AWS CLOUD TECHNICAL ESSENTIALS

(Notes by Aniruddha Repale)

INTRODUCTION :

AWS Cloud Technical Essentials is a fundamental-level course. It's designed to build your competence, confidence, and credibility with practical cloud skills that can help you innovate and advance your professional future. Here's a quick overview of the skills that you will learn in this course.

1: Getting Started with AWS Cloud

Initially, you will learn the definition of cloud computing and how to describe the cloud value proposition. You will learn how to differentiate between workloads that run on premises compared to workloads that run in the cloud, and how to create an AWS account. You will also get an overview of AWS, including how to differentiate between AWS Regions and Availability Zones, and the different ways that you can interact with AWS. Finally, you will learn best practices for using AWS Identity and Access Management (IAM).

2: AWS Compute & Networking

Then you will learn how AWS compute services differ from other AWS services. The content for this week covers the basic components of Amazon Elastic Compute Cloud (Amazon EC2) architecture, and how to differentiate between a container and a virtual machine. You will also learn about the features and advantages of using serverless technologies, in addition to basic networking concepts and the features of Amazon Virtual Private Cloud (Amazon VPC).

3: Storage & Databases on AWS

After it, you will learn important concepts for AWS storage services—such as buckets and objects for Amazon Simple Storage Service (Amazon S3), and how Amazon Elastic Block Store (Amazon EBS) is used on AWS. You will also explore databases on AWS, and the use cases for each AWS storage service.

4: Monitoring, Optimizing, and Going Serverless on AWS

Finally, you will learn about the benefits of monitoring on AWS, and how to optimize solutions on AWS. You will also learn about the function of Elastic Load Balancing (ELB), and how to differentiate between vertical scaling and horizontal scaling.

WHAT IS AWS ?

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 :

Infrastructure, like data centers and networking connectivity, still exists as the foundation of every cloud application. In AWS, this physical infrastructure makes up the AWS Global Infrastructure, in the form of Availability Zones and Regions.

REGIONS

Regions are geographic locations worldwide where AWS hosts its data centers. AWS Regions are named after the location where they reside. For example, in the United States, there is a Region in Northern Virginia called the Northern Virginia Region and a Region in Oregon called the Oregon Region. There are Regions in Asia Pacific, Canada, Europe, the Middle East, and South America, and AWS continues to expand to meet the needs of its customers.Each AWS Region is associated with a geographical name and a Region code.

Here are a few examples of Region codes:

- us-east-1: This is the first Region created in the east of the US. The geographical name for this Region is N. Virginia.
- ap-northeast-1: The first Region created in the northeast of Asia Pacific. The geographical name for this Region is Tokyo.

AWS Regions are independent from one another. This means that your data is not replicated from one Region to another, without your explicit consent and authorization.

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.

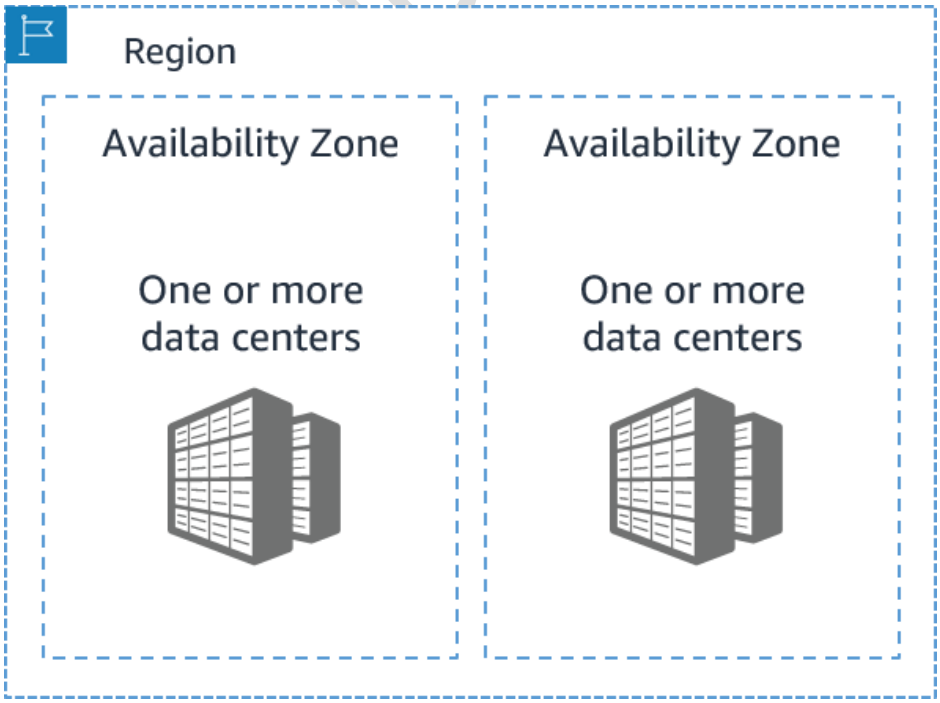
Latency. If your application is sensitive to latency, choose a Region that is close to your user base. This helps prevent long wait times for your customers. Synchronous applications such as gaming, telephony, WebSockets, and IoT are significantly affected by higher latency, but even asynchronous workloads, 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.

AVAILABILITY ZONES



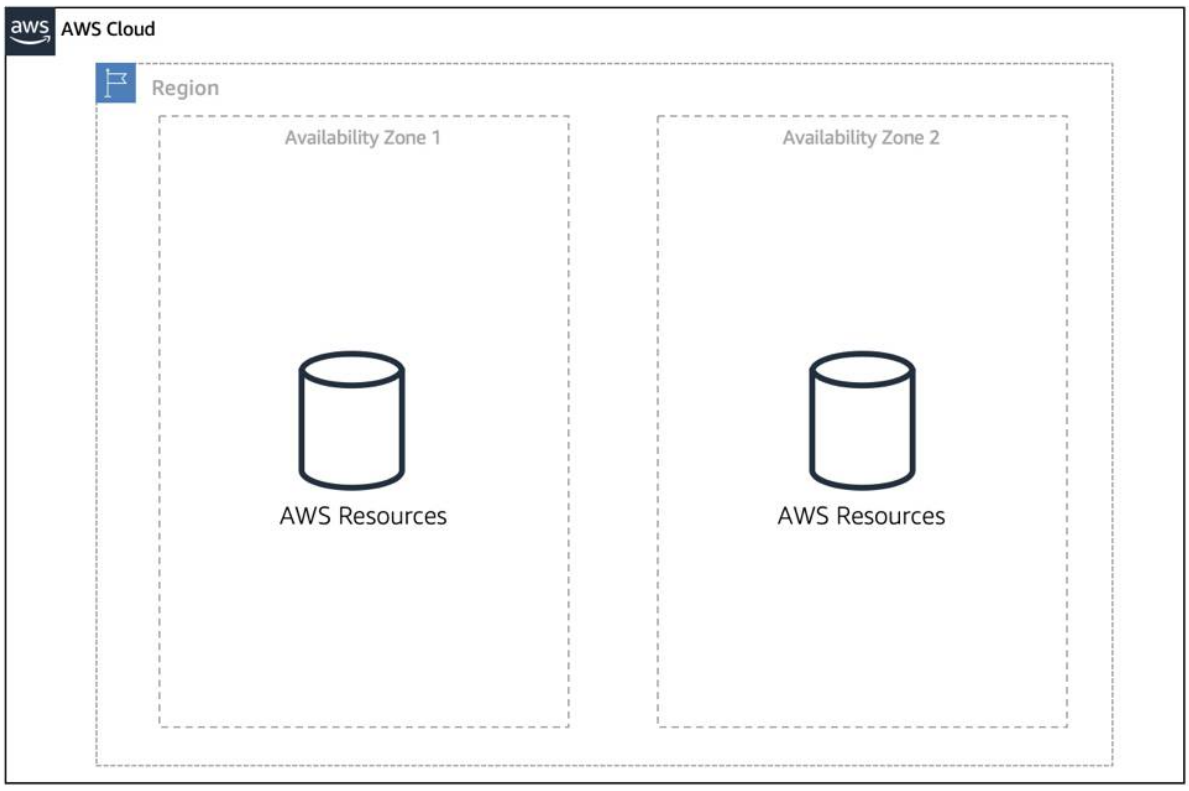
Inside every Region is a cluster of Availability Zones (AZ). An AZ consists of one or more data centers with redundant power, networking, and connectivity. These data centers operate in discrete facilities with undisclosed locations. They are connected using redundant high-speed and low-latency links.AZs also have a code name. Since they're located inside Regions, they can be addressed by appending a letter to the end of the Region code name. For example:

- us-east-1a: an AZ in us-east-1 (Northern Virginia Region)
- sa-east-1b: an AZ in sa-east-1 (São Paulo Region in South America)

If you see that a resource exists in us-east-1c, you know this resource is located in AZ c of the us-east-1 Region.

MAINTAIN RESILIENCY

To keep your application available, you need to maintain high availability and resiliency. A well-known best practice for cloud architecture is to use Region-scoped, managed services. These services come with availability and resiliency built in.When that is not possible, make sure the workload is replicated across multiple AZs. At a minimum, you should use two AZs. If one entire AZ fails, your application will have infrastructure up and running in at least a second AZ to take over the traffic.

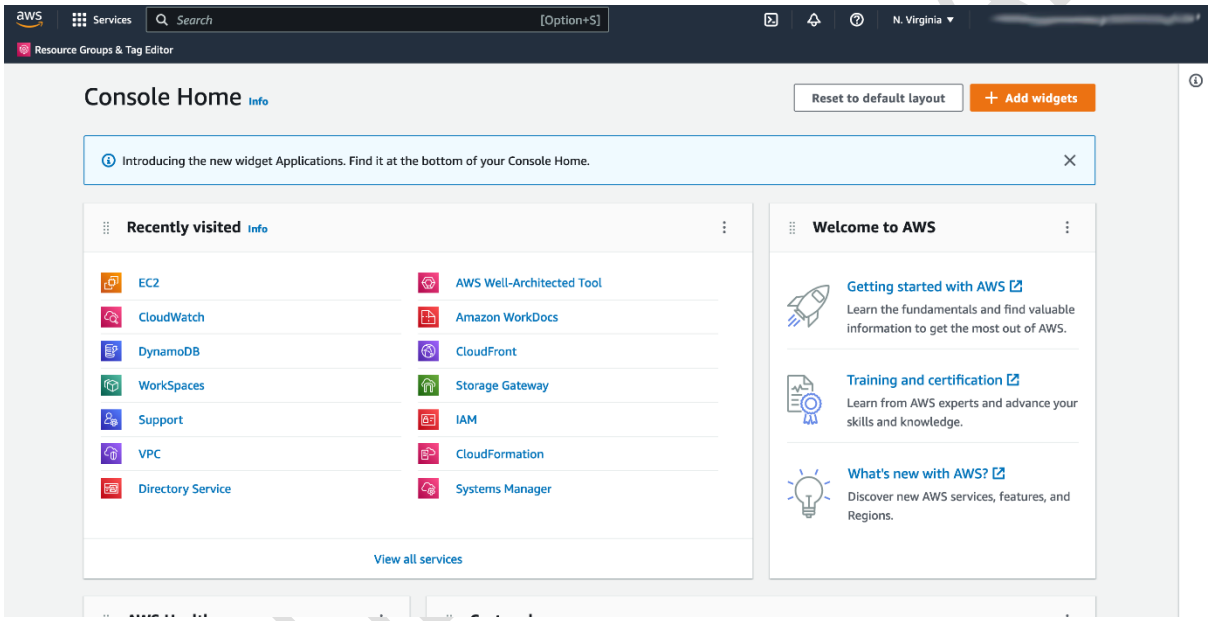


INTERACTING WITH AWS :

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)

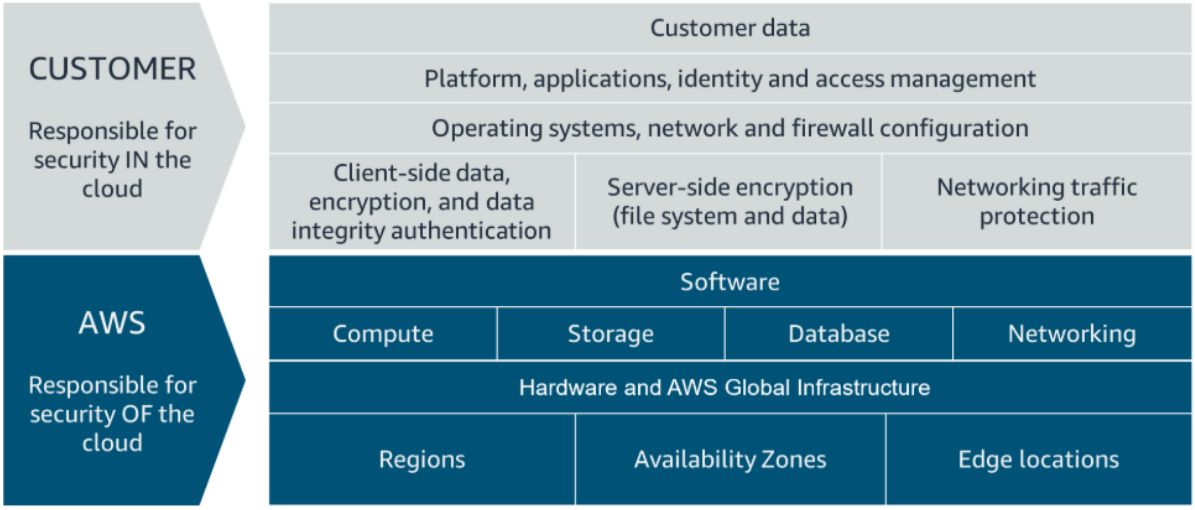
API calls to AWS can also be performed by executing code with programming languages. You can do this by using AWS Software Development Kits (SDKs). SDKs are open-source and maintained by AWS for the most popular programming languages, such as C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python, and 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 IN AWS :

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.



The level of responsibility AWS has depends on the service. AWS classifies services into three different categories.

1. **Infrastructure services** (Compute services, such as Amazon EC2) - AWS manages the underlying infrastructure and foundation services.
2. **Container services** (Services that require less management from the customer, such as Amazon RDS) - AWS manages the underlying infrastructure , foundation services, operating system, and application platform.
3. **Abstracted services** (Services that require very little management from the customer, such as 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 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.

✅ **Protect the AWS Root User :**

When you create an AWS account , you sign up with an email address and you create a password. The email address you sign up with becomes the root user of

AWS account. This root user has unrestricted access to everything in your account in most cases.

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 CLI or AWS API. 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.

Security:

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, ensuring security of login credentials associated with the root user is important. 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.

The Case for Multi-Factor Authentication:

When you create an AWS account and first log in to that account, you use single-factor authentication (username and password , security pin or a security token). However, sometimes a user's password is easy to guess.

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.

AWS Identity and Access Management:

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.

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.

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)

What is an IAM Group?

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.

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.

e.g. 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.

e.g. 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.

What is an IAM Policy?

To manage access and provide permissions to AWS services and resources, you create IAM policies and attach them to IAM users, groups, and roles. Whenever a user or role makes a request, AWS evaluates the policies associated with them. For example, if you have a developer inside the developers group who makes a request to an AWS service, AWS evaluates any policies attached to the developers group and any policies attached to the developer user to determine if the request should be allowed or denied.

Most policies are stored in AWS as JSON documents with several policy elements.

```
{
"Version": "2012-10-17",
"Statement": [{ "Effect": "Allow",
"Action": "*",
"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. Other actions can be like "iam: ChangePassword", "iam: GetUser" , etc.
- The **Resource** element specifies the object or objects that the policy statement covers. In above e.g., "*" represents all resources inside your AWS console. Other example of resource is "arn:aws:iam::123456789012:user/${aws:username}"

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*.

Role Based Access in AWS

AWS Role is a set of permissions that can be temporarily assigned to any AWS service, user, or application.
Roles don't have passwords or access keys. Instead, they are assumed for a short time to perform specific tasks.

Example: You create a role that allows access to an S3 bucket. An EC2 instance can "assume" this role to access the bucket without needing credentials.

An AWS IAM User is a permanent identity in your AWS account with a username and credentials (password/access key). You assign specific permissions to the user to let them access certain AWS services.

Example: You create a user named *developer1* who can log in to the AWS Console and work with EC2 or S3 based on permissions given.

👉 Use **IAM Users** for people who need regular access.
👉 Use **IAM Roles** when you want temporary, secure access — especially for AWS services or cross-account access.

AWS COMPUTE :

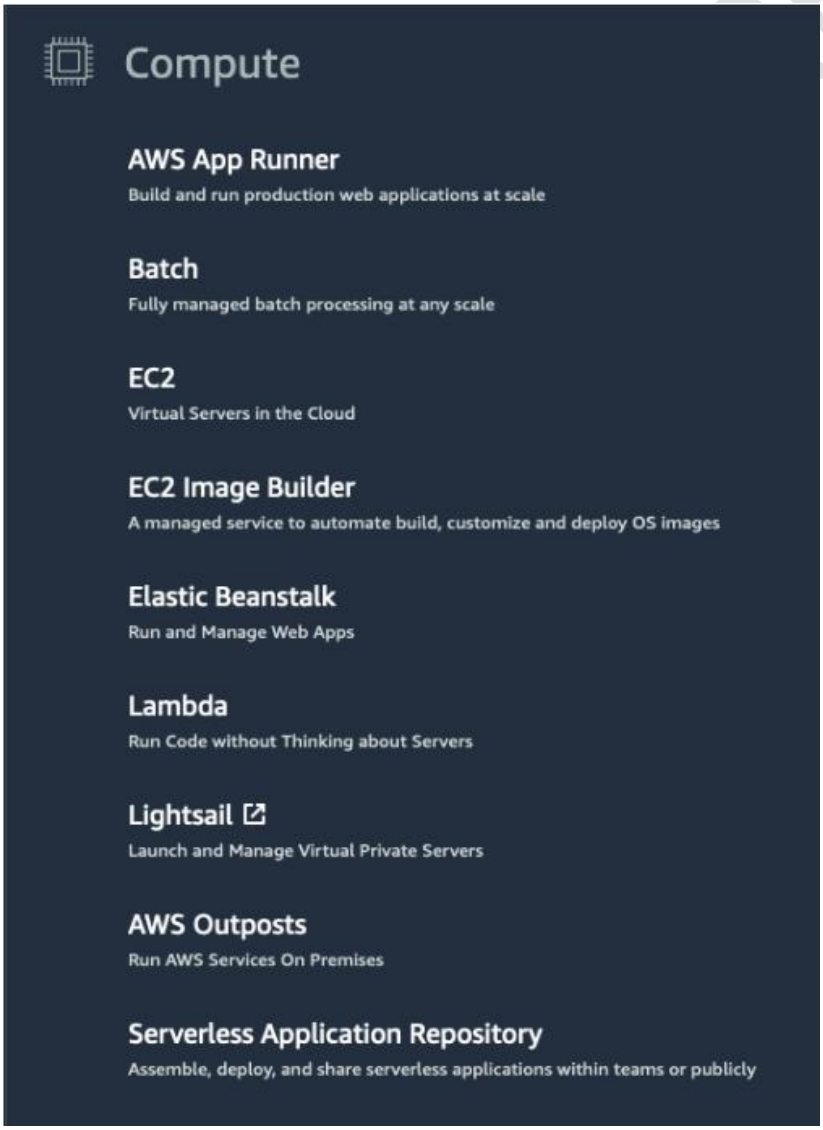
Compute as a Service on AWS :

The first building block you need to host an application is a server. These servers power your application by providing CPU, memory, and networking capacity to process users' requests and transform them into responses.

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.

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.



In AWS, these virtual machines are called Amazon Elastic Compute Cloud or Amazon EC2.

Introduction to Amazon Elastic Compute Cloud

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. It allows you to create virtual servers called EC2 instances. You can create and manage these instances through the AWS Management Console, the AWS CLI , AWS SDKs, or through automation tools and 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).

In the AWS Cloud, 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.

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.

One advantage of using AMIs is that they are reusable. 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.

Amazon EC2 Instance Lifecycle

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, in the instance type c5.large , **c5** determines the instance family and generation number (fifth generation of instances in an instance family that's optimized for generic computation) and **large** determines the amount of instance capacity.

Instance Families

1. General purpose : Provides a balance of compute, memory, and networking resources, and can be used for a variety of workloads.
2. Compute optimized : Ideal for compute-bound applications that benefit from high-performance processors.

3. Memory optimized : Designed to deliver fast performance for workloads that process large data sets in memory.

4. 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.

5. Storage optimized : Designed for workloads that require high, low-latency , sequential read and write access to large data sets on local storage.

Where Does Your EC2 Instance Live?

By default, your EC2 instances are placed in a network called the default Amazon Virtual Private Cloud (VPC). Inside this network, your instance resides in an Availability Zone of your choice. 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.

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**. 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.

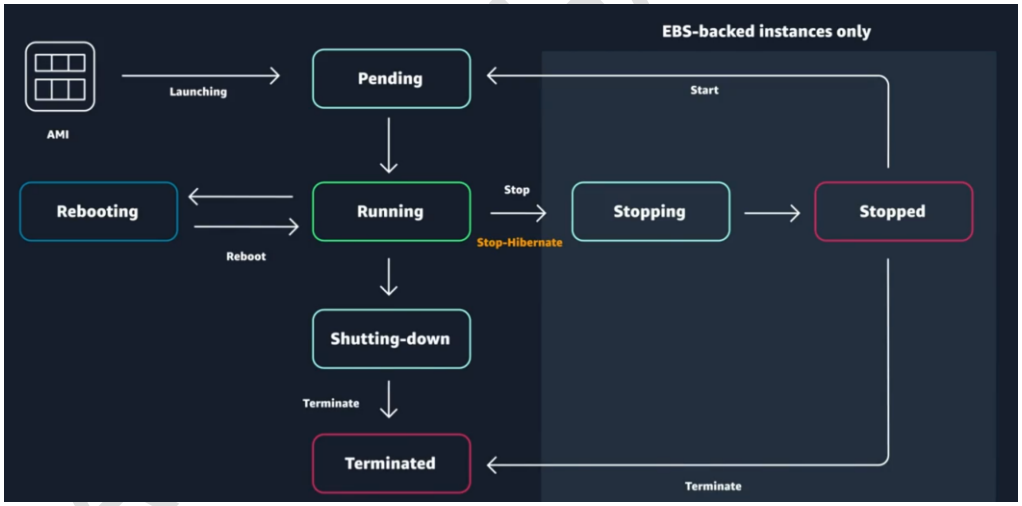
When your instance is **running** , it's ready to use. This is also the stage where billing begins. As soon as an instance is running, you are then able to take other actions on the instance, such as *reboot, terminate, stop, and stop-hibernate*.

When you **reboot** an instance , 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, 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 stop your instance, the data stored in memory (RAM) is lost. 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.

When you **terminate** an instance, 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.

When you **stop-hibernate** your instance, instance enters stopping phase and then the stopped phase. You can compare this to how you lock your laptop and shut the lid, but when you open it back up, everything is still in place where you left it. No boot sequences required. You are back up and running after a couple of seconds of the computer waking up. Since the state of the machine was written to memory when you stopped it, the state of the machine can be drawn from memory and put back into place. Then, you're back up and running. Simply , data stored in memory (RAM) is not lost.



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.

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.

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.

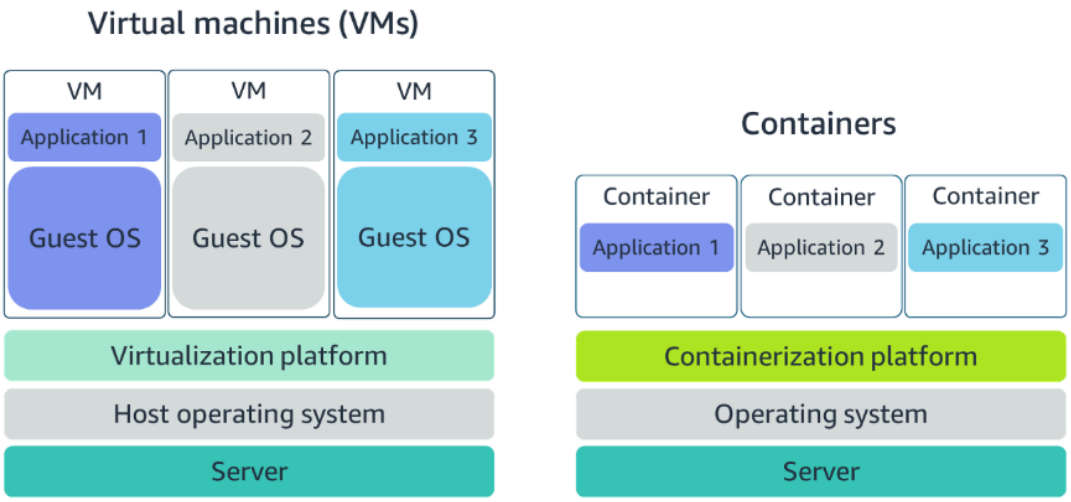
Save on Costs with Spot Instances

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. But , 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.

Container Services on AWS

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.

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. 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. 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).

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.

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.

NETWORKING ON AWS :

Networking is how you connect computers around the world and allow them to communicate with one another.

Think about sending a letter. 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.

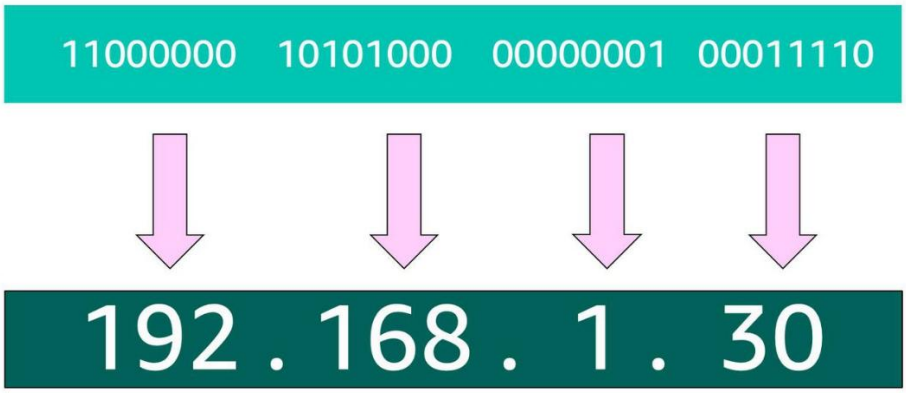
Here is an example of a 32-bit address in binary format:

```
11000000  10101000  00000001  00011110
```

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.

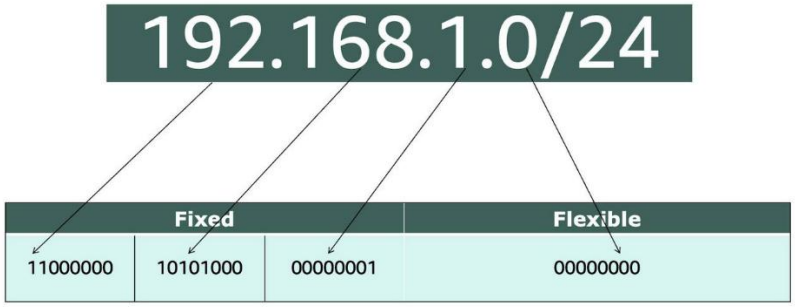
Use CIDR Notation:

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

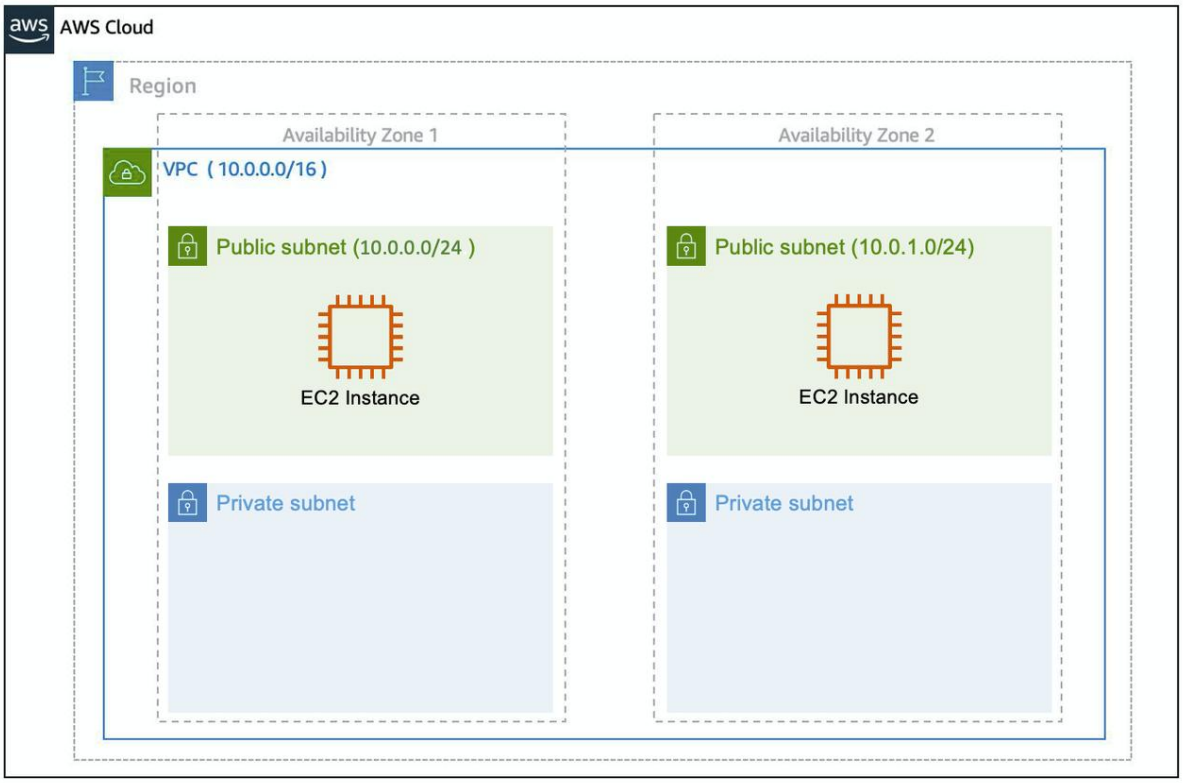
After you create your VPC, you need to create subnets inside of this network. Think of subnets as smaller networks inside your base network—or virtual area networks (VLANs) in a traditional, on-premises network. In an on-premises network, the typical use case for subnets is to isolate or optimize network 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.

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.

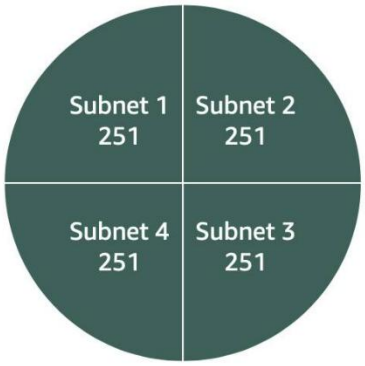
As you learned earlier in the trail, it's important to consider that "everything fails all the time." In this case, if one of these AZs fail, you still have your resources in another AZ available as backup.



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.



IP address	Reserved for
10.0.0.0	Network address
10.0.0.1	VPC local router
10.0.0.2	DNS server
10.0.0.3	Future use
10.0.3.255	Network broadcast address

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:

Destination	Target	Status	Propagated
10.2.0.0/16	local	active	No

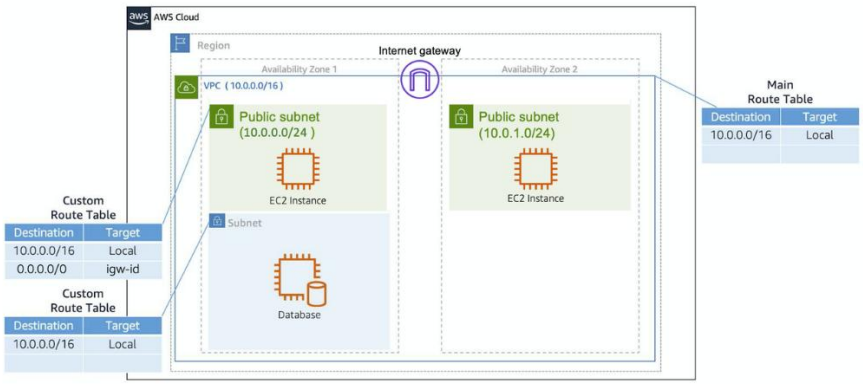
There are two main parts to this route table.

- 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. 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

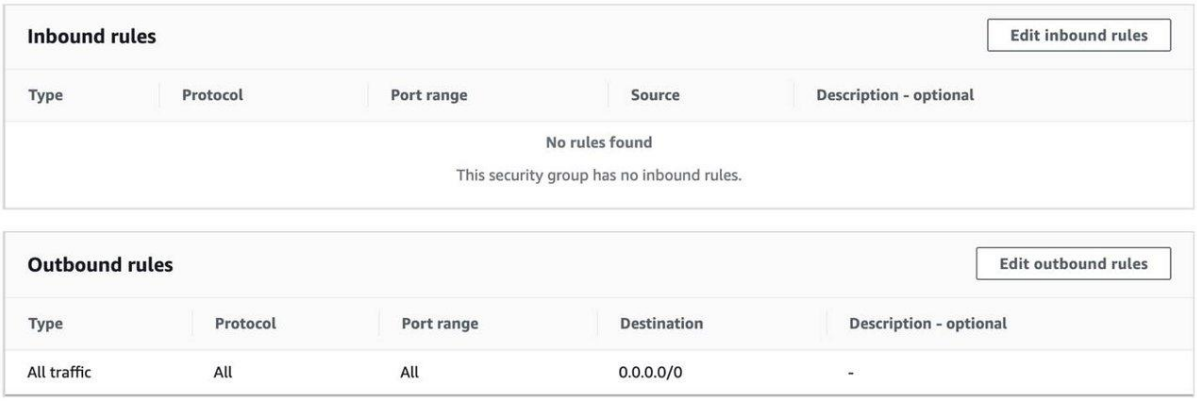
The default network ACL, shown in the table above, allows all traffic in and out of your subnet. To allow data to flow freely to your subnet, this is a good starting place. However, you may want to restrict data at the subnet level. For example, if you have a web application, you might restrict your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

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

STORAGE IN 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.

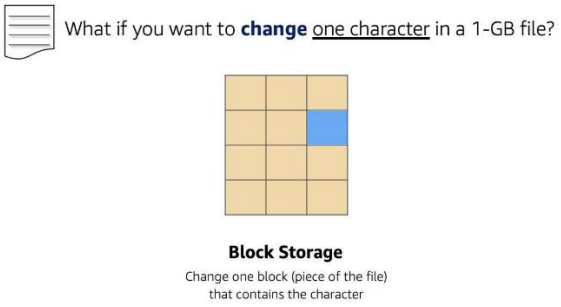
Each file has metadata such as file name, file size, and the date the file was created. The file also has a path, for example, computer/Cat_photos/cats-03.png. When you need to retrieve a file, your system can use the path to find it in the file hierarchy.

File storage is ideal when you require centralized access to files that need to be easily shared and managed by multiple host computers. Common use cases for file storage include:Large content repositories , Development environments,etc.

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.

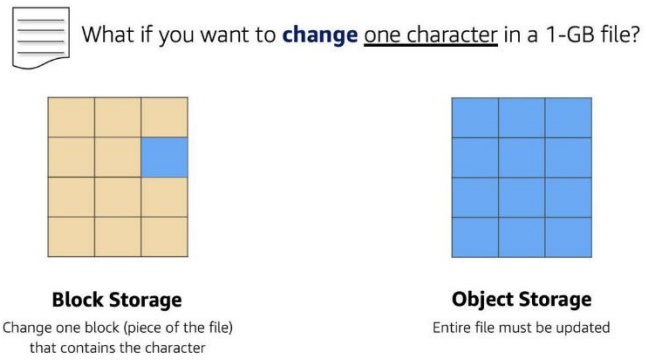


Since block storage is optimized for low-latency operations, it is a typical storage choice for high-performance enterprise workloads, such as databases or enterprise resource planning (ERP) systems, that require low-latency storage.

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.

Amazon EC2 Instance Storage and Amazon Elastic Block 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.

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.

- 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.
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.

Benefits of Using Amazon EBS

- 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.

Object Storage with Amazon S3

Unlike Amazon EBS, Amazon S3 is a standalone storage solution that isn't tied to compute. It enables you to retrieve your data from anywhere on the web. If you've ever used an online storage service to back up the data from your local machine, then you most likely have used a service similar to Amazon S3. The big difference between those online storage services and Amazon S3 is the storage type. Amazon S3 is an object storage service.

In Amazon S3, you have to 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.

The first part is choosing the **Region** you want the bucket to reside in. Typically, this will be a Region that you've used for other resources, such as your compute. When you choose a Region for your bucket, all objects you put inside that bucket are redundantly stored across multiple devices, across multiple Availability Zones.

The second part is choosing a **bucket name** which must be unique across all AWS accounts. Once you choose a name, that name is yours and cannot be claimed by anyone else unless you delete that bucket, which then releases the name for others to use.

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.

Note, you can have folders inside of buckets to help you organize objects. However, remember that there's no actual file hierarchy that supports this on the back end. It is instead a flat structure where all files and folders live at the same level. Using buckets and folders implies a hierarchy, which makes it easy to understand for the human eye.

S3 USE CASES

- **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.
- **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

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. Using IAM policies allows you to manage all policy information in one location.

UNDERSTAND S3 BUCKET POLICIES

S3 bucket policies are similar to 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.

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 of 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?

When you upload an object to Amazon S3 and you don't specify the storage class, you're uploading it to the default storage class—often referred to as standard storage. S3 storage classes let you change your storage tier as your data characteristics change. For example, if you are now accessing your old photos infrequently, you may want to change the storage class those photos are stored in to save on costs. 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. It 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. It offers the high durability, high throughput, and low latency of S3 Standard, with a low per-GB storage price and per-GB retrieval fee.
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. It 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:** It 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(lasting for very short time)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. Use the following table for more information about each of these services.

Service	Characteristic
Amazon Elastic File System (EFS)	Fully managed NFS file system.
Amazon FSx for Windows File Server	Fully managed file server built on Windows Server that supports the SMB protocol.
Amazon FSx for Lustre	Fully managed Lustre file system that integrates with S3.

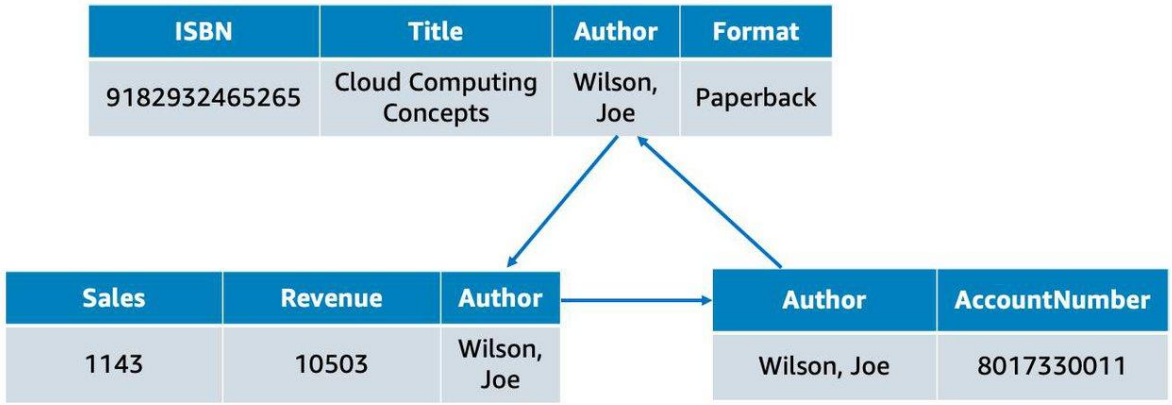
DATABASES ON AWS :

Choosing a database used to be a straightforward decision. There were only a few options to choose from. Since the 1970s, the database most commonly selected by businesses was a relational database.

Relational Database :

A relational database organizes data into tables. Data in one table can be linked to data in other tables to create relationships—hence, the relational part of the name.

A table stores data in rows and columns. A row, often called a record, contains all information about a specific entry. Columns describe attributes of that entry. Here's an example of three tables in a relational database.



This shows a table for books, a table for sales, and a table for authors. In the books table, each row includes the book ISBN, the title, the author, and the format. Each of these attributes is stored in its own column. The books table has something in common with the other two tables: the author attribute. That common column creates a relationship between the tables.

The tables, rows, columns, and relationships between them is referred to as a **logical schema**. With relational databases, a schema is fixed. Once the database is operational, it becomes difficult to change the schema. This requires most of the data modeling to be done upfront before the database is active.

Relational Database Management System :

A relational database management system (RDBMS) lets you create, update, and administer a relational database. Here are some common examples of relational database management systems:

- MySQL
- PostgresQL
- Oracle

- SQL server
- Amazon Aurora

You communicate with most RDBMS by using Structured Query Language (SQL) queries. Here's an example: SELECT * FROM table_name.This query selects all of the data from a particular table.

However, the real power of SQL queries is in creating more complex queries that let you pull data from several tables to piece together patterns and answers to business problems. For example, querying the sales table and the book table together to see sales in relation to an author's books. This is made possible by a join, which we talk about next.

THE BENEFITS OF USING A RELATIONAL DATABASE

There are many benefits to using a relational database. A few of them are listed here.

- **Joins:** You can join tables, enabling you to better understand relationships between your data.
- **Reduced redundancy:** You can store data in one table and reference it from other tables instead of saving the same data in different places.
- **Familiarity:** Relational databases have been a popular choice since the 1970s. Due to this popularity, technical professionals often have familiarity and experience with this type of database.
- **Accuracy:** Relational databases ensure that your data is persisted with high integrity and adheres to the ACID (atomicity, consistency, isolation, durability) principle.

USE CASES FOR RELATIONAL DATABASES

Applications that have a solid schema that doesn't change often, such as:

- Lift and shift applications that lifts an app from on-premises and shifts it to the cloud, with little or no modifications.

Applications that need persistent storage that follows the ACID principle, such as:

- Enterprise Resource Planning (ERP) applications
- Customer Relationship Management (CRM) applications
- Commerce and financial applications

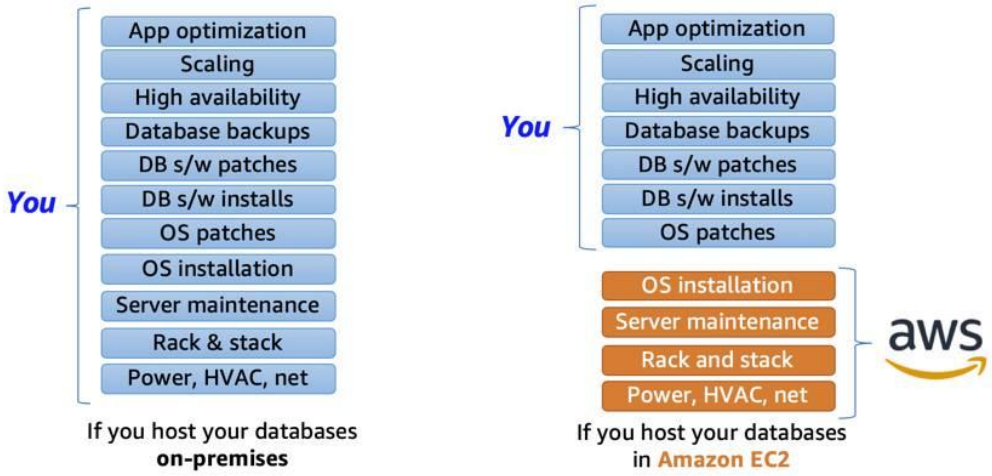
CHOOSE BETWEEN UNMANAGED AND MANAGED DATABASES

If you want to run a relational database on AWS, you first need to select how you want to run it: the unmanaged way or the managed way.

On-Premises Database

Let's say you operate a relational database on-premises (in your own data center). In this scenario, you are responsible for all aspects of operation, including the security and electricity of the data center, the management of the host machine, the management of the database on that host, as well as optimizing queries and managing customer data. You are responsible for absolutely everything, which means you have control over absolutely everything.

Unmanaged Database



Now, let's say you want to shift some of this work to AWS by running your relational database on Amazon EC2. If you host a database on Amazon EC2, AWS takes care of implementing and maintaining the physical infrastructure and hardware and installing the operating system of the EC2 instance. However, you're still responsible for managing the EC2 instance, managing the database on that host, optimizing queries, and managing customer data.

This is what is often referred to as the **unmanaged database** option on AWS. AWS is responsible for and has control over the hardware and underlying infrastructure, and you are responsible and have control over management of the host and database.

Managed Database

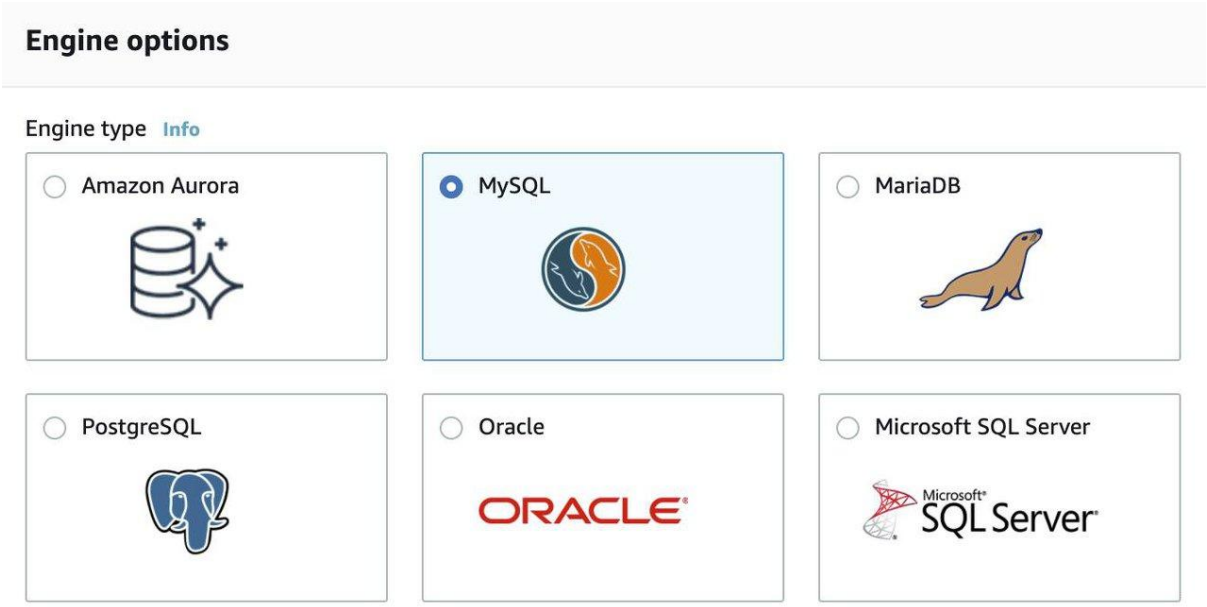
If you want to shift even more of the work to AWS, you can use a **managed database** service. These services provide the setup of both the EC2 instance and the database, and they provide systems for high availability, scalability, patching, and backups. However, you're still responsible for database tuning, query optimization, and of course, ensuring that your customer data is secure. This provides you ultimate convenience, but you have the least amount of control compared to the two previous options.

Amazon Relational Database Service

Amazon RDS enables you to create and manage relational databases in the cloud without the operational burden of traditional database management.

Amazon RDS helps you offload some of the 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. Amazon RDS supports most of the popular relational database management systems, ranging from commercial options, open source options, and even an AWS-specific option. Here are the supported Amazon RDS engines.

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

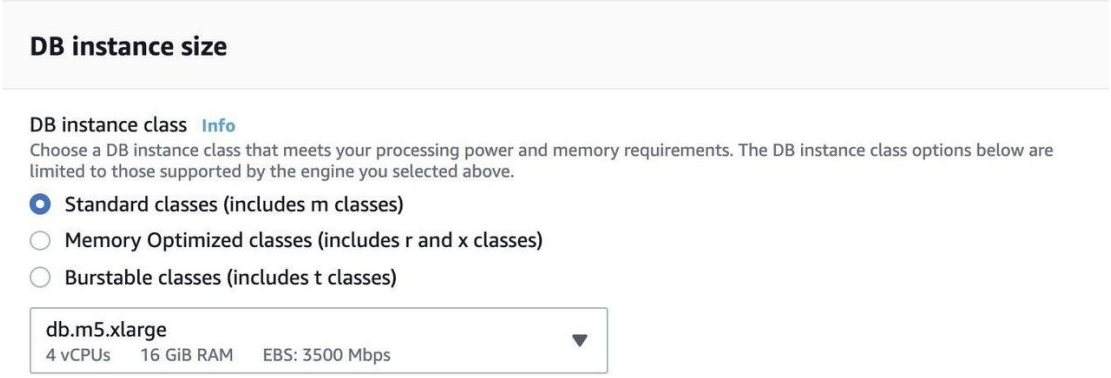


Note: The cloud native option, Amazon Aurora, is a MySQL and PostgreSQL-compatible database built for the cloud. It is more durable, more available, and provides faster performance than the Amazon RDS version of MySQL and PostgreSQL.

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. 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.



The DB instance you choose affects how much processing power and memory it has. 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.

Use AWS Identity and Access Management (IAM) Policies to Secure Amazon RDS

Network ACLs and security groups allow you to dictate the flow of traffic. If you want to restrict what actions and resources your employees can access, you can use IAM policies.

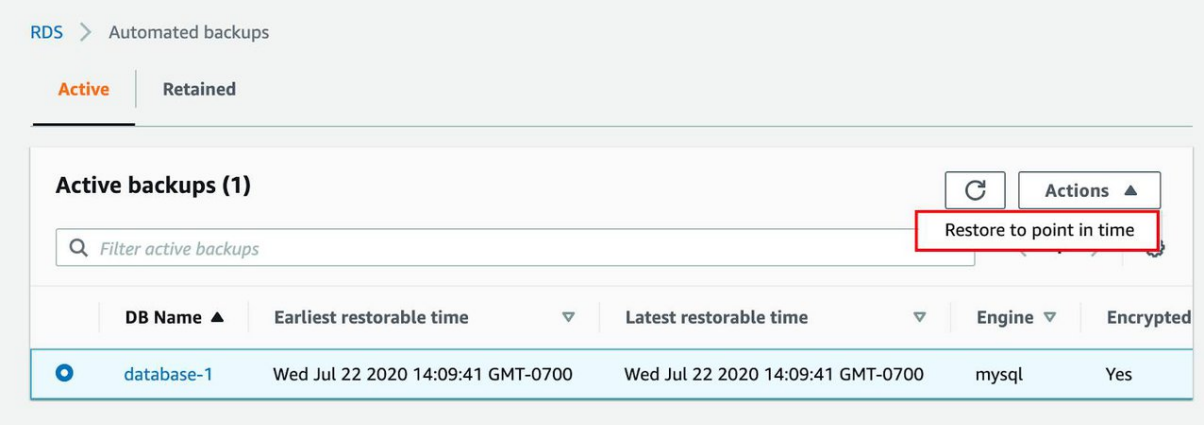
Back Up Your Data

To take regular backups of your RDS instance, you can use:

- Automatic backups
- Manual snapshots

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 of time 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 actually 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.



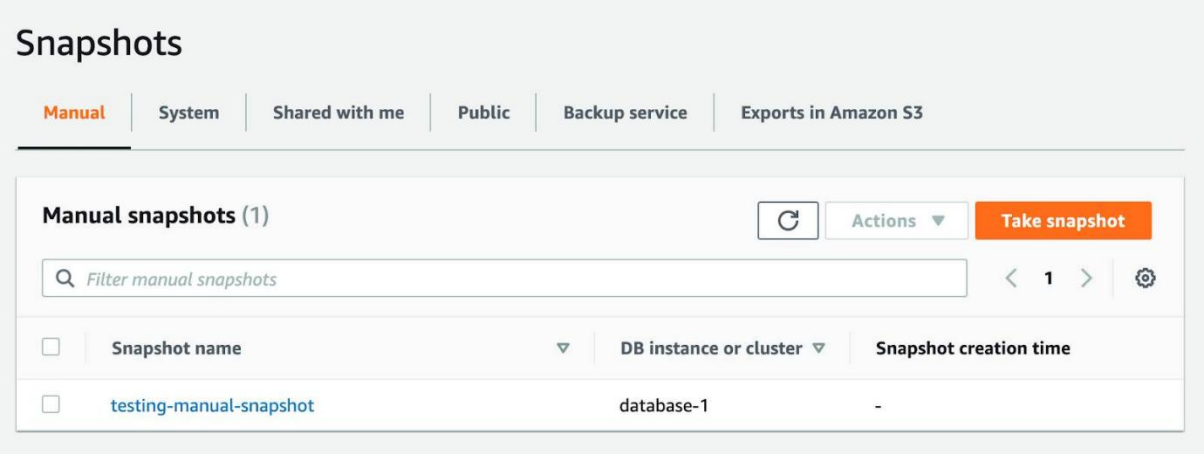
If you restore data from an automated backup, you have the ability to do point-in-time recovery. Point-in-time recovery creates a new DB instance using data restored from a specific point in time. This restoration method provides more granularity by restoring the full backup and rolling back transactions up to the specified time range.

Manual Snapshots

If you want to keep your automated backups longer than 35 days, use manual snapshots. Manual snapshots are similar to 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.

For example, to meet a compliance requirement that mandates you to keep database backups for a year, you would need to use manual snapshots to ensure those backups are retained for that period of time.

If you restore data from a manual snapshot, it creates a new DB instance using the data from the snapshot.



Which Backup Option Should I Use?

The answer, almost always, is both. Automated backups are beneficial for the point-in-time recovery. Manual snapshots allow you to retain backups for longer than 35 days.

Get Redundancy with Amazon RDS Multi-AZ

When you enable Amazon RDS Multi-AZ, Amazon RDS creates a redundant copy of your database in another AZ. You end up with two copies of your database: a primary copy in a subnet in one AZ and a standby copy in a subnet in a second AZ.

The primary copy of your database provides access to your data so that applications can query and display that information.

The data in the primary copy is synchronously replicated to the standby copy. The standby copy is not considered an active database, and does not get queried by applications.

To improve availability, Amazon RDS Multi-AZ ensures that you have two copies of your database running and that one of them is in the primary role. If there's an availability issue, such as the primary database losing connectivity, Amazon RDS triggers an automatic failover.

When you create a DB instance, a domain name system (DNS) name is provided. AWS uses that DNS name to failover to the standby database. In an automatic failover, the standby database is promoted to the primary role and queries are redirected to the new primary database.

To ensure that you don't lose Multi-AZ configuration, a new standby database is created by either:

- Demoting the previous primary to standby if it's still up and running
- Or standing up a new standby DB instance

Introduction to Amazon DynamoDB

What Is Amazon DynamoDB?

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic. You can scale up or scale down your tables' throughput capacity without downtime or performance degradation. You can use the AWS Management Console to monitor resource utilization and performance metrics.

DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All of your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability.

Core Components of Amazon DynamoDB

In DynamoDB, tables, items, and attributes are the core components that you work with. A table is a collection of items, and each item is a collection of attributes. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.

The following are the basic DynamoDB components:

Tables – Similar to other database systems, DynamoDB stores data in tables. A table is a collection of data. For example, see the example table called People that you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a Cars table to store information about vehicles that people drive.

Items – Each table contains zero or more items. An item is a group of attributes that is uniquely identifiable among all of the other items. In a People table, each item represents a person. For a Cars table, each item represents one vehicle. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is no limit to the number of items you can store in a table.

Attributes – Each item is composed of one or more attributes. An attribute is a fundamental data element, something that does not need to be broken down any further. For example, an item in a People table contains attributes called PersonID, LastName, FirstName, and so on. For a Department table, an item might have attributes such as DepartmentID, Name, Manager, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

Security with Amazon DynamoDB

DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

Choose the Right AWS Database Service

Database Type	Use Cases	AWS Service
Relational	Traditional applications, ERP, CRM, e-commerce	Amazon RDS, Amazon Aurora, Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	Amazon ElastiCache for Memcached, Amazon ElastiCache for Redis
Document	Content management, catalogs, user profiles	Amazon DocumentDB (with MongoDB compatibility)
Wide column	High-scale industrial apps for equipment maintenance, fleet management, and route optimization	Amazon Keyspaces (for Apache Cassandra)
Graph	Fraud detection, social networking, recommendation engines	Amazon Neptune
Time series	IoT applications, DevOps, industrial telemetry	Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	Amazon QLDB

As the industry changes, applications and databases change too. Today, with larger applications, you no longer see just one database supporting it. Instead, these applications are being broken into smaller services, each with their own purpose-built database supporting it.

MONITORING ON AWS :

When operating a website like the Employee Directory Application on AWS you may have questions like:

- How many people are visiting my site day to day?
- How can I track the number of visitors over time?
- How will I know if the website is having performance or availability issues?
- What happens if my Amazon EC2 instance runs out of capacity?
- Will I be alerted if my website goes down?

You need a way to collect and analyze data about the operational health and usage of your resources. The act of collecting, analyzing, and using data to make decisions or answer questions about your IT resources and systems is called monitoring. Monitoring enables you to have a near real-time pulse on your system and answer the questions listed above

Understand the Benefits of Monitoring

Monitoring gives you visibility into your resources, but the question now is, "Why is that important?" The following are some of the benefits of monitoring.

Respond to operational issues proactively before your end users are aware of them : It's a bad practice to wait for end users to let you know your application is experiencing an outage. Through monitoring, you can keep tabs on metrics like error response rate or request latency, over time, that help signal that an outage is going to occur. This enables you to automatically or manually perform actions to prevent the outage from happening—fixing the problem before your end users are aware of it.

Improve the performance and reliability of your resources : Monitoring the different resources that comprise your application provides you with a full picture of how your solution behaves as a system. Monitoring, if done well, can illuminate bottlenecks and inefficient architectures. This enables you to drive performance and reliability improvement processes.

Recognize security threats and events **:** When you monitor resources, events, and systems over time, you create what is called a baseline. A baseline defines what activity is normal. Using a baseline, you can spot anomalies like unusual traffic spikes or unusual IP addresses accessing your resources. When an anomaly occurs, an alert can be sent out or an action can be taken to investigate the event.

Make data-driven decisions for your business **:** Monitoring is not only to keep an eye on IT operational health. It also helps drive business decisions. For example, let's say

you launched a new feature for your app, and want to know whether it's being used. You can collect application-level metrics and view the number of users who use the new feature. With your findings, you decide whether to invest more time into improving the new feature.

Create more cost-effective solutions **:** Through monitoring, you can view resources that are being underutilized and rightsize your resources to your usage. This helps you optimize cost and make sure you aren't spending more money than necessary.

Enable Visibility

AWS resources create data you can monitor through metrics, logs, network traffic, events, and more. This data is coming from components that are distributed in nature, which can lead to difficulty in collecting the data you need if you don't have a centralized place to review it all. AWS has already done that for you with a service called Amazon CloudWatch.

Amazon CloudWatch is a monitoring and observability service that collects data like those mentioned in this module. CloudWatch provides actionable insights into your applications, and enables you to respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health. This unified view is important. You can use CloudWatch to:

- Detect anomalous behavior in your environments.
- Set alarms to alert you when something's not right.
- Visualize logs and metrics with the AWS Management Console.
- Take automated actions like scaling.
- Troubleshoot issues.
- Discover insights to keep your applications healthy.

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.

The employee directory app is built with various AWS services working together as building blocks. It would be difficult to monitor all of these different services independently, so CloudWatch acts as one centralized place where metrics are

gathered and analyzed. You already learned how EC2 instances post CPU utilization as a metric to CloudWatch. Different AWS resources post different metrics that you can monitor.

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. For example, you can get statistics for a specific EC2 instance by specifying the InstanceId dimension when you search.

Set Up Custom Metrics

Let's say for your application you wanted to record the number of page views your website gets. How would you record this metric to CloudWatch? It's an application-level metric, meaning that it's not something the EC2 instance would post to CloudWatch by default. This is where **custom metrics** come in. Custom metrics allows you to publish your own metrics to CloudWatch.

If you want to gain more granular visibility, you can use **high-resolution custom metrics**, which enable you to collect custom metrics down to a 1-second resolution. This means you can send one data point per second per custom metric. Other examples of custom metrics are:

- Web page load times
- Request error rates
- Number of processes or threads on your instance
- Amount of work performed by your application

Note: You can get started with custom metrics by programmatically sending the metric to CloudWatch using the PutMetricData API.

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.

Some services are set up to send log data to CloudWatch Logs with minimal effort, like AWS Lambda. With AWS Lambda, all you need to do is give the Lambda function the correct IAM permissions to post logs to CloudWatch Logs. Other services require more configuration. For example, 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.

You first need to decide what metric you want to set up an alarm for, then you define the threshold at which you want the alarm to trigger. Next, you define the specified time period of which the metric should cross the threshold for the alarm to be triggered.

For example, if you wanted to set up an alarm for an EC2 instance to trigger when the CPU utilization goes over a threshold of 80%, you also need to specify the time period the CPU utilization is over the threshold. You don't want to trigger an alarm based on short temporary spikes in the CPU. You only want to trigger an alarm if the CPU is elevated for a sustained amount of time, for example if it is over 80% for 5 minutes or longer, when there is a potential resource issue.

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.

An alarm can be triggered when it transitions from one state to another. Once an alarm is triggered, it can initiate an action. Actions can be an Amazon EC2 action, an Auto Scaling action, or a notification sent to Amazon Simple Notification Service (SNS).

Use CloudWatch Alarms to Prevent and Troubleshoot Issues

CloudWatch Logs uses metric filters to turn the log data into metrics that you can graph or set an alarm on. For the employee directory application, let's say you set up a metric filter for 500-error response codes.

Then, you define an alarm for that metric that will go into the ALARM state if 500-error responses go over a certain amount for a sustained time period. Let's say if it's more than five 500-error responses per hour, the alarm should enter the ALARM state. Next, you define an action that you want to take place when the alarm is triggered.

In this case, it makes sense to send an email or text alert to you so you can start troubleshooting the website, hopefully fixing it before it becomes a bigger issue. Once the alarm is set up, you feel comfortable knowing that if the error happens again, you'll be notified promptly.

You can set up different alarms for different reasons to help you prevent or troubleshoot operational issues. In the scenario just described, the alarm triggered an SNS notification that went to a person who looked into the issue manually. Another option is to have alarms trigger actions that automatically remediate technical issues.

For example, you can set up an alarm to trigger an EC2 instance to reboot, or scale services up or down. You can even set up an alarm to trigger an SNS notification, which then triggers an AWS Lambda function. The Lambda function then calls any AWS API to manage your resources, and troubleshoot operational issues. By using AWS services together like this, you respond to events more quickly.

OPTIMIZING SOLUTIONS ON AWS :

What Is Availability?

The availability of a system is typically expressed as a percentage of uptime in a given year or as a number of nines. Below, you can see a list of the percentages of availability based on the downtime per year, as well as its notation in nines.

Availability (%)	Downtime (per year)
90% ("one nine")	36.53 days
99% ("two nines")	3.65 days
99.9% ("three nines")	8.77 hours
99.95% ("three and a half nines")	4.38 hours
99.99% ("four nines")	52.60 minutes
99.995% ("four and a half nines")	26.30 minutes
99.999% ("five nines")	5.26 minutes

To increase availability, you need redundancy. This typically means more infrastructure: more data centers, more servers, more databases, and more replication of data. You can imagine that adding more of this infrastructure means a higher cost. Customers want the application to always be available, but you need to draw a line where adding redundancy is no longer viable in terms of revenue.

Manage Replication, Redirection, and High Availability

Create a Process for Replication

The first challenge is that you need to create a process to replicate the configuration files, software patches, and application itself across instances. The best method is to automate where you can.

Address Customer Redirection

The second challenge is how to let the clients, the computers sending requests to your server, know about the different servers. There are different tools that can be used here. The most common is using a Domain Name System (DNS) where the client uses one record which points to the IP address of all available servers. However, the time it takes to update that list of IP addresses and for the clients to become aware of such

change, sometimes called propagation, is typically the reason why this method isn't always used.

Another option is to use a load balancer which takes care of health checks and distributing the load across each server. Being between the client and the server, the load balancer avoids propagation time issues.

Understand the Types of High Availability

The last challenge to address when having more than one server is the type of availability you need—either be an active-passive or an active-active system.

- Active-Passive: With an active-passive system, only one of the two instances is available at a time. One advantage of this method is that for stateful applications where data about the client's session is stored on the server, there won't be any issues as the customers are always sent to the same server where their session is stored.
- Active-Active: A disadvantage of active-passive and where an active-active system shines is scalability. By having both servers available, the second server can take some load for the application, thus allowing the entire system to take more load. However, if the application is stateful, there would be an issue if the customer's session isn't available on both servers. Stateless applications work better for active-active systems.

Route Traffic with Amazon Elastic Load Balancing

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

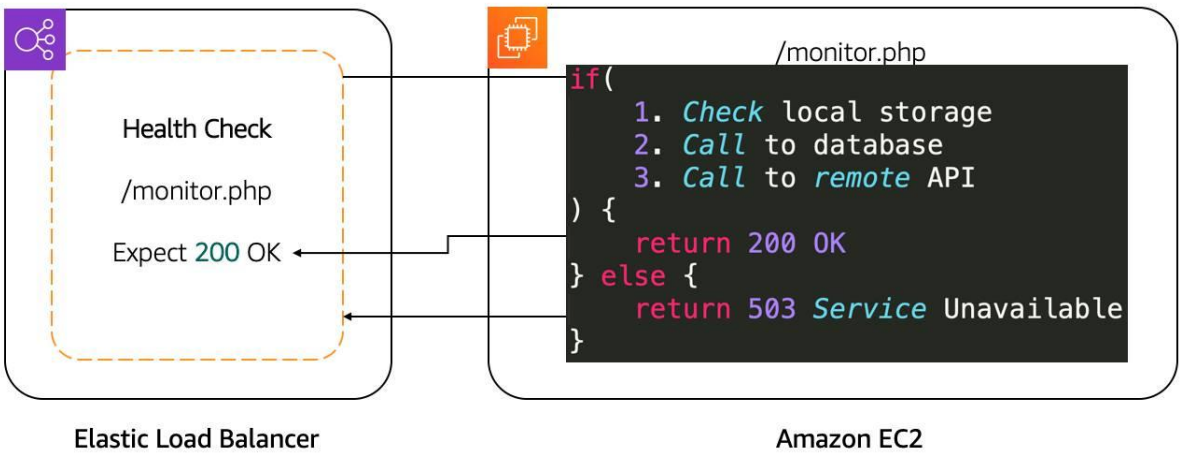
The ELB service provides a major advantage over using your own solution to do load balancing, in that you don't need to manage or operate it. It can distribute incoming application traffic across EC2 instances as well as containers, IP addresses, and AWS Lambda functions.

- 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

Taking the time to define an appropriate health check is critical. Only verifying that the port of an application is open doesn't mean that the application is working. It also doesn't mean that simply making a call to the home page of an application is the right way either.

For example, the employee directory application depends on a database, and S3. The health check should validate all of those elements. One way to do that would be to create a monitoring webpage like "/monitor" that will make a call to the database to ensure it can connect and get data, and make a call to S3. Then, you point the health check on the load balancer to the "/monitor" page.

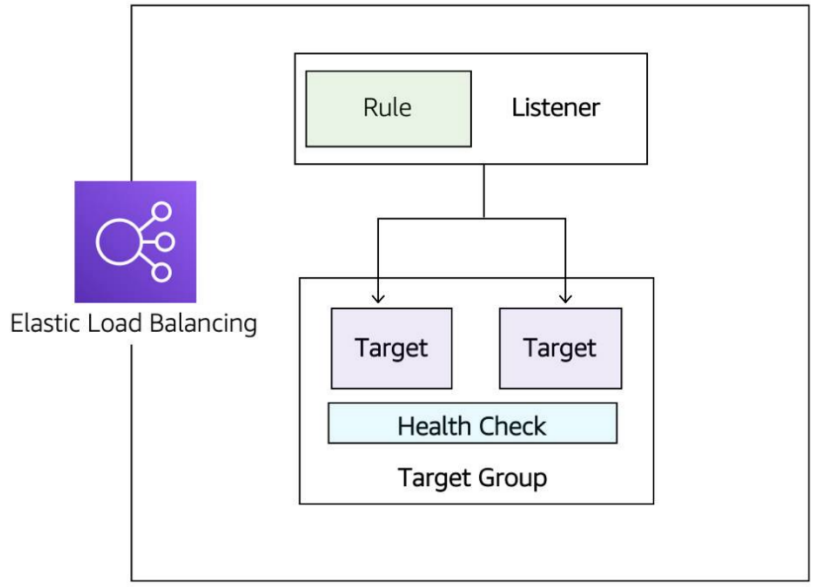


After determining the availability of a new EC2 instance, the load balancer starts sending traffic to it. If ELB determines that an EC2 instance is no longer working, it stops sending traffic to it and lets EC2 Auto Scaling know. EC2 Auto Scaling's responsibility is to remove it from the group and replace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

ELB COMPONENTS

The ELB service is made up of three main components.



- **Listeners:** The client connects to the listener. This is often referred to as client-side. To define a listener, a port must be provided as well as the protocol, depending on the load balancer type. There can be many listeners for a single load balancer.
- **Target groups:** The backend servers, or server-side, is defined in one or more target groups. This is where you define the type of backend you want to direct traffic to, such as EC2 Instances, AWS Lambda functions, or IP addresses. Also, a health check needs to be defined for each target group.
- **Rules:** To associate a target group to a listener, a rule must be used. Rules are made up of a condition that can be the source IP address of the client and a condition to decide which target group to send the traffic to.

APPLICATION LOAD BALANCER

Here are some primary features of Application Load Balancer (ALB).

ALB routes traffic based on request data. It makes routing decisions based on the HTTP protocol like the URL path (/upload) and host, HTTP headers and method, as well as the source IP address of the client. This enables granular routing to the target groups.

Send responses directly to the client. ALB has the ability to reply directly to the client with a fixed response like a custom HTML page. It also has the ability to send a redirect to the client which is useful when you need to redirect to a specific website or to redirect the request from HTTP to HTTPS, removing that work from your backend servers.

ALB supports TLS offloading. Speaking of HTTPS and saving work from backend servers, ALB understands HTTPS traffic. To be able to pass HTTPS traffic through ALB, an SSL certificate is provided by either importing a certificate via Identity and Access Management (IAM) or AWS Certificate Manager (ACM) services, or by creating one for free using ACM. This ensures the traffic between the client and ALB is encrypted.

Authenticate users. On the topic of security, ALB has the ability to authenticate the users before they are allowed to pass through the load balancer. ALB uses the OpenID Connect protocol and integrates with other AWS services to support more protocols like SAML, LDAP, Microsoft AD, and more.

Secure traffic. To prevent traffic from reaching the load balancer, you configure a security group to specify the supported IP address ranges.

ALB uses the round-robin routing algorithm. ALB ensures each server receives the same number of requests in general. This type of routing works for most applications.

ALB uses the least outstanding request routing algorithm. If the requests to the backend vary in complexity where one request may need a lot more CPU time than another, then the least outstanding request algorithm is more appropriate. It's also the right routing algorithm to use if the targets vary in processing capabilities. An outstanding request is when a request is sent to the backend server and a response hasn't been received yet.

For example, if the EC2 instances in a target group aren't the same size, one server's CPU utilization will be higher than the other if the same number of requests are sent to each server using the round-robin routing algorithm. That same server will have more outstanding requests as well. Using the least outstanding request routing algorithm would ensure an equal usage across targets.

ALB has sticky sessions. In the case where requests need to be sent to the same backend server because the application is stateful, then use the sticky session feature. This feature uses an HTTP cookie to remember across connections which server to send the traffic to. Finally, ALB is specifically for HTTP and HTTPS traffic. If your application uses a different protocol, then consider the Network Load Balancer (NLB).

NETWORK LOAD BALANCER

Here are some primary features of Network Load Balancer (NLB).Network Load Balancer supports TCP, UDP, and TLS protocols. HTTPS uses TCP and TLS as protocol. However, NLB operates at the connection layer, so it doesn't understand what a HTTPS request is. That means all features discussed above that are required to understand the HTTP and HTTPS protocol, like routing rules based on that protocol, authentication, and least outstanding request routing algorithm, are not available with NLB.

NLB uses a flow hash routing algorithm. The algorithm is based on:

- The protocol
- The source IP address and source port
- The destination IP address and destination port
- The TCP sequence number

If all of these parameters are the same, then the packets are sent to the exact same target. If any of them are different in the next packets, then the request may be sent to a different target.

NLB has sticky sessions. Different from ALB, these sessions are based on the source IP address of the client instead of a cookie.

NLB supports TLS offloading. NLB understands the TLS protocol. It can also offload TLS from the backend servers similar to how ALB works.

NLB handles millions of requests per second. While ALB can also support this number of requests, it needs to scale to reach that number. This takes time. NLB can instantly handle this amount of requests.

NLB supports static and elastic IP addresses. There are some situations where the application client needs to send requests directly to the load balancer IP address instead of using DNS. For example, this is useful if your application can't use DNS or if the connecting clients require firewall rules based on IP addresses. In this case, NLB is the right type of load balancer to use.

NLP preserves source IP address. NLB preserves the source IP address of the client when sending the traffic to the backend. With ALB, if you look at the source IP address of the requests, you will find the IP address of the load balancer. While with NLB, you would see the real IP address of the client, which is required by the backend application in some cases.

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:

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	✓	

Amazon EC2 Auto Scaling

Availability and reachability is improved by adding one more server. However, the entire system can again become unavailable if there is a capacity issue. Let's look at that load issue with both types of systems we discussed, active-passive and active-active.

Vertical Scaling

If there are too many requests sent to a single active-passive system, the active server will become unavailable and hopefully failover to the passive server. But this doesn't solve anything. With active-passive, you need vertical scaling. This means increasing the size of the server. With EC2 instances, you select either a larger type or a different

instance type. This can only be done while the instance is in a stopped state. In this scenario, the following steps occur:

1. Stop the passive instance. This doesn't impact the application since it's not taking any traffic.
2. Change the instance size or type, then start the instance again.
3. Shift the traffic to the passive instance, turning it active.
4. The last step is to stop, change the size, and start the previous active instance as both instances should match.

When the amount of requests reduces, the same operation needs to be done. Even though there aren't that many steps involved, it's actually a lot of manual work to do. Another disadvantage is that a server can only scale vertically up to a certain limit.

Once that limit is reached, the only option is to create another active-passive system and split the requests and functionalities across them. This could require massive application rewriting.This is where the active-active system can help. When there are too many requests, this system can be scaled horizontally by adding more servers.

Horizontal Scaling

As mentioned above, for the application to work in an active-active system, it's already created as stateless, not storing any client session on the server. This means that having two servers or having four wouldn't require any application changes. It would only be a matter of creating more instances when required and shutting them down when the traffic decreases.

The Amazon EC2 Auto Scaling service can take care of that task by automatically creating and removing EC2 instances based on metrics from Amazon CloudWatch.

You can see that there are many more advantages to using an active-active system in comparison with an active-passive. Modifying your application to become stateless enables scalability.

Integrate ELB with EC2 Auto Scaling

The ELB service integrates seamlessly with EC2 Auto Scaling. As soon as a new EC2 instance is added to or removed from the EC2 Auto Scaling group, ELB is notified. However, before it can send traffic to a new EC2 instance, it needs to validate that the application running on that EC2 instance is available.

This validation is done via the health checks feature of ELB. Monitoring is an important part of load balancers, as it should route traffic to only healthy EC2 instances. That's why ELB supports two types of health checks.

- Establishing a connection to a backend EC2 instance using TCP, and marking the instance as available if that connection is successful.
- Making an HTTP or HTTPS request to a webpage that you specify, and validating that an HTTP response code is returned.

Differentiate Between Traditional Scaling and Auto Scaling

With a traditional approach to scaling, you buy and provision enough servers to handle traffic at its peak. However, this means that at night time, there is more capacity than traffic. This also means you're wasting money. Turning off those servers at night or at times where the traffic is lower only saves on electricity.

The cloud works differently, with a pay-as-you-go model. It's important to turn off the unused services, especially EC2 instances that you pay for On-Demand. One could manually add and remove servers at a predicted time. But with unusual spikes in traffic, this solution leads to a waste of resources with over-provisioning or with a loss of customers due to under-provisioning.

The need here is for a tool that automatically adds and removes EC2 instances according to conditions you define—that's exactly what the EC2 Auto Scaling service does.

Use Amazon EC2 Auto Scaling

The EC2 Auto Scaling service works to add or remove capacity to keep a steady and predictable performance at the lowest possible cost. By adjusting the capacity to exactly what your application uses, you only pay for what your application needs. And even with applications that have steady usage, EC2 Auto Scaling can help with fleet management. If there is an issue with an EC2 instance, EC2 Auto Scaling can automatically replace that instance. This means that EC2 Auto Scaling helps both to scale your infrastructure and ensure high availability.

Configure EC2 Auto Scaling Components

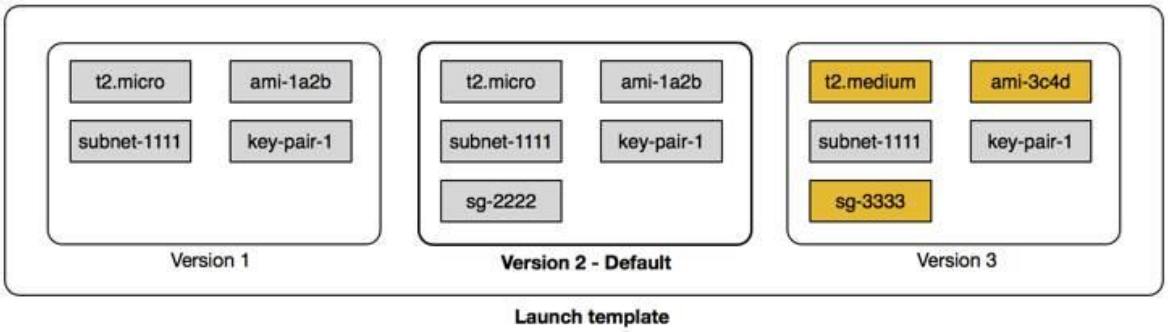
There are three main components to EC2 Auto Scaling.

- **Launch template or configuration:** What resource should be automatically scaled?
- **EC2 Auto Scaling Group:** Where should the resources be deployed?
- **Scaling policies:** When should the resources be added or removed?

Learn About Launch Templates

There are multiple parameters required to create EC2 instances: Amazon Machine Image (AMI) ID, instance type, security group, additional Amazon Elastic Block Store (EBS) volumes, and more. All this information is also required by EC2 Auto Scaling to create the EC2 instance on your behalf when there is a need to scale. This information is stored in a launch template.

You can use a launch template to manually launch an EC2 instance. You can also use it with EC2 Auto Scaling. It also supports versioning, which allows for quickly rolling back if there was an issue or to specify a default version of your launch template. This way, while iterating on a new version, other users can continue launching EC2 instances using the default version until you make the necessary changes.



You can create a launch template one of three ways.

- The fastest way to create a template is to use an existing EC2 instance. All the settings are already defined.
- Another option is to create one from an already existing template or a previous version of a launch template.
- The last option is to create a template from scratch. The following options will need to be defined: AMI ID, instance type, key pair, security group, storage, and resource tags.

Note: Another way to define what Amazon EC2 Auto Scaling needs to scale is by using a **launch configuration**. It's similar to the launch template, but it doesn't allow for versioning using a previously created launch configuration as a template. Nor does it allow for creating one from an already existing EC2 instance. For these reasons and to ensure that you're getting the latest features from Amazon EC2, use a launch template instead of launch configuration.

Get to Know EC2 Auto Scaling Groups

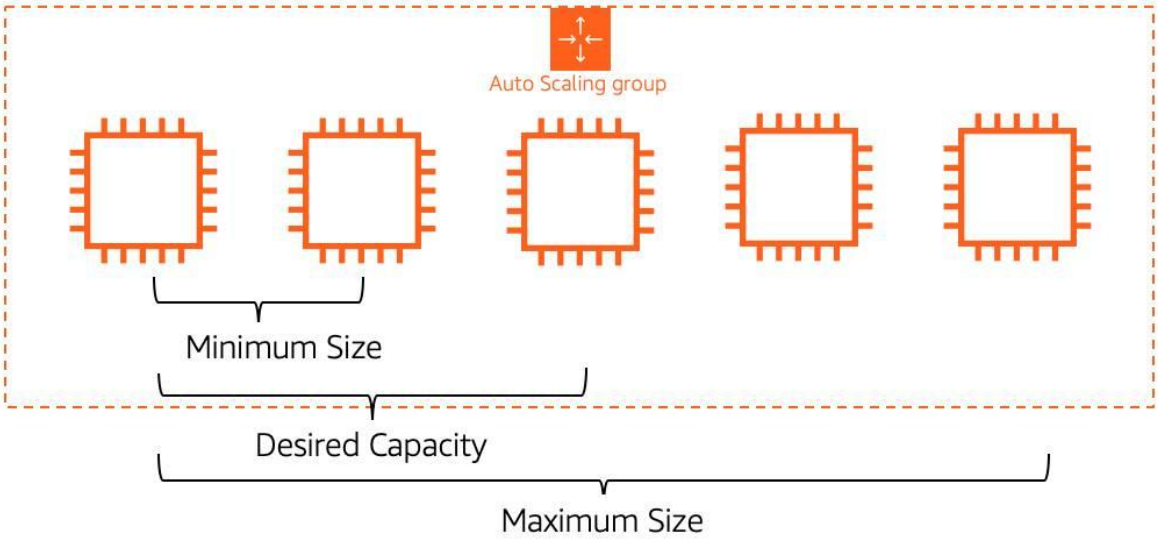
The next component that EC2 Auto Scaling needs is an EC2 Auto Scaling Group (ASG). An ASG enables you to define where EC2 Auto Scaling deploys your

resources. This is where you specify the Amazon Virtual Private Cloud (VPC) and subnets the EC2 instance should be launched in.

EC2 Auto Scaling takes care of creating the EC2 instances across the subnets, so it's important to select at least two subnets that are across different Availability Zones.

ASGs also allow you to specify the type of purchase for the EC2 instances. You can use On-Demand only, Spot only, or a combination of the two, which allows you to take advantage of Spot instances with minimal administrative overhead.To specify how many instances EC2 Auto Scaling should launch, there are three capacity settings to configure for the group size.

- **Minimum:** The minimum number of instances running in your ASG even if the threshold for lowering the amount of instances is reached.
- **Maximum:** The maximum number of instances running in your ASG even if the threshold for adding new instances is reached.
- **Desired capacity:** The amount of instances that should be in your ASG. This number can only be within or equal to the minimum or maximum. EC2 Auto Scaling automatically adds or removes instances to match the desired capacity number.

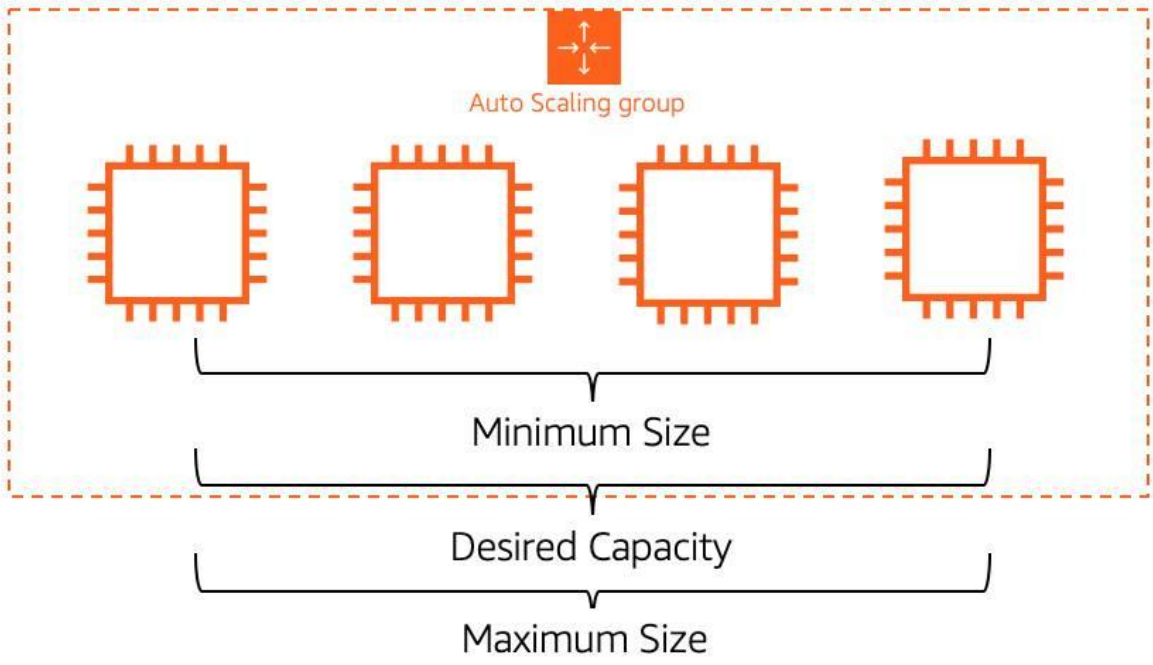


When EC2 Auto Scaling removes EC2 instances because the traffic is minimal, it keeps removing EC2 instances until it reaches a minimum capacity. Depending on your application, using a minimum of two is a good idea to ensure high availability, but you know how many EC2 instances at a bare minimum your application requires at all times. When reaching that limit, even if EC2 Auto Scaling is instructed to remove an instance, it does not, to ensure the minimum is kept.

On the other hand, when the traffic keeps growing, EC2 Auto Scaling keeps adding EC2 instances. This means the cost for your application will also keep growing. That's why it's important to set a maximum amount to make sure it doesn't go above your budget.

The desired capacity is the amount of EC2 instances that EC2 Auto Scaling creates at the time the group is created. If that number decreases, then EC2 Auto Scaling removes the oldest instance by default. If that number increases, then EC2 Auto Scaling creates new instances using the launch template.

Ensure Availability with EC2 Auto Scaling



Using different numbers for minimum, maximum, and desired capacity is used for dynamically adjusting the capacity. However, if you prefer to use EC2 Auto Scaling for fleet management, you can configure the three settings to the same number, for example four. EC2 Auto Scaling will ensure that if an EC2 instance becomes unhealthy, it replaces it to always ensure that four EC2 instances are available. This ensures high availability for your applications.

Enable Automation with Scaling Policies

By default, an ASG will be kept to its initial desired capacity. Although it's possible to manually change the desired capacity, you can also use scaling policies.

In the AWS Monitoring module, you learned about Amazon CloudWatch metrics and alarms. You use **metrics** to keep information about different attributes of your EC2 instance like the CPU percentage. You use **alarms** to specify an action when a

threshold is reached. Metrics and alarms are what scaling policies use to know when to act. For example, you set up an alarm that says when the CPU utilization is above 70% across the entire fleet of EC2 instances, trigger a scaling policy to add an EC2 instance.

There are three types of scaling policies: simple, step, and target tracking scaling.

Simple Scaling Policy

A simple scaling policy allows you to do exactly what's described above. You use a CloudWatch alarm and specify what to do when it is triggered. This can be a number of EC2 instances to add or remove, or a specific number to set the desired capacity to. You can specify a percentage of the group instead of using an amount of EC2 instances, which makes the group grow or shrink more quickly.

Once this scaling policy is triggered, it waits a cooldown period before taking any other action. This is important as it takes time for the EC2 instances to start and the CloudWatch alarm may still be triggered while the EC2 instance is booting. For example, you could decide to add an EC2 instance if the CPU utilization across all instances is above 65%. You don't want to add more instances until that new EC2 instance is accepting traffic.

However, what if the CPU utilization was now above 85% across the ASG? Only adding one instance may not be the right move here. Instead, you may want to add another step in your scaling policy. Unfortunately, a simple scaling policy can't help with that.

Step Scaling Policy

This is where a step scaling policy helps. Step scaling policies respond to additional alarms even while a scaling activity or health check replacement is in progress. Similar to the example above, you decide to add two more instances in case the CPU utilization is at 85%, and four more instances when it's at 95%.

Deciding when to add and remove instances based on CloudWatch alarms may seem like a difficult task. This is why the third type of scaling policy exists: target tracking.

Target Tracking Scaling Policy

If your application scales based on average CPU utilization, average network utilization (in or out), or based on request count, then this scaling policy type is the one to use. All you need to provide is the target value to track and it automatically creates the required CloudWatch alarms.

THANK YOU !