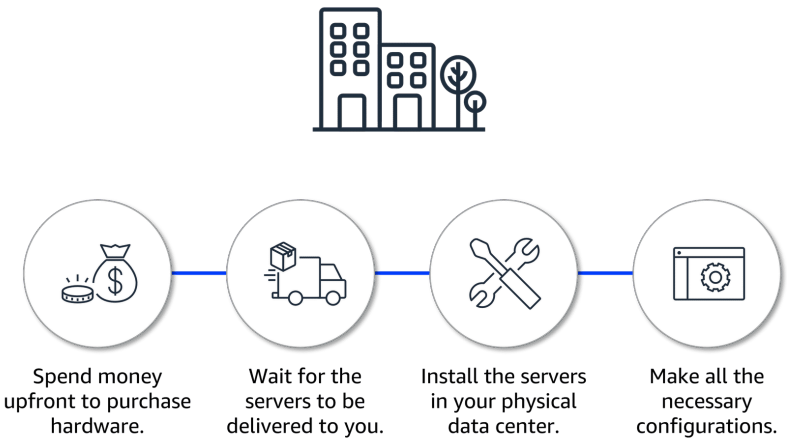
1. Summary: Introduction to Amazon EC2

This lesson introduces Amazon Elastic Compute Cloud (Amazon EC2), a core AWS service that provides on-demand compute capacity in the cloud.

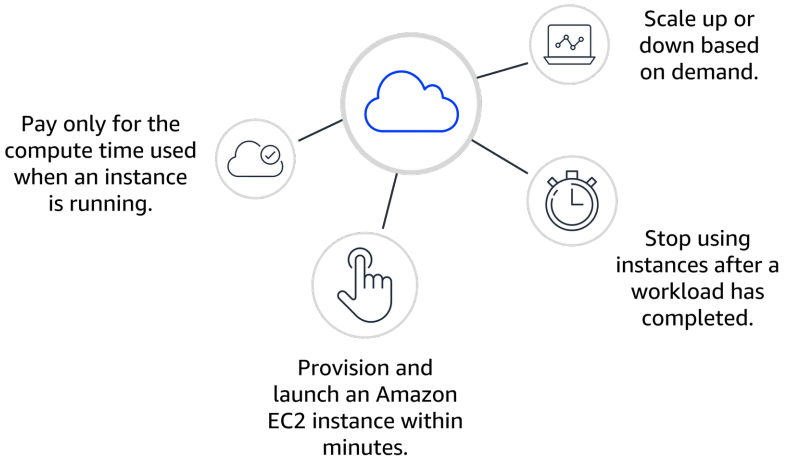
What is Compute?
Compute refers to the processing power needed to run applications. In the cloud, this is provided through virtual machines (VMs), which are essentially virtual servers that you can use without owning the physical hardware. In AWS, these virtual servers are called EC2 instances.

Benefits of EC2 vs. On-Premises Servers
Using EC2 instances offers significant advantages over managing traditional, physical servers in your own data center.

Challenges of on-premises resources



Benefits of using Cloud Resources



Feature	On-Premises Resources	Amazon EC2 (Cloud Resources)
Cost	High upfront investment in hardware. Fixed costs regardless of usage.	Pay only for what you use (variable cost). No upfront investment.
Speed	Time-consuming process (purchasing, delivery, installation).	Fast and convenient. Launch servers in minutes.
Flexibility	Locked into a specific capacity. Scaling is slow and expensive.	Highly flexible. Easily scale resources up or down (**vertical scaling**).
Management	You are responsible for all hardware maintenance and management.	AWS manages the underlying hardware for you.

How EC2 Works: Multi-tenancy and the Hypervisor

- **Multi-tenancy:** This is the concept where a single physical host machine is shared by multiple, isolated virtual machines (instances). These instances can belong to different customers.
- **Hypervisor:** A piece of software that runs on the host machine to manage the physical resources and ensure that each VM is completely isolated from the others. **AWS manages the host machine and the hypervisor**, so you don't have to worry about this layer.

Key Features of EC2

- **Full Control:** You choose the **Operating System** (Linux or Windows) and can install any software you need.
- **Resizable:** You can change the size of an instance (CPU, memory) as your needs change. This is known as **vertical scaling**.
- **Configurable Networking:** You control who can access your instance by configuring network and security settings.

The Basic Workflow for Using an EC2 Instance

1. **Launch an instance:** You start by selecting an **Amazon Machine Image (AMI)**, which is a template that contains the operating system and software configuration. You also choose an **instance type**, which defines the hardware resources (CPU, memory).
2. **Connect to the instance:** Once running, you can connect to it securely using tools like **SSH** for Linux or **Remote Desktop Protocol (RDP)** for Windows.

3. **Use the instance:** After connecting, you can use it just like a regular server: run commands, install applications, store files, etc.

2. Summary: Amazon EC2 Instance Types

This lesson explains that Amazon EC2 is not a one-size-fits-all service. It offers a wide variety of **instance types**, each optimized for specific kinds of workloads, much like a coffee shop needs different machines (espresso, drip, cold brew) to serve different customer orders efficiently.

Instance types are grouped into **instance families**, which provide different combinations of CPU, memory, storage, and networking capacity. Choosing the right instance type helps you optimize both performance and cost.

The Five Main EC2 Instance Families

1. **General Purpose**
    - **Description:** Provides a balanced mix of compute, memory, and networking resources.
    - **Use Cases:** Ideal for a wide variety of diverse workloads like web servers, code repositories, and small to medium databases. They are a great starting point if you're unsure about your application's specific performance needs.
2. **Compute Optimized**
    - **Description:** Optimized for applications that require high-performance processors.
    - **Use Cases:** Perfect for compute-intensive tasks such as high-performance computing (HPC), batch processing, scientific modeling, gaming servers, and machine learning inference.
3. **Memory Optimized**
    - **Description:** Designed to deliver fast performance for workloads that process large datasets in memory.
    - **Use Cases:** Ideal for memory-intensive tasks like high-performance databases, real-time big data analytics, and in-memory caches.
4. **Accelerated Computing**
    - **Description:** Uses hardware accelerators, or co-processors (like GPUs), to perform specific functions more efficiently than is possible with CPUs alone.
    - **Use Cases:** Suited for tasks like graphics processing, data pattern matching, floating-point calculations, and machine learning training.
5. **Storage Optimized**
    - **Description:** Designed for workloads that require high, sequential read and write access to very large datasets on local storage.
    - **Use Cases:** Ideal for I/O-intensive applications like large-scale databases, data warehousing, and distributed file systems.

Choosing an Instance Size Within each family, you can also select an **instance size** (e.g., small, medium, large, xlarge). A larger size provides more CPU, memory, and storage but also costs more. It's important to find the right balance between performance and cost to avoid paying for resources you don't need.

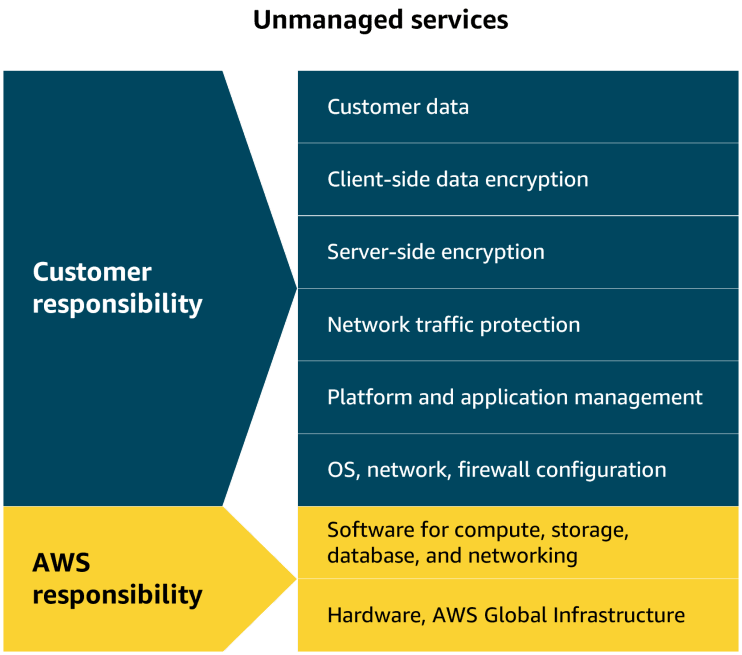
Key Takeaway: The cloud provides the flexibility to change your instance type or size if your initial choice isn't optimal. You can adapt quickly as your needs evolve.

3. Summary: How to Provision AWS Resources

This lesson explains that every action you take in AWS—from launching a server to configuring a setting—is an **API (Application Programming Interface) call**. It then covers the three primary methods you can use to make these API calls and interact with AWS services.

Three Ways to Interact with AWS Services

1. **AWS Management Console**
    - **What it is:** A web-based, graphical user interface (GUI) that allows you to manage AWS resources visually through your browser.
    - **Good for:** Beginners, visual tasks, setting up test environments, viewing bills, and managing non-technical tasks. It's the easiest way to get started and learn about services.
    - **Limitation:** Relying on manual point-and-click actions for repetitive tasks can be slow and prone to human error.
2. **AWS Command Line Interface (CLI)**
    - **What it is:** A tool that lets you manage AWS services from a text-based terminal or command line.
    - **Good for:** Advanced users and developers who want to automate tasks by writing scripts. It allows for efficient and repeatable management of resources.
3. **AWS Software Development Kits (SDKs)**
    - **What it is:** A set of tools and libraries that allow you to interact with AWS services directly from your application code, using various programming languages (like Python, Java, C++, etc.).
    - **Good for:** Developers who need to integrate AWS functionality directly into their applications.



Compute and the Shared Responsibility Model This lesson revisits the Shared Responsibility Model, specifically in the context of compute services, and introduces the concept of **managed vs. unmanaged services**.

- **Unmanaged Service:** A service where the customer has a high degree of control and responsibility.
- **Amazon EC2 is an Unmanaged Service:** When you use EC2, you are responsible for:
    - Managing the guest **Operating System (OS)**, including installing patches and updates.
    - Configuring **firewalls** (known as Security Groups in AWS).
    - Managing your applications and data security.
    - All necessary security configurations on the instance itself.

In this model, AWS is still responsible for the security *of* the cloud (the physical hardware and hypervisor), but you are responsible for security *in* the cloud (the OS, applications, and data on your EC2 instance).

4. Summary: Demo: Launching an Amazon EC2 Instance

This lesson provides a practical, step-by-step walkthrough of how to launch a basic web server using an Amazon EC2 instance through the AWS Management Console.

Key Configuration Steps in the Demo

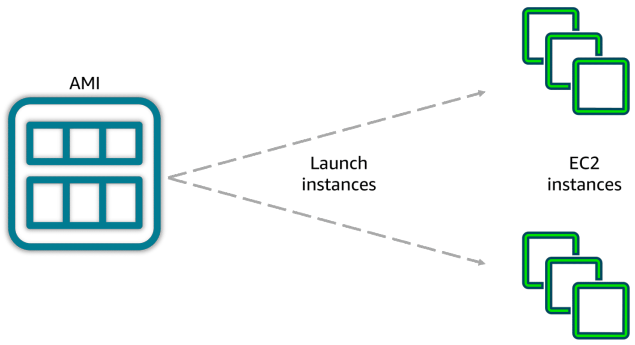
1. **Name the Instance:** The first step is to give the instance a descriptive name so it can be easily identified later.

2. **Choose an Amazon Machine Image (AMI):** An AMI is a template that defines the operating system and pre-installed software. The demo uses the default **Amazon Linux AMI**, which is suitable for a general-purpose web server.
3. **Select an Instance Type:** This determines the instance's hardware resources (CPU, memory). The demo selects the **t2.micro** type, which is a small instance eligible for the AWS Free Tier.
4. **Configure a Key Pair:** A key pair is used for secure SSH login. It consists of a public key (placed on the EC2 instance) and a private key (which you download and keep safe).
5. **Set Network Settings:** This configures how the instance can be accessed. Since it's a web server, the demo configures the settings to **allow HTTP traffic from the internet**.
6. **Configure Storage:** This step attaches storage to the instance. The demo allocates an **8 GB gp3 Elastic Block Store (EBS) volume**, which serves as the instance's hard drive.
7. **Use User Data:** The **User Data** field allows you to run a script automatically when the instance first launches. The demo uses a script to install and start the **Nginx web server**, turning the generic Linux instance into a functioning web server.

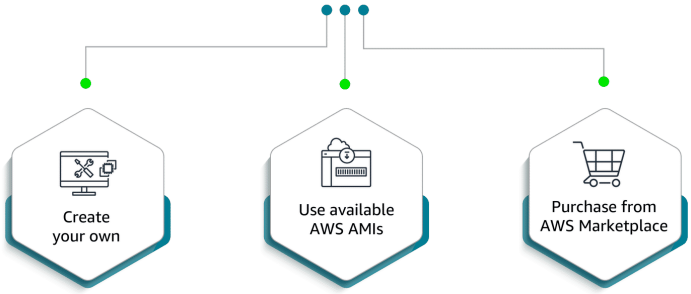
After launching, the demo confirms success by navigating to the instance's public IP address in a web browser, which displays the Nginx welcome page.

The Role of the Amazon Machine Image (AMI) The lesson further explains the importance of AMIs:

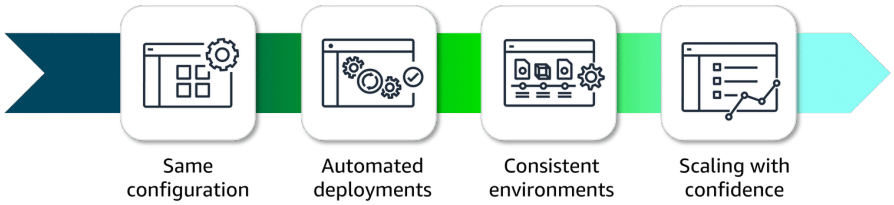
- **What it is:** An AMI is a pre-built virtual machine image containing everything needed to start an instance, including the OS, storage configuration, and any additional software.



- **How to get them:**
    1. **Create your own** custom AMIs.
    2. Use **pre-configured AMIs** provided by AWS.
    3. Purchase AMIs from the **AWS Marketplace** from third-party vendors.



- **Key Benefit - Repeatability:** AMIs ensure that every instance you launch from them is identical. This consistency is crucial for creating stable development/testing environments and for efficiently scaling applications, as it reduces errors and simplifies management.



5. Summary: Amazon EC2 Pricing

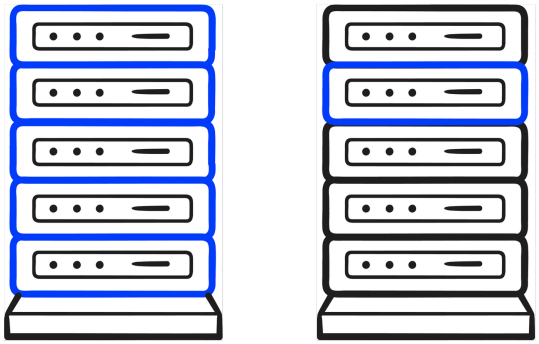
This lesson covers the various pricing options for Amazon EC2, allowing you to choose the most cost-effective solution for different workloads and usage patterns.

Primary Amazon EC2 Pricing Options

1. **On-Demand Instances**
    - **How it works:** Pay for compute capacity by the hour or second with no long-term commitments or upfront payments.
    - **Best for:** Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted. It's also ideal for getting started, testing, and development.
2. **Savings Plans**
    - **How it works:** Receive a significant discount (up to 72%) in exchange for a commitment to a consistent amount of usage (measured in $/hour) over a 1 or 3-year term.
    - **Best for:** Predictable and consistent workloads. This is a very flexible option as the savings automatically apply across different instance families, sizes, OS, and even other services like AWS Fargate and Lambda.
3. **Reserved Instances (RIs)**
    - **How it works:** Commit to a specific instance family in a specific region for a 1

or 3-year term to receive a large discount (up to 75%).
    - **Best for:** Steady-state or predictable usage where you know you'll need a specific type of instance for a long period. Payment options include all upfront, partial upfront, or no upfront.
4. **Spot Instances**
    - **How it works:** Request spare, unused EC2 capacity at a steep discount (up to 90% off the On-Demand price).
    - **The Catch:** AWS can reclaim the instance at any time with a two-minute warning.
    - **Best for:** Fault-tolerant, flexible, and interruptible workloads, such as batch processing, data analysis, or testing environments.
5. **Dedicated Hosts**
    - **How it works:** Pay for a physical server that is fully dedicated to your use.
    - **Best for:** Workloads with specific compliance, security, or software licensing requirements that demand physical isolation and control over instance placement.
6. **Dedicated Instances**
    - **How it works:** Pay for instances that run on hardware dedicated solely to your account.
    - **Best for:** Workloads that require physical isolation from other AWS customers but do not need the level of control over server placement that Dedicated Hosts provide.

Dedicated Hosts vs. Dedicated Instances It's important to understand the difference between these two options for physical isolation:



Dedicated Host Dedicated Instance

- **Dedicated Hosts:** You get an entire physical server for your exclusive use, giving you **complete control** over instance placement.
- **Dedicated Instances:** Your instances run on hardware that is dedicated to your account, providing physical isolation from other customers, but you **do not control** which specific server they run on.

Other Cost Optimization Tools

- **Capacity Reservations:** You can reserve compute capacity in a specific Availability Zone for any duration. This ensures you will always have access to that capacity when you need it for critical workloads. You pay the standard On-Demand rate for the reservation, whether you run instances in it or not.
- **Reserved Instance (RI) Flexibility:** RIs are flexible. The discount can automatically apply to different instance sizes within the same family and across different Availability Zones within the same Region.

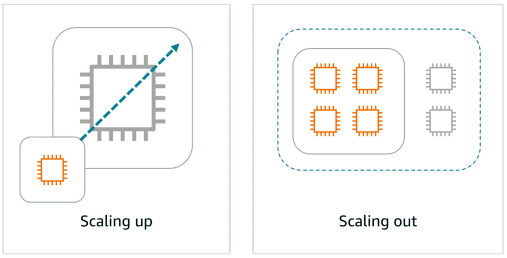
6. Summary: Scaling Amazon EC2

This lesson explains how AWS helps businesses handle fluctuating demand by automatically adjusting compute capacity, ensuring both customer satisfaction and cost efficiency.

Scalability vs. Elasticity

- **Scalability:** The ability of a system to handle a growing amount of work. It's about planning for long-term growth by adding more resources.
- **Elasticity:** The ability to automatically acquire resources when you need them and release them when you don't. It's about reacting to real-time changes in demand.

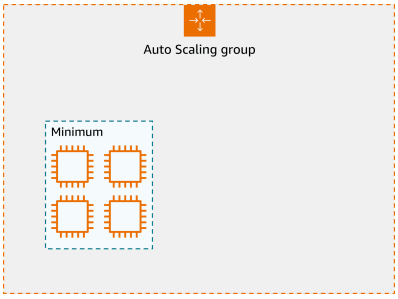
Two Ways to Scale



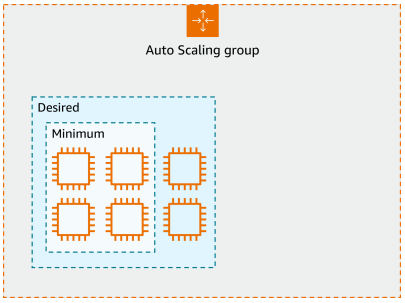
1. **Vertical Scaling (Scaling Up/Down):**
    - **What it is:** Increasing the power of an existing instance (e.g., adding more CPU or memory).
    - **Analogy:** Giving a single barista a more powerful coffee machine. This can help, but one person can still only do so much.
2. **Horizontal Scaling (Scaling Out/In):**
    - **What it is:** Adding more instances to your resource pool to handle work in parallel.
    - **Analogy:** Adding more baristas to the coffee shop. This is often more effective for handling a rush of customers, as multiple orders can be processed at the same time.

Amazon EC2 Auto Scaling This is the AWS service that automates the process of horizontal scaling. It automatically adds or removes EC2 instances in response to changing application demand.

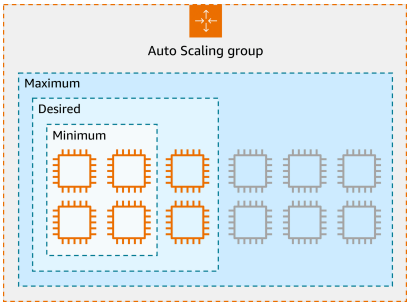
- **How it works:** It uses **Amazon CloudWatch** to monitor metrics (like CPU utilization or network traffic). When a metric crosses a certain threshold, it triggers a scaling action.
- **Auto Scaling Groups:** To use EC2 Auto Scaling, you create an **Auto Scaling group**, which is a collection of EC2 instances managed as a logical unit. You configure it with three key settings:
    1. **Minimum Capacity:** The smallest number of instances you always want running.



    2. **Desired Capacity:** The number of instances the group will aim to have running at any given time (it will scale to meet this number).



    3. **Maximum Capacity:** The largest number of instances the group is allowed to scale out to, which prevents over-provisioning and controls costs.



Key Takeaway: With Amazon EC2 Auto Scaling, you can create a cost-effective and highly available architecture that always has the right number of instances to meet demand—no more, no less.

7. Summary: Directing Traffic with Elastic Load Balancing

This lesson explains how to solve the challenge of distributing incoming traffic evenly across multiple EC2 instances, especially when using Auto Scaling.

The Problem: Uneven Traffic Distribution Even if you have multiple EC2 instances running (thanks to Auto Scaling), incoming user requests might all go to a single instance, overloading it while other instances sit idle.

- **Coffee Shop Analogy:** Imagine a coffee shop with three cashiers, but all the customers line up for just one of them. The other two cashiers are free, but the line for the popular cashier gets very long. To solve this, the shop adds a **host** at the entrance to direct incoming customers to the shortest line, ensuring the workload is balanced.

The Solution: Elastic Load Balancing (ELB)

Elastic Load Balancing (ELB) acts as the "host" for your application. It's a managed AWS service that automatically distributes incoming application traffic across multiple targets, such as EC2 instances.

- **How it works:** ELB serves as a single point of contact for all incoming traffic. It receives requests and then routes them to the available EC2 instances based on a chosen routing method (e.g., to the instance with the least connections).
- **"Elastic":** The load balancer itself can automatically scale its capacity up or down to handle changes in incoming traffic.

ELB and EC2 Auto Scaling: A Powerful Combination ELB and EC2 Auto Scaling are separate services, but they are designed to work together to create a scalable and highly available application.

1. A user sends a request to your application.
2. The request first hits the **Elastic Load Balancer**.
3. The ELB distributes the request to one of the healthy EC2 instances in your **Auto Scaling group**.
4. If traffic increases, **EC2 Auto Scaling** adds more instances. The new instances automatically register with the ELB and start receiving traffic.
5. If traffic decreases, **EC2 Auto Scaling** removes instances. The ELB stops sending traffic to the instances that are being terminated.

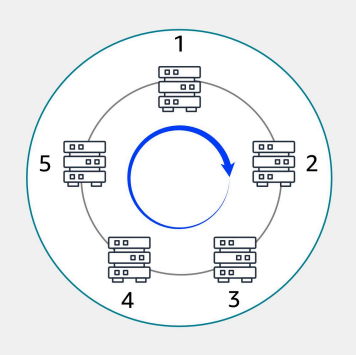
This combination ensures that your application can handle fluctuating demand efficiently while maintaining consistent performance.

Key Benefits of ELB

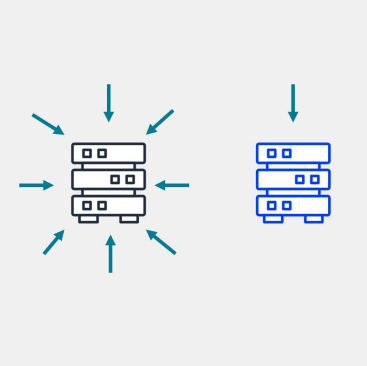
- **Efficient Traffic Distribution:** Prevents any single instance from being overloaded.
- **Automatic Scaling:** The load balancer itself scales to meet traffic demands.
- **Simplified Management:** Decouples your application tiers. For example, your front-end servers only need to know the address of the load balancer, not the individual addresses of all the back-end servers. ELB handles the complexity of tracking which instances are available.
- **High Availability:** By routing traffic only to healthy instances, ELB increases the fault tolerance of your application.

Routing Methods To optimize traffic distribution, ELB uses several routing methods:

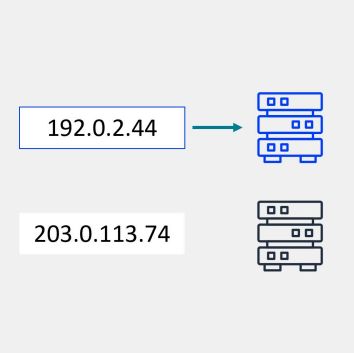
- **Round Robin:** Distributes traffic evenly across all available servers in a cyclic manner.



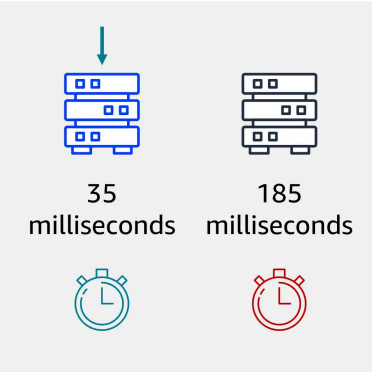
- **Least Connections:** Routes traffic to the server with the fewest active connections, maintaining a balanced load.



- **IP Hash:** Uses the client's IP address to consistently route traffic to the same server (useful for persistent sessions).



- **Least Response Time:** Directs traffic to the server with the fastest response time, minimizing latency.

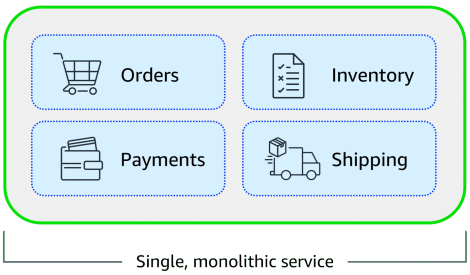


8. Summary: Messaging and Queuing

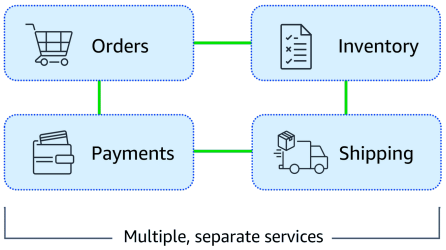
This lesson introduces the concept of decoupling application components to improve reliability and resilience, focusing on two key AWS services: Amazon SQS and Amazon SNS.

Tightly Coupled vs. Loosely Coupled Architectures

- **Tightly Coupled (Monolithic):** Application components communicate directly with each other. If one component fails, it can cause a chain reaction, leading to the failure of other components or the entire system.
    - **Analogy:** A cashier hands a coffee order directly to a barista. If the barista is busy or on break, the cashier is stuck and cannot take the next order.



- **Loosely Coupled (Microservices):** Application components are independent. If one component fails, the others can continue to function normally. This is achieved by placing a buffer or message system between them.
    - **Analogy:** The cashier places the coffee order on an order board (a queue). The barista can then pick up the next order from the board whenever they are ready. The cashier is never blocked, and no orders are lost.



AWS Services for Decoupling

1. ***Amazon Simple Queue Service (Amazon SQS)***
    - ***What it is:*** *A fully managed message queuing service. It allows you to send, store, and receive messages between software components at any volume.*
    - ***How it works:*** *One component (a producer) sends a message to an SQS* ***queue****. Another component (a consumer) retrieves the message from the queue to process it. The message is stored reliably in the queue until it is processed. This is a* ***pull-based*** *system.*
    - ***Use Case:*** *Decoupling application tiers. For example, if a front-end service needs to send a task to a back-end processing service, it can place the task in an SQS queue.*
2. ***Amazon Simple Notification Service (Amazon SNS)***
    - ***What it is:*** *A fully managed publish-subscribe (pub/sub) messaging service.*
    - ***How it works:*** *A component (a publisher) sends a message to an SNS* ***topic****. The topic then immediately pushes that message out to all the different subscribers of that topic. Subscribers can be web servers, email addresses, SMS, or other AWS services. This is a* ***push-based*** *system.*
    - ***Use Case:*** *Fanning out notifications. For example, sending an order*

confirmation to a customer via email and SMS simultaneously.

3. ***Amazon EventBridge***
    - ***What it is:*** *A serverless event bus service that helps connect different parts of an application using events.*
    - ***How it works:*** *EventBridge routes events from various* ***sources*** *(like custom apps, AWS services, or third-party software) to relevant* ***targets*** *(like Lambda functions, SQS queues, etc.). It simplifies the process of receiving, filtering, transforming, and delivering events.*
    - ***Use Case:*** *Building event-driven architectures. For example, in a food delivery app, an "order placed" event can be routed simultaneously to the payment service, restaurant notification system, and inventory management system.*

Key Difference: SQS vs. SNS

- ***SQS (Queue):*** *Messages are held in a queue until a consumer* ***pulls*** *them for processing. It's a* ***one-to-one*** *communication model (one message is processed by one consumer).*
- ***SNS (Topic):*** *Messages are immediately* ***pushed*** *to all subscribers. It's a* ***one-to-many*** *communication model (one message is sent to many subscribers).*