# **AWS Technical Essentials**

Contents

[**AWS Technical Essentials** 1](#_Toc171082581)

[**Module - 1: Introduction to Amazon Web Services** 7](#_Toc171082582)

[**1.1. What is AWS?** 7](#_Toc171082583)

[**Cloud computing deployment models** 7](#_Toc171082584)

[**Real-time Scenario** 8](#_Toc171082585)

[**Summary** 8](#_Toc171082586)

[**Six advantages of cloud computing** 9](#_Toc171082587)

[**Resources** 9](#_Toc171082588)

[**1.2. AWS Global Infrastructure** 9](#_Toc171082589)

[**Regions** 9](#_Toc171082590)

[**Availability Zones** 10](#_Toc171082591)

[**Scope of AWS services** 11](#_Toc171082592)

[**Maintaining resiliency** 11](#_Toc171082593)

[**Edge locations** 12](#_Toc171082594)

[**Resources** 12](#_Toc171082595)

[**1.3. Interacting with AWS** 12](#_Toc171082596)

[**AWS Management Console** 12](#_Toc171082597)

[**AWS CLI** 13](#_Toc171082598)

[**AWS SDKs** 13](#_Toc171082599)

[**Resources** 14](#_Toc171082600)

[**1.4. Security and the AWS Shared Responsibility Model** 14](#_Toc171082601)

[**AWS responsibility** 15](#_Toc171082602)

[**Customer responsibility** 15](#_Toc171082603)

[**Resources** 16](#_Toc171082604)

[**1.5. Protecting the AWS Root User** 16](#_Toc171082605)

[**AWS Root User** 16](#_Toc171082606)

[**AWS root user credentials** 17](#_Toc171082607)

[**AWS root user best practices** 17](#_Toc171082608)

[**Multi-factor authentication** 18](#_Toc171082609)

[**Resources** 19](#_Toc171082610)

[**1.6. AWS Identity and Access Management** 20](#_Toc171082611)

[**Authentication and authorization** 20](#_Toc171082612)

[**What is IAM?** 20](#_Toc171082613)

[**IAM best practices** 23](#_Toc171082614)

[**Resources** 23](#_Toc171082615)

[**1.7. Assessment** 24](#_Toc171082616)

[**Module - 2: AWS Compute** 25](#_Toc171082617)

[**2.1. Compute As a Service** 25](#_Toc171082618)

[**Servers** 25](#_Toc171082619)

[**Choosing the right compute option** 25](#_Toc171082620)

[**Resources** 25](#_Toc171082621)

[**2.2. Getting Started with Amazon EC2** 26](#_Toc171082622)

[**Amazon EC2** 26](#_Toc171082623)

[**Amazon Machine Image** 26](#_Toc171082624)

[**Relationship between AMIs and EC2 instances** 26](#_Toc171082625)

[**Finding AMIs** 27](#_Toc171082626)

[**Configuring EC2** 27](#_Toc171082627)

[**Amazon EC2 instance types** 27](#_Toc171082628)

[**Instance families** 28](#_Toc171082629)

[**EC2 instance locations** 29](#_Toc171082630)

[**Architecting for high availability** 29](#_Toc171082631)

[**Resources** 29](#_Toc171082632)

[**2.3. Amazon EC2 Instance Lifecycle** 30](#_Toc171082633)

[**Pricing** 31](#_Toc171082634)

[**Resources** 32](#_Toc171082635)

[**2.4. Container Services** 33](#_Toc171082636)

[**Containers** 33](#_Toc171082637)

[**Difference between VMs and containers** 33](#_Toc171082638)

[**Orchestrating containers** 34](#_Toc171082639)

[**Resources** 36](#_Toc171082640)

[**2.5. Introduction to Serverless** 37](#_Toc171082641)

[**Removing the undifferentiated heavy lifting** 37](#_Toc171082642)

[**Go Serverless** 37](#_Toc171082643)

[**Resources** 37](#_Toc171082644)

[**2.6. Serverless with AWS Fargate** 38](#_Toc171082645)

[**Resources** 38](#_Toc171082646)

[**2.7. Serverless with AWS Lambda** 39](#_Toc171082647)

[**Running code on AWS Lambda** 39](#_Toc171082648)

[**How Lambda works** 39](#_Toc171082649)

[**Resources** 40](#_Toc171082650)

[**2.8. Assessment** 41](#_Toc171082651)

[**3. AWS Networking** 42](#_Toc171082652)

[**3.1. Introduction to Networking** 42](#_Toc171082653)

[**Networking defined** 42](#_Toc171082654)

[**Networking basics** 42](#_Toc171082655)

[**IP addresses** 42](#_Toc171082656)

[**IPv4 notation** 42](#_Toc171082657)

[**CIDR notation** 43](#_Toc171082658)

[**Resources** 43](#_Toc171082659)

[**3.2. Amazon VPC** 44](#_Toc171082660)

[**Creating a subnet** 44](#_Toc171082661)

[**High availability with a VPC** 45](#_Toc171082662)

[**Reserved IPs** 45](#_Toc171082663)

[**Gateways** 46](#_Toc171082664)

[**AWS Direct Connect** 47](#_Toc171082665)

[**Resources** 47](#_Toc171082666)

[**3.3. Amazon VPC Routing** 48](#_Toc171082667)

[**Main route table** 48](#_Toc171082668)

[**Custom route tables** 48](#_Toc171082669)

[**Resources** 48](#_Toc171082670)

[**3.4. Amazon VPC Security** 49](#_Toc171082671)

[**Secure subnets with network access control lists** 49](#_Toc171082672)

[**Secure EC2 instances with security groups** 50](#_Toc171082673)

[**Resources** 51](#_Toc171082674)

[**3.5. Assessment** 52](#_Toc171082675)

[**4. AWS Storage** 53](#_Toc171082676)

[**4.1. Storage Types** 53](#_Toc171082677)

[**File storage** 53](#_Toc171082678)

[**Block storage** 54](#_Toc171082679)

[**Object storage** 55](#_Toc171082680)

[**Relating back to traditional storage systems** 55](#_Toc171082681)

[**Resources** 55](#_Toc171082682)

[**4.2. File Storage with Amazon EFS and Amazon FSx** 56](#_Toc171082683)

[**Amazon Elastic File System (Amazon EFS)** 56](#_Toc171082684)

[**Amazon FSx** 56](#_Toc171082685)

[**Resources** 57](#_Toc171082686)

[**4.3. Block Storage with Amazon EC2 Instance Store & Amazon EBS** 58](#_Toc171082687)

[**Amazon EC2 instance store** 58](#_Toc171082688)

[**Amazon EBS** 58](#_Toc171082689)

[**Scaling Amazon EBS volumes** 59](#_Toc171082690)

[**Amazon EBS use cases** 59](#_Toc171082691)

[**EBS volume types** 59](#_Toc171082692)

[**Amazon EBS benefits** 60](#_Toc171082693)

[**Amazon EBS snapshots** 61](#_Toc171082694)

[**Resources** 61](#_Toc171082695)

[**4.4. Object Storage with Amazon S3** 62](#_Toc171082696)

[**Amazon S3** 62](#_Toc171082697)

[**Amazon S3 concepts** 62](#_Toc171082698)

[**Amazon S3 bucket names** 62](#_Toc171082699)

[**Object key names** 63](#_Toc171082700)

[**Amazon S3 use cases** 63](#_Toc171082701)

[**Security in Amazon S3** 64](#_Toc171082702)

[**Amazon S3 and IAM policies** 64](#_Toc171082703)

[**Amazon S3 bucket policies** 64](#_Toc171082704)

[**Amazon S3 encryption** 64](#_Toc171082705)

[**Amazon S3 storage classes** 64](#_Toc171082706)

[**Amazon S3 versioning** 66](#_Toc171082707)

[**Managing your storage lifecycle** 67](#_Toc171082708)

[**Resources** 67](#_Toc171082709)

[**4.5. Assessment** 68](#_Toc171082710)

[**5. Databases on AWS** 69](#_Toc171082711)

[**5.1. Introduction to Databases on AWS** 69](#_Toc171082712)

[**History behind enterprise databases** 69](#_Toc171082713)

[**Relational databases** 69](#_Toc171082714)

[**Relational database management system** 69](#_Toc171082715)

[**Relational database benefits** 69](#_Toc171082716)

[**Relational database use cases** 70](#_Toc171082717)

[**Choose between unmanaged and managed databases** 70](#_Toc171082718)

[**Resources** 71](#_Toc171082719)

[**5.2. Amazon RDS** 72](#_Toc171082720)

[**Amazon RDS overview** 72](#_Toc171082721)

[**Database instances** 72](#_Toc171082722)

[**Storage on Amazon RDS** 72](#_Toc171082723)

[**Amazon RDS in an Amazon Virtual Private Cloud** 73](#_Toc171082724)

[**Backup data** 73](#_Toc171082725)

[**Choosing a backup option** 73](#_Toc171082726)

[**Redundancy with Amazon RDS Multi-AZ** 74](#_Toc171082727)

[**Amazon RDS security** 74](#_Toc171082728)

[**Resources** 75](#_Toc171082729)

[**5.3. Purpose-Built Databases** 76](#_Toc171082730)

[**Purpose-built databases for all application needs** 76](#_Toc171082731)

[**Amazon DynamoDB** 76](#_Toc171082732)

[**Amazon ElastiCache** 76](#_Toc171082733)

[**Amazon MemoryDB for Redis** 76](#_Toc171082734)

[**Amazon DocumentDB (with MongoDB compatibility)** 76](#_Toc171082735)

[**Amazon Keyspaces (for Apache Cassandra)** 76](#_Toc171082736)

[**Amazon Neptune** 77](#_Toc171082737)

[**Amazon Timestream** 77](#_Toc171082738)

[**Amazon Quantum Ledger Database (Amazon QLDB)** 77](#_Toc171082739)

[**Resources** 77](#_Toc171082740)

[**5.4. Amazon DynamoDB** 78](#_Toc171082741)

[**DynamoDB overview** 78](#_Toc171082742)

[**DynamoDB core components** 78](#_Toc171082743)

[**DynamoDB use cases** 79](#_Toc171082744)

[**DynamoDB security** 79](#_Toc171082745)

[**Resources** 80](#_Toc171082746)

[**5.5. Choosing the Right Database Service** 81](#_Toc171082747)

[**AWS database services** 81](#_Toc171082748)

[**Breaking up applications and databases** 81](#_Toc171082749)

[**Resources** 81](#_Toc171082750)

[**5.6. Assessment** 82](#_Toc171082751)

[**6. Monitoring, Load Balancing & Scaling** 83](#_Toc171082752)

[**6.1. Monitoring** 83](#_Toc171082753)

[**Purpose of monitoring** 83](#_Toc171082754)

[**Use metrics to solve problems** 83](#_Toc171082755)

[**Types of metrics** 83](#_Toc171082756)

[**Monitoring benefits** 84](#_Toc171082757)

[**Resources** 84](#_Toc171082758)

[**6.2. Amazon Cloud Watch** 85](#_Toc171082759)

[**Visibility using CloudWatch** 85](#_Toc171082760)

[**How CloudWatch works** 85](#_Toc171082761)

[**CloudWatch concepts** 85](#_Toc171082762)

[**Custom metrics** 86](#_Toc171082763)

[**CloudWatch dashboards** 86](#_Toc171082764)

[**Amazon CloudWatch Logs** 87](#_Toc171082765)

[**CloudWatch alarms** 88](#_Toc171082766)

[**Resources** 88](#_Toc171082767)

[**6.3. Solution Optimization** 89](#_Toc171082768)

[**Availability** 89](#_Toc171082769)

[**Why improve application availability?** 89](#_Toc171082770)

[**Resources** 90](#_Toc171082771)

[**6.4. Traffic Routing with Elastic Load Balancing** 91](#_Toc171082772)

[**Load balancers** 91](#_Toc171082773)

[**ELB features** 91](#_Toc171082774)

[**Health checks** 91](#_Toc171082775)

[**ELB components** 92](#_Toc171082776)

[**Types of load balancers** 93](#_Toc171082777)

[**Selecting between ELB types** 94](#_Toc171082778)

[**Resources** 94](#_Toc171082779)

[**6.5. Amazon EC2 Auto Scaling** 95](#_Toc171082780)

[**Capacity issues** 95](#_Toc171082781)

[**Traditional scaling compared to auto scaling** 96](#_Toc171082782)

[**Amazon EC2 Auto Scaling features** 96](#_Toc171082783)

[**Configure Amazon EC2 Auto Scaling components** 96](#_Toc171082784)

[**Scaling policies** 98](#_Toc171082785)

[**Resources** 98](#_Toc171082786)

[**6.6. Assessment** 99](#_Toc171082787)

[**7.** **Innovation with AWS** 100](#_Toc171082788)

[**What is Amazon CodeWhisperer** 100](#_Toc171082789)

[**Installation** 100](#_Toc171082790)

[**Compatibility** 100](#_Toc171082791)

[**Support** 100](#_Toc171082792)

[**Code Generation** 100](#_Toc171082793)

[**Resources** 100](#_Toc171082794)

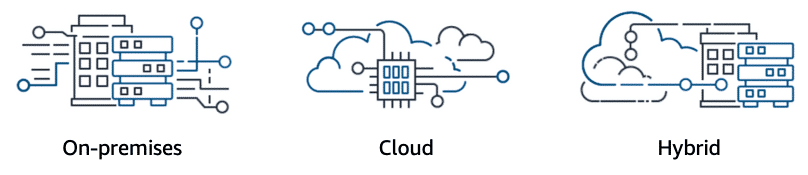
[**8.** **Assessment** 101](#_Toc171082795)

# **Module - 1: Introduction to Amazon Web Services**

## **1.1. What is AWS?**

### **Cloud computing deployment models**

Cloud computing provides developers and IT departments with the ability to focus on what matters most by avoiding work like procurement, maintenance, and capacity planning. Several deployment strategies have emerged to help meet specific needs of different users. Each type of deployment method provides you with different levels of control, flexibility, and management.



###### **On-premises**

Companies and organizations hosted and maintained hardware such as compute, storage, and networking equipment in their own data centres. They often allocated entire infrastructure departments to take care of their data centres, which resulted in costly operations that made some workloads and experimentation impossible.

As internet use 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 emerged.

###### **Cloud**

Cloud computing is the on-demand delivery of IT resources over the internet with primarily pay-as-you-go pricing. With cloud computing, companies do not have to manage and maintain their own hardware and data centres. Instead, companies like Amazon Web Services (AWS) own and maintain data centres and provide virtual data centre technologies and services to companies and users over the internet.

###### **Hybrid**

This type of deployment is a way to connect infrastructure and applications between cloud-based resources and existing resources that are not located in the cloud. The most common method of hybrid deployment between the cloud and existing on-premises infrastructure connects cloud resources to internal systems to extend and grow an organization's infrastructure into the cloud.

### **Real-time Scenario**

To help differentiate between running workloads on-premises and in the cloud, consider a scenario in which developers must deploy a new application feature. Before they deploy, the team wants to test the feature in a separate quality assurance (QA) environment that has the same configurations as production. In an on-premises solution, an additional environment requires you to buy and install hardware, connect the necessary cabling, provision power, install operating systems, and more. These tasks can be time consuming and expensive. Meanwhile, the new feature’s time-to-market increases while the developers wait for the QA environment.

In contrast, by running your application in the cloud, you can replicate an entire production environment in a matter of minutes or even seconds. Instead of physically installing hardware and connecting cabling, the solution is managed over the internet.

### **Summary**

Using cloud computing saves time during setup and removes redundant and unnecessary tasks. If you look at any application, you will see that some of its aspects are very important to your business, like the code. However, other aspects are no different than any other application that you might make—for example, the compute that the code runs on. Some repetitive common tasks don’t differentiate your business, like installing virtual machines (VMs) or storing backups. By removing these tasks, you can focus on what is strategically unique to your business and let AWS handle the time-consuming tasks that don’t separate you from your competitors, we refer to this as removing "undifferentiated heavy lifting". That's where AWS fits into all of this.

### **Six advantages of cloud computing**

1. Pay-as-you-go.
2. Benefit from massive economies of scale.
3. Stop guessing capacity.
4. Increase speed and agility.
5. Realize cost savings.
6. Go global in minutes.

### **Resources**

* AWS website: [What Is Cloud Computing?](https://aws.amazon.com/what-is-cloud-computing/)
* AWS whitepaper: [Types of Cloud Computing](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/types-of-cloud-computing.html)
* AWS website: [Cloud Computing with AWS](https://aws.amazon.com/what-is-aws/)
* AWS whitepaper: [Overview of Amazon Web Services](https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws-overview/aws-overview.pdf)

## **1.2. AWS Global Infrastructure**

Infrastructure, like data centres 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 Regions and Availability Zones.

### **Regions**

Regions are geographic locations worldwide where AWS hosts its data centres. AWS Regions are named after the location where they reside. AWS has Regions in Asia Pacific, China, Europe, the Middle East, North America, and South America.Each AWS Region is associated with a geographical name and a Region code.

Here are examples of region codes:

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

###### **Choosing the right AWS Region**

AWS Regions are independent from one another. Without explicit customer consent and authorization, data is not replicated from one Region to another. When you decide which AWS Region to host your applications and workloads, consider four main aspects: latency, price, service availability, and compliance.

**1. Latency**

If your application is sensitive to latency (the delay between a request for data and the response), 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, Web Sockets, and Internet of Things (IoT) are significantly affected by high latency. Asynchronous workloads, such as ecommerce applications, can also suffer from user connectivity delays.

**2. Pricing**

Due to the local economy and the physical nature of operating data centres, prices vary from one Region to another. Internet connectivity imported equipment costs, customs, real estate, and other factors impact a Region's pricing.

**3. Service Availability**

Some services might not be available in some Regions. The AWS documentation provides a table that shows the services available in each Region.

**4. Data Compliance**

Enterprise companies often must comply with regulations that require customer data to be stored in a specific geographic territory. Choose a Region that meets your compliance requirements.

### **Availability Zones**

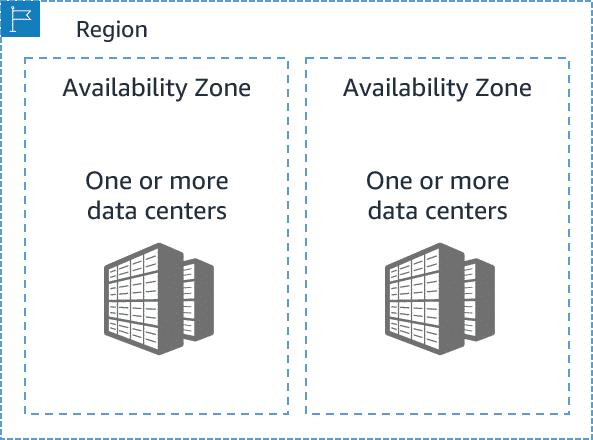
Inside every Region is a cluster of Availability Zones. An Availability Zone consists of one or more data centres with redundant power, networking, and connectivity. These data centres operate in discrete facilities in undisclosed locations. They are connected using redundant high-speed and low-latency links.

Availability Zones also have code names. Because they are located inside regions, they can be addressed by appending a letter to the end of the Region code name.

Here are examples of Availability Zone codes:

* **us-east-1a** is an Availability Zone in us-east-1 (N. Virginia Region).
* **sa-east-1b** is an Availability Zone in sa-east-1 (São Paulo Region).

Therefore, if you see that a resource exists in us-east-1c, you can infer that the resource is in Availability Zone c of the us-east-1 Region.



### **Scope of AWS services**

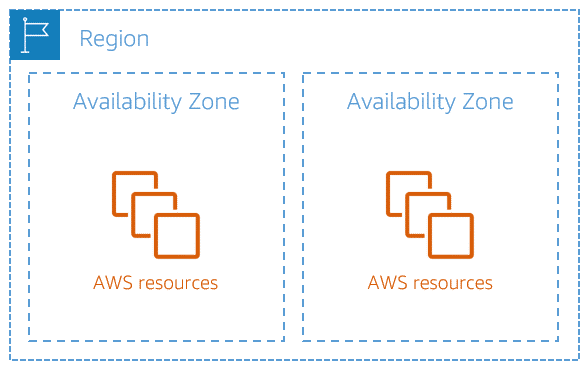
Depending on the AWS service that you use, your resources are either deployed at the Availability Zone, Region, or Global level. Each service is different, so you must understand how the scope of a service might affect your application architecture.

When you operate a Region-scoped service, you only need to select the Region that you want to use. If you are not asked to specify an individual Availability Zone to deploy the service in, this is an indicator that the service operates on a Region-scope level. For Region-scoped services, AWS automatically performs actions to increase data durability and availability.

On the other hand, some services ask you to specify an Availability Zone. With these services, you are often responsible for increasing the data durability and high availability of these resources.

### **Maintaining resiliency**

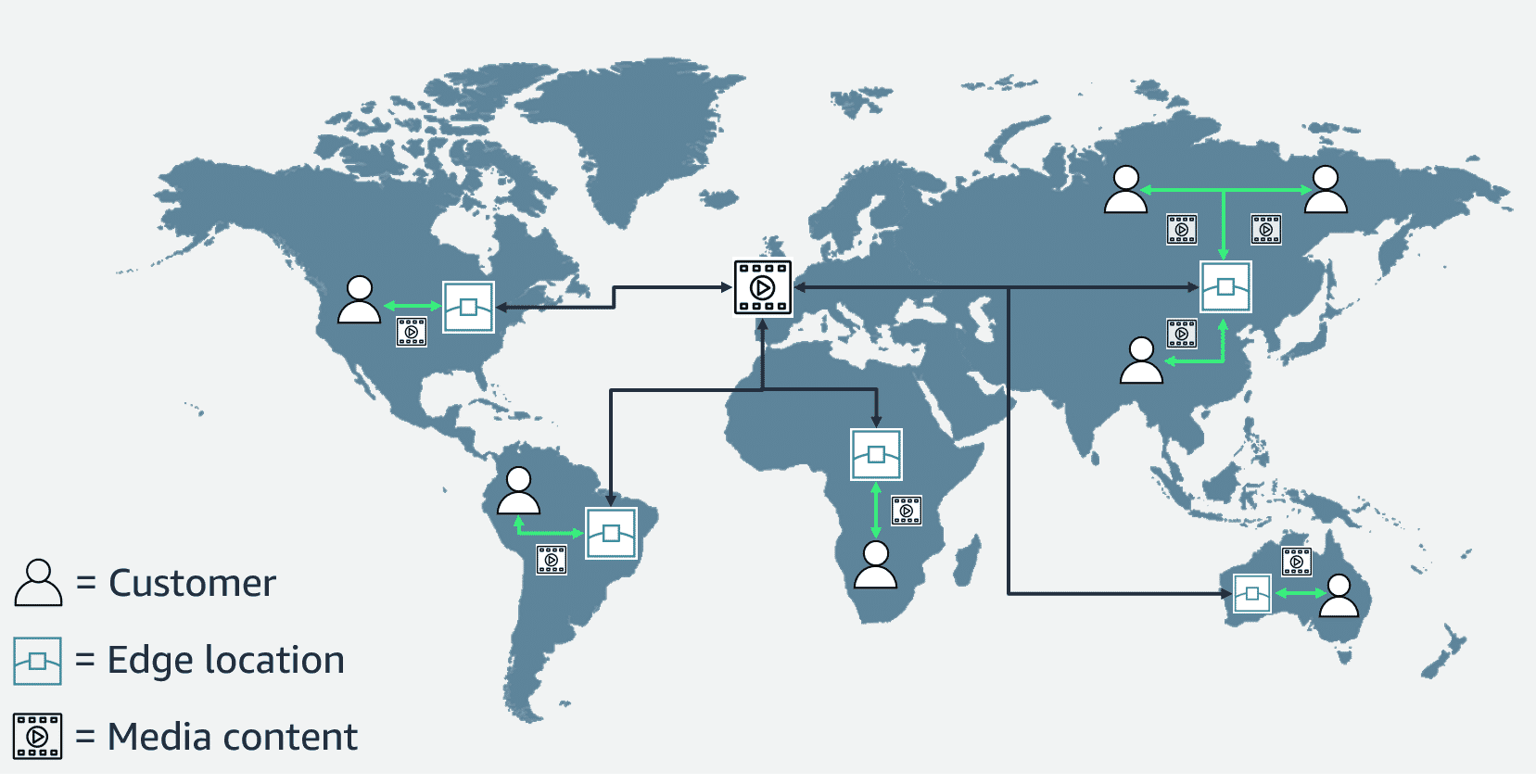
To keep your application available, you must maintain high availability and resiliency. A well-known best practice for cloud architecture is to use Region-scoped, managed services. These services come with availability and resiliency built in. When that is not possible, make sure your workload is replicated across multiple Availability Zones. At a minimum, you should use two Availability Zones. That way, if an Availability Zone fails, your application will have infrastructure up and running in a second Availability Zone to take over the traffic.



### **Edge locations**

Edge locations are global locations where content is cached. For example, if your media content is in London and you want to share video files with your customers in Sydney, you could have the videos cached in an edge location closest to Sydney. This would make it possible for your customers to access the cached videos more quickly than accessing them from London. Currently, there are over 400+ edge locations globally.

Amazon CloudFront delivers your content through a worldwide network of edge locations. When a user requests content that is being served with CloudFront, the request is routed to the location that provides the lowest latency. So that content is delivered with the best possible performance. CloudFront speeds up the distribution of your content by routing each user request through the AWS backbone network to the edge location that can best serve your content.



### **Resources**

* AWS website: [Global Infrastructure](https://aws.amazon.com/about-aws/global-infrastructure/)
* AWS whitepaper: [AWS Global Infrastructure Documentation](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/global-infrastructure.html)
* AWS website: [AWS Regions and Availability Zones](https://aws.amazon.com/about-aws/global-infrastructure/regions_az/)
* AWS reference guide: [AWS Service Endpoints](https://docs.aws.amazon.com/general/latest/gr/rande.html)
* AWS website: [AWS Regional Services](https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/)
* AWS developer guide: [Amazon CloudFront](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html)

## **1.3. Interacting with AWS**

Every action that 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, AWS Command Line Interface (AWS CLI), or AWS SDKs.

### **AWS Management Console**

One way to manage cloud resources is through the web-based console, where you log in and choose the desired service. This can be the easiest way to create and manage resources when you first begin working with the cloud.

### **AWS CLI**

Consider the scenario where you run many servers on AWS for your application’s frontend. You want to run a report to collect data from all the servers. You need to do this programmatically every day because the server details might change. Instead of manually logging in to the console and then copying and pasting information, you can schedule an AWS CLI script with an API call to pull this data for you.  
  
The AWS CLI is a unified tool that you can use to manage AWS services. You can download and configure one tool that you can use to control multiple AWS services from the command line and automate them with scripts. The AWS CLI is open source, and installers are available for Windows, Linux, and macOS.  
  
For example, you run the following API call against a service, using the AWS CLI:

**aws s3api list-buckets**

You will get a response like the following one, listing the buckets in your AWS accounts:

{

"Owner": {

"DisplayName": "tech-essentials",

"ID": "d9881f40b83adh2896eb276f44ffch53677faec805422c83dfk60cc335a7da92"

},

"Buckets": [

{

"CreationDate": "2023-01-10T15:50:20.000Z",

"Name": "aws-tech-essentials"

},

{

"CreationDate": "2023-01-10T16:04:15.000Z",

"Name": "aws-tech-essentials-employee-directory-app"

}

]

}

### **AWS SDKs**

API calls to AWS can also be performed by running code with programming languages. You can do this by using AWS 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, Ruby, Rust, and Swift.  
  
Developers commonly use AWS SDKs to integrate their application source code with AWS services. For example, consider an application with a frontend that runs in Python. Every time the application receives a photo, it uploads the file to a storage service. This action can be achieved in the source code by using the AWS SDK for Python (Boto3).

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

import boto3

ec2 = boto3.client('ec2')

response = ec2.describe\_instances()

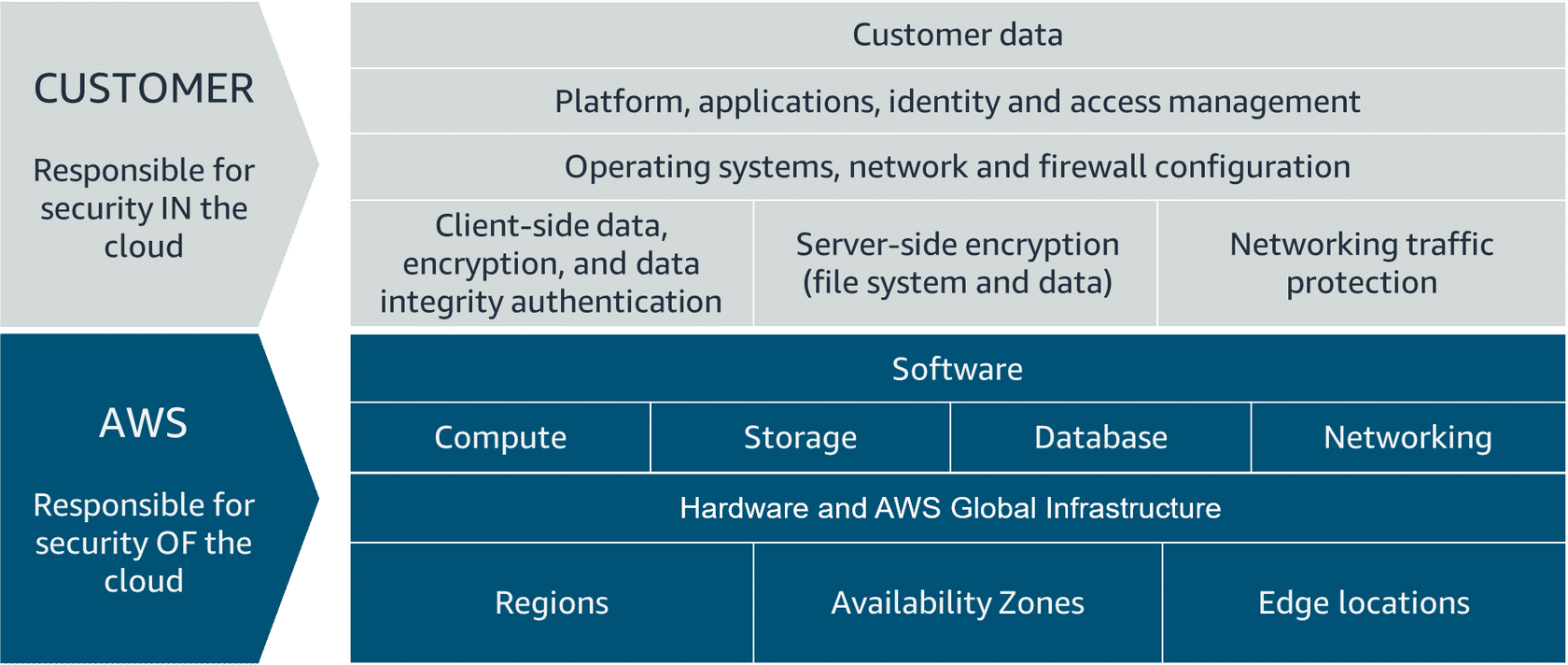
print(response)

### **Resources**

* AWS getting started guide: [What Is the AWS Management Console?](https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/)
* AWS website: [AWS Command Line Interface](https://aws.amazon.com/cli/)
* AWS website: [Tools to Build on AWS](https://aws.amazon.com/developer/tools/)

## **1.4. Security and the AWS Shared Responsibility Model**

When you work 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. The distinction of responsibility is commonly referred to as security ***of*** the cloud as compared to security ***in*** the cloud.



### **AWS responsibility**

AWS is responsible for security of the cloud. This means that AWS protects and secures the infrastructure that runs the services offered in the AWS Cloud.

AWS is responsible for the following:

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

The level of responsibility that AWS has depends on the service. AWS classifies services into two categories.

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

### **Customer responsibility**

Customers are responsible for security in the cloud. When using any AWS service, the customer is responsible for properly configuring the service and their applications, in addition to ensuring that their data is secure.  
  
The customers' level of responsibility depends on the AWS service. Some services require the customer to perform all the necessary security configuration and management tasks. Other more abstracted services require customers to only manage the data and control access to their resources. Using the two categories of AWS services, customers can determine their level of responsibility for each AWS service that they use.

|  |  |  |
| --- | --- | --- |
| **Category** | **Examples of AWS Services in the Category** | **Customer Responsibility** |
| **Infrastructure services** | Compute services, such as Amazon Elastic Compute Cloud (Amazon EC2) | Customers' control the operating system and application platform, in addition to encrypting, protecting, and managing customer data. |
| **Abstracted services** | Services that require very little management from the customer, such as Amazon Simple Storage Service (Amazon S3) | Customers are responsible for customer data, encrypting the data, and protecting it through network firewalls and backups. |

Due to the varying levels of effort, customers must consider which AWS services they use and review the level of responsibility required to secure each service. They must also review how the AWS shared responsibility model aligns with the security standards in their IT environment in addition to any applicable laws and regulations.  
  
A key concept is that customers maintain complete control of their data and are responsible for managing the security related to their content.

For example, you are responsible for the following:

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

### **Resources**

* AWS website: [Shared Responsibility Model](https://aws.amazon.com/compliance/shared-responsibility-model/)

## **1.5. Protecting the AWS Root User**

### **AWS Root User**

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

### **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 that were used to create the account. This allows you to access the AWS Management Console. The second set of credentials is called access keys, which allow you to make programmatic requests from the AWS Command Line Interface (AWS CLI) or AWS API.  
  
Access keys consist of two parts:

* **Access key ID:** for example, ***A2lAl5EXAMPLE***
* **Secret access key:**for example, ***wJalrFE/KbEKxE***

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

**Delete your access keys to stay safe!**

If you don't have an access key for your AWS account root user, don't create one unless you absolutely need to.

If you have an access key for your AWS account root user and want to delete the key, follow these steps:

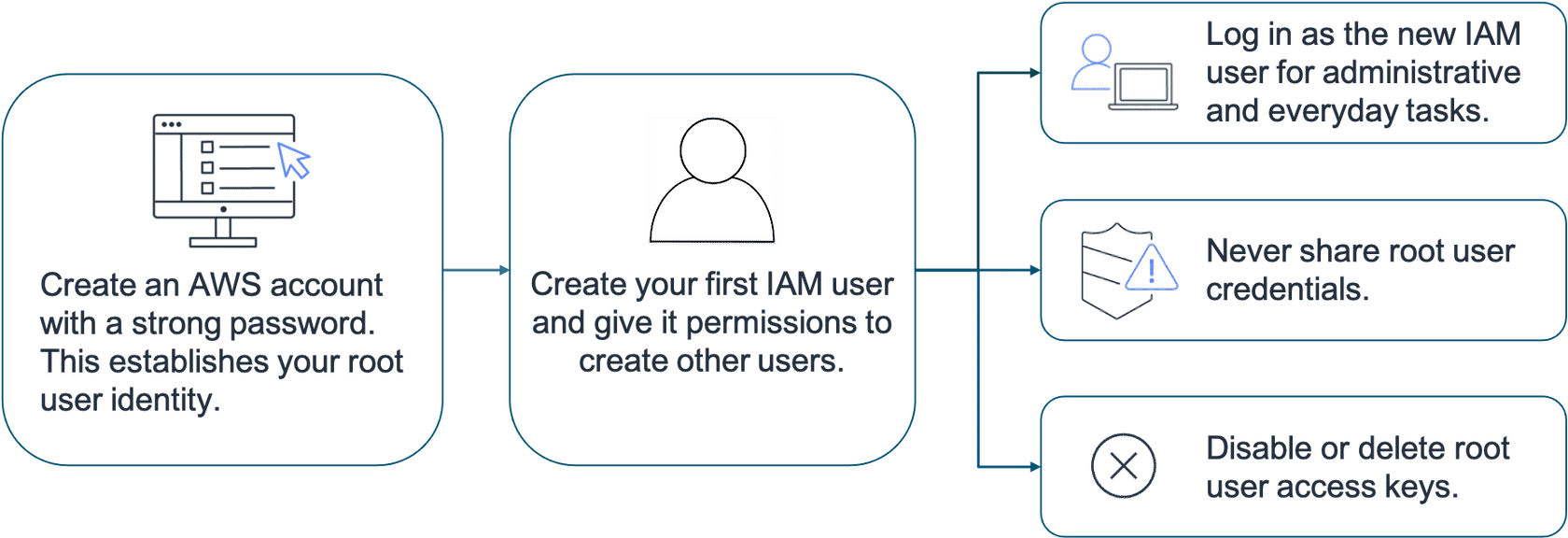
1. In the AWS Management Console, navigate to your username in the navigation bar. Go to the **My Security Credentials** page and sign in with the root user’s credentials.
2. Open the **Access keys** section.
3. Under **Actions**, choose **Delete**.
4. Choose **Yes**.

### **AWS root user best practices**

The root user has complete access to all AWS services and resources in your account, including your billing and personal information. Therefore, you should securely lock away the credentials associated with the root user and not use the root user for everyday tasks.

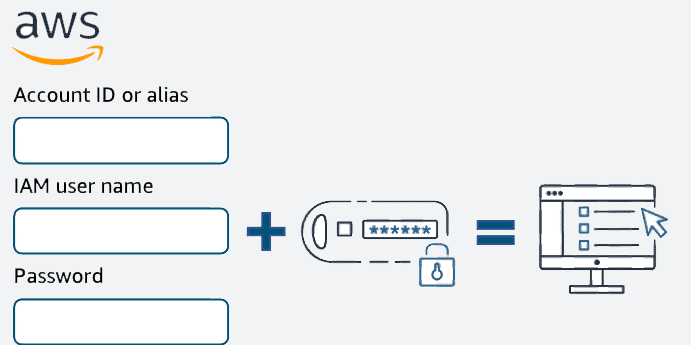
To ensure the safety of the root user, follow these best practices:

* Choose a strong password for the root user.
* Enable multi-factor authentication (MFA) 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.
* Create an Identity and Access Management (IAM) user for administrative tasks or everyday tasks.



### **Multi-factor authentication**

Sometimes a user’s password is easy to guess. This is why using multi-factor authentication (MFA) is important in preventing unwanted account access.



MFA requires two or more authentication methods to verify an identity.

There are three categories of information that MFA pulls from

* 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 a fingerprint or face scanning technology.

With a combination of this information, systems can provide a layered approach to account access. So even if the first method of authentication is cracked by a malicious actor, the second method of authentication, such as a fingerprint, provides another level of security. This extra layer of security can help protect your most important accounts, which is why you should activate MFA on your AWS root user.

**MFA on AWS**

If you activate MFA on your root user, you must present a piece of identifying information from both the something you know category and the something you have category. The first piece of identifying information the user enters is an email and password combination. The second piece of information is a temporary numeric code provided by an MFA device.

**Supported MFA devices.**

AWS supports a variety of MFA mechanisms, such as virtual MFA devices, hardware time-based one-time password (TOTP) tokens, and FIDO security keys.

| **Device** | **Description** | **Supported Devices** |
| --- | --- | --- |
| **Virtual MFA** | A software app that runs on a phone or other device that provides a one-time passcode. These applications can run on unsecured mobile devices, and because of that, they might not provide the same level of security as hardware or FIDO security keys. | Twilio Authy Authenticator, Duo Mobile, LastPass Authenticator, Microsoft Authenticator, Google Authenticator, Symantec VIP |
| **Hardware TOTP token** | A hardware device, generally a key fob or display card device, that generates a one-time, six-digit numeric code based on the time-based one-time password (TOTP) algorithm. | Key fob, display card |
| **FIDO security keys** | FIDO-certified hardware security keys are provided by third-party providers such as Yubico. You can plug your FIDO security key into a USB port on your computer and enable it using the instructions that follow. | [FIDO Certified products](https://fidoalliance.org/certification/fido-certified-products) |

### **Resources**

* AWS user guide: [Enabling a Hardware TOTP token (Console)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_physical.html)
* AWS user guide: [Enabling a FIDO Security Key (Console)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_fido.html)
* AWS user guide: [Enabling Virtual Multi-Factor Authentication Device (Console)](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html)
* AWS website: [Multi-Factor Authentication for IAM](https://aws.amazon.com/iam/features/mfa/)
* AWS reference guide: [Tasks That Require Root User Credentials](https://docs.aws.amazon.com/accounts/latest/reference/root-user-tasks.html)

## **1.6. AWS Identity and Access Management**

### **Authentication and authorization**

When you configure access to any account, two terms come up frequently: **authentication**and **authorization**.

You must fully understand them to properly configure access management on AWS.

**Authentication**

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

**Authorization**

After you’re authenticated and in your AWS account, authorization comes in. Authorization is the process of giving users permission to access AWS resources and services. Authorization determines whether a user can perform certain actions, such as read, edit, delete, or create resources. Authorization answers the question, “What actions can you perform?”

**What is IAM?**

AWS Identity and Access Management (IAM) is an AWS service that helps you 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 sharing your set of access keys or password. You can also provide granular access to those working in your account, so people and services only have permissions to the resources that they need. For example, to provide a user of your AWS account with read-only access to a particular AWS service, you can granularly select which actions and which resources in that service that they can access.

**IAM features**

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

* Global
* Integrated with AWS services.
* Shared access
* Multi-factor authentication
* Identity federation
* Free to use.

**IAM user**

An IAM user represents a person or service that interacts with AWS. You define the user in your AWS account. Any activity done by that user is billed to your account. When you create a user, that user can sign in to gain access to the AWS resources inside your account.

You can also add more users to your account as needed. For example, for your application, you could create individual users in your AWS account that correspond to the people who are working on your application. Each person should have their own login credentials to prevent sharing credentials between users.

**IAM user credentials**

An IAM user consists of a name and a set of credentials.

When you create a user, you can provide them with the following types of access:

* Access to the AWS Management Console
* Programmatic access to the AWS CLI and AWS API

To access the console, provide the user with a username and password. For programmatic access, AWS generates a set of access keys that can be used with the AWS CLI and AWS API. IAM user credentials are considered permanent, which means that they stay with the user until there’s a forced rotation by admins.  
When you create an IAM user, you can grant permissions directly at the user level. This can seem like a good idea if you have only one or a few users. However, as the number of users increases, keeping up with permissions can become more complicated. Fortunately, you can group IAM users and attach permissions at the group level.

**IAM groups**

An IAM group is a collection of users. All users in the group inherit the permissions assigned to the group. This makes it possible to give permissions to multiple users at once. It’s a more convenient and scalable way of managing permissions for users in your AWS account. This is why using IAM groups is a best practice.  
For example, you might organize your IAM groups by developers, security, and admins. You could then place all your IAM users into their respective groups.  
This provides a way to see who has what permissions in your organization. It also helps you scale when new people join, leave, and change roles in your organization.

Keep in mind the following features of groups:

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

The root user can perform all actions on all resources inside an AWS account by default. This contrasts with creating new IAM users, new groups, or new roles. To allow an IAM identity to perform specific actions in AWS, such as implement resources, you must grant the IAM user the necessary permissions.  
  
The way you grant permissions in IAM is by using IAM policies.

**IAM policies**

To manage access and provide permissions to AWS services and resources, you create IAM policies and attach them to an IAM identity. Whenever an IAM identity 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.

**IAM policy examples**

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

The following example provides admin access through an IAM identity-based policy.

{

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

"Statement": [{

"Effect": "Allow",

"Action": "\*",

"Resource": "\*"

}]

}

This policy has four major JSON elements: ***Version***, ***Effect***, ***Action***, and ***Resource***.

* The ***Version***element defines the version of the policy language. It specifies the language syntax rules that are needed by AWS to process a policy. To use all the available policy features, include **"Version": "2012-10-17"** before the **"Statement"**element in your policies.
* The ***Effect***element specifies whether the policy 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 example policy, the action is **"\*"**. This is called a wildcard, and it is used to symbolize every action inside your AWS account.
* The ***Resource***element specifies the object or objects that the policy statement covers. In the policy example, the resource is the wildcard **"\*"**. This represents all resources inside your AWS console.

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

The next example shows a more granular IAM policy.

{

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

"Statement": [

{

"Sid": "DenyS3AccessOutsideMyBoundary",

"Effect": "Deny",

"Action": [

"s3:\*"

],

"Resource": "\*",

"Condition": {

"StringNotEquals": {

"aws:ResourceAccount": [

"222222222222"

]

}

}

}

]

}

This policy uses a *Deny* effect to block access to Amazon S3 actions unless the Amazon S3 resource that's being accessed is in account *222222222222*. This ensures that any Amazon S3 principals are accessing only the resources that are inside of a trusted AWS account.

### **IAM best practices**

This section summarizes some of the most important IAM best practices that you must be familiar with before building solutions in AWS.

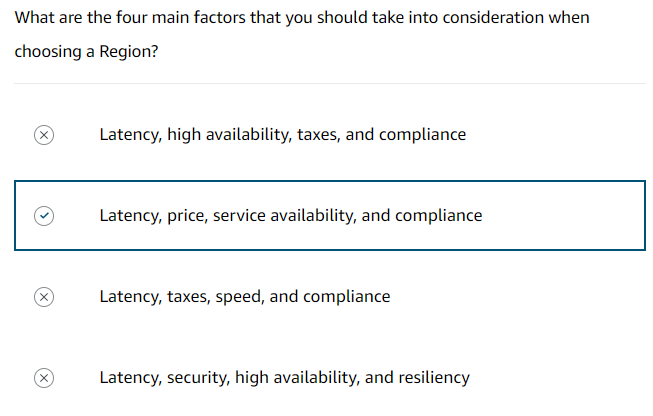
* Lock down the AWS root user.
* Follow the principle of least privilege.
* Use IAM appropriately.
* Use IAM roles when possible.
* Consider using an identity provider.
* Regularly review and remove unused users, roles, and other credentials

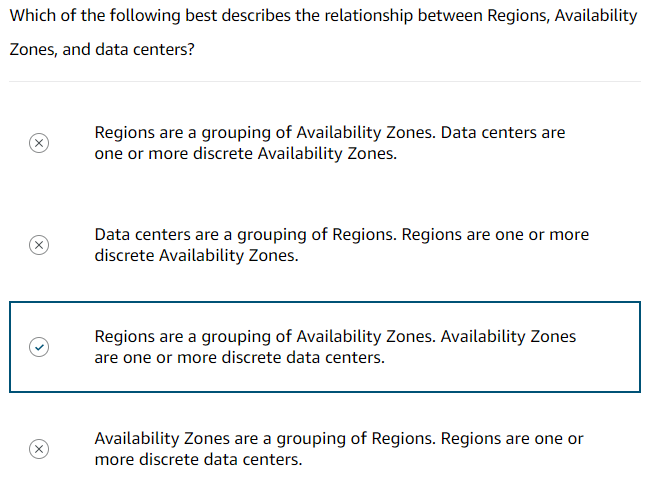
### **Resources**

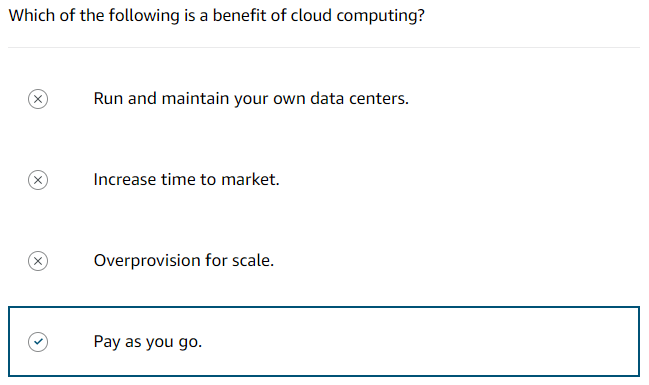
* AWS user guide: [What Is IAM?](https://docs.aws.amazon.com/en_us/IAM/latest/UserGuide/introduction.html)
* AWS user guide: [IAM Identities (User, User Groups, and Roles)](https://docs.aws.amazon.com/en_us/IAM/latest/UserGuide/id.html)
* AWS user guide: [Access Management for AWS Resources](https://docs.aws.amazon.com/en_us/IAM/latest/UserGuide/access.html)
* AWS user guide: [Security Best Practices in IAM](https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html)
* AWS blog: [How to Create and Manage Users within AWS IAM Identity Centre](https://aws.amazon.com/blogs/security/how-to-create-and-manage-users-within-aws-sso/)

## **1.7. Assessment**

,







# **Module - 2: AWS Compute**

## **2.1. Compute As a Service**

### **Servers**

The first building block that you need to host an application is a server. Servers can usually handle HTTP requests and send responses to clients following the client-server model. Although any API-based communication also falls under this model.

A client is a person or computer that sends a request. A server handling the requests is a computer, or collection of computers, connected to the internet serving websites to internet users. Servers power your application by providing CPU, memory, and networking capacity to process users’ requests and transform them into responses.

For context, common HTTP servers include the following:

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

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

### **Choosing the right compute option**

First, you need to know which compute service to use for each use case. At a fundamental level, three types of compute options are available: virtual machines (VMs), container services, and serverless.

A virtual machine will often be the easiest compute option to understand. This is because a virtual machine emulates a physical server and allows you to install an HTTP server to run your applications, for example. To run virtual machines, you install a hypervisor on a host machine. In its simplest form, a hypervisor is software or firmware that makes it possible to share physical hardware resources across one or more virtual machines. The hypervisor provisions the resources to create and run your VMs.  
In AWS, Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure and resizable compute capacity in the cloud. You can provision virtual servers called EC2 instances. Behind the scenes, AWS operates and manages the host machines and the hypervisor layer. AWS also installs the virtual machine operating system, called the guest operating system.  
Beneath the surface, some AWS compute services use Amazon EC2 or use virtualization concepts. You should understand this service before advancing to container services and serverless compute.

### **Resources**

* AWS whitepaper: [Compute Services](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/compute-services.html)
* AWS website: [Compute for Any Workload](https://aws.amazon.com/products/compute/)

## **2.2. Getting Started with Amazon EC2**

### **Amazon EC2**

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. With this service, you can provision virtual servers called EC2 instances.

With Amazon EC2, you can do the following:

* Provision and launch one or more EC2 instances in minutes.
* Stop or shut down EC2 instances when you finish running a workload.
* Pay by the hour or second for each instance type (minimum of 60 seconds).

You can create and manage EC2 instances through the AWS Management Console, AWS CLI, AWS SDKs, automation tools, and infrastructure orchestration services.

To create an EC2 instance, you must define the following:

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

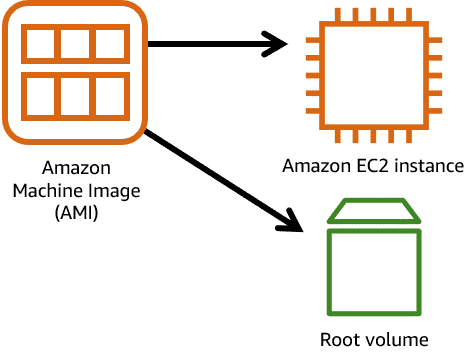
### **Amazon Machine Image**

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, the operating system installation is not your responsibility. Instead, it's built into the AMI that you choose.

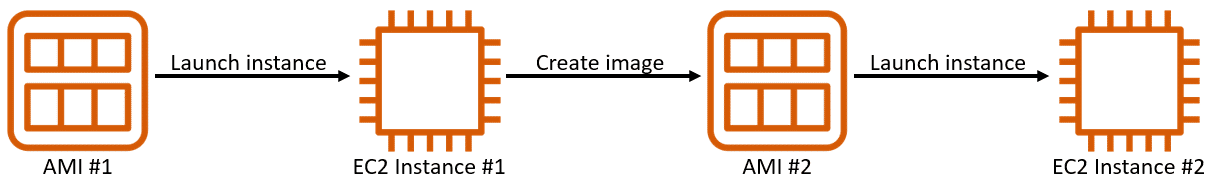
An AMI includes the operating system, storage mapping, architecture type, launch permissions, and any additional preinstalled software applications.

### **Relationship between AMIs and EC2 instances**

EC2 instances are live instantiations (or versions) of what is defined in an AMI, as a cake is a live instantiation of a cake recipe. If you are familiar with software development, you can also see this kind of relationship between a class and an object. In this case, the AMI is how you model and define your instance. The EC2 instance is the entity you interact with, where you can install your web server and serve your content to users.  
  
When you launch a new instance, AWS allocates a virtual machine that runs on a hypervisor. Then the AMI that you selected is copied to the root device volume, which contains the image that is used to boot the volume. In the end, you get a server that you can connect to and install packages and additional software on. In the example, you install a web server along with the properly configured source code of your employee directory application.



One advantage of using AMIs is that they are reusable. You might choose a Linux-based AMI and configure the HTTP server, application packages, and additional software that you need to run your application. If you want to create another EC2 instance with the same configurations, you could create and configure a new EC2 instance to match the first instance. Or you could create an AMI from your running instance and use the AMI to start a new instance. That way, your new instance would have the same configurations as your current instance because the configurations set in the AMIs are the same.



### **Finding AMIs**

There are multiple ways to select an AMI.

1. Quick Start AMIs
2. AWS Marketplace AMIs
3. My AMIs
4. Community AMIs
5. Custom image

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

An AMI includes the operating system, storage mapping, architecture type, launch permissions, and any additional preinstalled software applications.

### **Configuring EC2**

Now that you know how to select an operating system for your EC2 instance, you are ready to choose other configurations to create your EC2 instance, such as the instance type, network, and storage.

For an application like the employee directory application, you need instances with enough capacity to process customer requests. Your instance sizing will depend on both the demands of your application and the anticipated size of your user base.  
Forecasting server capacity for an on-premises application requires difficult decisions involving significant upfront capital spending. In contrast, changes to the allocation of your cloud-based services can be made with a simple API call. Because of the AWS pay-as-you-go model, you can match your infrastructure capacity to your application’s demand, instead of the other way around.

### **Amazon EC2 instance types**

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. Instance types consist of a prefix identifying the type of workloads they’re optimized for, followed by a size.

For example, the instance type c5n.xlarge can be broken down as follows:

* **First position –** Indicates the instance family. Here **‘c’** indicates compute optimized.
* **Second position –** Indicates the generation of the instance. Here **‘5’** is 5th generation.
* **Remaining letters before the period –** In this case, **n** indicates additional attributes, such as local NVMe storage.
* **After the period –** After the period, indicates the instance size. Here, it's **‘xlarge’**.

### **Instance families**

Each instance family is optimized to fit different use cases. The following table describes instance families and some typical workloads.

|  |  |  |
| --- | --- | --- |
| **Instance family** | **Description** | **Use Cases** |
| **General purpose** | These provide a balance of compute, memory, and networking resources, and can be used for a variety of workloads. | Ideal for applications that use these resources in equal proportions, such as web servers and code repositories. |
| **Compute optimized** | These are ideal for compute-bound applications that benefit from high-performance processors. | Well-suited for batch processing workloads, media transcoding, high performance web servers, high performance computing (HPC), scientific modelling, dedicated gaming servers and ad server engines, machine learning inference, and other compute intensive applications |
| **Memory optimized** | These are designed to deliver fast performance for workloads that process large datasets in memory. | Memory-intensive applications, such as high-performance databases, distributed web-scale in-memory caches, mid-size in-memory databases, real-time big-data analytics, and other enterprise applications |
| **Accelerated computing** | These 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 in software running on CPUs. | Machine learning, HPC, computational fluid dynamics, computational finance, seismic analysis, speech recognition, autonomous vehicles, and drug discovery |
| **Storage optimized** | These are designed for workloads that require high sequential read and write access to large datasets on local storage. They are optimized to deliver tens of thousands of low-latency random I/O operations per second (IOPS) to applications that replicate their data across different instances. | NoSQL databases (Cassandra, MongoDB and Redis), in-memory databases, scale-out transactional databases, data warehousing, Elasticsearch, and analytics |
| **HPC optimized** | HPCs are purpose built to offer the best price performance for running HPC workloads at scale on AWS. | Ideal for applications that benefit from high-performance processors, such as large, complex simulations and deep learning workloads |

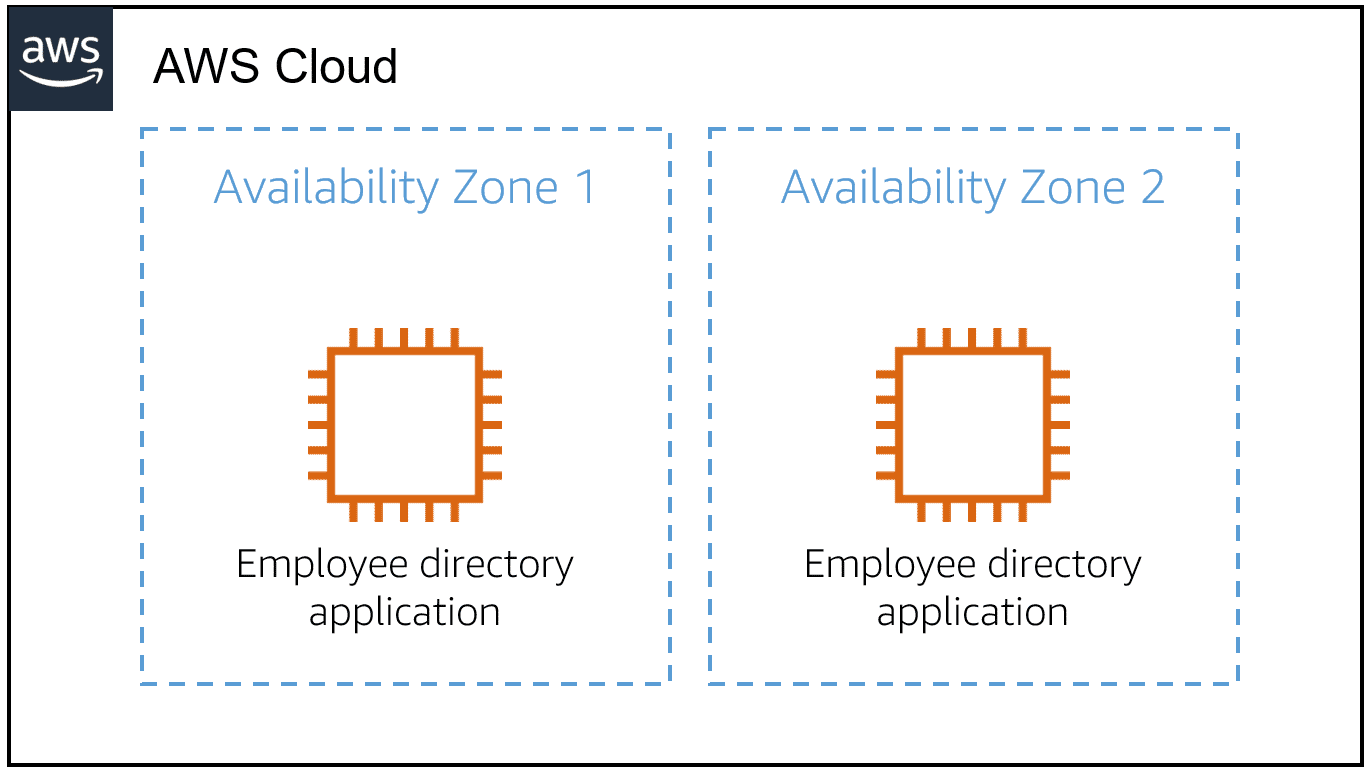
### **EC2 instance locations**

Unless otherwise specified, when you launch EC2 instances, they are placed in a default virtual private cloud (VPC). The default VPC is suitable for getting started quickly and launching public EC2 instances without having to create and configure your own VPC.

Any resource that 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 in it.  
You should change this default setting to choose your own custom VPCs and restrict access with additional routing and connectivity mechanisms.

### **Architecting for high availability**

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

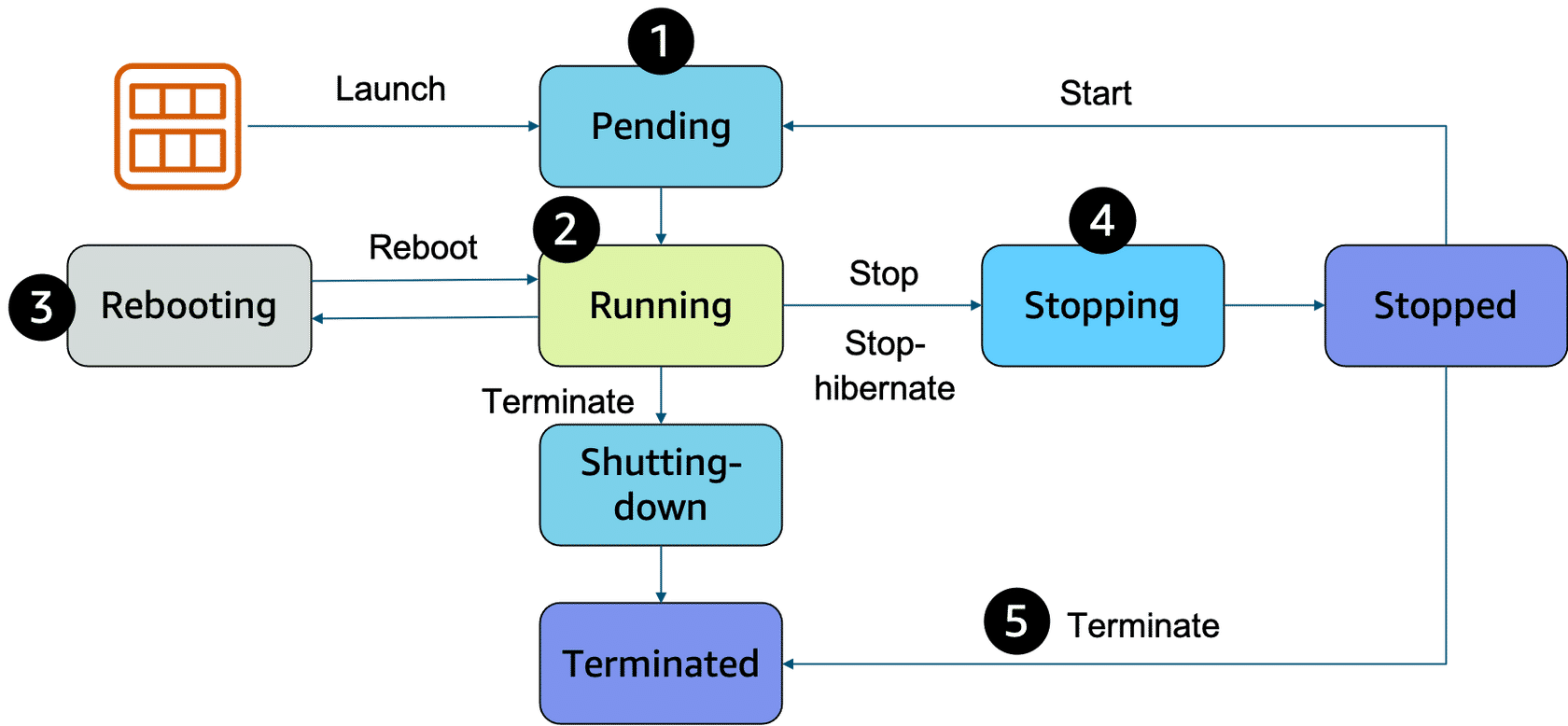


### **Resources**

* AWS website: [Amazon EC2](https://aws.amazon.com/ec2/)
* AWS user guide: [Amazon Machine Images (AMI)](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html)
* AWS user guide: [Create an Amazon EBS-Backed Linux AMI](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html)
* AWS user guide: [What Is EC2 Image Builder?](https://docs.aws.amazon.com/imagebuilder/latest/userguide/what-is-image-builder.html)
* AWS user guide: [Default VPCs](https://docs.aws.amazon.com/vpc/latest/userguide/default-vpc.html)
* AWS whitepaper: [Reliability Pillar - AWS Well-Architected Framework](https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html?ref=wellarchitected-wp)

## **2.3. Amazon EC2 Instance Lifecycle**

An EC2 instance transitions between different states from the moment you create it until its termination.



1. When you launch an instance, it enters the **pending**state. When an 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.
2. 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 can take other actions on the instance, such as reboot, terminate, stop, and stop-hibernate.
3. 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 keeps its public DNS name (IPv4) and private and public IPv4 addresses. An IPv6 address (if applicable) remains on the same host computer and maintains its public and private IP address, in addition to any data on its instance store volumes.
4. When you stop your instance, it enters the **stopping** and then **stopped** state. This is like when you shut down your laptop. You can stop and start an instance if it has an Amazon Elastic Block Store (Amazon EBS) volume as its root device. When you stop and start an instance, your instance can be placed on a new underlying physical server. Your instance retains its private IPv4 addresses and if your instance has an IPv6 address, it retains its IPv6 address. When you put the instance into stop-hibernate, the instance enters the stopped state, but saves the last information or content into memory, so that the start process is faster.
5. When you **terminate**an instance, the instance stores are erased, and you lose both the public IP address and private IP address of the machine. Termination of an instance means that you can no longer access the machine. As soon as the status of an instance changes to **shutting down**or **terminated,**you stop incurring charges for that instance.

###### **Difference between stop and stop-hibernate.**

When you stop an instance, it enters the stopping state until it reaches the stopped state. AWS does not charge usage or data transfer fees for your instance after you stop it. But storage for any Amazon EBS volumes is still charged. While your instance is in the stopped state, you can modify some attributes, like the instance type. When you stop your instance, the data from the instance memory (RAM) is lost.  
  
When you stop-hibernate an instance, Amazon EC2 signals the operating system to perform hibernation (suspend-to-disk), which saves the contents from the instance memory (RAM) to the EBS root volume. You can hibernate an instance only if hibernation is turned on and the instance meets the hibernation prerequisites.

### **Pricing**

One of the ways to reduce costs with Amazon EC2 is to choose the right pricing option for the way that your applications run. AWS offers a variety of pricing options to address different workload scenarios.

###### **On-Demand Instances**

With On-Demand Instances, you pay for compute capacity per hour or per second, depending on which instances that you run. There are no long-term commitments or upfront payments required. Billing begins whenever the instance is running, and billing stops when the instance is in a stopped or terminated state. You can increase or decrease your compute capacity to meet the demands of your application and only pay the specified hourly rates for the instance that you use.

On-Demand Instances are recommended for the following use cases:

* Users who prefer the low cost and flexibility of Amazon EC2 without upfront payment or long-term commitments.
* Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted.
* Applications being developed or tested on Amazon EC2 for the first time.

###### **Spot Instances**

For applications that have flexible start and end times. With Amazon EC2 Spot Instances, you can request spare Amazon EC2 computing capacity for up to 90 percent off the On-Demand price.

Spot Instances are recommended for the following use cases:

* Applications that have flexible start and end times.
* Applications that are only feasible at very low compute prices.
* Users with fault-tolerant or stateless workloads

With Spot Instances, you set a limit on how much you want to pay for the instance hour. This is compared against the current Spot price that AWS determines. Spot Instance prices adjust gradually based on long-term trends in supply and demand for Spot Instance capacity. If the amount that you pay is more than the current Spot price and there is capacity, you will receive an instance.

###### **Savings Plans**

Savings Plans are a flexible pricing model that offers low usage prices for a 1-year or 3-year term commitment to a consistent amount of usage. For workloads having predictable and consistent usage, Savings Plans can provide significant savings than On Demand instances.

Savings Plans are recommended for the following use cases:

* Workloads with a consistent and steady-state usage.
* Those using different instance types and compute solutions in different locations.
* Those who can make commitment to use Amazon EC2 over a 1-year or 3-year term.

###### **Reserved Instances**

For applications with steady state usage that might require reserved capacity. With this option, you save up to 75 percent compared to On-Demand Instance pricing. 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.

With Reserved Instances, you can choose the type that best fits your applications needs.

* **Standard Reserved Instances:** These provide the most significant discount (up to 72 percent off On-Demand pricing) and are best suited for steady-state usage.
* **Convertible Reserved Instances:** These provide a discount (up to 54 percent off On-Demand pricing) and the capability to change the attributes of the Reserved Instance if the exchange results in the creation of Reserved Instances of equal or greater value. Like Standard Reserved Instances, Convertible Reserved Instances are best suited for steady-state usage.
* **Scheduled Reserved Instances:** These are available to launch within the time windows that you reserve. You can match your capacity reservation to a predictable recurring schedule that only requires a fraction of a day, a week, or a month.

###### **Dedicated Hosts**

A Dedicated Host is a physical Amazon EC2 server that is dedicated for your use. Dedicated Hosts can help you reduce costs because you can use your existing server-bound software licenses, such as Windows Server, SQL Server, and Oracle licenses. And they can also help you meet compliance requirements. Amazon EC2 Dedicated Host is also integrated with AWS License Manager, a service that helps you manage your software licenses, including Microsoft Windows Server and Microsoft SQL Server licenses.

* Can be purchased on demand (hourly).
* Can be purchased as a Reservation for up to 70 percent off the On-Demand price.

### **Resources**

* AWS user guide: [Amazon EC2: Instance Lifecycle](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html)
* AWS user guide: [Hibernation Prerequisites](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/hibernating-prerequisites.html)
* AWS website: [Amazon EC2 Pricing](https://aws.amazon.com/ec2/pricing/)
* AWS website: [Amazon EC2 On-Demand Pricing](https://aws.amazon.com/ec2/pricing/on-demand/)
* AWS website: [Amazon EC2 Spot Instances Pricing](https://aws.amazon.com/ec2/spot/pricing/)
* AWS website: [Amazon EC2 Reserved Instances Pricing](https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/)

## **2.4. Container Services**

AWS offers a broad spectrum of compute offerings that give you the flexibility to choose the right tool for the job. The three main categories of compute are virtual machines (VMs), containers, and serverless. No one-size-fits-all compute service exists because it depends on your needs.  
The key is to understand what each option offers. Then you can build an appropriate cloud architecture for your use case.   
Containers can host a variety of different workloads, including web applications, lift and shift migrations, distributed applications, and streamlining of development, test, and production environments.

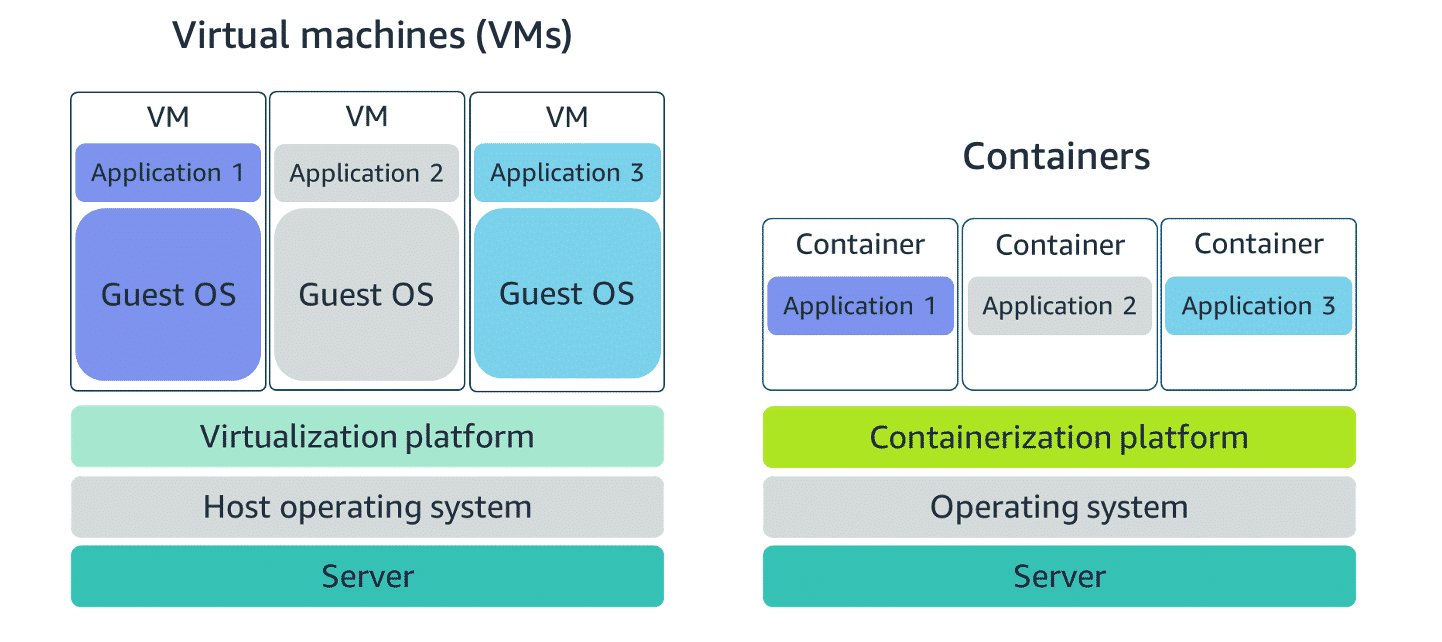
### **Containers**

A container is a standardized unit that packages your code and its dependencies. This package is designed to run reliably on any platform, because the container creates its own independent environment.

With containers, workloads can be carried from one place to another, such as from development to production or from on-premises environments to the cloud.

An example of a containerization platform is Docker. Docker is a popular container runtime that simplifies the management of the entire operating system stack required for container isolation, including networking and storage. Docker helps customers create, package, deploy, and run containers.

### **Difference between VMs and containers**



Compared to virtual machines (VMs), containers share the same operating system and kernel as the host that they are deployed on. But virtual machines contain their own operating system. Each virtual machine must maintain a copy of an operating system, which results in a degree of wasted resources. A container is more lightweight. Containers spin up quicker, almost instantly. This difference in startup time becomes instrumental when designing applications that must scale quickly during I/O bursts.

Containers can provide speed, but virtual machines offer the full strength of an operating system and more resources, like package installation, dedicated kernel, and more.

### **Orchestrating containers**

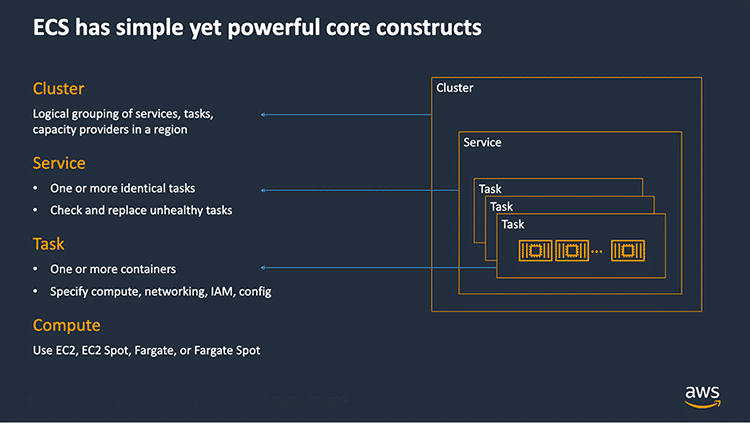
In AWS, containers can run on EC2 instances. For example, you might have a large instance and run a few containers on that instance. Although running one instance is uncomplicated to manage, it lacks high availability and scalability. Most companies and organizations run many containers on many EC2 instances across several Availability Zones.  
If you’re trying to manage your compute at a large scale, you should consider the following:

* 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

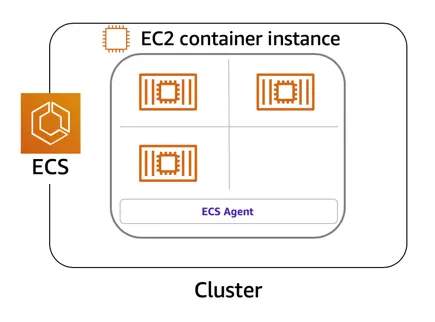
This coordination is handled by a container orchestration service. AWS offers two container orchestration services: Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS).

###### **Managing containers with Amazon ECS**

Amazon ECS is an end-to-end container orchestration service that helps you spin up new containers. With Amazon ECS, your containers are defined in a task definition that you use to run an individual task or a task within a service. You have the option to run your tasks and services on a serverless infrastructure that's managed by another AWS service called AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of EC2 instances that you manage.



If you choose to have more control by running and managing your containers on a cluster of Amazon EC2 instances, you will also need to install the Amazon ECS container agent on your EC2 instances. Note that an EC2 instance with the container agent installed is often called a container instance. This container agent is open source and responsible for communicating to the Amazon ECS service about cluster management details. You can run the agent on both Linux and Windows AMIs.



When the Amazon ECS container instances are up and running, you can perform actions that include, but are not limited to, the following:

* Launching and stopping containers
* Getting cluster state
* Scaling in and out
* Scheduling the placement of containers across your cluster
* Assigning permissions
* Meeting availability requirements

To prepare your application to run on Amazon ECS, you create a task definition. The task definition is a text file, in JSON format, that describes one or more containers. A task definition is like a blueprint that describes the resources that you need to run a container, such as CPU, memory, ports, images, storage, and networking information.  
Here is a simple task definition that you can use for your corporate directory application. In this example, this runs on the Nginx web server.

{

"family": "webserver",

"containerDefinitions": [ {

"name": "web",

"image": "nginx",

"memory": "100",

"cpu": "99"

} ],

"requiresCompatibilities": [ "FARGATE" ],

"networkMode": "awsvpc",

"memory": "512",

"cpu": "256"

}

###### **Using Kubernetes with Amazon EKS**

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services. By bringing software development and operations together by design, Kubernetes created a rapidly growing ecosystem that is very popular and well established in the market.

If you already use Kubernetes, you can use Amazon EKS to orchestrate the workloads in the AWS Cloud. Amazon EKS is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes.

Amazon EKS is conceptually like Amazon ECS, but with the following differences:

* In Amazon ECS, the machine that runs the containers is an EC2 instance that has an ECS agent installed and configured to run and manage your containers. This instance is called a container instance. In Amazon EKS, the machine that runs the containers is called a worker node or Kubernetes node.
* An ECS container is called a task. An EKS container is called a pod.
* Amazon ECS runs on AWS native technology. Amazon EKS runs on 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 could be the tool for you.

### **Resources**

* AWS website: [Containers on AWS](https://aws.amazon.com/containers/services/)
* External website: [Docker: Use Containers to Build, Share and Run Your Applications](https://www.docker.com/resources/what-container)
* AWS website: [Amazon Elastic Container Service (Amazon ECS)](https://aws.amazon.com/ecs/)
* External website: [Github: Amazon ECS Agent](https://github.com/aws/amazon-ecs-agent)
* AWS developer guide: [Amazon ECS Container Instances](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_instances.html)
* External website: [Coursera course: Building Containerized Applications on AWS](https://www.coursera.org/learn/containerized-apps-on-aws)
* AWS website: [Amazon Elastic Kubernetes Service (EKS)](https://aws.amazon.com/eks/)
* AWS user guide: [Amazon EKS User Guide](https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html)

## **2.5. Introduction to Serverless**

### **Removing the undifferentiated heavy lifting**

If you run your code on Amazon EC2, AWS is responsible for the physical hardware. You are still responsible for the logical controls, such as the guest operating system, security and patching, networking, security, and scaling.  
  
As covered in the previous Container Services lesson, you choose to have more control by running and managing your containers on Amazon ECS and Amazon EKS. By doing so, AWS is still responsible for more of the container management, such as deploying containers across EC2 instances and managing the container cluster. However, when running Amazon ECS and Amazon EKS on Amazon EC2, you are still responsible for maintaining the underlying EC2 instances.  
  
Is there a way to remove some of this undifferentiated heavy lifting? Yes! If you want to deploy your workloads and applications without having to manage any EC2 instances, you can do that on AWS with serverless compute.

### **Go Serverless**

With serverless compute, you can spend time on the things that differentiate your application, rather than spending time on ensuring availability, scaling, and managing servers.

Every definition of serverless mentions the following four aspects:

* There are no servers to provision or manage.
* It scales with usage.
* You never pay for idle resources.
* Availability and fault tolerance are built in.

AWS has developed serverless services for all three layers of the application stack.

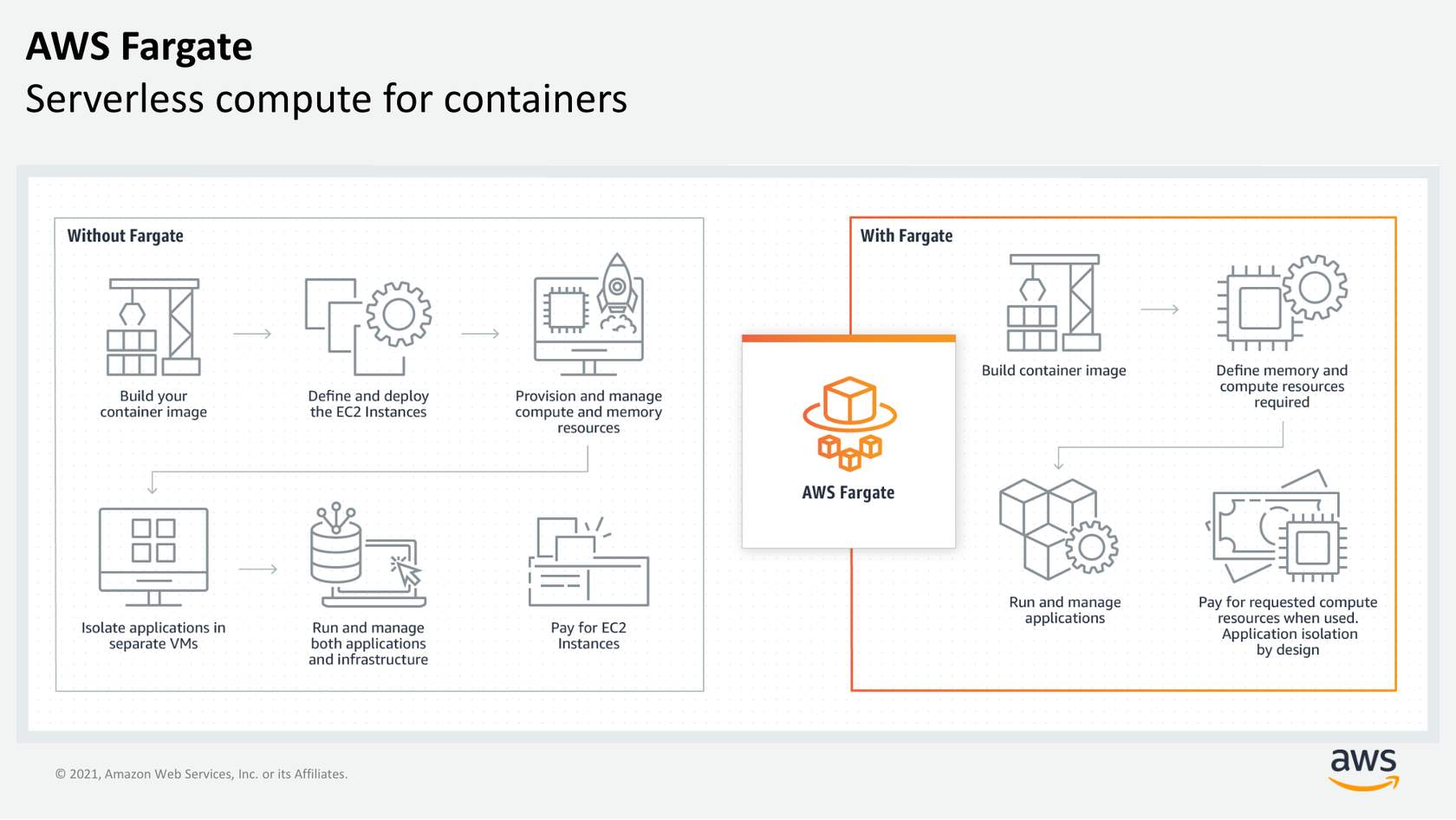
### **Resources**

* AWS website: [Serverless on AWS](https://aws.amazon.com/serverless/#:~:text=Serverless%20is%20the%20native%20architecture,services%20without%20thinking%20about%20servers.)
* AWS website: [AWS Serverless Resources](https://aws.amazon.com/serverless/resources/?serverless.sort-by=item.additionalFields.createdDate&serverless.sort-order=desc)

## **2.6. Serverless with AWS Fargate**

Fargate abstracts the EC2 instance so that you’re not required to manage the underlying compute infrastructure. However, with Fargate, you can use all the same Amazon ECS concepts, APIs, and AWS integrations. It natively integrates with IAM and Amazon Virtual Private Cloud (Amazon VPC). With native integration with Amazon VPC, you can launch Fargate containers inside your network and control connectivity to your applications.

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



### **Resources**

* AWS website: [AWS Fargate](https://aws.amazon.com/fargate/?c=ser&sec=srv)
* AWS website: [Getting Started with Serverless Computing](https://aws.amazon.com/serverless/resources/?serverless.sort-by=item.additionalFields.createdDate&serverless.sort-order=desc)
* External site: [Coursera course: Building Modern Python Applications on AWS](https://www.coursera.org/learn/building-modern-python-applications-on-aws)

## **2.7. Serverless with AWS Lambda**

### **Running code on AWS Lambda**

If you want to deploy your workloads and applications without having to manage any EC2 instances or containers, you can use Lambda.

With Lambda, you can run code without provisioning or managing servers. You can run code for virtually any type of application or backend service. This includes data processing, real-time stream processing, machine learning, WebSockets, IoT backends, mobile backends, and web applications like your employee directory application!

Lambda runs your code on a high availability compute infrastructure and requires no administration from the user. You upload your source code in one of the languages that Lambda supports, and Lambda takes care of everything required to run and scale your code with high availability. There are no servers to manage. You get continuous scaling with subsecond metering and consistent performance.

### **How Lambda works**

The Lambda function is the foundational principle of AWS Lambda. You have the option of configuring your Lambda functions using the Lambda console, Lambda API, AWS CloudFormation, or AWS Serverless Application Model (AWS SAM). You can invoke your function directly by using the Lambda API, or you can configure an AWS service or resource to invoke your function in response to an event.

###### **Function**

A function is a resource that you can invoke to run your code in Lambda. Lambda runs instances of your function to process events.

When you create the Lambda function, it can be authored in several ways:

* You can create the function from scratch.
* You can use a blueprint that AWS provides.
* You can select a container image to deploy for your function.
* You can browse the AWS Serverless Application Repository.

###### **Trigger**

Triggers describe when a Lambda function should run. A trigger integrates your Lambda function with other AWS services and event source mappings. So, you can run your Lambda function in response to certain API calls or by reading items from a stream or queue. This increases your ability to respond to events in your console without having to perform manual actions.

###### **Event**

An event is a JSON-formatted document that contains data for a Lambda function to process. The runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event.

###### **Application Environment**

An application environment provides a secure and isolated runtime environment for your Lambda function. An application environment manages the processes and resources that are required to run the function.

###### **Deployment package**

You deploy your Lambda function code using a deployment package.

Lambda supports two types of deployment packages:

* **A .zip file archive –** This contains your function code and its dependencies. Lambda provides the operating system and runtime for your function.
* **A container image –** This is compatible with the Open Container Initiative (OCI) specification. You add your function code and dependencies to the image. You must also include the operating system and a Lambda runtime.

###### **Runtime**

The runtime provides a language-specific environment that runs in an application environment. When you create your Lambda function, you specify the runtime that you want your code to run in. You can use built-in runtimes, such as Python, Node.js, Ruby, Go, Java, or .NET Core. Or you can implement your Lambda functions to run on a custom runtime.

###### **Lambda function handler**

The AWS Lambda function handler is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method. When the handler exits or returns a response, it becomes available to handle another event.

You can use the following general syntax when creating a function handler in Python.

**def handler\_name (event, context):**

**...**

**return some\_value**

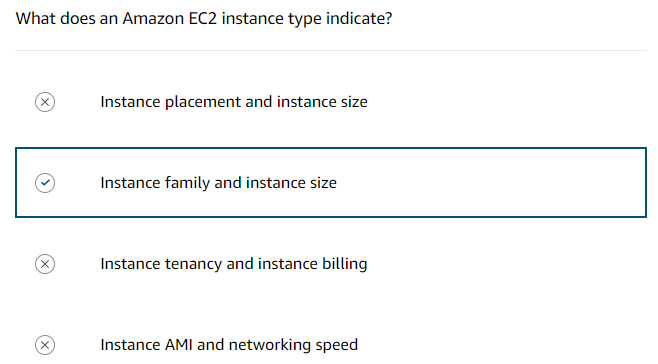
###### **Billing granularity**

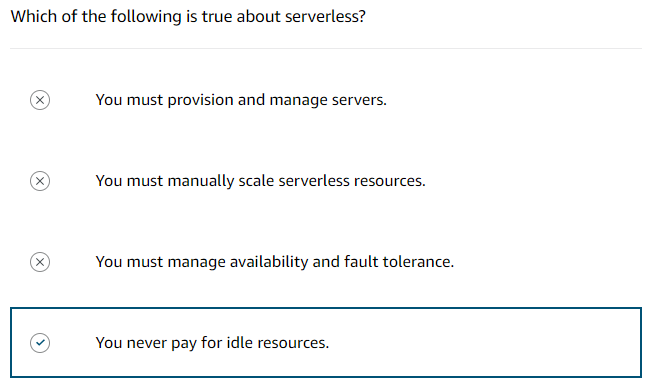
With Lambda, you can run code without provisioning or managing servers, and you pay only for what you use. You are charged for the number of times that your code is invoked (requests) and for the time that your code runs, rounded up to the nearest 1 millisecond (ms) of duration.

### **Resources**

* AWS website: [Building Applications with Serverless Architectures](https://aws.amazon.com/lambda/serverless-architectures-learn-more/)
* AWS blog: [Best Practices for Organizing Larger Serverless Applications](https://aws.amazon.com/blogs/compute/best-practices-for-organizing-larger-serverless-applications/)
* AWS developer guide: [Configuring AWS Lambda Functions](https://docs.aws.amazon.com/lambda/latest/dg/lambda-functions.html)
* AWS blog: [10 Things Serverless Architects Should Know](https://aws.amazon.com/blogs/architecture/ten-things-serverless-architects-should-know/)
* AWS workshop: [AWS Alien Attack: A Serverless Adventure](https://alienattack.workshop.aws/)
* AWS blog: [Resize Images with Amazon S3, AWS Lambda, and Amazon API Gateway](https://aws.amazon.com/blogs/compute/resize-images-on-the-fly-with-amazon-s3-aws-lambda-and-amazon-api-gateway/)
* AWS blog: [New for AWS Lambda – 1ms Billing Granularity Adds Cost Savings](https://aws.amazon.com/blogs/aws/new-for-aws-lambda-1ms-billing-granularity-adds-cost-savings/)

## **2.8. Assessment**





# **3. AWS Networking**

## **3.1. Introduction to Networking**

### **Networking defined**

Networking is how you connect computers around the world and allow them to communicate with one another. AWS has built a network of resources using data centres, Availability Zones, and Regions.

### **Networking basics**

One way to think about networking is to 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 cannot properly deliver the letter. In the digital world, computers handle the delivery of messages in a similar way as a letter. This is called routing.

### **IP addresses**

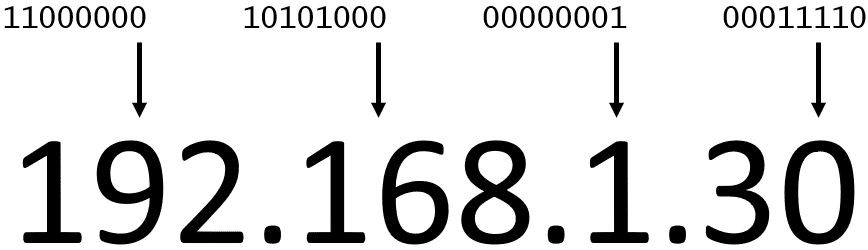
Just like each home has a mailing address, each computer has an IP address. IP address uses a combination of bits, 0s and 1s.  
  
Here is an example of a 32-bit address in binary format:

11000000 10101000 00000001 00011110 is a 32-bit address written in binary format.

It’s called 32-bit because you have 32 digits.

### **IPv4 notation**

Typically, you don’t see an IP address in its binary format. Instead, it’s converted into decimal format and noted as an IPv4 address.  
  
In the following diagram, 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 Classless Inter-Domain Routing (CIDR) comes in.

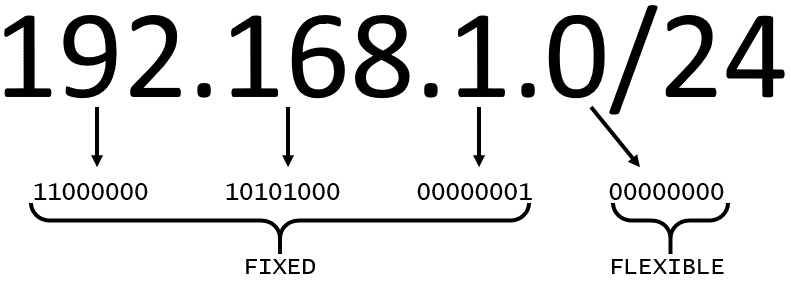
### **CIDR notation**

192.168.1.30 is a single IP address. If you want 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 to use CIDR notation. CIDR notation is a compressed way of representing a range of IP addresses. Specifying a range determines how many IP addresses are available to you.  
  
CIDR notation is shown here.



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 (the last 8 bits) 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 that 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 16 IP addresses. The largest IP range you can have been a /16, which provides 65,536 IP addresses.

### **Resources**

* External website: [Stanford: Introduction to Computer Networking](https://web.stanford.edu/class/cs101/network-1-introduction.html)
* AWS user guide: [How Amazon VPC Works](https://docs.aws.amazon.com/vpc/latest/userguide/how-it-works.html)
* External website: [Wikipedia: Classless Inter-Domain Routing](https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)
* External website: [CIDR.xyz: An interactive IP Address and CIDR Range Visualizer](https://cidr.xyz/)

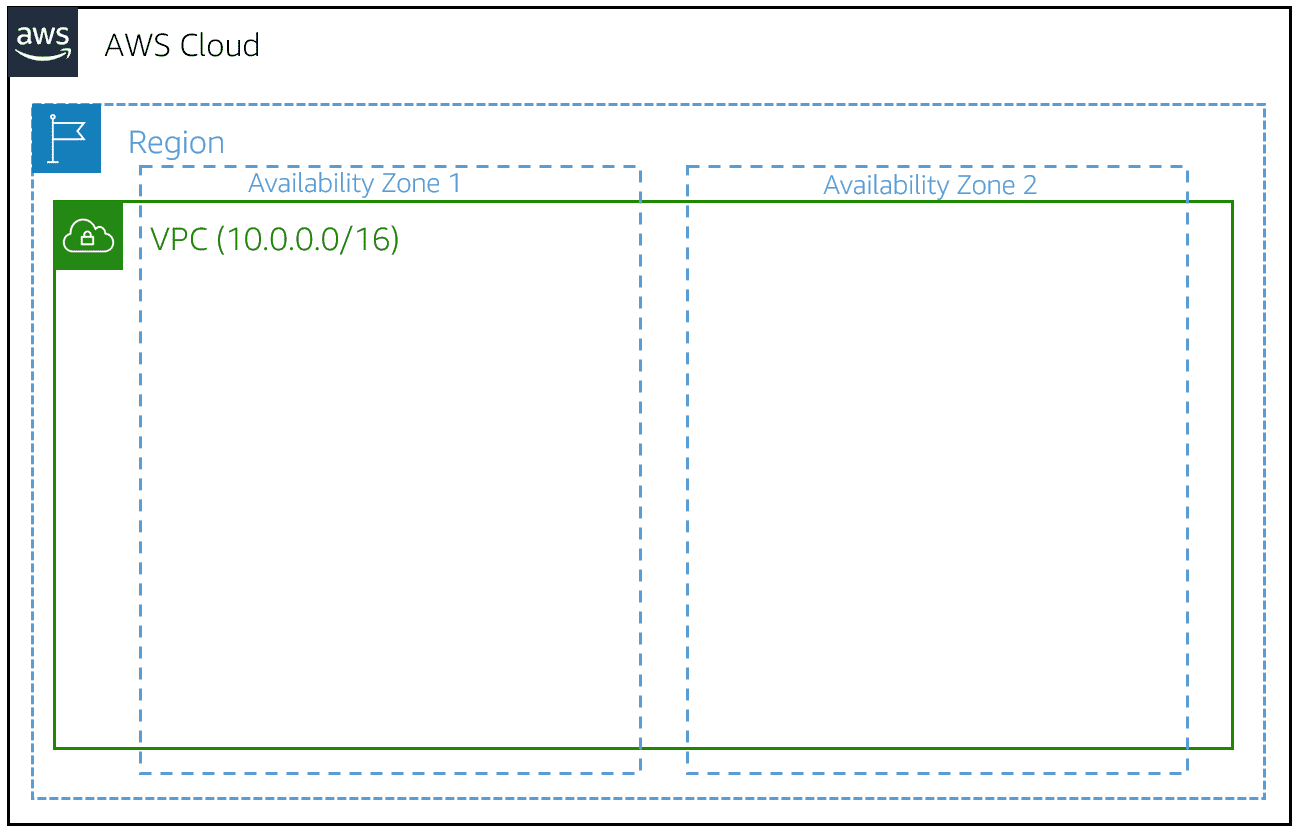
## **3.2. Amazon VPC**

A virtual private cloud (VPC) is an isolated network that you create in the AWS Cloud, similar to a traditional network in a data centre.

When you create an Amazon VPC, you must choose three main factors:

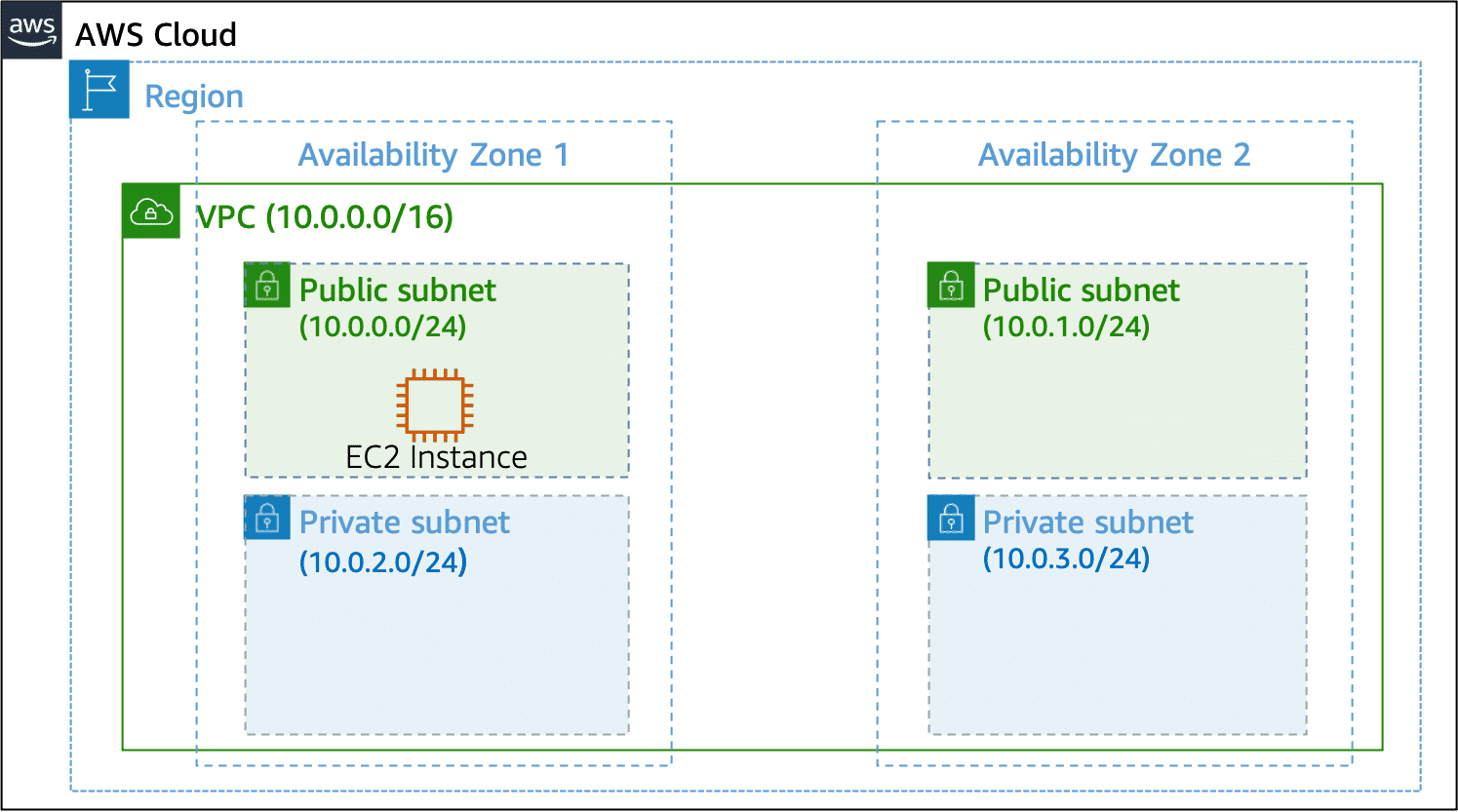
* **Name of the VPC**
* **Region where the VPC will live** – A VPC spans all the Availability Zones within the selected Region.
* **IP range for the VPC in CIDR notation** – This determines the size of your network. Each VPC can have up to five CIDRs: one primary and four secondaries for IPv4. Each of these ranges can be between /28 (in CIDR notation) and /16 in size.

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



### **Creating a subnet**

After you create your VPC, you must create subnets inside the network. Think of subnets as smaller networks inside your base network, or virtual local 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 to provide high availability and connectivity options for your resources. Use a public subnet for resources that must be connected to the internet and a private subnet for resources that won't be connected to the internet.



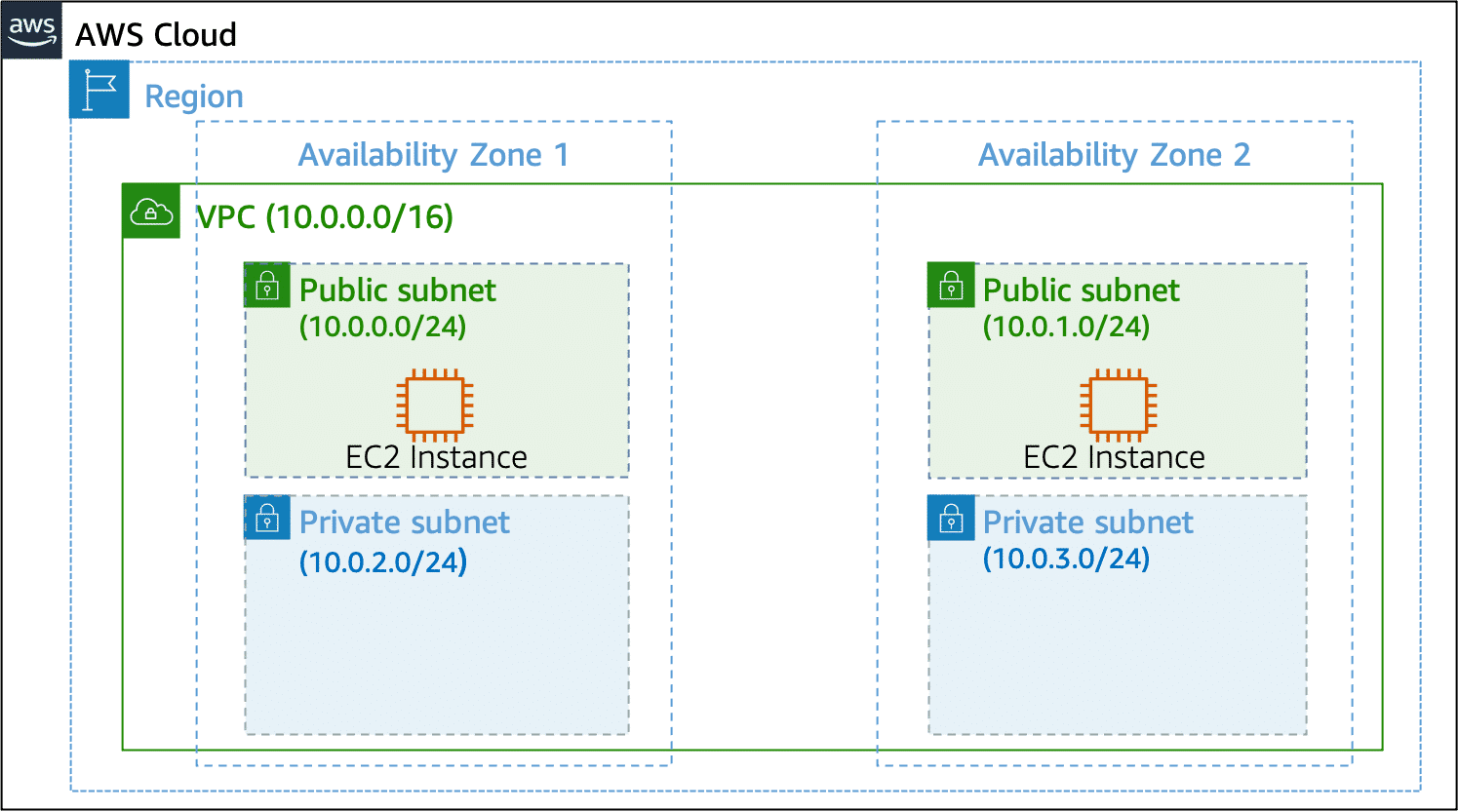
When you create a subnet, you must specify the following:

* **VPC** that you want your subnet to live in - Here: VPC (10.0.0.0/16)
* **Availability Zone** that you want your subnet to live in - Here: Availability Zone 1
* **IPv4 CIDR block for your subnet**, which must be a subset of the VPC CIDR block - Here: 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 that you choose.

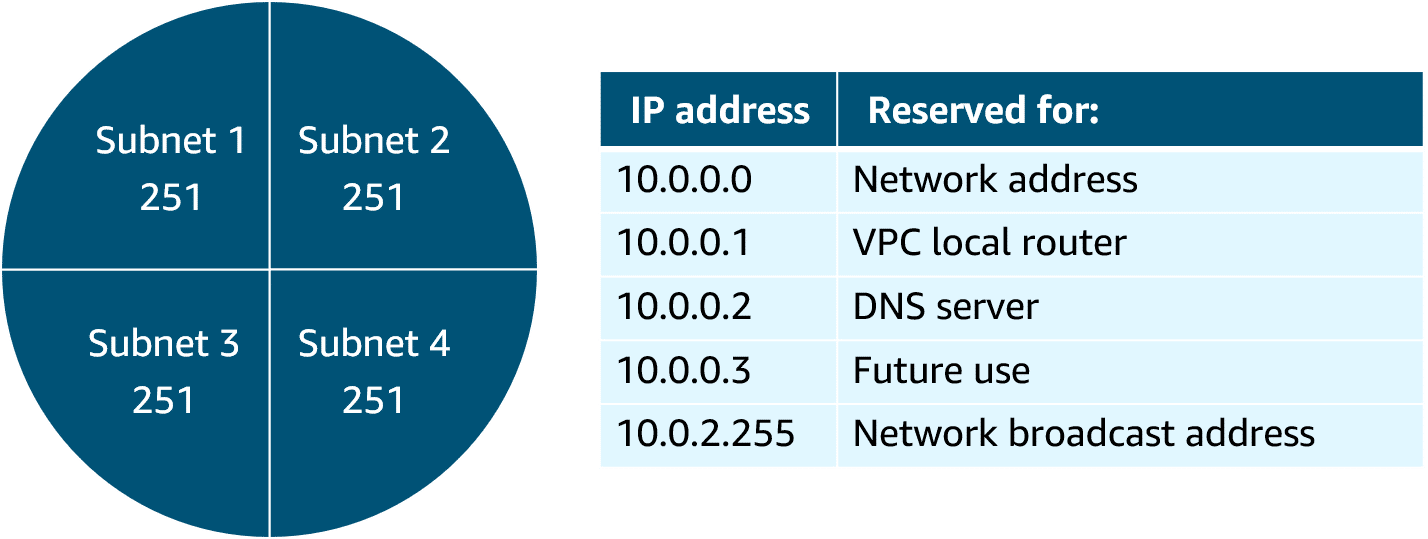
### **High availability with a VPC**

When you create your subnets, keep high availability in mind. To maintain redundancy and fault tolerance, create at least two subnets configured in two Availability Zones.



### **Reserved IPs**

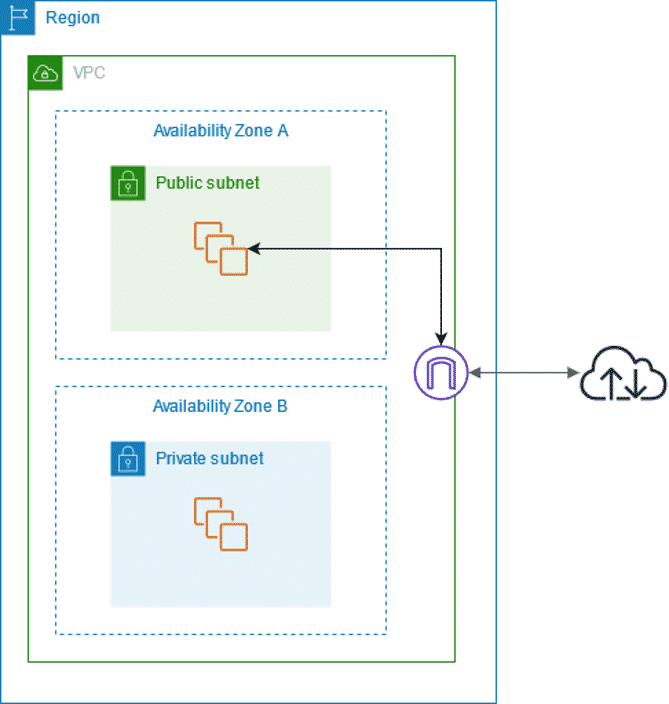
For AWS to configure your VPC appropriately, AWS reserves five IP addresses in each subnet. These IP addresses are used for routing, Domain Name System (DNS), and network management.  
For example, consider a VPC with the IP range 10.0.0.0/22. The VPC includes 1,024 total IP addresses. This is then divided 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.



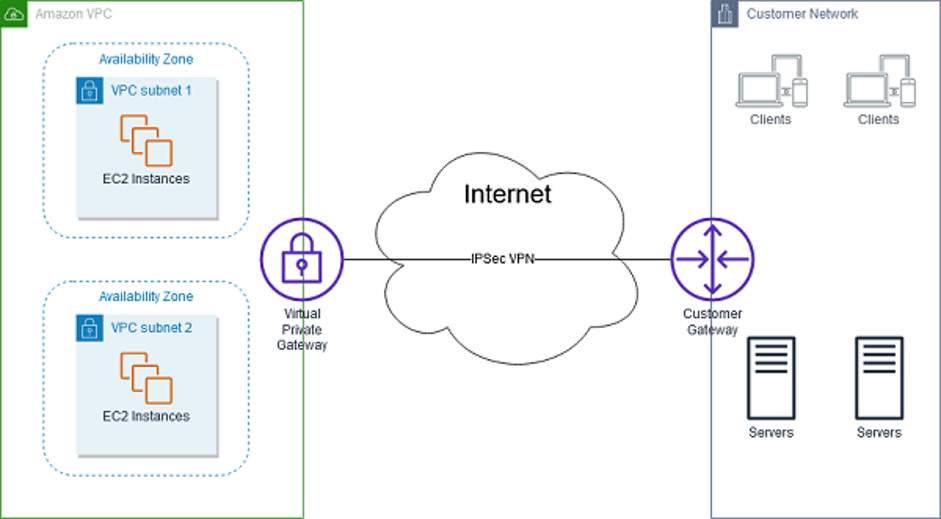
The five reserved IP addresses can impact how you design your network. A common starting place for those who are new to the cloud is to create a VPC with an IP range of /16 and create subnets with an IP range of /24. This provides a large amount of IP addresses to work with at both the VPC and subnet levels.

### **Gateways**

**Internet gateway**To activate internet connectivity for your VPC, you must create an internet gateway. Just as a modem connects your computer to the internet, the internet gateway connects your VPC to the internet. An internet gateway is highly available and scalable. After you create an internet gateway, you attach it to your VPC.

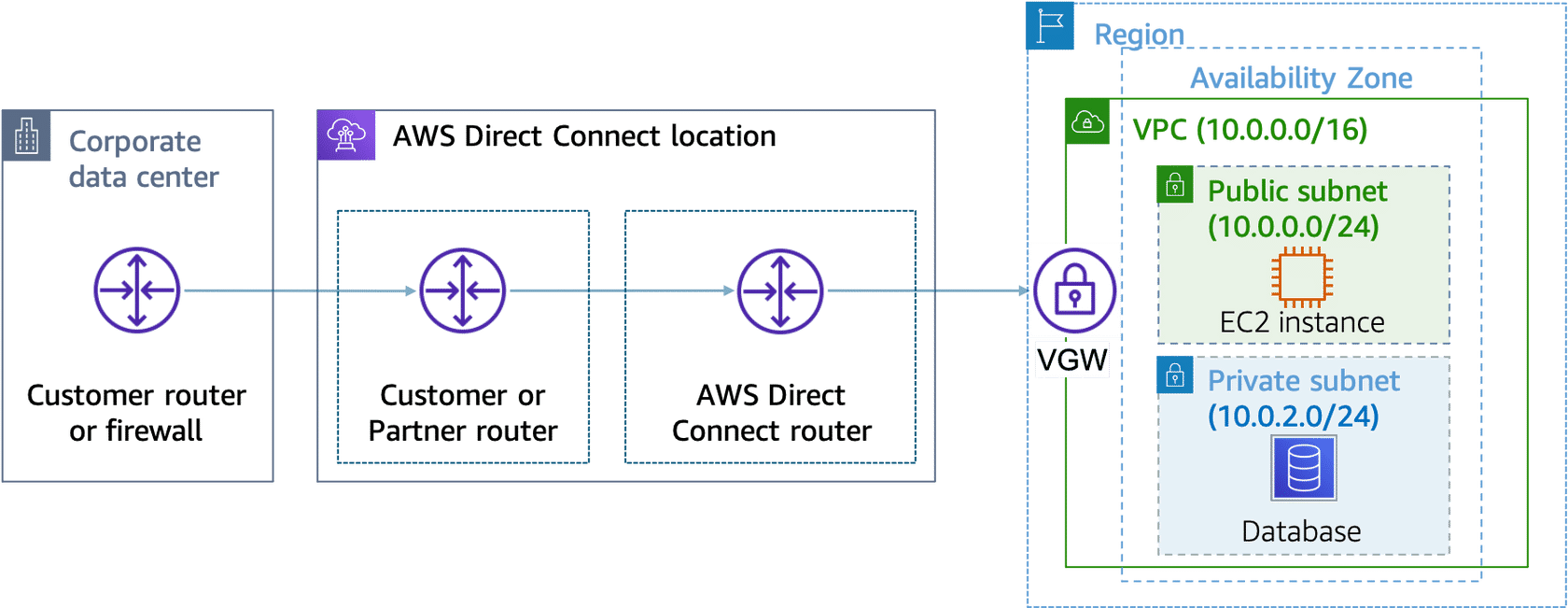


**Virtual private gateway**A virtual private gateway connects your VPC to another private network. When you create and attach a virtual private gateway to a VPC, the gateway acts as anchor on the AWS side of the connection. On the other side of the connection, you will need to connect a customer gateway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



### **AWS Direct Connect**

To establish a secure physical connection between your on-premises data centre and your Amazon VPC, you can use AWS Direct Connect. With AWS Direct Connect, your internal network is linked to an AWS Direct Connect location over a standard Ethernet Fiber-optic cable. This connection allows you to create virtual interfaces directly to public AWS services or to your VPC.



### **Resources**

* AWS user guide: [What Is Amazon VPC?](https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html)
* AWS user guide: [How Amazon VPC Works](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html)
* AWS user guide: [How AWS Site-to-Site VPN Works](https://docs.aws.amazon.com/vpn/latest/s2svpn/how_it_works.html)
* AWS user guide: [What Is AWS Direct Connect](https://docs.aws.amazon.com/directconnect/latest/UserGuide/Welcome.html)?

## **3.3. Amazon VPC Routing**

### **Main route table**

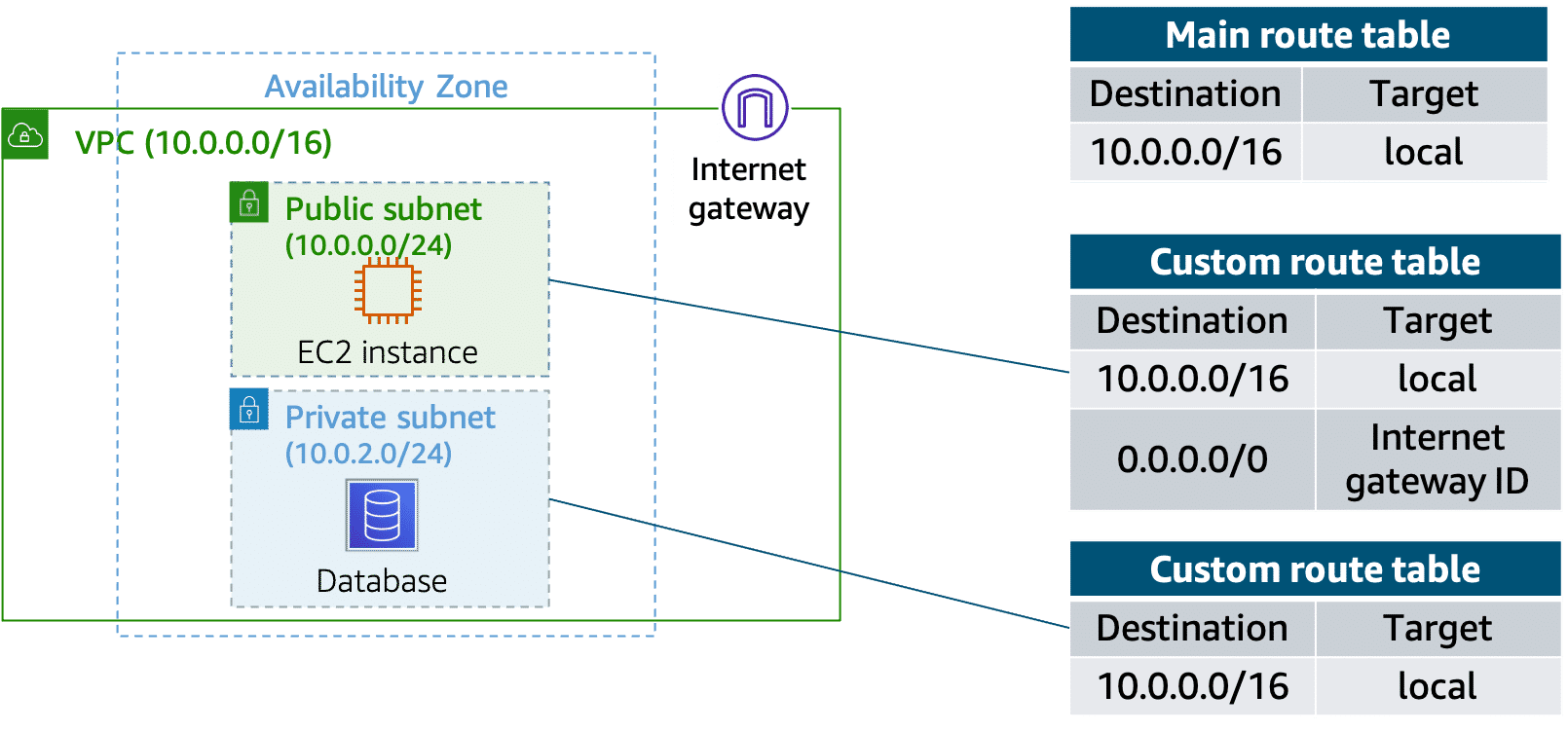
When you create a VPC, AWS creates a route table called the main route table. A route table contains a set of rules, called routes, that are used to determine where network traffic is directed. AWS assumes that when you create a new VPC with subnets, you want traffic to flow between them. Therefore,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

The following rules apply to the main route table:

* You cannot delete the main route table.
* You cannot set a gateway route table as the main route table.
* You can replace the main route table with a custom subnet route table.
* You can add, remove, and modify routes in the main route table.
* You can explicitly associate a subnet with the main route table, even if it's already implicitly associated.

### **Custom route tables**

The main route table is used implicitly by subnets that do not have an explicit route table association. However, you might want to provide different routes on a per-subnet basis for traffic to access resources outside of the VPC. For example, your application might consist of a frontend and a database. You can create separate subnets for the resources and provide different routes for each of them.  
If you associate a subnet with a custom route table, the subnet will use it instead of the main route table. Each custom route table that you create will have the local route already inside it, allowing communication to flow between all resources and subnets inside the VPC. You can protect your VPC by explicitly associating each new subnet with a custom route table and leaving the main route table in its original default state.

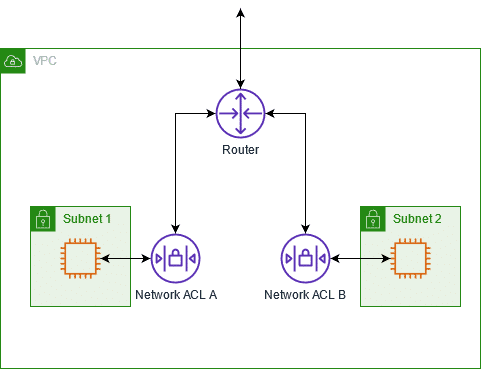


### **Resources**

* AWS user guide: [Configure Route Tables](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html)

## **3.4. Amazon VPC Security**

### **Secure subnets with network access control lists**



Think of a network access control list (network ACL) as a virtual firewall at the subnet level. A network ACL lets you 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 is an example of a default ACL for a VPC that supports IPv4.

##### **Default network ACL**

| **Inbound** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Allow or 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** | **Destination** | **Allow or 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 preceding table, allows all traffic in and out of the subnet. To allow data to flow freely to the subnet, this is a good starting place.  
However, you might 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.

##### **Custom network ACL**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inbound** | | | | | |
| **Rule #** | **Source IP** | **Protocol** | **Port** | **Allow or 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 or Deny** | **Comments** |
| 120 | 0.0.0.0/0 | TCP | 1025-65535 | ALLOW | Allows outbound responses to clients on the internet (serving people visiting the web servers in the subnet) |
| \* | 0.0.0.0/0 | All | All | DENY | Denies all outbound traffic not already handled by a preceding rule (not modifiable) |

Notice that in the custom network ACL in the preceding example, you allow inbound 443 and outbound range 1025–65535. That’s because HTTPS uses port 443 to initiate a connection and will respond to an ephemeral port. Network ACLs are considered stateless, so you need to include both the inbound and outbound ports used for the protocol. If you don’t include the outbound range, your server would respond but the traffic would never leave the subnet. Because network ACLs are configured by default to allow incoming and outgoing traffic, you don’t need to change their initial settings unless you need additional security layers.

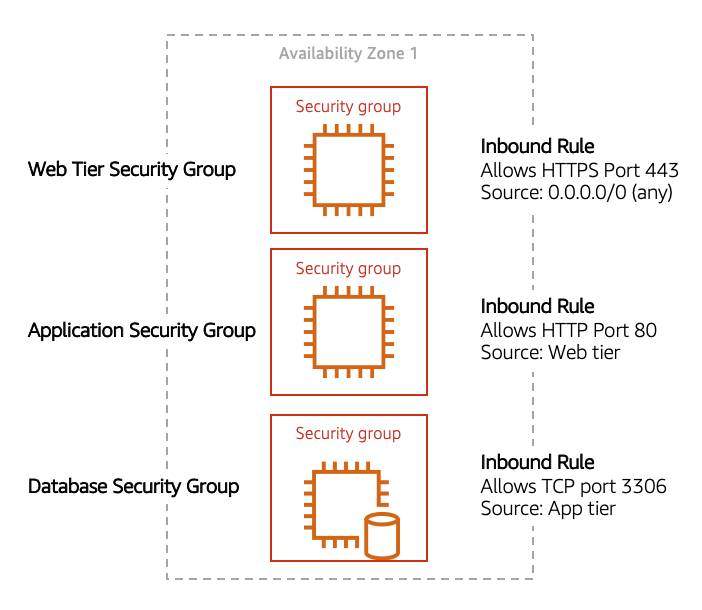
### **Secure EC2 instances with security groups**

The next layer of security is for your EC2 instances. Here, you can create a virtual firewall called a security group. The default configuration of a security group blocks all inbound traffic and allows all outbound traffic. To allow inbound traffic, you must create inbound rules. You might be wondering, “Wouldn’t this block all EC2 instances from receiving the response of any customer requests?” Well, security groups are stateful. Security groups will remember if a connection is originally initiated by the EC2 instance or from the outside, and temporarily allow traffic to respond without modifying the inbound rules.  
If you want your EC2 instance to accept traffic from the internet, you must open up inbound ports. If you have a web server, you might need to accept HTTP and HTTPS requests to allow that type of traffic into your security group. You can create an inbound rule that will allow port 80 (HTTP) and port 443 (HTTPS), as shown.

##### **Security group inbound rules**

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

You learned earlier that subnets can be used to segregate traffic between computers in your network. Security groups can be used in the same way. A common design pattern is to organize resources into different groups and create security groups for each to control network communication between them.

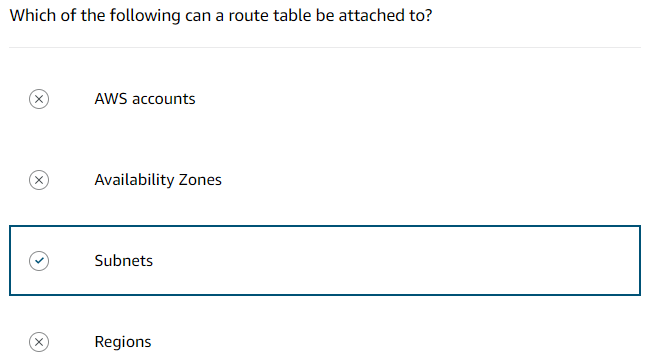


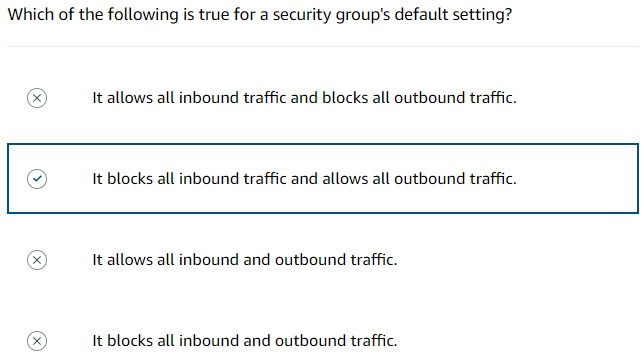
This example defines three tiers and isolates each tier with defined security group rules. In this case, internet traffic to the web tier is allowed over HTTPS. Web tier to application tier traffic is allowed over HTTP, and application tier to database tier traffic is allowed over MySQL. This is different from traditional on-premises environments, in which you isolate groups of resources with a VLAN configuration. In AWS, security groups allow you to achieve the same isolation without tying the security groups to your network.

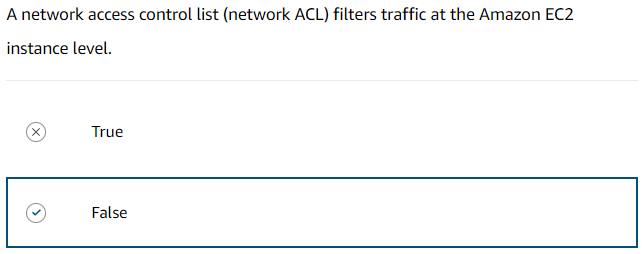
### **Resources**

* AWS user guide: [Configure Route Tables](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html)
* AWS user guide: [Example Routing Options](https://docs.aws.amazon.com/vpc/latest/userguide/route-table-options.html)
* AWS user guide: [Working with Route Tables](https://docs.aws.amazon.com/vpc/latest/userguide/WorkWithRouteTables.html)
* AWS user guide: [Control Traffic to Subnets Using Network ACLs](https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html)
* AWS user guide: [Control Traffic to Resources Using Security Groups](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html)

## **3.5. Assessment**



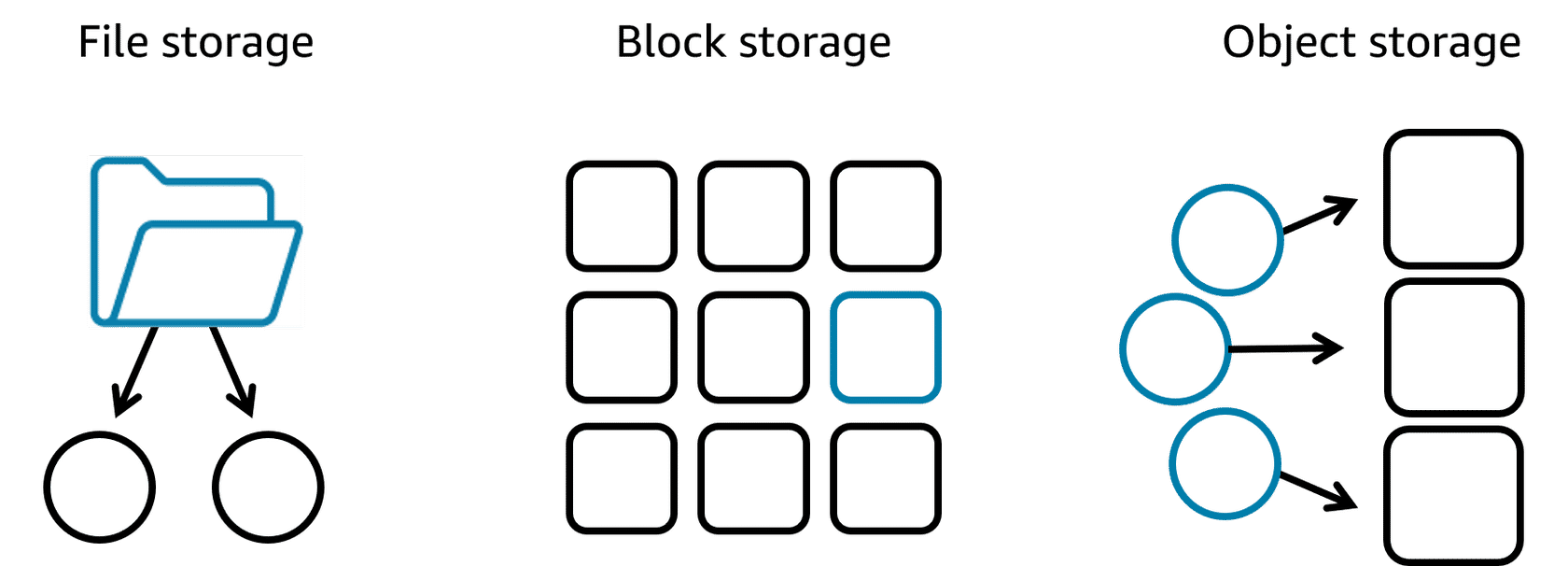




# **4. AWS Storage**

## **4.1. Storage Types**

AWS storage services are grouped into three categories: file storage, block storage, and object storage. In file storage, data is stored as files in a hierarchy. In block storage, data is stored in fixed-size blocks. And in object storage, data is stored as objects in buckets.



### **File storage**

Files are organized in a tree-like hierarchy that consist of folders and subfolders.

Each file has metadata such as file name, file size, and the date the file was created. The file also has a path, for example, computer/Application files/Cat photos/cats-03.png. When you need to retrieve a file, your system can use the path to find it in the file hierarchy.  
File storage is ideal when you require centralized access to files that must be easily shared and managed by multiple host computers. Typically, this storage is mounted onto multiple hosts, and requires file locking and integration with existing file system communication protocols.

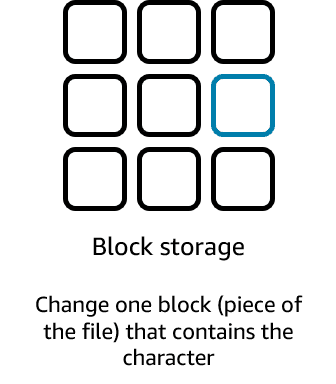
##### **Use cases for file storage**

* **Web serving -** Cloud file storage solutions follow common file-level protocols, file naming conventions, and permissions that developers are familiar with. Therefore, file storage can be integrated into web applications.
* **Analytics -** Many analytics workloads interact with data through a file interface and rely on features such as file lock or writing to portions of a file. Cloud-based file storage supports common file-level protocols and has the ability to scale capacity and performance. Therefore, file storage can be conveniently integrated into analytics workflows.
* **Media and entertainment -** Many businesses use a hybrid cloud deployment and need standardized access using file system protocols (NFS or SMB) or concurrent protocol access. Cloud file storage follows existing file system semantics. Therefore, storage of rich media content for processing and collaboration can be integrated for content production, digital supply chains, media streaming, broadcast playout, analytics, and archive.
* **Home directories -** Businesses wanting to take advantage of the scalability and cost benefits of the cloud are extending access to home directories for many of their users. Cloud file storage systems adhere to common file-level protocols and standard permissions models. Therefore, customers can lift and shift applications that need this capability to the cloud.

### **Block storage**

Block storage splits files into fixed-size chunks of data called blocks that have their own addresses. Each block is an individual piece of data storage. Because each block is addressable, blocks can be retrieved efficiently. Think of block storage as a more direct route to access the data.  
When data is requested, the 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. Besides the address, no additional metadata is associated with each block.

##### **Changing one character in a 1-GB file with block storage**



If you want to change one 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.

##### **Use cases for block storage**

Because block storage is optimized for low-latency operations, it is a preferred storage choice for high-performance enterprise workloads and transactional, mission-critical, and I/O-intensive applications.

* **Transactional workloads -** Organizations that process time-sensitive and mission-critical transactions store such workloads into a low-latency, high-capacity, and fault-tolerant database. Block storage allows developers to set up a robust, scalable, and highly efficient transactional database. Because each block is a self-contained unit, the database performs optimally, even when the stored data grows.
* **Containers -** Developers use block storage to store containerized applications on the cloud. Containers are software packages that contain the application and its resource files for deployment in any computing environment. Like containers, block storage is equally flexible, scalable, and efficient. With block storage, developers can migrate the containers seamlessly between servers, locations, and operating environments.
* **Virtual machines -** Block storage supports popular virtual machine (VM) hypervisors. Users can install the operating system, file system, and other computing resources on a block storage volume. They do so by formatting the block storage volume and turning it into a VM file system. So, they can readily increase or decrease the virtual drive size and transfer the virtualized storage from one host to another.

### **Object storage**

In object storage, files are stored as objects. Objects, much like files, are treated as a single, distinct unit of data when stored. Objects are stored in a bucket using a flat structure, meaning there are no folders, directories, or complex hierarchies. Each object contains a unique identifier. This identifier, along with any additional metadata, is bundled with the data and stored.

##### **Changing one character in a 1-GB file with object storage**

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

##### **Use cases for object storage**

With object storage, you can store almost any type of data, and there is no limit to the number of objects stored, which makes it readily scalable. Object storage is generally useful when storing large or unstructured data sets.

* **Data archiving -** Cloud object storage is excellent for long-term data retention. You can cost-effectively archive large amounts of rich media content and retain mandated regulatory data for extended periods of time. You can also use cloud object storage to replace on-premises tape and disk archive infrastructure. This storage solution provides enhanced data durability, immediate retrieval times, better security and compliance, and greater data accessibility.
* **Backup and recovery -** You can configure object storage systems to replicate content so that if a physical device fails, duplicate object storage devices become available. This ensures that your systems and applications continue to run without interruption. You can also replicate data across multiple data centres and geographical regions.
* **Rich media -** With object storage, you can accelerate applications and reduce the cost of storing rich media files such as videos, digital images, and music. By using storage classes and replication features, you can create cost-effective, globally replicated architecture to deliver media to distributed users.

### **Relating back to traditional storage systems**

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

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

Adding storage in a traditional data centre is a rigid process—the storage solutions must be purchased, installed, and configured. With cloud computing, the process is more flexible. You can create, delete, and modify storage solutions within a matter of minutes.

### **Resources**

* AWS website: [Cloud Computing Concepts Hub](https://aws.amazon.com/what-is/?faq-hub-cards.sort-by=item.additionalFields.sortDate&faq-hub-cards.sort-order=desc&awsf.tech-category=tech-category%23storage)

## **4.2. File Storage with Amazon EFS and Amazon FSx**

### **Amazon Elastic File System (Amazon EFS)**

Amazon Elastic File System (Amazon EFS) is a set-and-forget file system that automatically grows and shrinks as you add and remove files. There is no need for provisioning or managing storage capacity and performance. Amazon EFS can be used with AWS compute services and on-premises resources. You can connect tens, hundreds, and even thousands of compute instances to an Amazon EFS file system at the same time, and Amazon EFS can provide consistent performance to each compute instance.

With the Amazon EFS simple web interface, you can create and configure file systems quickly without any minimum fee or setup cost. You pay only for the storage used and you can choose from a range of storage classes designed to fit your use case.

|  |  |
| --- | --- |
| **Standard storage classes** | **One zone storage classes** |
| EFS Standard and EFS Standard-Infrequent Access (Standard-IA) offer Multi-AZ resilience and the highest levels of durability and availability. | EFS One Zone and EFS One Zone-Infrequent Access (EFS One Zone-IA) provide additional savings by saving your data in a single availability zone. |

### **Amazon FSx**

Amazon FSx is a fully managed service that offers reliability, security, scalability, and a broad set of capabilities that make it convenient and cost effective to launch, run, and scale high-performance file systems in the cloud. With Amazon FSx, you can choose between four widely used file systems: Lustre, NetApp ONTAP, OpenZFS, and Windows File Server. You can choose based on your familiarity with a file system or based on your workload requirements for feature sets, performance profiles, and data management capabilities.

##### **Amazon FSx for NetApp ONTAP**

It is a fully managed service. It combines the familiar features, performance, capabilities, and API operations of on-premises NetApp file systems with the agility, scalability, and simplicity of a fully managed AWS service. FSx for ONTAP can serve as a drop-in replacement for existing ONTAP deployments, giving customers the ability to launch and run ONTAP file systems in the cloud. FSx for ONTAP provides rich data management features and flexible shared file storage that are broadly accessible from Linux, Windows, and macOS compute instances running in AWS or on premises.

##### **Amazon FSx for OpenZFS**

Itis a fully managed file storage service that helps you to move data residing in on-premises ZFS or other Linux-based file servers to AWS without changing your application code or how you manage data. With FSx for OpenZFS, you no longer have to worry about setting up and provisioning files servers and storage volumes. You also don't have to deal with replicating data, installing, and patching file server software, detecting, and addressing hardware failures, or manually performing backups.

FSx for OpenZFS delivers leading performance for latency-sensitive and small-file workloads with popular NAS data management capabilities (snapshots, and cloning), at a lower price than commercially licensed alternatives.

##### **Amazon FSx for Windows File Server**

It provides fully managed, highly reliable and scalable Microsoft Windows file servers, backed by a fully native Windows file system. FSx for Windows File Server provides file storage that is accessible over the Service Message Block (SMB) protocol and has the ability to serve as a drop-in replacement for existing Windows file server deployments.

As a fully managed service, FSx for Windows File Server removes the administrative tasks of setting up and provisioning file servers and storage volumes and provides ease of use for customers building and running Windows applications.

##### **Amazon FSx for Lustre**

The open-source Lustre file system is designed for applications that require fast storage, where you want your storage to keep up with your compute. **Amazon FSx for Lustre** makes it convenient and cost effective to launch, run, and scale the popular high-performance file system. You can link FSx for Lustre file systems to data repositories on Amazon Simple Storage Service (Amazon S3) or to on-premises data stores.

FSx for Lustre delivers the highest levels of throughput (up to 1+ TB/s) and IOPS (millions). Customers can seamlessly integrate, access, and process their Amazon S3 datasets using the Lustre high-performance file system.

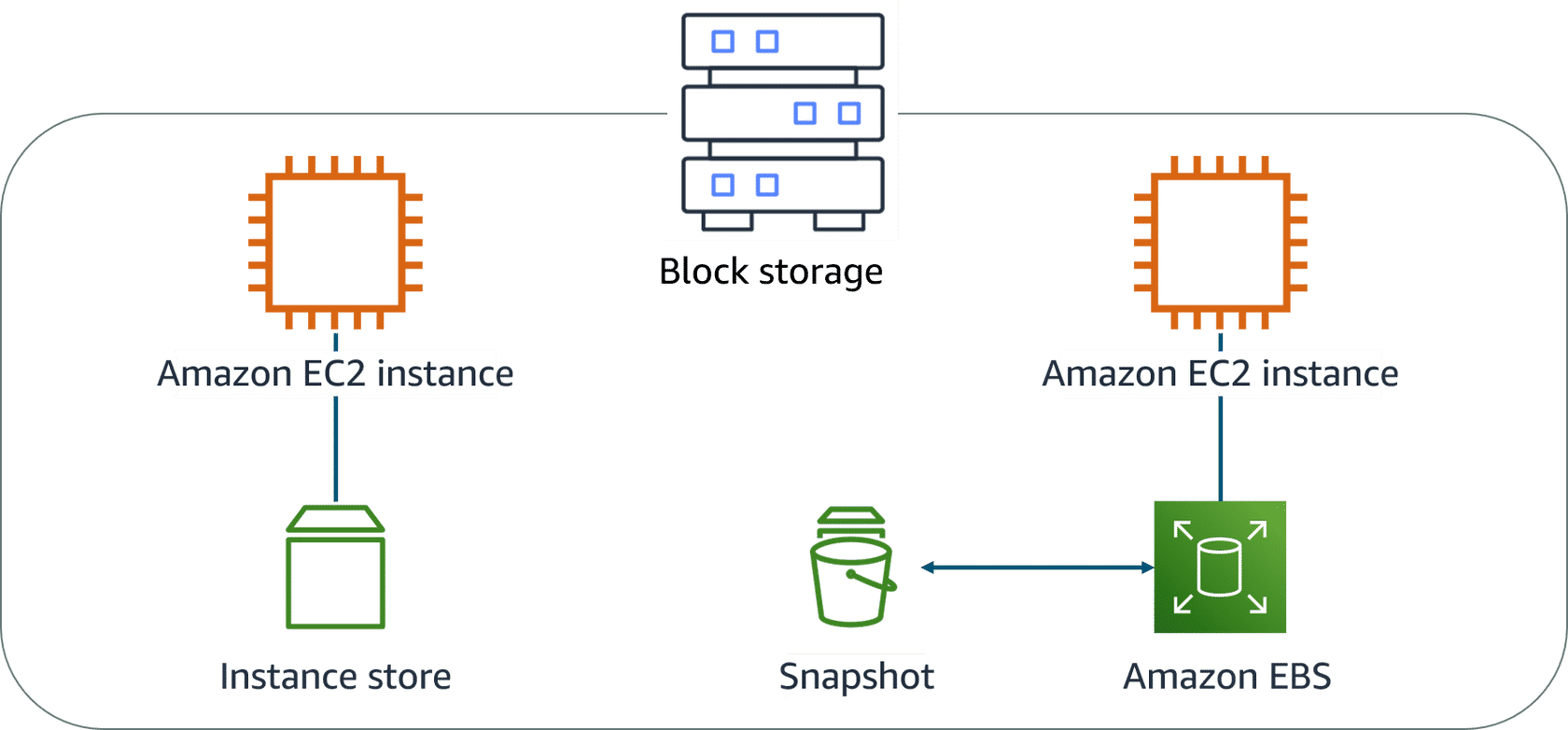
### **Resources**

* AWS website: [Amazon EFS](https://aws.amazon.com/efs/)
* AWS website: [Amazon FSx for NetApp ONTAP](https://aws.amazon.com/fsx/netapp-ontap/)
* AWS website: [Amazon FSx for OpenZFS](https://aws.amazon.com/fsx/openzfs/)
* AWS website: [Amazon FSx for Windows File Server](https://aws.amazon.com/fsx/windows/?nc=sn&loc=1)
* AWS website: [Amazon FSx for Lustre](https://aws.amazon.com/fsx/lustre/?nc=sn&loc=1)

## **4.3. Block Storage with Amazon EC2 Instance Store & Amazon EBS**

### **Amazon EC2 instance store**

Amazon Elastic Compute Cloud (Amazon EC2) instance store provides temporary block-level storage for an instance. This storage is located on disks that are physically attached to the host computer. This ties the lifecycle of the data to the lifecycle of the EC2 instance. If you delete the instance, the instance store is also deleted. Because of this, instance store is considered ephemeral storage.   
Instance store is ideal if you host applications that replicate data to other EC2 instances, such as Hadoop clusters. For these cluster-based workloads, having the speed of locally attached volumes and the resiliency of replicated data helps you achieve data distribution at high performance. It’s also ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content.



### **Amazon EBS**

Amazon Elastic Block Store (Amazon EBS) is block-level storage that you can attach to an Amazon EC2 instance. You can compare this to how you much attach an external drive to your laptop. This attachable storage is called an EBS volume.

* **Detachable:** 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.
* **Distinct:** The external drive is separate from the computer. That means that if an accident occurs and the computer goes down, you still have your data on your external drive. The same is true for EBS volumes.
* **Size-limited:**You’re limited to the size of the external drive because it has a fixed limit to how scalable it can be. This also relates to Amazon EBS, because a volume also has a max limitation of how much content you can store on it.
* **1-to-1 connection:**Most 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.

**AWS announced the Amazon EBS multi-attach feature that permits Provisioned IOPS SSD (io1 or io2) 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.**

### **Scaling Amazon EBS volumes**

You can scale EBS volumes in two ways.

1. **Increase the volume size** only if it doesn’t increase above the maximum size limit. Depending on the volume selected, Amazon EBS currently supports a maximum volume size of 64 tebibytes (TiB). For example, if you provision a 5-TiB io2 Block Express volume, you can choose to increase the size of your volume until you get to 64 TiB.
2. **Attach multiple volumes** to a single EC2 instance. Amazon 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 must retrieve data quickly and have data persist long term. Volumes are commonly used in the following scenarios.

* **Operating systems** – Boot and root volumes can be used to store an operating system. The root device for an instance launched from an Amazon Machine Image (AMI) is typically an EBS volume. These are commonly referred to as EBS-backed AMIs.
* **Databases** – As a storage layer for databases running on Amazon EC2 that will scale with your performance needs and provide consistent and low-latency performance.
* **Enterprise applications** – Amazon EBS provides high availability and high durability block storage to run business-critical applications.
* **Big data analytics engines** – Amazon EBS offers data persistence, dynamic performance adjustments, and the ability to detach and reattach volumes, so you can resize clusters for big data analytics.

### **EBS volume types**

EBS volumes are organized into two main categories: solid-state drives (SSDs) and hard-disk drives (HDDs). SSDs are used for transactional workloads with frequent read/write operations with small I/O size. HDDs are used for large streaming workloads that need high throughput performance. AWS offers two types of each.

##### **SSD volumes**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **General Purpose SSD volumes** | | **Provisioned IOPS SSD volumes** | | |
| **Volume type** | gp3 | gp2 | io2 Block Express | io2 | io1 |
| **Description** | Provides a balance of price and performance for a wide variety of transactional workloads | | Provides high-performance SSD designed for latency-sensitive transactional workloads | | |
| **Volume size** | 1 GiB–16 TiB | | 4 GiB–64 TiB | 4GiB–16 TiB | |
| **Max IOPS** **per volume** | 16,000 | | 256,000 | 64,000 | |
| **Max throughput per volume** | 1,000 MiB/s | 250 MiB/s | 4,000 MiB/s | 1,000 MiB/s | |
| **Amazon EBS Multi-attach** | Not supported | | Supported | | |

##### **HDD volumes**

|  |  |  |
| --- | --- | --- |
|  | **Throughput Optimized HDD volumes** | **Cold HDD volumes** |
| **Volume type** | st1 | sc1 |
| **Description** | A low-cost HDD designed for frequently accessed, throughput-intensive workloads | The lowest cost HDD designed for less frequently accessed workloads |
| **Volume size** | 125 GiB–16 TiB | |
| **Max IOPS per volume** | 500 | 250 |
| **Max throughput per volume** | 500 MiB/s | 250 MiB/s |
| **Amazon EBS Multi-attach** | Not supported | |

### **Amazon EBS benefits**

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

### **Amazon EBS snapshots**

Errors happen. One error is not backing up data and then inevitably losing it. To prevent this from happening to you, always back up your data, even in AWS. Because your EBS volumes consist of the data from your EC2 instance, you should make 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 S3.

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

### **Resources**

* AWS user guide: [Amazon EC2 Instance Store](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html)
* AWS user guide: [Amazon Elastic Block Store (Amazon EBS)](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html)
* AWS FAQs: [Amazon EBS](https://aws.amazon.com/ebs/faqs/)

## **4.4. Object Storage with Amazon S3**

### **Amazon S3**

Amazon Simple Storage Service (Amazon S3) is a standalone storage solution that isn’t tied to compute. With Amazon S3, you can retrieve your data from anywhere on the web. If you have used an online storage service to back up the data from your local machine, 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. Object storage stores data in a flat structure. An object is a file combined with metadata. You can store as many of these objects as you want. All the characteristics of object storage are also characteristics of Amazon S3.

### **Amazon S3 concepts**

In Amazon S3, you store your objects in containers called buckets. You can’t upload an object, not even a single photo, to Amazon S3 without creating a bucket first. When you store an object in a bucket, the combination of a bucket name, key, and version ID uniquely identifies the object.  When you create a bucket, you specify, at the very minimum, two details: the bucket name and the AWS Region that you want the bucket to reside in.

* **Bucket Name** - When you choose a bucket name, it must be unique across all AWS accounts. AWS stops you from choosing a bucket name that has already been chosen by someone else in another AWS account. After you choose a name, that name is yours and cannot be claimed by anyone else unless you delete the bucket, which then releases the name for others to use.
* **AWS region** - When you choose a Region for your bucket, all objects that you put inside the bucket will be redundantly stored across multiple devices, across multiple Availability Zones.

### **Amazon S3 bucket names**

Amazon S3 supports global buckets. Therefore, each bucket name must be unique across all AWS accounts in all AWS Regions within a partition. A partition is a grouping of regions, of which AWS currently has three: Standard Regions, China Regions, and AWS GovCloud (US). When naming a bucket, choose a name that is relevant to you or your business. For example, you should avoid using AWS or Amazon in your bucket name.

The following are some examples of the rules that apply for naming buckets in Amazon S3.

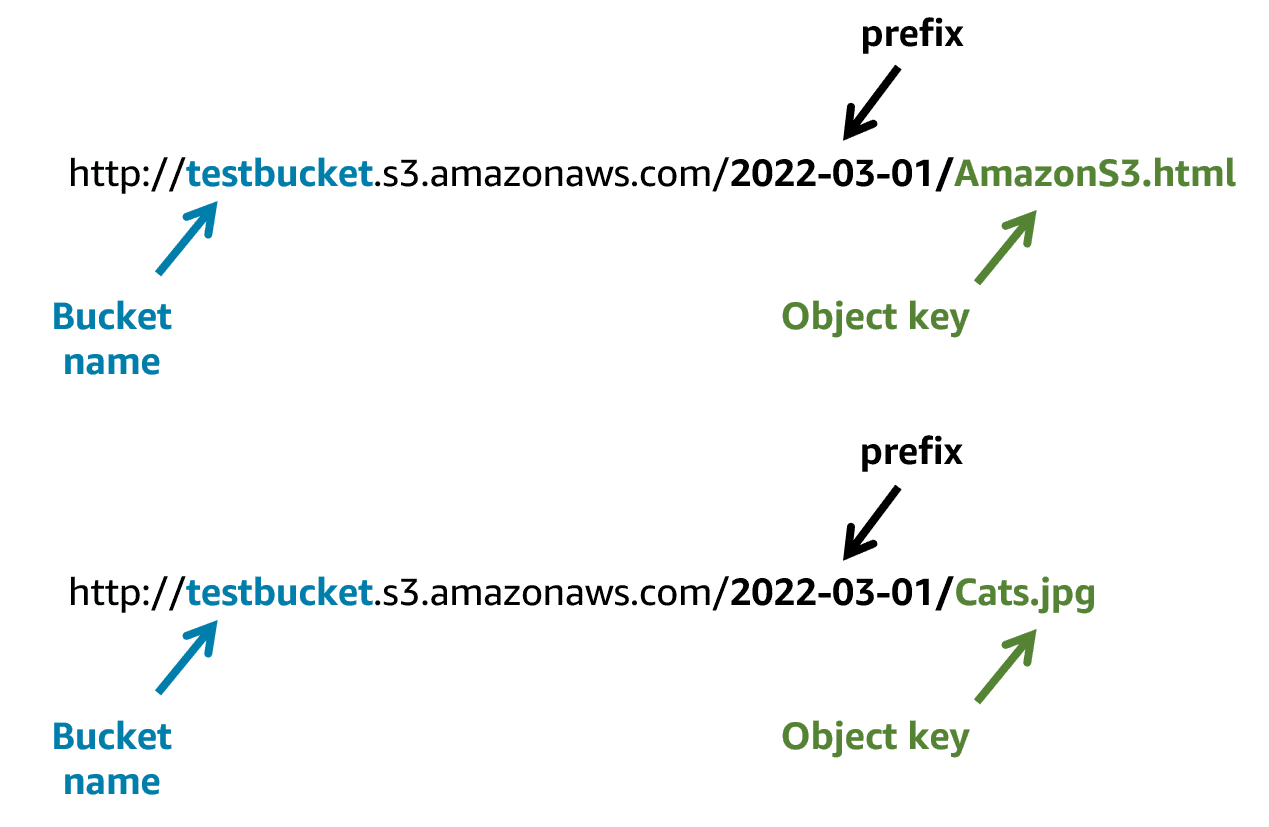
* Bucket names must be between 3 (min) and 63 (max) characters long.
* Bucket names can consist only lowercase letters, numbers, dots (.), and hyphens (-).
* Bucket names must begin and end with a letter or number.
* Buckets must not be formatted as an IP address.
* A bucket name cannot be used by another AWS account in the same partition until the bucket is deleted.

If your application automatically creates buckets, choose a bucket naming scheme that is unlikely to cause naming conflicts and will choose a different bucket name, should one not be available.

### **Object key names**

The object key (key name) uniquely identifies the object in an Amazon S3 bucket. When you create an object, you specify the key name. Amazon S3 model is a flat structure, meaning there is no hierarchy of sub-buckets or subfolders. However, the Amazon S3 console does support the concept of folders. By using key name prefixes and delimiters, you can imply a logical hierarchy.

For example, suppose your bucket called testbucket has two objects with the following object keys: 2022-03-01/AmazonS3.html and 2022-03-01/Cats.jpg. The console uses the key name prefix, 2022-03-01, and delimiter (/) to present a folder structure.



Amazon S3 supports buckets and objects, and there is no hierarchy. However, by using prefixes and delimiters in an object key name, the Amazon S3 console and the AWS SDKs are able to infer hierarchy and introduce the concept of folders.

### **Amazon S3 use cases**

**1. Backup and storage** – Amazon S3 is a natural place to back up files because it is highly redundant. AWS stores your EBS snapshots in Amazon S3 to take advantage of its high availability.

**2. Media hosting** – Because you can store unlimited objects, and each individual object can be up to 5 TB, Amazon S3 is an ideal location to host video, photo, and music uploads.

**3. Software delivery** – You can use Amazon S3 to host your software applications that customers can download.

**4. Data lakes** – AmazonS3 is an optimal foundation for a data lake because of its virtually unlimited scalability. You can increase storage from gigabytes to petabytes of content, paying only for what you use.

**5. Static websites** – You can configure your S3 bucket to host a static website of HTML, CSS, and client-side scripts.

**6. Static content** – Because of the limitless scaling, the support for large files, and the fact that you can access any object over the web at any time, Amazon S3 is the perfect place to store static content.

### **Security in Amazon S3**

Everything in Amazon S3 is private by default. This means that all Amazon S3 resources, such as buckets 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.  
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.

Amazon S3 provides several security management features: IAM policies, S3 bucket policies, and encryption to develop and implement your own security policies.

### **Amazon S3 and IAM policies**

When IAM policies are attached to your resources (buckets and objects) or IAM users, groups, and roles, the policies define which actions they can perform. Access policies that you attach to your resources are referred to as resource-based policiesand access policies attached to users in your account are called user policie**s**.

You should use IAM policies for private buckets in the following two scenarios:

* You have many buckets with different permission requirements. Instead of defining many different S3 bucket policies, you can use IAM policies.
* You want all policies to be in a centralized location. By using IAM policies, you can manage all policy information in one location.

### **Amazon S3 bucket policies**

S3 bucket policies are defined in a JSON format. Unlike IAM policies, which are attached to resources and users, S3 bucket policies can only be attached to S3 buckets. The policy that is placed on the bucket applies to every object in that bucket. S3 bucket policies specify what actions are allowed or denied on the bucket.

You should use S3 bucket policies in the following scenarios:

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

For examples of bucket policies, see the [Bucket Policy Examples](https://docs.aws.amazon.com/en_us/AmazonS3/latest/userguide/example-bucket-policies.html)

### **Amazon S3 encryption**

Amazon S3 reinforces encryption in transit (as it travels to and from Amazon S3) and at rest. To protect data, Amazon S3 automatically encrypts all objects on upload and applies server-side encryption with S3-managed keys as the base level of encryption for every bucket in S3.

### **Amazon S3 storage classes**

When you upload an object to Amazon S3 and you don’t specify the storage class, you upload it to the default storage class, often referred to as standard storage.   
Amazon S3 storage classes let you change your storage tier when your data characteristics change. For example, if you are accessing your old photos infrequently, you might want to change the storage class for the photos to save costs.

|  |  |
| --- | --- |
| **Storage Class** | **Description** |
| **S3 Standard** | This is considered general-purpose storage for cloud applications, dynamic websites, content distribution, mobile and gaming applications, and big data analytics. |
| **S3 Intelligent-Tiering** | This tier is useful if your data has unknown or changing access patters. S3 Intelligent-Tiering stores objects in three tiers: a frequent access tier, an infrequent access tier, and an archive instance 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. |
| **S3 Standard-Infrequent Access (S3 Standard-IA)** | This tier is for data that is accessed less frequently but requires rapid access when needed. S3 Standard-IA offers the high durability, high throughput, and low latency of S3 Standard, with a low per-GB storage price and per-GB retrieval fee. This storage tier is ideal if you want to store long-term backups, disaster recovery files, and so on. |
| **S3 One Zone-Infrequent Access (S3 One Zone-IA)** | Unlike other S3 storage classes that store data in a minimum of three Availability Zones, S3 One Zone-IA stores data in a single Availability Zone, which makes it less expensive than S3 Standard-IA. S3 One Zone-IA is ideal for customers who want a lower-cost option for infrequently accessed data, but do not require the availability and resilience of S3 Standard or S3 Standard-IA. It's a good choice for storing secondary backup copies of on-premises data or easily re-creatable data. |
| **S3 Glacier Instant Retrieval** | Use S3 Glacier Instant Retrieval for archiving data that is rarely accessed and requires millisecond retrieval. Data stored in this storage class offers a cost savings of up to 68 percent compared to the S3 Standard-IA storage class, with the same latency and throughput performance. |
| **S3 Glacier Flexible Retrieval** | S3 Glacier Flexible Retrieval offers low-cost storage for archived data that is accessed 1–2 times per year. With S3 Glacier Flexible Retrieval, your data can be accessed in as little as 1–5 minutes using an expedited retrieval. You can also request free bulk retrievals in up to 5–12 hours. It is an ideal solution for backup, disaster recovery, offsite data storage needs, and for when some data occasionally must be retrieved in minutes. |
| **S3 Glacier Deep Archive** | S3 Glacier Deep Archive is the lowest-cost Amazon S3 storage class. It supports long-term retention and digital preservation for data that might be accessed once or twice a year. Data stored in this class has a default retrieval time of 12 hours. It is designed for customers that retain data sets for 7–10 years or longer, to meet regulatory compliance requirements. Examples include those in highly regulated industries, such as the financial services, healthcare, and public sectors. |
| **S3 on Outposts** | Amazon S3 on Outposts delivers object storage to your on-premises AWS Outposts environment using S3 API's and features. For workloads that require satisfying local data residency requirements or need to keep data close to on premises applications for performance reasons, the S3 Outposts storage class is the ideal option. |

### **Amazon S3 versioning**

Amazon S3 identifies objects in part by using the object name. Without Amazon S3 versioning, every time you upload an object, it will overwrite the original object.  
This can be an issue for several reasons, including the following:

* **Common names:** You or someone else who has access to the bucket might not have intended to overwrite an object; but once it's overwritten, the original object can't be accessed.
* **Version preservation:** You might want to preserve different versions of an object. Without versioning, if you wanted to create a new version of object, you would need to upload the object and choose a different name for it. Having several objects all with slight differences in naming variations can cause confusion and clutter in S3 buckets.

To counteract these issues, you can use Amazon S3 versioning. Versioning keeps multiple versions of a single object in the same bucket. This preserves old versions of an object without using different names, which helps with object recovery from accidental deletions, accidental overwrites, or application failures.

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

The following are examples:

* Deleting an object does not remove the object permanently. Instead, Amazon S3 puts a marker on the object that shows that you tried to delete it. If you want to restore the object, you can remove the marker and the object is reinstated.
* 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.

##### **Versioning states**

Buckets can be in one of three states. The versioning state applies to all objects in the bucket. Storage costs are incurred for all objects in your bucket, including all versions. To reduce your Amazon S3 bill, you might want to delete previous versions of your objects when they are no longer needed.

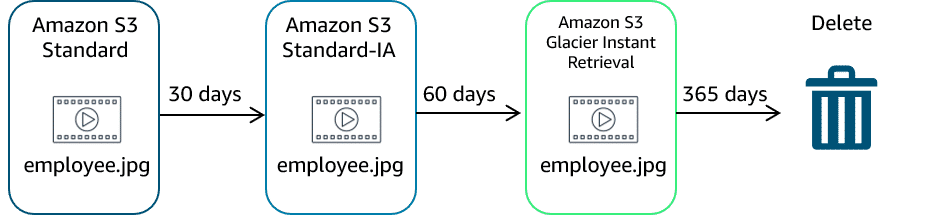
1. **Un-versioned (default)** – No new and existing objects in the bucket have a version.
2. **Versioning-enabled** – Versioning is enabled for all objects in the bucket. After you version-enable a bucket, it can never return to an un-versioned state. However, you can suspend versioning on that bucket.
3. **Versioning-suspended** – Versioning is suspended for new objects. All new objects in the bucket will not have a version. However, all existing objects keep their object versions.

### **Managing your storage lifecycle**

If you keep manually changing your objects, such as your employee photos, from storage tier to storage tier, you might want to automate the process by configuring their Amazon S3 lifecycle. When you define a lifecycle configuration for an object or group of objects, you can choose to automate between two types of actions: transition and expiration.

* **Transition actions** define when objects should transition to another storage class.
* **Expiration actions** define when objects expire and should be permanently deleted.

For example, you might transition objects to S3 Standard-IA storage class 30 days after you create them. Or you might archive objects to the S3 Glacier Deep Archive storage class 1 year after creating them.



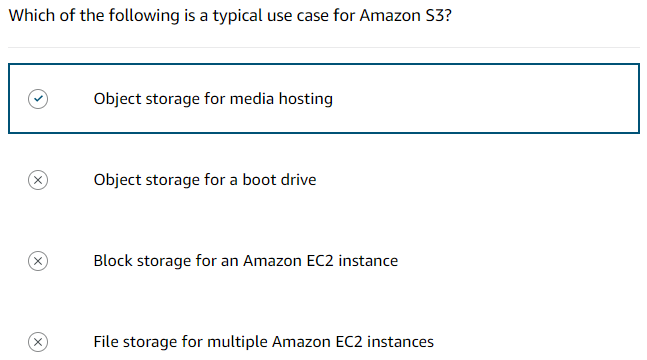
The following use cases are good candidates for the use of lifecycle configuration rules:

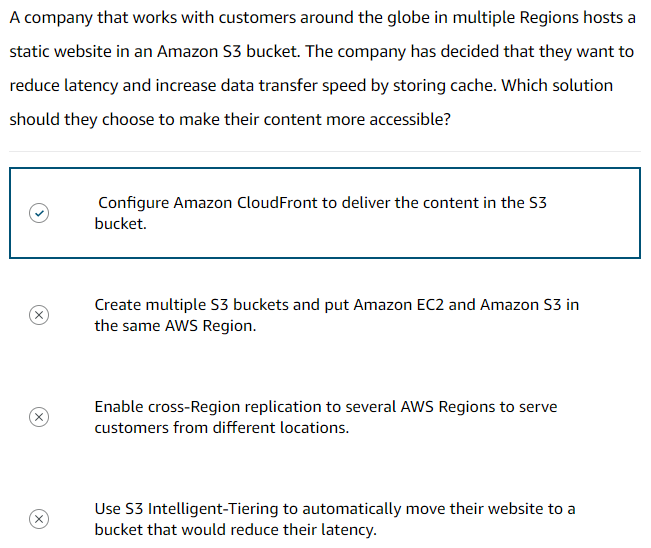
* **Periodic logs:**If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you might want to delete them.
* **Data that changes in access frequency:** Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them. But your organization or regulations might require you to archive them for a specific period. After that, you can delete them.

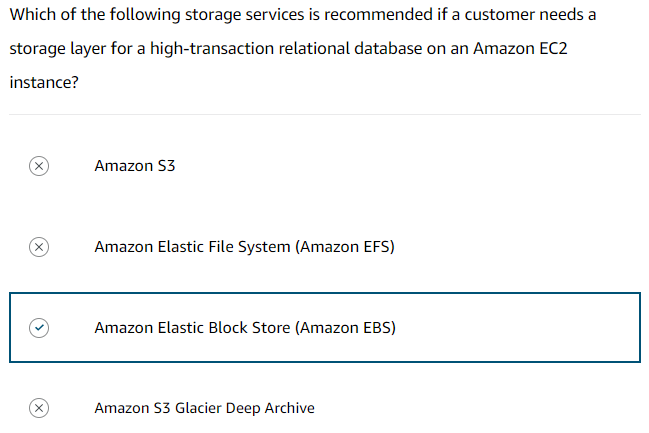
### **Resources**

* AWS website: [Amazon S3](https://aws.amazon.com/s3/)
* AWS website: [Amazon S3 Storage Classes](https://aws.amazon.com/s3/storage-classes/)
* AWS user guide: [Using Versioning in S3 Buckets](https://docs.aws.amazon.com/AmazonS3/latest/userguide/Versioning.html)
* AWS user guide: [Bucket Naming Rules](https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html)
* AWS user guide: [Bucket Policy Examples](https://docs.aws.amazon.com/en_us/AmazonS3/latest/userguide/example-bucket-policies.html)

## **4.5. Assessment**







# **5. Databases on AWS**

## **5.1. Introduction to Databases on AWS**

### **History behind enterprise databases**

Choosing a database used to be a straightforward decision. Customers had only a few options to choose from. Typically, they would consider a few vendors and then, inevitably, choose one for all their applications. Businesses often selected a database technology before they fully understood their use case. Since the 1970s, the database type most commonly selected by businesses was a relational database.

### **Relational databases**

A relational database organizes data into tables. Data in one table can link 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 an entry.   
  
The tables, rows, columns, and relationships between them is called a logical schema. With relational databases, a schema is fixed. After the database is operational, it becomes difficult to change the schema. Because of this, most of the data modelling is done up front before the database is active.

### **Relational database management system**

With a relational database management system (RDBMS), you can create, update, and administer a relational database.

Some common examples of RDBMSs include the following:

* MySQL
* PostgreSQL
* Oracle
* Microsoft SQL Server
* Amazon Aurora

You communicate with an RDBMS by using structured query language (SQL) queries. However, the power of SQL queries is in creating more complex queries that pull data from several tables to identify patterns and answers to business problems.

### **Relational database benefits**

* **Complex SQL** - With SQL, you can **join**multiple tables so you can 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** - Because relational databases have been a popular choice since the 1970s, technical professionals often have familiarity and experience with them.
* **Accuracy** - Relational databases ensure that your data has high integrity and adheres to the atomicity, consistency, isolation, and durability (ACID) principle.

### **Relational database use cases**

Much of the world runs on relational databases. In fact, they’re at the core of many mission-critical applications, some of which you might use in your day-to-day life.

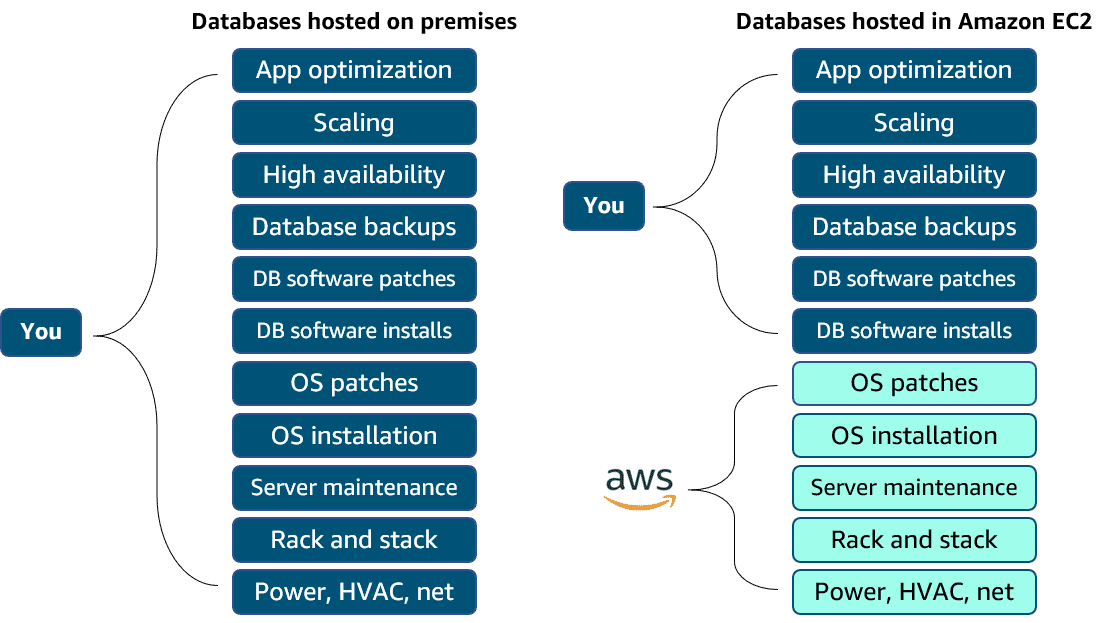
* **Applications that have a fixed schema** – **These are applications that have a fixed schema** and don't change often. An example is a lift-and-shift application that lifts an app from on-premises and shifts it to the cloud, with little or no modifications.
* **Applications that need persistent storage** – **These are applications that need persistent storage**and follow 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 trade your on-premises database for a relational database on AWS, you first need to select how you want to run it—managed or unmanaged. Managed services and unmanaged services are handled similar to the shared responsibility model. The shared responsibility model distinguishes between AWS security responsibilities and the customer’s security responsibilities. Likewise, managed compared to unmanaged can be understood as a trade-off between convenience and control.

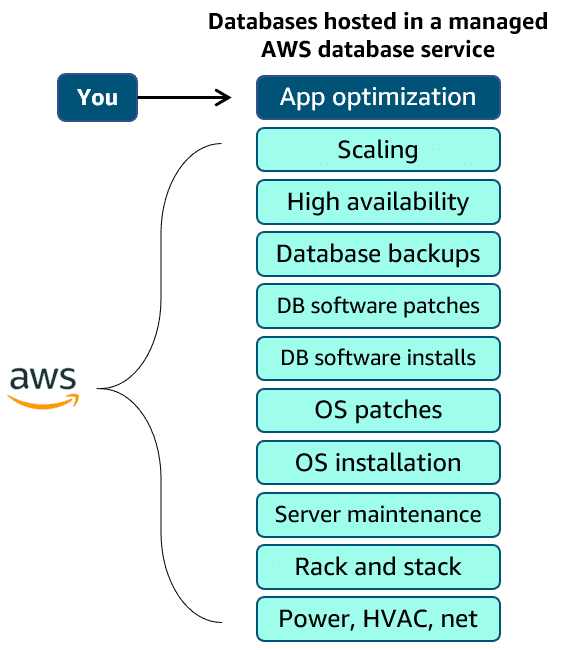
##### **Unmanaged databases**

If you operate a relational database on premises, you are responsible for all aspects of operation. You are responsible for absolutely everything, which means you have control over absolutely everything. Now, suppose you want to shift some of the work to AWS by running your relational database on Amazon Elastic Compute Cloud (Amazon EC2). If you host a database on Amazon EC2, AWS implements and maintains the physical infrastructure and hardware and installs the EC2 instance operating system (OS). However, you are still responsible for managing the EC2 instance, managing the database on that host, optimizing queries, and managing customer data. This is called an unmanaged database option. In this option, AWS is responsible for and has control over the hardware and underlying infrastructure. You are responsible for and have control over management of the host and database.



##### **Managed databases**

To shift 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, in this model, you’re still responsible for database tuning, query optimization, and ensuring that your customer data is secure. This option provides the ultimate convenience, but the least amount of control compared to the two previous options.



### **Resources**

* AWS website: [What Is a Relational Database?](https://aws.amazon.com/relational-database/)
* AWS website: [AWS Cloud Databases](https://aws.amazon.com/products/databases/)

## **5.2. Amazon RDS**

### **Amazon RDS overview**

Amazon RDS is a managed database service customers can use to create and manage relational databases in the cloud without the operational burden of traditional database management. With Amazon RDS, you can offload some of the unrelated work of creating and managing a database. You can focus on the tasks that differentiate your application, instead of focusing on infrastructure-related tasks, like provisioning, patching, scaling, and restoring.  
Amazon RDS supports most of the popular RDBMSs, ranging from commercial options to open-source options and even a specific AWS option.

Supported Amazon RDS engines include the following:

* **Commercial:** Oracle, SQL Server
* **Open source:** MySQL, PostgreSQL, MariaDB
* **Cloud native:**Aurora

### **Database instances**

Amazon RDS is built from compute and storage. The compute portion is called the database (DB) instance, which runs the DB engine. Depending on the engine selected, the instance will have different supported features and configurations. A DB instance can contain multiple databases with the same engine, and each DB can contain multiple tables.  
Underneath the DB instance is an EC2 instance. However, this instance is managed through the Amazon RDS console instead of the Amazon EC2 console. When you create your DB instance, you choose the instance type and size. The DB instance class you choose affects how much processing power and memory it has.

**Instance Classes:**

* 1. **Standard Classes -** Standard instances provide a balance of compute, memory, and network resources. They are a good choice for many database workloads.
  2. **Memory optimized classes -** Memory optimized instances accelerate performance for workloads that process large datasets in memory.
  3. **Burstable classes -** Burstable performance instances provide a baseline level of CPU performance with the ability to burst above the baseline.

### **Storage on Amazon RDS**

The storage portion of DB instances for Amazon RDS use Amazon Elastic Block Store (Amazon EBS) volumes for database and log storage. This includes MySQL, MariaDB, PostgreSQL, Oracle, and SQL Server. When using Aurora, data is stored in cluster volumes, which are single, virtual volumes that use solid-state drives (SSDs). A cluster volume contains copies of your data across three Availability Zones in a single AWS Region. For nonpersistent, temporary files, Aurora uses local storage.

Amazon RDS provides three storage types: General Purpose SSD (also called gp2 and gp3), Provisioned IOPS SSD (also called io1), and Magnetic (also called standard). They differ in performance characteristics and price, which means you can tailor your storage performance and cost to the needs of your database workload.

**Storage types:**

* 1. **General Purpose SSD -** These volumes offer cost-effective storage. This is ideal for a broad range of workloads running on medium-sized DB instances. Best suited for development and testing environments.
  2. **Provisioned IOPS SSD -** This type of storage is designed to meet the needs of I/O-intensive workloads. For example, database workloads requiring low I/O latency and consistent I/O throughput. Best suited for production environments.
  3. **Magnetic -** Amazon RDS also supports magnetic storage for backward compatibility. We recommend that you use General Purpose SSD or Provisioned IOPS SSD for any new storage needs. The maximum amount of storage for DB instances on magnetic storage is less than that of the other storage types.

### **Amazon RDS in an Amazon Virtual Private Cloud**

When you create a DB instance, you select the Amazon Virtual Private Cloud (Amazon VPC) your databases will live in. Then, you select the subnets that will be designated for your DB. This is called a DB subnet group, and it has at least two Availability Zones in its Region. The subnets in a DB subnet group should be private, so they don’t have a route to the internet gateway. This ensures that your DB instance, and the data inside it, can be reached only by the application backend. Access to the DB instance can be restricted further by using network access control lists (network ACLs) and security groups.

### **Backup data**

##### **Automated Backups**

Automated backups are turned on by default. This backs up your entire DB instance and your transaction logs. Set a backup window that is the period of time that automatic backups occur. Typically, set the window during a time when your database experiences little activity.

* **Retaining backups:**Automated backups are retained between 0 and 35 days. The 0 days setting stops automated backups from happening. If you set it to 0, it will also delete all existing automated backups. This is not ideal. The benefit of automated backups that you can do point-in-time recovery.
* **Point-in-time recovery:** This 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**

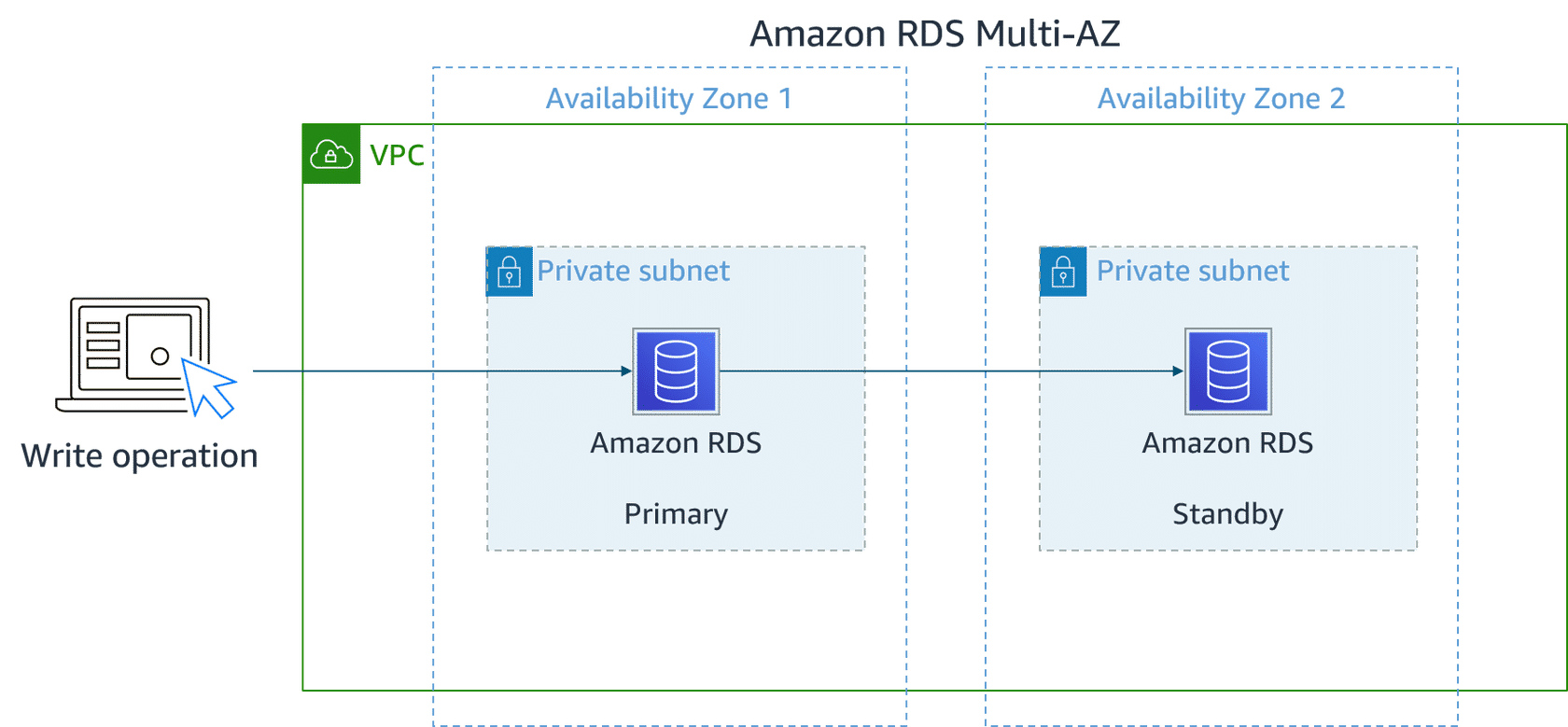
If you want to keep your automated backups longer than 35 days, use manual snapshots. Manual snapshots are similar to taking Amazon EBS snapshots, except you manage them in the Amazon RDS console. These are backups that you can initiate at any time. They exist until you delete them. If you restore data from a manual snapshot, it creates a new DB instance using the data from the snapshot.

### **Choosing a backup option**

It is advisable to deploy both backup options. Automated backups are beneficial for point-in-time recovery. With manual snapshots, you can retain backups for longer than 35 days.

### **Redundancy with Amazon RDS Multi-AZ**

In an Amazon RDS Multi-AZ deployment, Amazon RDS creates a redundant copy of your database in another Availability Zone. You end up with two copies of your database—a primary copy in a subnet in one Availability Zone and a standby copy in a subnet in a second Availability Zone.  
The primary copy of your database provides access to your data so that applications can query and display the information. The data in the primary copy is synchronously replicated to the standby copy. The standby copy is not considered an active database, and it does not get queried by applications.



If an availability issue arises, such as the primary database loses connectivity, Amazon RDS initiates an automatic failover.

When you create a DB instance, a Domain Name System (DNS) name is provided. AWS uses that DNS name to fail over 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 help ensure that you don't lose Multi-AZ configuration, there are two ways you can create a new standby database. They are as follows:

* Demote the previous primary to standby if it's still up and running.
* Stand up a new standby DB instance.

The reason you can select multiple subnets for an Amazon RDS database is because of the Multi-AZ configuration. You will want to ensure that you have subnets in different Availability Zones for your primary and standby copies.

### **Amazon RDS security**

When it comes to security in Amazon RDS, you have control over managing access to your Amazon RDS resources, such as your databases on a DB instance. How you manage access will depend on the tasks you or other users need to perform in Amazon RDS. Network ACLs and security groups help users dictate the flow of traffic. If you want to restrict the actions and resources others can access, you can use AWS Identity and Access Management (IAM) policies.

* **IAM -** Use IAM policies to assign permissions that determine who can manage Amazon RDS resources. For example, you can use IAM to determine who can create, describe, modify, and delete DB instances, tag resources, or modify security groups.
* **Security groups -** Use security groups to control which IP addresses or Amazon EC2 instances can connect to your databases on a DB instance. When you first create a DB instance, all database access is prevented except through rules specified by an associated security group.
* **Amazon RDS encryption -** Use Amazon RDS encryption to secure your DB instances and snapshots at rest.
* **SSL or TLS -** Use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connections with DB instances running the MySQL, MariaDB, PostgreSQL, Oracle, or SQL Server database engines.

### **Resources**

* AWS website: [Amazon RDS](https://aws.amazon.com/rds/)
* AWS user guide: [Configuring an Amazon RDS DB Instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_RDS_Configuring.html)
* AWS user guide: [Backing up and restoring an Amazon RDS DB instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_CommonTasks.BackupRestore.html)
* AWS user guide: [Security in Amazon RDS](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.html)

## **5.3. Purpose-Built Databases**

### **Purpose-built databases for all application needs**

Relational databases were the default option. They were widely used in nearly all applications. A relational database is like a multi-tool. It can do many things, but it is not perfectly suited to any one particular task. It might not always be the best choice for your business needs. The one-size-fits-all approach of using a relational database for everything no longer works.

AWS offers a broad and deep portfolio of purpose-built databases that support diverse data models. You can pick the best database to solve a specific problem and break away from restrictive commercial databases. You can focus on building applications that meet the needs of your organization.

### **Amazon DynamoDB**

Fully managed NoSQL database that provides fast, consistent performance at any scale. It has a flexible billing model, tight integration with infrastructure as code (IaC), and a hands-off operational model. DynamoDB has become the database of choice for two categories of applications: high-scale applications and serverless applications. It can work for nearly all online transaction processing (OLTP) application workloads.

### **Amazon ElastiCache**

Fully managed, in-memory caching solution. It provides support for two open-source, in-memory cache engines: Redis and Memcached.

### **Amazon MemoryDB for Redis**

Redis-compatible, durable, in-memory database service that delivers ultra-fast performance. With MemoryDB, you can achieve microsecond read latency, single-digit millisecond write latency, high throughput, and Multi-AZ durability for modern applications, like those built with microservices architectures. You can use MemoryDB as a fully managed, primary database to build high-performance applications.

### **Amazon DocumentDB (with MongoDB compatibility)**

Fully managed document database. A document database is a type of NoSQL database you can use to store and query rich documents in your application. These types of databases work well for the following use cases: content management systems, profile management, and web and mobile applications. It has API compatibility with MongoDB.

### **Amazon Keyspaces (for Apache Cassandra)**

Scalable, highly available, and managed Apache Cassandra compatible database service. Apache Cassandra is a popular option for high-scale applications that need top-tier performance. Amazon Keyspaces is a good option for high-volume applications with straightforward access patterns. With Amazon Keyspaces, you can run your Cassandra workloads on AWS using the same Cassandra Query Language (CQL) code, Apache 2.0 licensed drivers, and tools that you use today.

### **Amazon Neptune**

Fully managed graph database. A graph database is a good choice for highly connected data with a rich variety of relationships. Companies often use graph databases for recommendation engines, fraud detection, and knowledge graphs.

### **Amazon Timestream**

Timestream is a fast, scalable, and serverless time series database service for Internet of Things (IoT) and operational applications. It makes it easy to store and analyse trillions of events per day up to 1,000 times faster and for as little as one-tenth of the cost of relational databases. Time series data is a sequence of data points recorded over a time interval. It is used for measuring events that change over time, such as stock prices over time or temperature measurements over time.

### **Amazon Quantum Ledger Database (Amazon QLDB)**

With traditional databases, you can overwrite or delete data, so developers use techniques, such as audit tables and audit trails to help track data lineage. These approaches can be difficult to scale and put the burden of ensuring that all data is recorded on the application developer. Amazon QLDB is a purpose-built ledger database that provides a complete and cryptographically verifiable history of all changes made to your application data.

### **Resources**

* AWS website: [AWS Cloud Databases](https://aws.amazon.com/products/databases/)

## **5.4. Amazon DynamoDB**

### **DynamoDB overview**

Fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. With DynamoDB, you can offload the administrative burdens of operating and scaling a distributed database.

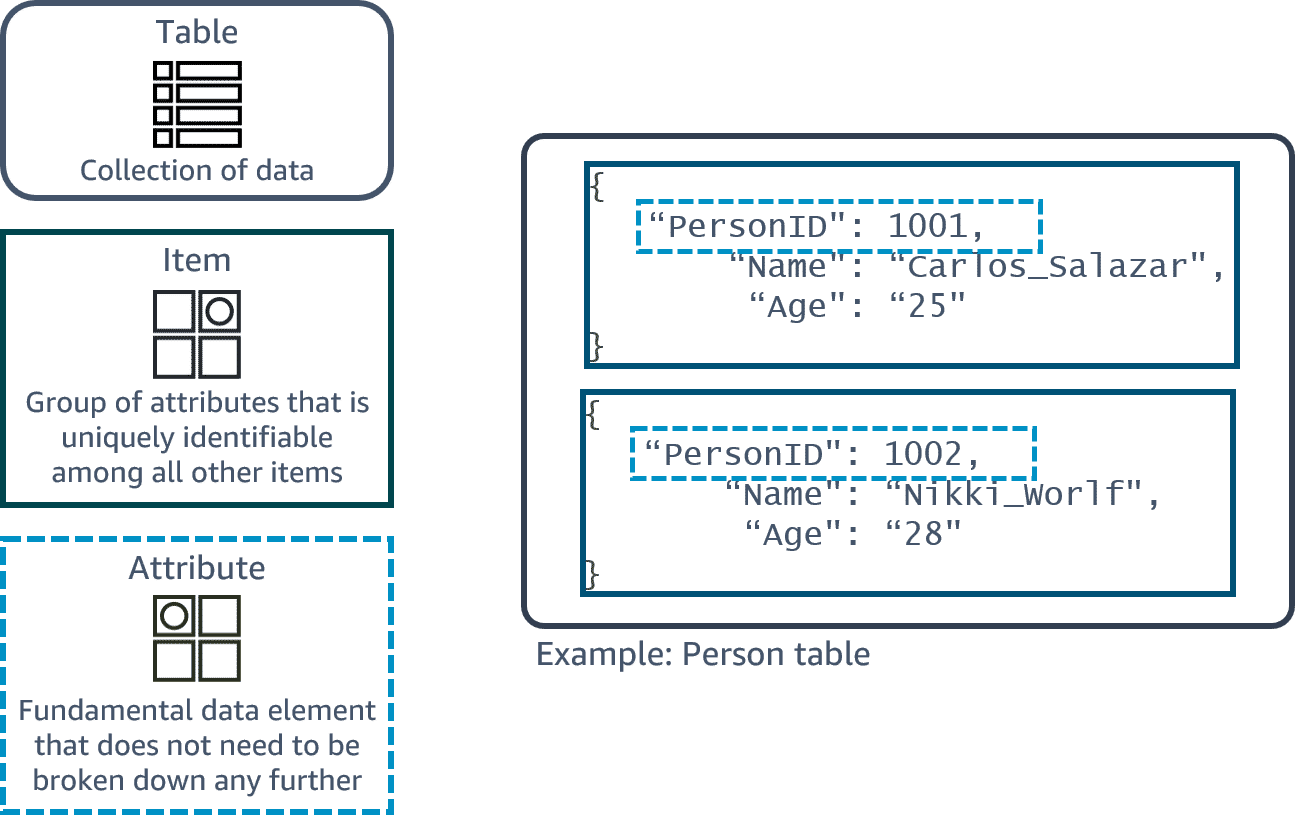
With DynamoDB, you can do the following:

* Create database tables that can store and retrieve any amount of data and serve any level of request traffic.
* Scale up or scale down your tables' throughput capacity without downtime or performance degradation.
* Monitor resource usage and performance metrics using the AWS Management Console.

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

### **DynamoDB core components**

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.



### **DynamoDB use cases**

Fully managed service that handles the operations work. You can offload the administrative burdens of operating and scaling distributed databases to AWS.

You might want to consider using DynamoDB in the following circumstances:

* You are experiencing scalability problems with other traditional database systems.
* You are actively engaged in developing an application or service.
* You are working with an OLTP workload.
* You care deploying a mission-critical application that must be highly available at all times without manual intervention.
* You require a high level of data durability, regardless of your backup-and-restore strategy.

DynamoDB is used in a wide range of workloads because of its simplicity, from low-scale operations to ultrahigh-scale operations, such as those demanded by Amazon.com.

**Potential Use Cases:**

* **Develop software applications** – Build internet-scale applications supporting user-content metadata and caches that require high concurrency and connections for millions of users and millions of requests per second.
* **Create media metadata stores** – Scale throughput and concurrency for analysis of media and entertainment workloads, such as real-time video streaming and interactive content. Deliver lower latency with multi-Region replication across Regions.
* **Scale gaming platforms** – Focus on driving innovation with no operational overhead. Build out your game platform with player data, session history, and leaderboards for millions of concurrent users.
* **Deliver seamless retail experiences** – Use design patterns for deploying shopping carts, workflow engines, inventory tracking, and customer profiles. DynamoDB supports high-traffic, extreme-scaled events and can handle millions of queries per second.

### **DynamoDB security**

DynamoDB provides a number of security features to consider as you develop and implement your own security policies. They include the following:

* DynamoDB provides a highly durable storage infrastructure designed for mission-critical and primary data storage. Data is redundantly stored on multiple devices across multiple facilities in a DynamoDB Region.
* All user data stored in DynamoDB is fully encrypted at rest. DynamoDB encryption at rest provides enhanced security by encrypting all your data at rest using encryption keys stored in AWS Key Management Service (AWS KMS).
* IAM administrators control who can be authenticated and authorized to use DynamoDB resources. You can use IAM to manage access permissions and implement security policies.
* As a managed service, DynamoDB is protected by the AWS global network security procedures.

**Security Best Practices**:

* Use AWS CloudTrail to monitor AWS managed key usage - If you are using an AWS managed key for encryption at rest, usage of the key is recorded in AWS CloudTrail. CloudTrail can tell you who made the request, the services used, actions performed, parameters for the action, and response elements returned.
* Use IAM roles to authenticate access to DynamoDB - For users, applications, and other AWS services to access DynamoDB, they must include valid AWS credentials in their AWS API requests. Use IAM roles to obtain temporary access keys.
* Use IAM policy conditions for fine-grained access control - When you grant permissions in DynamoDB, you can specify conditions that determine how a permissions policy takes effect. Implementing least privilege is key in reducing security risk and the impact that can result from errors or malicious intent.
* Monitor DynamoDB operations using CloudTrail - When activity occurs in DynamoDB, that activity is recorded in a CloudTrail event. For an ongoing record of events in DynamoDB and in your AWS account, create a trail to deliver log files to an Amazon Simple Storage Service (Amazon S3) bucket.

### **Resources**

* AWS website: [Amazon DynamoDB](https://aws.amazon.com/dynamodb/)
* AWS developer guide: [What is Amazon DynamoDB?](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html)
* AWS blog: [AWS Database Blog](https://aws.amazon.com/blogs/database/how-to-determine-if-amazon-dynamodb-is-appropriate-for-your-needs-and-then-plan-your-migration/)

## **5.5. Choosing the Right Database Service**

### **AWS database services**

following table provides a quick look at the AWS database portfolio.

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

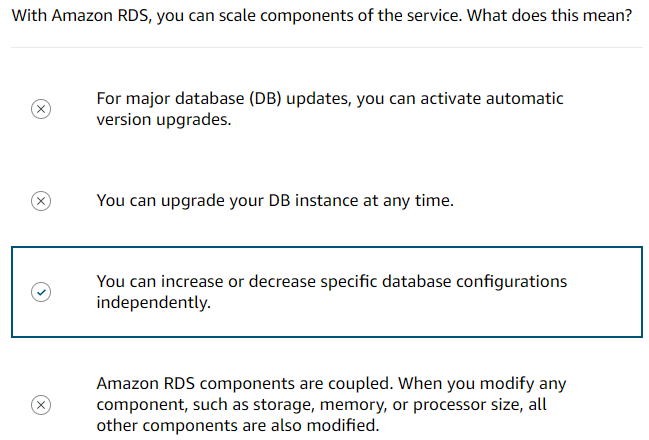
### **Breaking up applications and databases**

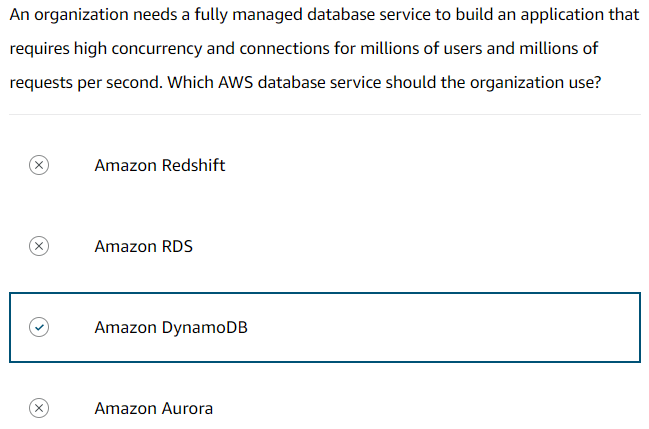
As the industry changes, applications and databases change too. Today, with larger applications, you no longer see just one database supporting them. Instead, applications are broken into smaller services, each with its own purpose-built database supporting it. This shift removes the idea of a one-size-fits-all database and replaces it with a complimentary database strategy. You can give each database the appropriate functionality, performance, and scale the workload requires.

### **Resources**

* AWS website: [AWS Cloud Databases](https://aws.amazon.com/products/databases/)
* AWS blog: [AWS Database Blog](https://aws.amazon.com/blogs/database/?nc=sn&loc=4)
* AWS website: [Database Freedom](https://aws.amazon.com/products/databases/freedom/?nc=sn&loc=5)

## **5.6. Assessment**





# **6. Monitoring, Load Balancing & Scaling**

## **6.1. Monitoring**

### **Purpose of monitoring**

When operating a website, you might have questions like the following:

* 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 Elastic Compute Cloud (Amazon EC2) instance runs out of capacity?
* Will I be alerted if my website goes down?

The act of collecting, analysing, and using data to make decisions or answer questions about your IT resources and systems is called monitoring. Monitoring provides a near real-time pulse on your system and helps answer the previous questions. You can use the data you collect to watch for operational issues caused by events like overuse of resources, application flaws, resource misconfiguration, or security-related events. Think of the data collected through monitoring as outputs of the system, or metrics.

### **Use metrics to solve problems**

Each individual data point that a resource creates is a metric. Metrics that are collected and analysed over time become statistics, such as average CPU utilization over time showing a spike. One way to evaluate the health of an EC2 instance is through CPU utilization. When analysing CPU utilization, take a process that exceeds a specific threshold for an unusual length of time. Use that abnormal event as a cue to either manually or automatically resolve the issue through actions like scaling the instance. Other examples of metrics that EC2 instances have are network utilization, disk performance, memory utilization, and the logs created by the applications running on top of Amazon EC2.

### **Types of metrics**

1. **Amazon Simple Storage Service (Amazon S3) metrics**

* Size of objects stored in a bucket.
* Number of objects stored in a bucket.
* Number of HTTP request made to a bucket

1. **Amazon Relational Database Service (Amazon RDS) metrics**

* Database connections
* CPU utilization of an instance
* Disk space consumption

1. **Amazon EC2 metrics**

* CPU utilization
* Network utilization
* Disk performance
* Status checks

### **Monitoring benefits**

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

* **Respond proactively** – **Respond to operational issues proactively before your end users are aware of them.** Waiting for end users to let you know when your application is experiencing an outage is a bad practice. Through monitoring, you can keep tabs on metrics like error response rate and request latency. Over time, the metrics help signal when an outage is going to occur. You can automatically or manually perform actions to prevent the outage from happening and fix the problem before your end users are aware of it.
* **Improve performance and reliability** – **Monitoring can improve the performance and reliability of your resources.**Monitoring the various 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 helps you drive performance and improve reliability.
* **Recognize security threats and events** – **By monitoring, you can recognize security threats and events.**When you monitor resources, events, and systems over time, you create what is called a baseline. A baseline defines normal activity. 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** – **Monitoring helps you make data-driven decisions for your business.**Monitoring keeps an eye on IT operational health and drives business decisions. For example, suppose you launched a new feature for your cat photo app and now you want to know if 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 can decide whether to invest more time into improving the new feature.
* **Create cost-effective solutions** – **Through monitoring, you can create more cost-effective solutions.**You can view resources that are underused and rightsize your resources to your usage. This helps you optimize cost and make sure you aren’t spending more money than necessary.

### **Resources**

* AWS user guide: [Best practices for monitoring](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_best_practices.html)
* AWS website: [Monitoring and Observability](https://aws.amazon.com/cloudops/monitoring-and-observability/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&blog-posts-cards.sort-by=item.additionalFields.createdDate&blog-posts-cards.sort-order=desc)
* AWS user guide: [Monitor Amazon EC2](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_ec2.html)
* AWS user guide: [Monitoring Amazon S3](https://docs.aws.amazon.com/AmazonS3/latest/userguide/monitoring-overview.html)
* AWS user guide: [Monitoring metrics in an Amazon RDS instance](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Monitoring.html)

## **6.2. Amazon Cloud Watch**

### **Visibility using CloudWatch**

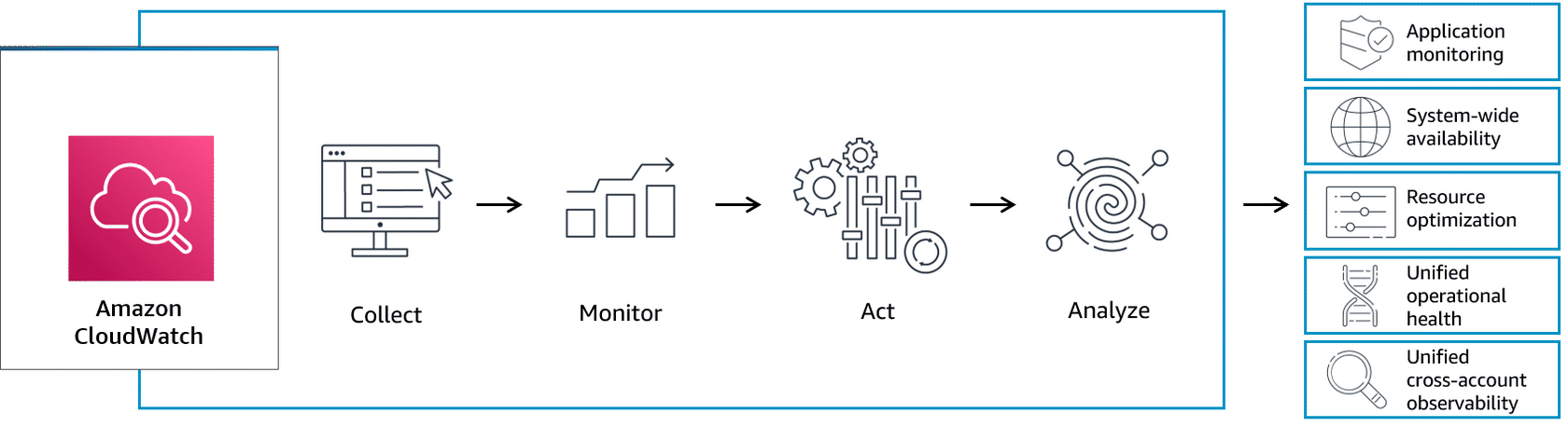
AWS resources create data that you can monitor through metrics, logs, network traffic, events, and more. CloudWatch is a monitoring and observability service that collects your resource data and provides actionable insights into your applications.

You can use CloudWatch to do the following:

* Detect anomalous behaviour in your environments.
* Set alarms to alert you when something is 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.

### **How CloudWatch works**

It is a managed service that you can use for monitoring.

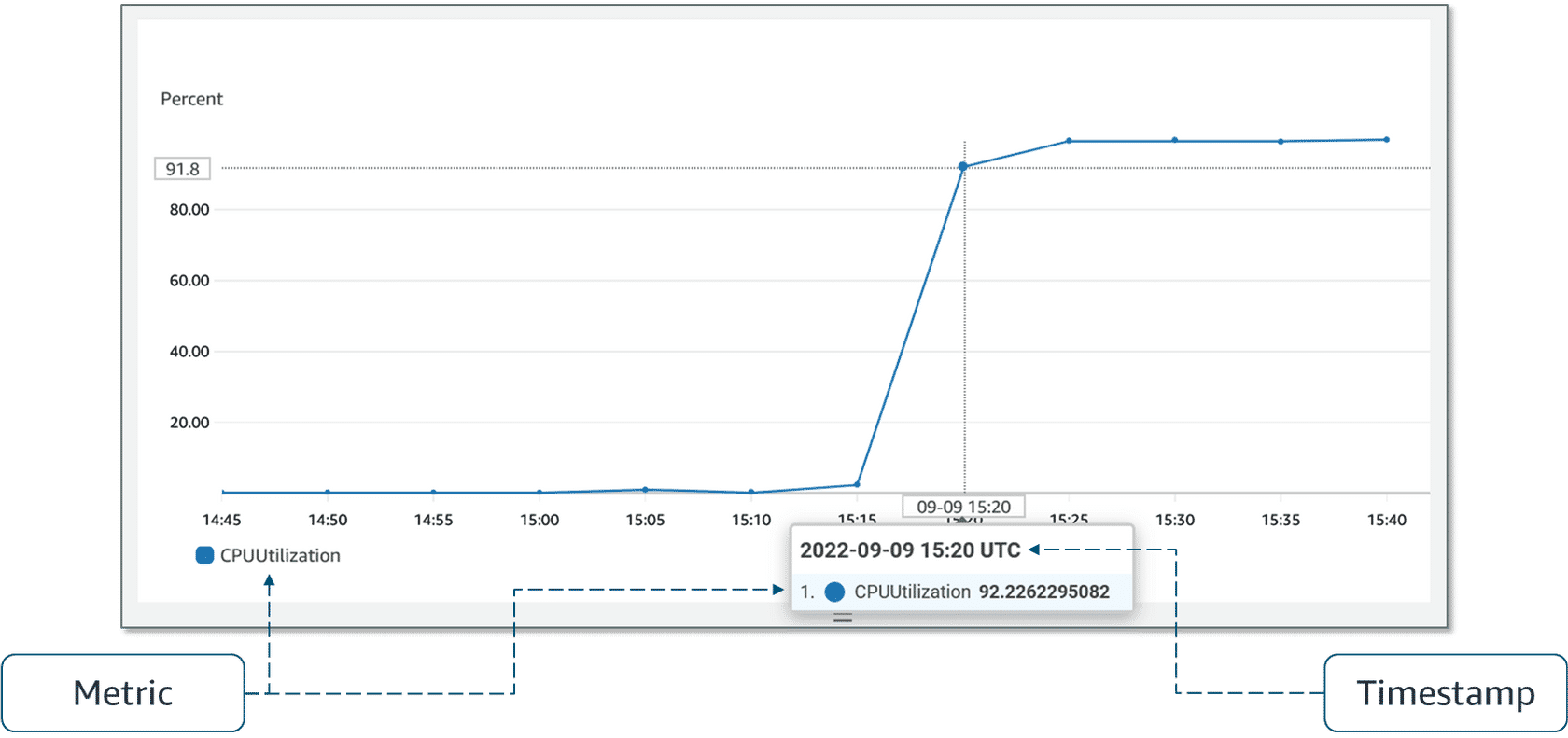


CloudWatch acts as a centralized place where metrics are gathered and analysed.

Many AWS services automatically send metrics to CloudWatch for free at a rate of 1 data point per metric per 5-minute interval without any extra cost. 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 incurs a fee.

### **CloudWatch concepts**

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor and the data points as representing the values of that variable over time. Every metric data point must be associated with a timestamp.



AWS services that send data to CloudWatch attach dimensions to each metric. A dimension is a name and value pair that is part of the metric’s identity. You can use dimensions to filter the results that CloudWatch returns. For example, many Amazon EC2 metrics publish **InstanceId**as a dimension name and the actual instance ID as the value for that dimension.

By default, many AWS services provide metrics at no charge for resources such as EC2 instances, Amazon Elastic Block Store (Amazon EBS) volumes, and Amazon RDS database (DB) instances. For a charge, you can activate features such as detailed monitoring or publishing your own application metrics on resources such as your EC2 instances.

### **Custom metrics**

Suppose you have an application, and you want to record the number of page views your website gets. How would you record this metric with CloudWatch? First, it's an application-level metric. That means it’s not something the EC2 instance would post to CloudWatch by default. This is where custom metrics come in. With custom metrics, you can publish your own metrics to CloudWatch. If you want to gain more granular visibility, you can use high-resolution custom metrics, which make it possible for you to collect custom metrics down to a 1-second resolution. This means you can send 1 data point per second per custom metric.

Some examples of custom metrics include the following:

* Webpage load times
* Request error rates
* Number of processes or threads on your instance
* Amount of work performed by your application.

### **CloudWatch dashboards**

Once you provision your AWS resources and they are sending metrics to CloudWatch, you can visualize and review that data using CloudWatch dashboards. Dashboards are customizable home pages you can configure for data visualization for one or more metrics through 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 AWS Regions into a single dashboard 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 is concerned, with AWS Identity and Access Management (IAM) policies, you control who has access to view or manage your CloudWatch dashboards.

### **Amazon CloudWatch Logs**

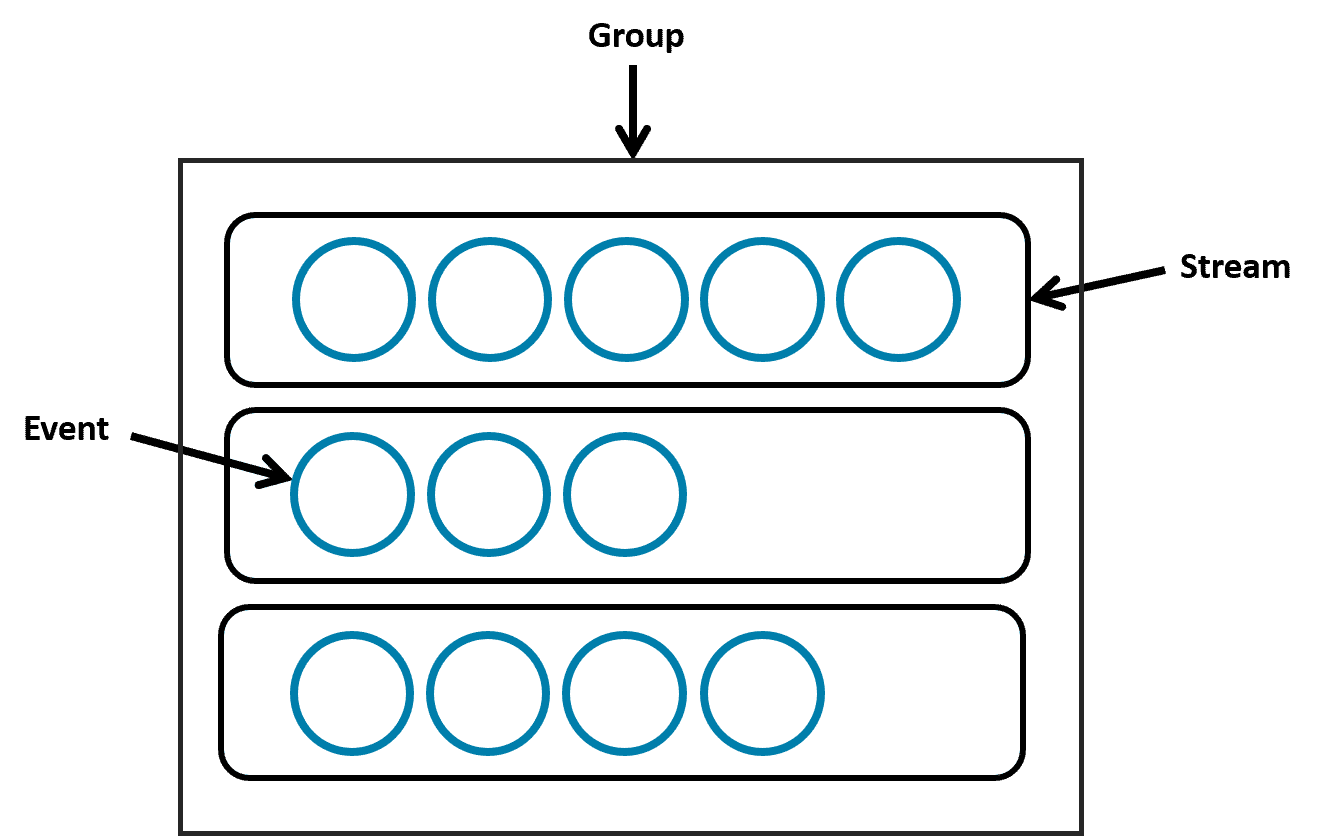
CloudWatch Logs is centralized place for logs to be stored and analyzed. With this service, you can monitor, store, and access your log files from applications running on EC2 instances, AWS Lambda functions, and other sources.

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

Some services, like Lambda, are set up to send log data to CloudWatch Logs with minimal effort. With 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, to send your application logs from an EC2 instance into CloudWatch Logs, you need to install and configure the CloudWatch Logs agent on the EC2 instance. With the CloudWatch Logs agent, EC2 instances can automatically send log data to CloudWatch Logs.

##### **CloudWatch Logs terminology**

Log data sent to CloudWatch Logs can come from different sources, so it’s important you understand how they’re organized.



**Event:**

* A log event is a record of activity recorded by the application or resource being monitored.
* It has a timestamp and an event message.

**Group**

* A log group is composed of log streams that all share the same retention and permissions settings.
* For example, suppose you have multiple EC2 instances hosting your application and you send application log data to CloudWatch Logs. You can group the log streams from each instance into one log group.

**Stream**

* Log events are 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 filter or query for insights.

### **CloudWatch alarms**

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

First, you must decide which metric you want to set up an alarm for, and then you define the threshold that will invoke the alarm. Next, you define the threshold's time period. You don’t want to invoke an alarm based on short, temporary spikes in the CPU. You only want to invoke an alarm if the CPU is elevated for a sustained amount of time. For example, if CPU utilization exceeds 80 percent for 5 minutes or longer, there might be a resource issue. To set up an alarm you need to choose the metric, threshold, and time period.

An alarm can be invoked when it transitions from one state to another. After an alarm is invoked, it can initiate an action. Actions can be an Amazon EC2 action, an automatic scaling action, or a notification sent to Amazon Simple Notification Service (Amazon SNS).

Three possible states of an alarm:

1. **OK -** The metric is within the defined threshold. Everything appears to be operating like normal.
2. **ALARM -** The metric is outside the defined threshold. This might be an operational issue.
3. **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.

##### **Prevent and troubleshoot issues with CloudWatch alarms**

CloudWatch Logs uses metric filters to turn the log data into metrics that you can graph or set an alarm on.

After the alarm is set up, you know that if the error happens again, you will be notified promptly. You can set up different alarms for different reasons to help you prevent or troubleshoot operational issues.

Another option is to have alarms invoke actions that automatically remediate technical issues. For example, you can set up an alarm to invoke an EC2 instance to reboot or scale services up or down. You can even set up an alarm to invoke an Amazon SNS notification that invokes a 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 can respond to events more quickly.

**Resources**

* AWS website: [Amazon CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/)
* AWS website: [Amazon CloudWatch](https://aws.amazon.com/cloudwatch/)
* AWS user guide: [Getting Started with Amazon CloudWatch](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/GettingStarted.html)
* AWS user guide: [What Is Amazon CloudWatch Logs?](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html)
* AWS user guide: [AWS services that publish CloudWatch metrics](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/aws-services-cloudwatch-metrics.html)
* AWS user guide: [View available metrics](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/viewing_metrics_with_cloudwatch.html)
* AWS website: [Amazon SNS](https://aws.amazon.com/sns/)

## **6.3. Solution Optimization**

### **Availability**

The availability of a system is typically expressed as a percentage of uptime in a given year or as a number of nines. In the following table is a list of availability percentages based on the downtime per year and its notation in nines.

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

To increase availability, you need redundancy. This typically means more infrastructure—more data centres, 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.

### **Why improve application availability?**

In the current application, one EC2 instance hosts the application. The photos are served from Amazon S3, and the structured data is stored in Amazon DynamoDB. That single EC2 instance is a single point of failure for the application. Even if the database and Amazon S3 are highly available, customers have no way to connect if the single instance becomes unavailable. One way to solve this single point of failure issue is to add one more server in a second Availability Zone.

**Adding a second Availability Zone**

The physical location of a server is important. In addition to potential software issues at the operating system (OS) or application level, you must also consider hardware issues. They might be in the physical server, the rack, the data centre, or even the Availability Zone hosting the virtual machine. To remedy the physical location issue, you can deploy a second EC2 instance in a second Availability Zone. This second instance might also solve issues with the OS and the application.

However, when there is more than one instance, it brings new challenges, such as the following:

1. **Replication process –**The first challenge with multiple EC2 instances is that you need to create a process to replicate the configuration files, software patches, and application across instances. The best method is to automate where you can.
2. **Customer redirection –**The second challenge is how to notify the clients—the computers sending requests to your server—about the different servers. You can use various tools here. The most common is using a Domain Name System (DNS) where the client uses one record that points to the IP address of all available servers.  
   However, this method isn't always used because of propagation — the time frame it takes for DNS changes to be updated across the Internet.  
   Another option is to use a load balancer, which takes care of health checks and distributing the load across each server. Situated between the client and the server, a load balancer avoids propagation time issues. You will learn more about load balancers in the next section.
3. **Types of high availability –**The last challenge to address when there is more than one server is the type of availability you need: active-passive or active-active.

* **Active-passive systems** – 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. This is because the customers are always sent to the server where their session is stored.
* **Active-active systems** – A disadvantage of an active-passive system is scalability. This is where an active-active system shines. With both servers available, the second server can take some load for the application, and the entire system can 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.

### **Resources**

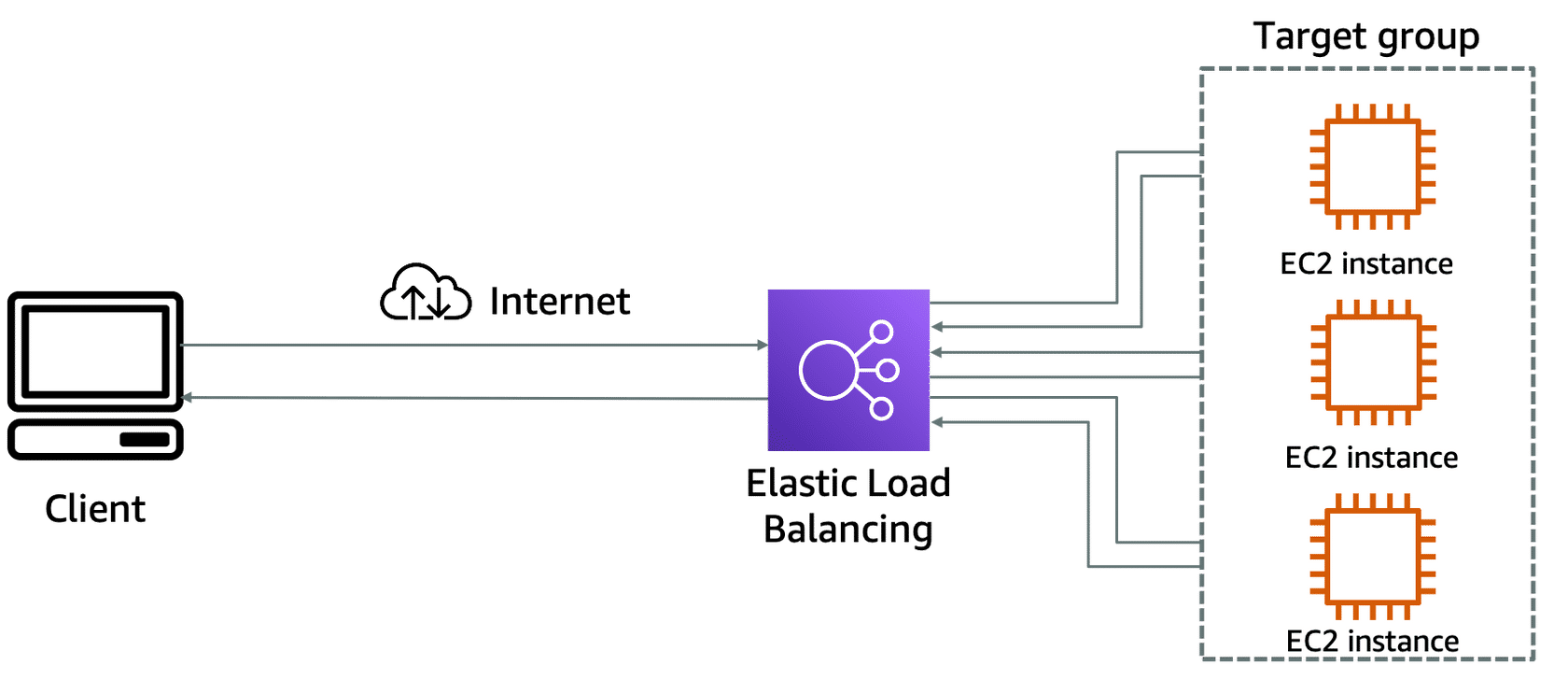
* AWS whitepaper: [High availability and scalability on AWS](https://docs.aws.amazon.com/whitepapers/latest/real-time-communication-on-aws/high-availability-and-scalability-on-aws.html)
* AWS documentation: [Reliability Pillar – AWS Well-Architected Framework](https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html)
* AWS website: [Amazon EC2 Auto Scaling](https://aws.amazon.com/ec2/autoscaling/)

## **6.4. Traffic Routing with Elastic Load Balancing**

### **Load balancers**

Load balancing refers to the process of distributing tasks across a set of resources. You can use a load balancer to distribute the requests across all the servers hosting the application. To do this, the load balancer needs to take all 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 an application starts from a client's browser. The request is sent to a load balancer. Then, it’s sent to one of the EC2 instances that hosts the application. The return traffic goes back through the load balancer and back to the client's browser.  
Although it is possible to install your own software load balancing solution on EC2 instances, AWS provides the ELB service for you.



### **ELB features**

The ELB service provides a major advantage over using your own solution to do load balancing. Mainly, you don’t need to manage or operate ELB. It can distribute incoming application traffic across EC2 instances, containers, IP addresses, and Lambda functions. Other key features include the following:

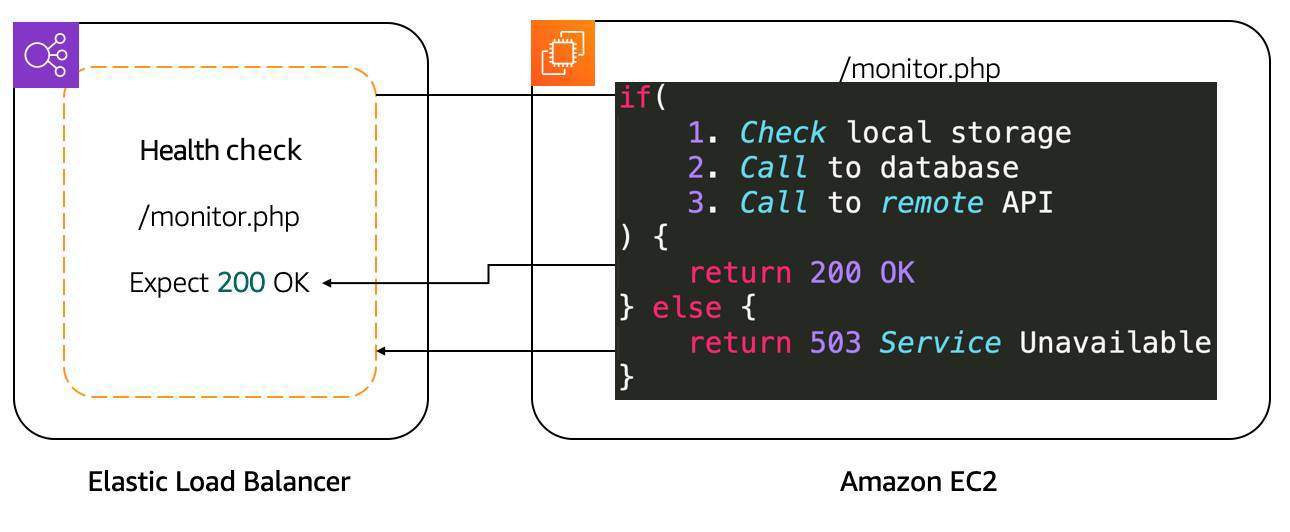
* **Hybrid mode –** Because ELB can load balance to IP addresses, it can work in a hybrid mode, which means it also load balances to on-premises servers.
* **High availability –**ELB is highly available. The only option you must ensure is that the load balancer's targets are deployed across multiple Availability Zones.
* **Scalability –** 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**

Monitoring is an important part of load balancers because they should route traffic to only healthy EC2 instances. That’s why ELB supports two types of health checks as follows:

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

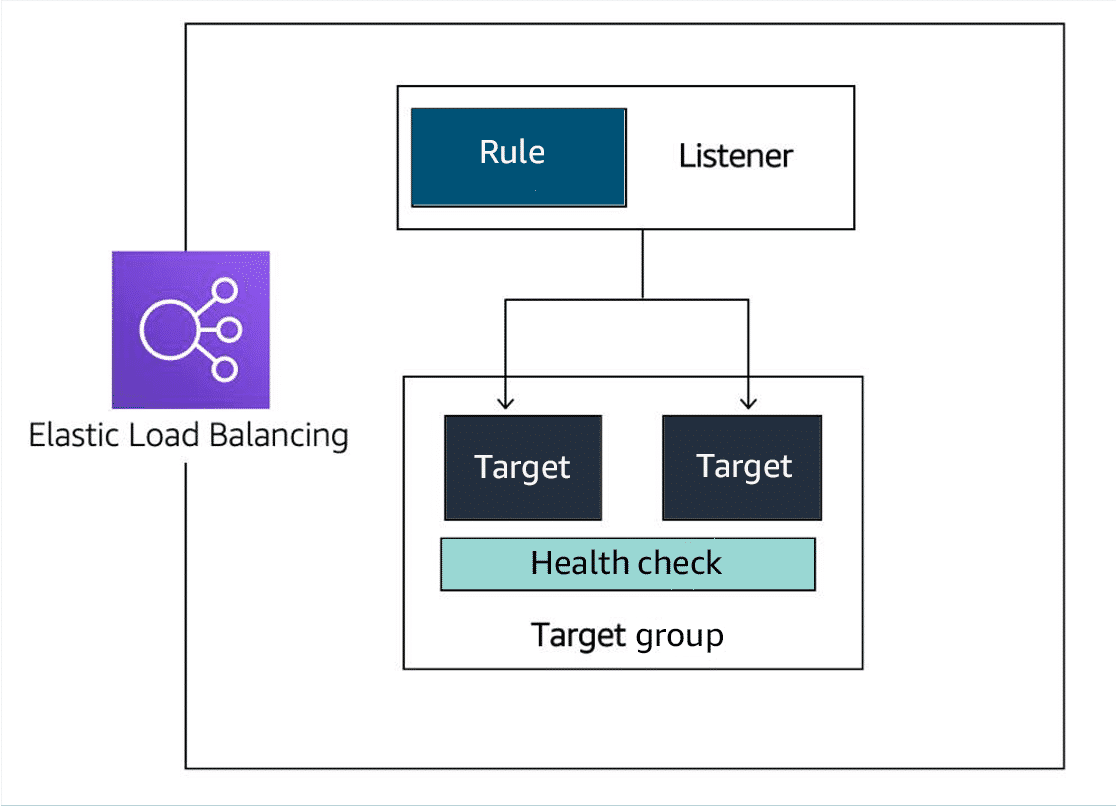
Taking time to define an appropriate health check is critical. Only verifying that the port of an application is open doesn’t mean that the application is working. It also doesn’t mean that making a call to the home page of an application is the right way either.  
For example, the Employee Directory application depends on a database and Amazon S3. The health check should validate all the elements. One way to do that is to create a monitoring webpage, such as**/monitor**. It will make a call to the database to ensure that it can connect, get data, and make a call to Amazon 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 informs Amazon EC2 Auto Scaling. It is the responsibility of Amazon EC2 Auto Scaling to remove that instance from the group and replace it with a new EC2 instance.   
If Amazon EC2 Auto Scaling has a scaling policy that calls for a scale down action, it informs ELB that the EC2 instance will be terminated. ELB can prevent Amazon EC2 Auto Scaling from terminating an EC2 instance until all connections to the instance end. It also prevents any new connections. This feature is called connection draining.

### **ELB components**

The ELB service is made up of three main components: rules, listeners, and target groups.



1. **Rule –** To associate a target group to a listener, you must use a rule. Rules are made up of two conditions. The first condition is the source IP address of the client. The second condition decides which target group to send the traffic to.
2. **Listener –** The client connects to the listener. This is often called client side. To define a listener, a port must be provided in addition to the protocol, depending on the load balancer type. There can be many listeners for a single load balancer.
3. **Target Group –** The backend servers, or server side, are 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, Lambda functions, or IP addresses. Also, a health check must be defined for each target group.

### **Types of load balancers**

1. **Application Load Balancer (ALB)**
   * User authorization
   * Rich metrics and logging
   * Redirects
   * Fixed response
2. **Network Load Balancer (NLB)**
   * TCP and User Datagram Protocol (UDP) connection based.
   * Source IP preservation
   * Low latency
3. **Gateway Load Balancer (GLB).**
   * Health checks
   * Gateway Load Balancer Endpoints
   * Higher availability for third-party virtual appliances

##### **Application Load Balancer**

An Application Load Balancer functions at Layer 7 of the Open Systems Interconnection (OSI) model. It is ideal for load balancing HTTP and HTTPS traffic. After the load balancer receives a request, it evaluates the listener rules in priority order to determine which rule to apply. It then routes traffic to targets based on the request content.

**Primary features of an Application Load Balancer:**

1. **Routes traffic based on request data** – An ALB makes routing decisions based on the HTTP and HTTPS protocol. The ALB could use the URL path (/upload) and host, HTTP headers and method, or the source IP address of the client.
2. **Sends responses directly to the client** – An ALB can reply directly to the client with a fixed response, such as a HTML page. It can send a redirect to the client. This is useful when you must redirect to a specific website or redirect a request from HTTP to HTTPS.
3. **Uses TLS offloading** – An ALB understands HTTPS traffic. To pass HTTPS traffic through an Application Load Balancer, an SSL certificate is provided one of the following ways:

* Importing a certificate by way of IAM or ACM services
* Creating a certificate for free using ACM

This ensures that the traffic between the client and ALB is encrypted.

1. **Authenticates users** – An Application Load Balancer can authenticate users before they can pass through the load balancer. The ALB uses the OpenID Connect (OIDC) protocol and integrates with other AWS services to support protocols, such as the following:

* SAML
* Lightweight Directory Access Protocol (LDAP)
* Microsoft Active Directory

1. **Secures traffic** – To prevent traffic from reaching the load balancer, you configure a security group to specify the supported IP address ranges.
2. **Supports sticky sessions** – If requests must be sent to the same backend server because the application is stateful, use the sticky session feature. This feature uses an HTTP cookie to remember which server to send the traffic to across connections.

##### **Network Load Balancer**

A Network Load Balancer is ideal for load balancing TCP and UDP traffic. It functions at Layer 4 of the OSI model, routing connections from a target in the target group based on IP protocol data.

**Primary features of Network Load Balancers:**

* 1. **Sticky sessions -** Routes requests from the same client to the same target.
  2. **Low latency -** Offers low latency for latency-sensitive applications.
  3. **Source IP address -** Preserves the client-side source IP address.
  4. **Static IP support -** Automatically provides a static IP address per AZ (subnet).
  5. **Elastic IP address support -** Lets users assign a custom, fixed IP address per AZ (subnet).
  6. **DNS failover -** Uses Amazon Route 53 to direct traffic to load balancer nodes in other zones.

##### **Gateway Load Balancer**

A Gateway Load Balancer helps you to deploy, scale, and manage your third-party appliances, such as firewalls, intrusion detection and prevention systems, and deep packet inspection systems. It provides a gateway for distributing traffic across multiple virtual appliances while scaling them up and down based on demand.

**Primary features of Gateway Load Balancers:**

1. **High availability -** Ensures high availability and reliability by routing traffic through healthy virtual appliances.
2. **Monitoring -** Can be monitored using CloudWatch metrics.
3. **Streamlined deployments -** Can deploy a new virtual appliance.
4. **Private connectivity -** Connects internet gateways, virtual private clouds (VPCs), and other network resources over a private network.

### **Selecting between ELB types**

| **Feature** | **ALB** | **NLB** | **GLB** |
| --- | --- | --- | --- |
| **Load Balancer Type** | Layer 7 | Layer 4 | Layer 3 gateway and Layer 4 load balancing |
| **Target Type** | IP, instance, Lambda | IP, instance, ALB | IP, instance |
| **Protocol Listeners** | HTTP, HTTPS | TCP, UDP, TLS | IP |
| **Static IP and Elastic IP Address** |  | Yes |  |
| **Preserve Source IP Address** | Yes | Yes | Yes |
| **Fixed Response** | Yes |  |  |
| **User Authentication** | Yes |  |  |

### **Resources**

* AWS website: [Elastic Load Balancing features](https://aws.amazon.com/elasticloadbalancing/features/#Product_comparisons)
* AWS website: [AWS Certificate Manager](https://aws.amazon.com/certificate-manager/)
* AWS documentation: [Authenticate users using an Application Load Balancer](https://docs.aws.amazon.com/elasticloadbalancing/latest/application/listener-authenticate-users.html)
* AWS developer guide: [How AWS WAF works](https://docs.aws.amazon.com/waf/latest/developerguide/how-aws-waf-works.html)
* AWS blog: [Introducing AWS Gateway Load Balancer](https://aws.amazon.com/blogs/aws/introducing-aws-gateway-load-balancer-easy-deployment-scalability-and-high-availability-for-partner-appliances/)

## **6.5. Amazon EC2 Auto Scaling**

### **Capacity issues**

You can improve availability and reachability by adding one more server. However, the entire system can again become unavailable if there is a capacity issue. This section looks at load issues for both active-passive systems and active-active systems. These issues are addressed through scaling.

##### **VERTICAL SCALING**

**Increase the instance size.** If too many requests are sent to a single active-passive system, the active server will become unavailable and, hopefully, fail over to the passive server. But this doesn’t solve anything.  
With active-passive systems, 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 be done only 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 because it’s not taking any traffic.
2. **Change the instance size or type,** and then start the instance again.
3. **Shift the traffic** to the passive instance, turning it active.
4. **Stop, change the size, and start** the previous active instance because both instances should match.

When the number of requests reduces, you must do the same operation. Even though there aren’t that many steps involved, it’s actually a lot of manual work. Another disadvantage is that a server can only scale vertically up to a certain limit. When that limit is reached, the only option is to create another active-passive system and split the requests and functionalities across them. This can require massive application rewriting.  
This is where the active-active system can help. When there are too many requests, you can scale this system horizontally by adding more servers.

##### **HORIZONTAL SCALING**

**Add additional instances.**As mentioned, for the application to work in an active-active system, it’s already created as stateless, not storing any client sessions on the server. This means that having two or four servers wouldn’t require any application changes. It would only be a matter of creating more instances when required and shutting them down when 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. We will learn more about this service in this lesson.  
  
You can see that there are many more advantages to using an active-active system in comparison with an active-passive system. Modifying your application to become stateless provides scalability.

### **Traditional scaling compared to auto scaling**

With a traditional approach to scaling, you buy and provision enough servers to handle traffic at its peak. You must turn off the unused services, especially EC2 instances. You can 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 a loss of customers because of 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 Amazon EC2 Auto Scaling service does.

### **Amazon EC2 Auto Scaling features**

The Amazon EC2 Auto Scaling service adds and removes capacity to keep a steady and predictable performance at the lowest possible cost. This means Amazon EC2 Auto Scaling helps scale your infrastructure and ensure high availability.

**Features:**

1. Automatic scaling - Automatically scales in and out based on demand.
2. Scheduled scaling - Scales based on user-defined schedules.
3. Fleet management - Automatically replaces unhealthy EC2 instances.
4. Predictive scaling - Uses machine learning (ML) to help schedule the optimum number of EC2 instances.
5. Purchase options - Includes multiple purchase models, instance types, and Availability Zones.
6. Amazon EC2 availability - Comes with the Amazon EC2 service.

##### **ELB with Amazon EC2 Auto Scaling**

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

### **Configure Amazon EC2 Auto Scaling components**

There are three main components of Amazon EC2 Auto Scaling.

1. **Launch template or configuration:**Which resources should be automatically scaled?
2. **Amazon EC2 Auto Scaling groups:**Where should the resources be deployed?
3. **Scaling policies:**When should the resources be added or removed?

##### **Launch templates and configurations**

Multiple parameters are required to create EC2 instances—Amazon Machine Image (AMI) ID, instance type, security group, additional Amazon EBS volumes, and more. All this information is also required by Amazon 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 or for use with Amazon EC2 Auto Scaling. It also supports versioning, which can be used for quickly rolling back if there's an issue or a need to specify a default version of the 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.

A launch template specifies instance configuration information, such as the ID of the AMI, instance type, and security groups. You can have multiple versions of a launch template with a subset of the full parameters.

You can create a launch template in one of three ways as follows:

1. Use an existing EC2 instance. All the settings are already defined.
2. Create one from an already existing template or a previous version of a launch template.
3. Create a template from scratch. These parameters will need to be defined: AMI ID, instance type, key pair, security group, storage, and resource tags.

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 you cannot use a previously created launch configuration as a template. You also cannot create a template from an already existing EC2 instance. For these reasons, and to ensure that you get the latest features from Amazon EC2, AWS recommends you use a launch template instead of a launch configuration.

##### **Amazon EC2 Auto Scaling groups**

An Auto Scaling group helps you define where Amazon EC2 Auto Scaling deploys your resources. This is where you specify the Amazon Virtual Private Cloud (Amazon VPC) and subnets the EC2 instance should be launched in. Amazon EC2 Auto Scaling takes care of creating the EC2 instances across the subnets, so select at least two subnets that are across different Availability Zones.

You have three capacity settings to configure for the group size.

1. **Minimum Capacity –** This is the minimum number of instances running in your Auto Scaling group, even if the threshold for lowering the number of instances is reached. When Amazon EC2 Auto Scaling removes EC2 instances because the traffic is minimal, it keeps removing EC2 instances until it reaches a minimum capacity. When reaching that limit, even if Amazon EC2 Auto Scaling is instructed to remove an instance, it does not.

**Note:** Depending on your application, using a minimum of two is recommended to ensure high availability.

1. **Desired Capacity –** The desired capacity is the number of EC2 instances that Amazon EC2 Auto Scaling creates at the time the group is created. This number can only be within or equal to the minimum or maximum. If that number decreases, Amazon EC2 Auto Scaling removes the oldest instance by default. If that number increases, Amazon EC2 Auto Scaling creates new instances using the launch template.
2. **Maximum Capacity –** This is the maximum number of instances running in your Auto Scaling group, even if the threshold for adding new instances is reached. When traffic keeps growing, Amazon EC2 Auto Scaling keeps adding EC2 instances. This means the cost for your application will also keep growing. That's why you must set a maximum amount to ensure it doesn't go above your budget.

### **Scaling policies**

By default, an Auto Scaling group will be kept to its initial desired capacity. While it is possible to manually change the desired capacity, you can also use scaling policies.  
You use metrics to keep information about different attributes of your EC2 instance, such as 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 can set up an alarm that states when the CPU utilization is above 70 percent across the entire fleet of EC2 instances. It will then invoke a scaling policy to add an EC2 instance.

##### **SIMPLE SCALING POLICY**

You use a CloudWatch alarm and specify what to do when it is invoked. This can include adding or removing a number of EC2 instances or specifying a number of instances to set the desired capacity to. You can specify a percentage of the group instead of using a number of EC2 instances, which makes the group grow or shrink more quickly.

After the scaling policy is invoked, it enters a cooldown period before taking any other action. This is important because it takes time for the EC2 instances to start, and the CloudWatch alarm might still be invoked while the EC2 instance is booting. For example, you might decide to add an EC2 instance if the CPU utilization across all instances is above 65 percent. You don’t want to add more instances until that new EC2 instance is accepting traffic. However, what if the CPU utilization is now above 85 percent across the Auto Scaling group? Adding one instance might not be the right move. Instead, you might want to add another step in your scaling policy. This is where a step scaling policy helps.

##### **STEP SCALING POLICY**

Step scaling policies respond to additional alarms even when a scaling activity or health check replacement is in progress. You might decide to add two more instances when CPU utilization is at 85 percent and four more instances when it is at 95 percent. Deciding when to add and remove instances based on CloudWatch alarms might seem like a difficult task. This is why the third type of scaling policy exists—target tracking.

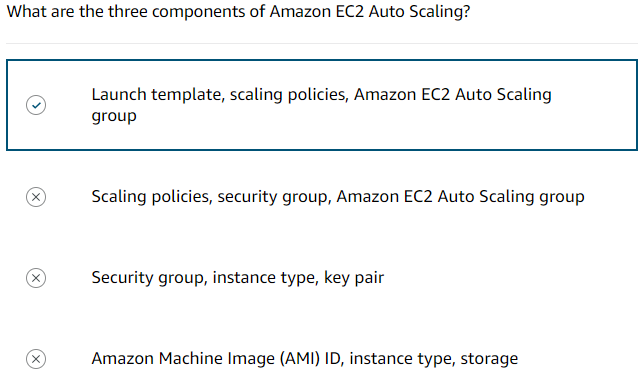
##### **TARGET TRACKING SCALING POLICY**

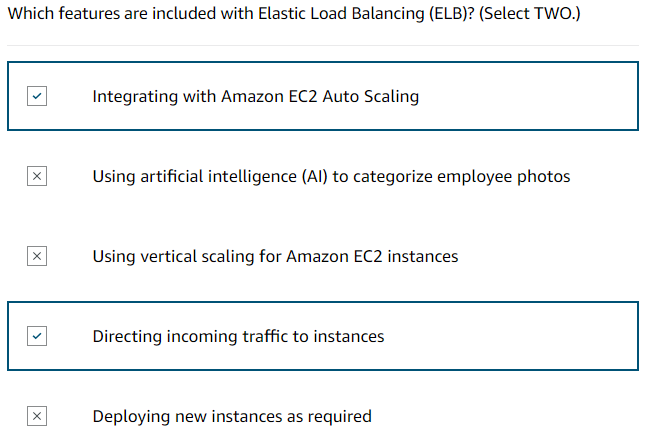
If your application scales based on average CPU utilization, average network utilization (in or out), or request count, then this scaling policy type is the one to use. All you need to provide is the target value to track, and it automatically creates the required CloudWatch alarms.

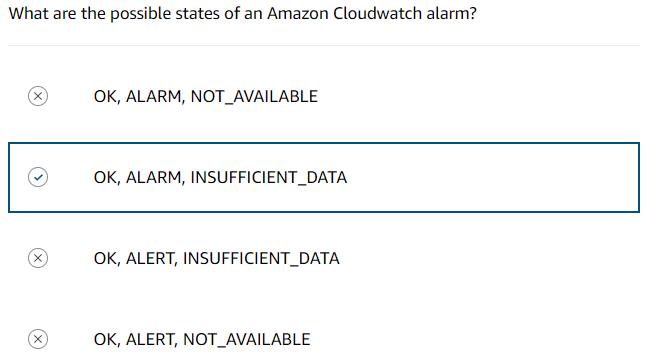
### **Resources**

* AWS website: [Amazon EC2 Auto Scaling](https://aws.amazon.com/ec2/autoscaling/)
* AWS website: [Amazon EC2 Auto Scaling FAQs](https://aws.amazon.com/ec2/autoscaling/faqs/)
* AWS user guide: [Set capacity limits on your Auto Scaling group](https://docs.aws.amazon.com/autoscaling/ec2/userguide/asg-capacity-limits.html)
* AWS user guide: [Step and simple scaling policies for Amazon EC2 Auto Scaling](https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html)
* AWS user guide: [Target tracking scaling policies for Amazon EC2 Auto Scaling](https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-target-tracking.html)
* AWS user guide: [Create an Auto Scaling group using a launch template](https://docs.aws.amazon.com/autoscaling/ec2/userguide/create-asg-launch-template.html)

## **6.6. Assessment**







# **Innovation with AWS**

### **What is Amazon CodeWhisperer**

Amazon CodeWhisperer is a machine learning-powered code generator that provides you with code recommendations in real time. As you write code, Amazon CodeWhisperer analyses your code and comments as you write code in your integrated development environment (IDE). CodeWhisperer then automatically generates suggestions based on your existing code and comments. Additionally, CodeWhisperer analyzes the surrounding code, ensuring the generated code matches your style, naming conventions, and seamlessly integrates into the existing context.

### **Installation**

Developers can easily access CodeWhisperer by downloading and installing the AWS Toolkit IDE extension. You can also activate CodeWhisperer from directly within the AWS Lambda and AWS Cloud9 console code editors.

### **Compatibility**

CodeWhisperer seamlessly integrates with popular tools like Visual Studio (VS) Code, JetBrains IDEs (IntelliJ IDEA, PyCharm, etc.), Amazon SageMaker Studio, JupyterLab, AWS Cloud9, and the AWS Lambda console.

### **Support**

It supports a wide range of programming languages and development environments, including Python, Java, JavaScript, TypeScript, C#, Go, Rust, PHP, Ruby, Kotlin, C, C++, Shell scripting, SQL, and Scala.

### **Code Generation**

User actions will vary depending on the tools you're integrating with. Take a look at the table below for an example of user actions available in VS Code.

|  |  |
| --- | --- |
| **User actions** | |
| Previous and Next recommendation | Left arrow and Right arrow |
| Accept a recommendation | Tab |
| Reject a recommendation | ESC |
| Manually trigger code generation when typing a comment | MacOS [option]-[c], Windows [Alt]-[c] |

### **Resources**

* AWS Website: [Getting started(opens in a new tab)](https://aws.amazon.com/codewhisperer/resources/)
* AWS Skill Builder: [Amazon CodeWhisperer - Getting started(opens in a new tab)](https://explore.skillbuilder.aws/learn/course/internal/view/elearning/16405/amazon-codewhisperer-getting-started)
* AWS Workshop: [Amazon CodeWhisperer workshop(opens in a new tab)](https://catalog.us-east-1.prod.workshops.aws/workshops/a33a5d69-1417-4d5f-acc9-ae5c7fba665b/en-US/)
* AWS Workshop: [Learn how to build an event-driven serverless app](https://catalog.us-east-1.prod.workshops.aws/workshops/63320e83-6abc-493d-83d8-f822584fb3cb/en-US/)

# **Assessment**

