.

S3 Bucket policies can only be placed on buckets and cannot be used for folders or objects. However, the policy that is placed on the bucket applies to every object in that bucket. You should use S3 bucket policies when:

- You need a simple way to do cross-account access to S3, without using IAM roles.
- Your IAM policies bump up against the defined size limit. S3 bucket policies have a larger size limit.

ENCRYPT S3

Amazon S3 reinforces encryption in transit (as it travels to and from Amazon S3) and at rest. To protect data at rest, you can use:

- **Server-side encryption:** This allows Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects.
- **Client-side encryption:** Encrypt your data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and all related tools.

To encrypt in transit, you can use client-side encryption or Secure Sockets Layer (SSL).

USE VERSIONING TO PRESERVE OBJECTS

As you know, Amazon S3 identifies objects in part by using the object name. For example, when you upload an employee photo to S3, you may name the object employee.jpg and store it in a folder called employees. If you don't use Amazon S3 versioning, anytime you upload an object called employee.jpg to the employees' folder, it overwrites the original file. This can be an issue for several reasons.

- employee.jpg is a common name for an employee photo object. You or someone else who has access to that bucket might not have intended to overwrite it, and now that you have, you no longer have access to the original file.

- You may want to preserve different versions of employee.jpg. Without versioning, if you wanted to create a new version of employee.jpg, you would need to upload the object and choose a different name for it. Having several objects all with slight differences in naming variations may cause confusion and clutter in your bucket.

So, what do you do? You use S3 versioning! Versioning enables you to keep multiple versions of a single object in the same bucket. This allows you to preserve old versions of an object without having to use different naming constructs, in case you need to recover from accidental deletions, accidental overwrites, or even application failures. Let's see how this works.

If you enable versioning for a bucket, Amazon S3 automatically generates a unique version ID for the object being stored. In one bucket, for example, you can have two objects with the same key, but different version IDs, such as employeephoto.gif (version 111111) and employeephoto.gif (version 121212). Versioning-enabled buckets let you recover objects from accidental deletion or overwrite.

- Deleting an object does not remove the object permanently. Instead, Amazon S3 puts a marker on the object that shows you tried to delete it. If you want to restore the object, you can remove this marker, and it reinstates the object.
- If you overwrite an object, it results in a new object version in the bucket. You still have access to previous versions of the object.

UNDERSTAND VERSIONING STATES

Buckets can be in one of three states.

- **Unversioned (the default):** No new or existing objects in the bucket have a version.
- **Versioning-enabled**: This enables versioning for all objects in the bucket.
- **Versioning-suspended**: This suspends versioning for new objects. All new objects in the bucket will not have a version. However, all existing objects keep their object versions.

The versioning state applies to all the objects in that bucket. Keep in mind that storage costs are incurred for all objects in your bucket and all versions of those objects. To reduce your S3 bill, you may want to delete previous versions of your objects that are no longer in use.

WHAT ARE AMAZON S3 STORAGE CLASSES?

There are six S3 storage classes.

1. **Amazon S3 Standard:** This is considered general purpose storage for cloud applications, dynamic websites, content distribution, mobile and gaming applications, and big data analytics.
2. **Amazon S3 Intelligent-Tiering:** This tier is useful if your data has unknown or changing access patterns. S3 Intelligent-Tiering stores objects in two tiers, a frequent access tier and an infrequent access tier. Amazon S3 monitors access patterns of your data, and automatically moves your data to the most cost-effective storage tier based on frequency of access.

3. **Amazon S3 Standard-Infrequent Access (S3 Standard-IA):** S3 Standard-IA is for data that is accessed less frequently but requires rapid access when needed. S3 Standard-IA offers the high durability, high throughput, and low latency of S3 Standard, with a low per-GB storage price and per-GB retrieval fee. This storage tier is ideal if you want to store long-term backups, disaster recovery files, and so on.
4. **Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA):** Unlike other S3 storage classes which store data in a minimum of three Availability Zones (AZs), S3 One Zone-IA stores data in a single AZ and costs 20% less than S3 Standard-IA. S3 One Zone-IA is ideal for customers who want a lower-cost option for infrequently accessed data but do not require the availability and resilience of S3 Standard or S3 Standard-IA. It's a good choice for storing secondary backup copies of on-premises data or easily re-creatable data.
5. **Amazon S3 Glacier Instant Retrieval:** Amazon S3 Glacier Instant Retrieval is an archive storage class that delivers the lowest-cost storage for long-lived data that is rarely accessed and requires retrieval in milliseconds.
6. **Amazon S3 Glacier Flexible Retrieval:** S3 Glacier Flexible Retrieval delivers low-cost storage, up to 10% lower cost (than S3 Glacier Instant Retrieval), for archive data that is accessed 1—2 times per year and is retrieved asynchronously.
7. **Amazon S3 Glacier Deep Archive:** S3 Glacier Deep Archive is Amazon S3's lowest-cost storage class and supports long-term retention and digital preservation for data that may be accessed once or twice in a year. It is designed for customers—particularly those in highly regulated industries, such as the Financial Services, Healthcare, and Public Sectors—that retain data sets for 7 to 10 years or longer to meet regulatory compliance requirements.
8. **Amazon S3 Outposts:** Amazon S3 on Outposts delivers object storage to your on-premises AWS Outposts environment.

Choose the Right Storage Service

Amazon EC2 Instance Store

Instance store is **ephemeral block storage**. This is preconfigured storage that exists on the same physical server that hosts the EC2 instance and **cannot be detached from Amazon EC2**. You can think of it as a built-in drive for your EC2 instance. Instance store is generally well-suited for temporary storage of information that is constantly changing, such as buffers, caches, and scratch data. It is not meant for data that is persistent or long-lasting. If you need persistent long-term block storage that can be detached from Amazon EC2 and provide you more management flexibility, such as increasing volume size or creating snapshots, then you should use Amazon EBS.

Amazon EBS

Amazon EBS is meant for data that changes frequently and needs to persist through instance stops, terminations, or hardware failures. Amazon EBS has two different types of volumes: SSD-backed volumes and HDD-backed volumes. SSD-backed volumes have the following characteristics.

- Performance depends on IOPS (input/output operations per second).
- Ideal for transactional workloads such as databases and boot volumes.

HDD-backed volumes have the following characteristics:

- Performance depends on MB/s.
- Ideal for throughput-intensive workloads, such as big data, data warehouses, log processing, and sequential data I/O.

Here are a few important features of Amazon EBS that you need to know when comparing it to other services.

- It is **block storage**.
- You pay for what you provision (you have to provision storage in advance).
- EBS volumes are replicated across multiple servers in a single Availability Zone.
- Most EBS volumes can only be attached to a single EC2 instance at a time.

Amazon S3

If your data doesn't change that often, Amazon S3 might be a more cost-effective and scalable storage solution. S3 is ideal for storing static web content and media, backups and archiving, data for analytics, and can even be used to host entire static websites with custom domain names. Here are a few important features of Amazon S3 to know about when comparing it to other services.

- It is **object storage**.
- You pay for what you use (you don't have to provision storage in advance).
- Amazon S3 replicates your objects across multiple Availability Zones in a Region.
- Amazon S3 is not storage attached to compute.

Amazon Elastic File System (Amazon EFS) and Amazon FSx

For file storage that can mount on to multiple EC2 instances, you can use Amazon Elastic File System (Amazon EFS) or Amazon FSx.

- It is **file storage.**
- You pay for what you use (you don’t have to provision storage in advance).
- Amazon EFS and Amazon FSx can be mounted onto multiple EC2 instances.

DATABASES ON AWS

What Is Amazon RDS?

Amazon RDS enables you to create and manage relational databases in the cloud without the operational burden of traditional database management. For example, if you sell healthcare equipment and your goal is to be the number one seller in the Pacific Northwest, building out a database doesn't directly help you achieve that goal though having a database is necessary to achieve the goal. Amazon RDS helps you offload some of this unrelated work of creating and managing a database. You can focus on the tasks that differentiate your application, instead of infrastructure-related tasks such as provisioning, patching, scaling, and restoring.

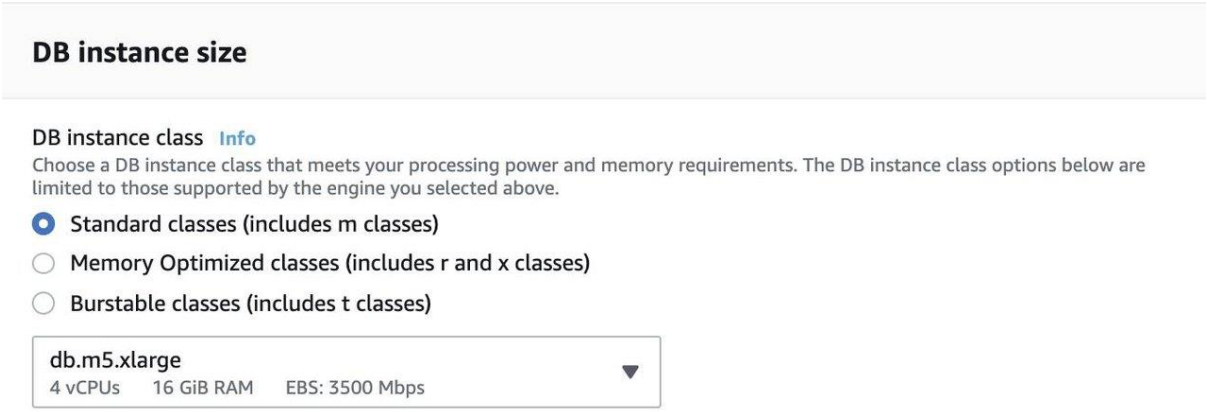
Here are the supported Amazon RDS engines.

- **Commercial:** Oracle, SQL Server
- **Open Source:** MySQL, PostgreSQL, MariaDB
- **Cloud Native:** Amazon Aurora

Understand DB Instances

Just like the databases that you would build and manage yourself, Amazon RDS is built off of compute and storage. The compute portion is called the DB (database) instance, which runs the database engine. Depending on the engine of the DB instance you choose, the engine will have different supported features and configurations. A DB instance can contain multiple databases with the same engine, and each database can contain multiple tables. Underneath the DB instance is an EC2 instance. However, this instance is managed through the Amazon RDS console instead of the Amazon EC2 console. When you create your DB instance, you choose the instance type and size. Amazon RDS supports three instance families.

- **Standard**, which include general-purpose instances
- **Memory Optimized**, which are optimized for memory-intensive applications
- **Burstable Performance**, which provides a baseline performance level, with the ability to burst to full CPU usage.



Much like a regular EC2 instance, the DB instance uses Amazon Elastic Block Store (EBS) volumes as its storage layer. You can choose between three types of EBS volume storage.

- General purpose (SSD)
- Provisioned IOPS (SSD)
- Magnetic storage (not recommended)

Work with Amazon RDS in an Amazon Virtual Private Cloud

When you create a DB instance, you select the Amazon Virtual Private Cloud (VPC) that your databases will live in. Then, you select the subnets that you want the DB instances to be placed in. This is referred to as a **DB subnet group**. To create a DB subnet group, you specify:

- The Availability Zones (AZs) that include the subnets you want to add
- The subnets in that AZ where your DB instance are placed

The subnets you add should be private, so they don't have a route to the internet gateway. This ensures your DB instance, and the cat data inside of it, can only be reached by the app backend. Access to the DB instance can be further restricted by using network access control lists (ACLs) and security groups. With these firewalls, you can control, at a granular level, what type of traffic you want to allow into your database. Using these controls provide layers of security for your infrastructure. It reinforces that only the backend instances have access to the database.

Back Up Your Data

You don't want to lose any of that precious cat information. To take regular backups of your RDS instance, you can use:

- Automatic backups
- Manual snapshots

Automatic Backups

Automated backups are turned on by default. This backs up your entire DB instance (not just individual databases on the instance), and your transaction logs. When you create your DB instance, you set a backup window that is the period that automatic backups occur. Typically, you want to set these windows during a time when your database experiences little activity because it can cause increased latency and downtime.

You can retain your automated backups between 0 and 35 days. You might ask yourself, "Why set automated backups for 0 days?" The 0 days setting disables automatic backups from happening. Keep in mind that if you set it to 0, it will also delete all existing automated backups. This is not ideal, as the benefit of having automated backups is having the ability to do point-in-time recovery.

Manual Snapshots

If you want to keep your automated backups longer than 35 days, use manual snapshots. Manual snapshots are like taking EBS snapshots, except you manage them in the RDS console. These are backups that you can initiate at any time, that exist until you delete them.

Introduction to Amazon CloudWatch

How CloudWatch Works

The great thing about CloudWatch is that all you need to get started is an AWS account. It is a managed service, which enables you to focus on monitoring, without managing any underlying infrastructure.

Many AWS services send metrics automatically for free to CloudWatch at a rate of one data point per metric per 5-minute interval, without you needing to do anything to turn on that data collection. This by itself gives you visibility into your systems without you needing to spend any extra money to do so. This is known as **basic monitoring**. For many applications, basic monitoring does the job.

For applications running on EC2 instances, you can get more granularity by posting metrics every minute instead of every 5 minutes using a feature like **detailed monitoring**. Detailed monitoring has an extra fee associated.

Break Down Metrics

Each metric in CloudWatch has a timestamp and is organized into containers called **namespaces**. Metrics in different namespaces are isolated from each other—you can think of them as belonging to different categories.

AWS services that send data to CloudWatch attach **dimensions** to each metric. A dimension is a name/value pair that is part of the metric's identity. You can use dimensions to filter the results that CloudWatch returns.

Understand the CloudWatch Dashboards

Once you've provisioned your AWS resources and they are sending metrics to CloudWatch, you can then visualize and review that data using the CloudWatch console with dashboards. Dashboards are customizable home pages that you use for data visualization for one or more metrics through the use of widgets, such as a graph or text.

You can build many custom dashboards, each one focusing on a distinct view of your environment. You can even pull data from different Regions into a single dashboard in order to create a global view of your architecture.

CloudWatch aggregates statistics according to the period of time that you specify when creating your graph or requesting your metrics. You can also choose whether your metric widgets display live data. Live data is data published within the last minute that has not been fully aggregated.

You are not bound to using CloudWatch exclusively for all your visualization needs. You can use external or custom tools to ingest and analyze CloudWatch metrics using the GetMetricData API.

As far as security goes, you can control who has access to view or manage your CloudWatch dashboards through AWS Identity and Access Management (IAM) policies that get associated with IAM users, IAM groups, or IAM roles.

Get to Know CloudWatch Logs

CloudWatch can also be the centralized place for logs to be stored and analyzed, using CloudWatch Logs. CloudWatch Logs can monitor, store, and access your log files from applications running on Amazon EC2 instances, AWS Lambda functions, and other sources.

CloudWatch Logs allows you to query and filter your log data. For example, let's say you're looking into an application logic error for your application, and you know that when this error occurs it will log the stack trace. Since you know it logs the error, you query your logs in CloudWatch Logs to find the stack trace. You also set up **metric filters** on logs, which turn log data into numerical CloudWatch metrics that you graph and use on your dashboards.

If you want to send your application logs from an EC2 instance into CloudWatch Logs, you need to first install and configure the CloudWatch Logs agent on the EC2 instance.

The CloudWatch Logs agent enables Amazon EC2 instances to automatically send log data to CloudWatch Logs. The agent includes the following components.

- A plug-in to the AWS Command Line Interface (CLI) that pushes log data to CloudWatch Logs.
- A script that initiates the process to push data to CloudWatch Logs.
- A cron job that ensures the daemon is always running.

After the agent is installed and configured, you can then view your application logs in CloudWatch Logs.

Learn the CloudWatch Logs Terminology

Log data sent to CloudWatch Logs can come from different sources, so it's important you understand how they're organized and the terminology used to describe your logs.

Log event: A log event is a record of activity recorded by the application or resource being monitored, and it has a timestamp and an event message.

Log stream: Log events are then grouped into log streams, which are sequences of log events that all belong to the same resource being monitored. For example, logs for an EC2 instance are grouped together into a log stream that you can then filter or query for insights.

Log groups: Log streams are then organized into log groups. A log group is composed of log streams that all share the same retention and permissions settings. For example, if you have multiple EC2 instances hosting your application and you are sending application log data to CloudWatch Logs, you can group the log streams from each instance into one log group. This helps keep your logs organized.

Configure a CloudWatch Alarm

You can create CloudWatch alarms to automatically initiate actions based on sustained state changes of your metrics. You configure when alarms are triggered and the action that is performed.

Keeping all that in mind, to set up an alarm you need to choose the metric, the threshold, and the time period. An alarm has three possible states.

- **OK:** The metric is within the defined threshold. Everything appears to be operating like normal.
- **ALARM:** The metric is outside of the defined threshold. This could be an operational issue.
- **INSUFFICIENT_DATA:** The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

WHAT'S A LOAD BALANCER?

Load balancing refers to the process of distributing tasks across a set of resources. In the case of the corporate directory application, the resources are EC2 instances that host the application, and the tasks are the different requests being sent. It's time to distribute the requests across all the servers hosting the application using a load balancer.

To do this, you first need to enable the load balancer to take all of the traffic and redirect it to the backend servers based on an algorithm. The most popular algorithm is round-robin, which sends the traffic to each server one after the other.

A typical request for the application would start from the browser of the client. It's sent to a load balancer. Then, it's sent to one of the EC2 instances that hosts the application. The return traffic would go back through the load balancer and back to the client browser. Thus, the load balancer is directly in the path of the traffic.

Although it is possible to install your own software load balancing solution on EC2 instances, AWS provides a service for that called Elastic Load Balancing (ELB).

FEATURES OF ELB

The ELB service provides a major advantage over using your own solution to do load balancing, in that you don't need to manage or operate it. It can distribute incoming application traffic across EC2 instances as well as containers, IP addresses, and AWS Lambda functions.

- The fact that ELB can load balance to IP addresses means that it can work in a hybrid mode as well, where it also load balances to on-premises servers.
- ELB is highly available. The only option you have to ensure is that the load balancer is deployed across multiple Availability Zones.
- In terms of scalability, ELB automatically scales to meet the demand of the incoming traffic. It handles the incoming traffic and sends it to your backend application.

HEALTH CHECKS

After determining the availability of a new EC2 instance, the load balancer starts sending traffic to it. If ELB determines that an EC2 instance is no longer working, it stops sending traffic to it and lets EC2 Auto Scaling know. EC2 Auto Scaling's responsibility is to remove it from the group and replace it with a new EC2 instance. Traffic only sends to the new instance if it passes the health check.

In the case of a scale down action that EC2 Auto Scaling needs to take due to a scaling policy, it lets ELB know that EC2 instances will be terminated. ELB can prevent EC2 Auto Scaling from terminating the EC2 instance until all connections to that instance end, while preventing any new connections. That feature is called **connection draining**.

SELECT BETWEEN ELB TYPES

Selecting between the ELB service types is done by determining which feature is required for your application. Below you can find a list of the major features that you learned in this unit and the previous.

Feature	**Application Load Balancer**	**Network Load Balancer**
Protocols	HTTP, HTTPS	TCP, UDP, TLS
Connection draining (deregistration delay)	✓	
IP addresses as targets	✓	✓
Static IP and Elastic IP address		✓
Preserve Source IP address		✓
Routing based on Source IP address, path, host, HTTP headers, HTTP method, and query string	✓	
Redirects	✓	
Fixed response	✓	
User authentication	✓	