Summary: Introduction to Serverless Computing

This lesson introduces the spectrum of AWS compute services, categorized by the level of management and control they offer. This ranges from unmanaged services that provide maximum control, to serverless services that offer maximum convenience by abstracting away the underlying infrastructure.

The Spectrum of AWS Compute Services

The choice of service depends on the balance a user needs between fine-grained control and ease of use.

- **Coffee Shop Analogy:**
  - **Unmanaged:** A high-end, fully customizable espresso machine. It offers total control over every aspect of the coffee-making process (grind, temperature, pressure) but requires significant effort, maintenance, and expertise.
  - **Managed/Serverless:** A coffee maker that uses pods. It's incredibly convenient—just pop in a pod and press a button. You have less control over the final product, but it saves a lot of time and effort.

1. Unmanaged Services

- **Description:** Services where the customer has a high degree of control over the infrastructure and is responsible for many management tasks.
- **Primary Example: Amazon EC2**.
- **Responsibilities:** While AWS manages the physical hardware and hypervisor (security *of* the cloud), the **customer is responsible for**:
  - Managing the guest Operating System (OS), including patching and updates.
  - Scaling the instances.
  - All application and security configurations *on* the instance.

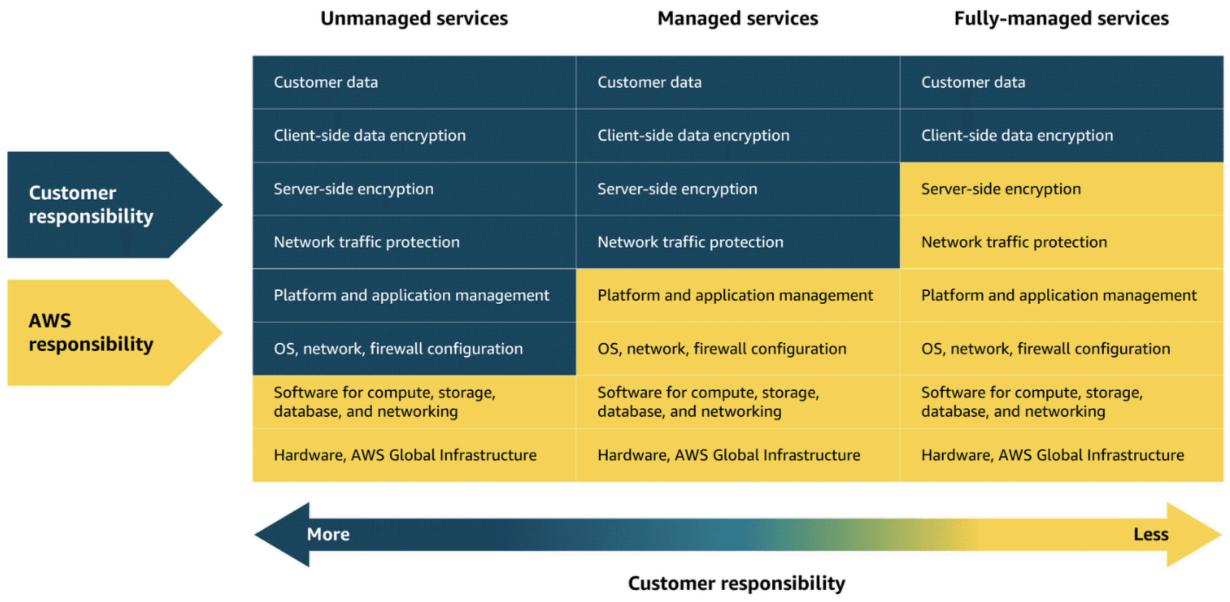
2. Managed Services

- **Description:** Services where AWS handles more of the operational overhead, allowing the customer to focus more on their application and less on infrastructure management.
- **Examples:** Elastic Load Balancing (ELB), Amazon SQS, Amazon SNS.
- **Responsibilities:** The customer is primarily responsible for **configuring the service** to meet their needs. AWS manages the underlying servers, scaling, high availability, and maintenance.

3. Serverless (Fully-Managed) Services

- **Description:** The highest level of abstraction. With serverless computing, you cannot see or access the underlying infrastructure at all. AWS fully manages provisioning, scaling, high availability, and maintenance.
- **Primary Example: AWS Lambda**.
- **Responsibilities:** The customer is responsible only for their **application code**. All infrastructure management is handled by AWS. This allows developers to focus entirely

on writing and deploying code without worrying about servers.


Unmanaged services
Managed services
Fully-managed services
Customer responsibility
AWS responsibility
Customer data
Client-side data encryption
Server-side encryption
Network traffic protection
Platform and application management
OS, network, firewall configuration
Software for compute, storage, database, and networking
Hardware, AWS Global Infrastructure
More
Less
Customer responsibility

Key Takeaway: The Shared Responsibility Model in Compute

The type of compute service you choose directly impacts your role in the Shared Responsibility Model.

Service Type	AWS Responsibility	Customer Responsibility
Unmanaged (EC2)	Physical hardware, hypervisor	OS, scaling, patching, application security, data
Managed (ELB)	Underlying servers, scaling, HA, maintenance	Configuration of the service
Serverless (Lambda)	All infrastructure, scaling, HA, maintenance	Application code security and management

AWS offers this range of services to cater to different workloads and requirements, giving you the flexibility to choose the right balance of customization and convenience for your specific needs.

Summary: AWS Lambda

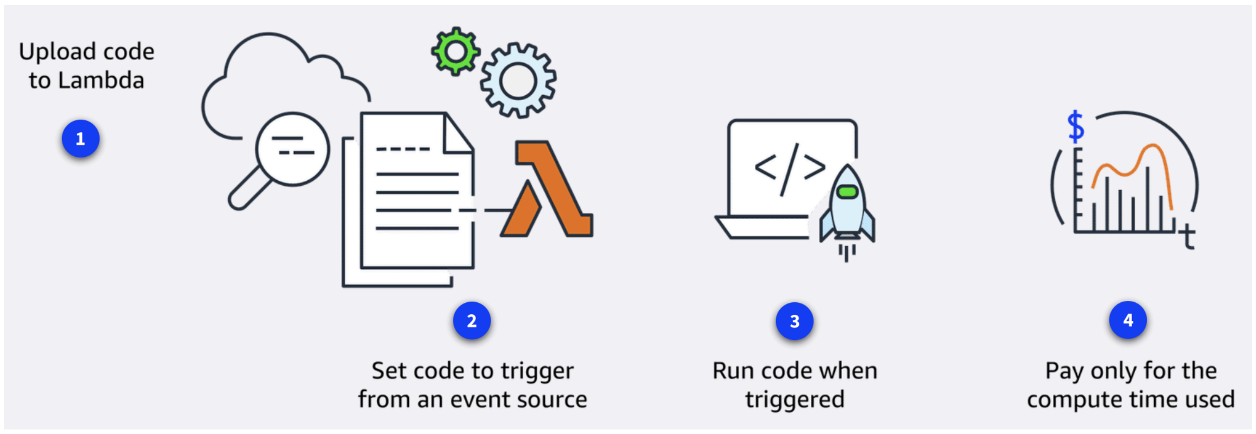
This lesson dives into AWS Lambda, a core serverless compute service that allows you to run code without provisioning or managing any servers, also known as a "Function as a Service"

(FaaS).

What is AWS Lambda? Lambda is an event-driven compute service. Instead of running on a persistent server, your code is packaged into a **Lambda function** that only executes in response to an event, known as a **trigger**.

- **Analogy: The Crab Classifier App** Imagine an app where a user uploads a photo of a crab. Instead of having a server always running and waiting for uploads, a Lambda function can be **triggered** by the file upload event. The function then runs, classifies the crab, and shuts down. You only pay for the few seconds it took to process the image.

How Lambda Works: The Trigger-Function Model



The process is straightforward:

1. **Upload Code:** You package your application code into a Lambda function.
2. **Set a Trigger:** You configure an event source to trigger the function. This trigger can be an upload to an Amazon S3 bucket, a new message in an SQS queue, an API call, a schedule, or an event from many other AWS services.
3. **Code Runs When Triggered:** Lambda automatically executes your function only when the trigger event occurs. AWS handles all the underlying infrastructure, scaling, patching, and high availability.
4. **Pay Only for Compute Time:** You are billed only for the time your code is actually running, measured in milliseconds. There is no charge when the function is idle.

Key Characteristics of Lambda

- **Serverless:** No servers to provision or manage. AWS handles all the infrastructure for you.
- **Automatic Scaling:** Lambda automatically scales your application by running code in response to each trigger. It can scale from a few requests per day to thousands per second.
- **Event-Driven:** Functions are designed to respond to specific events, making them ideal for building reactive, microservices-based architectures.
- **Time Limit:** Lambda functions have a maximum execution duration of **15 minutes**. For longer-running processes, other services might be more suitable.
- **Language Support:** Lambda supports numerous programming languages (like Python, Node.js, Java) through **runtimes**. You can also provide your own custom runtime for other languages.

Common Use Cases

Lambda is ideal for short, event-driven tasks.

- **Real-time Image Processing:** A social media app can use Lambda to automatically resize and apply filters to images as soon as they are uploaded.
- **Personalized Content Delivery:** A news app can trigger a Lambda function to fetch and process articles, tailoring recommendations for a user when they open the app.
- **Real-time Event Handling:** An online game can use Lambda to process in-game events like scoring a point or updating a leaderboard in real-time.

Demo Summary: Connecting SQS to Lambda

The demo illustrated a practical, event-driven workflow.

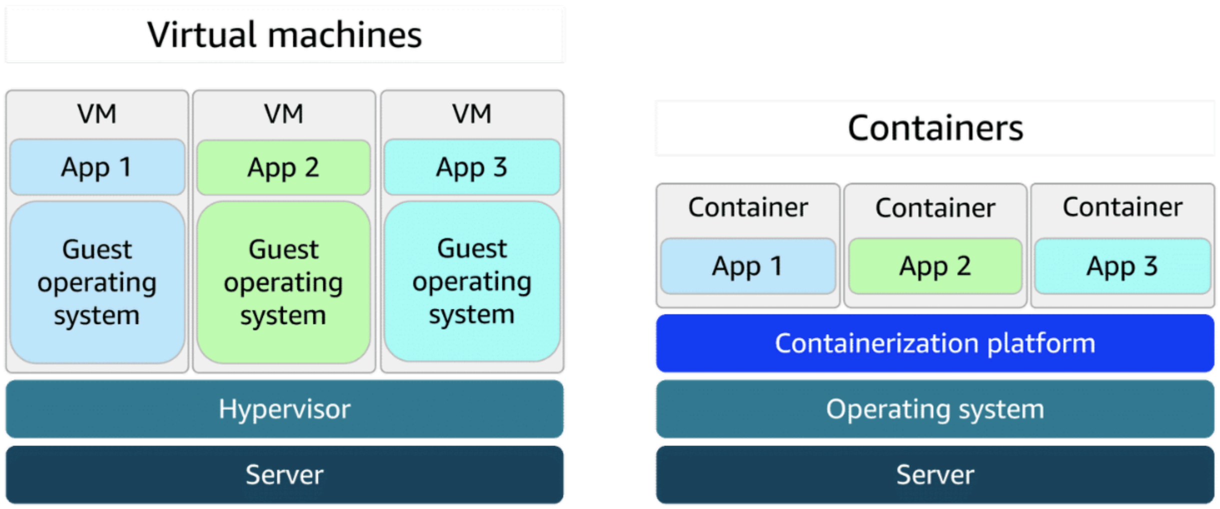
- **Goal:** To have a Lambda function automatically process messages from an Amazon SQS queue.
- **Architecture:** `SQS Queue -> Trigger -> Lambda Function`
- **Steps:**
  - Messages were added to an existing SQS queue.
  - A Lambda function was created using a pre-configured **blueprint** designed for SQS processing.
  - An **Execution Role** (IAM role) was created to grant the Lambda function the necessary **permissions** to read messages from the SQS queue.
  - The SQS queue was configured as the **trigger** for the Lambda function.
- **Verification:**
  - The messages immediately disappeared from the SQS queue, indicating they had been processed.
  - The logs for the Lambda function, viewed in **Amazon CloudWatch Logs**,

showed the content of the messages, confirming that the function ran successfully and processed them.

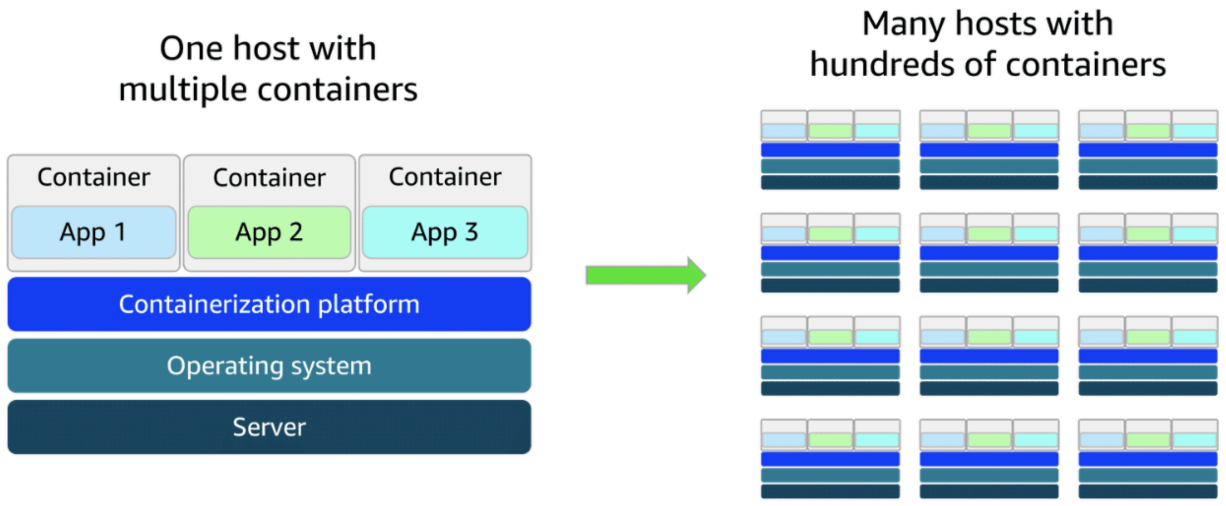
Summary: Containers and Orchestration on AWS

This lesson introduces containers as a solution to the common development problem of applications working on one machine but failing on others. It then covers the AWS services used to store, manage, and run containerized applications at scale.

The Challenge: The "It Works on My Machine" Problem Deployments often fail because of inconsistencies between a developer's local environment and the production environment. **Containers** solve this by packaging an application's code, runtime, dependencies, and configurations into a single, portable unit. This ensures the application runs consistently everywhere.



- **Containers vs. Virtual Machines (VMs):**
    - **Containers:** Lightweight and fast because they share the host machine's operating system.
    - **VMs:** Heavier and slower because each VM runs a full, separate operating system on top of a hypervisor.


One host with multiple containers
Container
Container
Container
App 1
App 2
App 3
Containerization platform
Operating system
Server
Many hosts with hundreds of containers

The Need for Orchestration While running a few containers is easy, managing hundreds or thousands across multiple servers becomes incredibly complex. **Container orchestration** services automate the deployment, scaling, failure recovery, and management of containers at scale.

AWS Container Services

AWS provides a complete suite of services for a container workflow, which can be broken down into three categories: Registry, Orchestration, and Compute.

1. Registry: Where to Store Container Images

- **Amazon Elastic Container Registry (Amazon ECR):** A fully managed container registry used to store, manage, and deploy your container images. Orchestration tools pull images from ECR to run them.

2. Orchestration: How to Manage the Containers

AWS offers two primary orchestration services:

- **Amazon Elastic Container Service (Amazon ECS):** A streamlined, highly scalable, and AWS-native container orchestration service. It's ideal for users who want deep integration with other AWS services and a simpler management experience.
- **Amazon Elastic Kubernetes Service (Amazon EKS):** A managed service for running the open-source **Kubernetes** platform on AWS. It's ideal for users who are already familiar with Kubernetes or need the control, flexibility, and large community support that it offers, especially for large-scale or hybrid deployments.

3. Compute: Where the Containers Run

Both ECS and EKS need underlying compute resources to run the containers. You have two options, known as "launch types":

- **Amazon EC2 Launch Type:**
  - **What it is:** You run your containers on a cluster of EC2 instances that you provision and manage.
  - **Best for:** Workloads that require full control over the underlying infrastructure, specific hardware configurations, or custom networking.
- **AWS Fargate Launch Type:**
  - **What it is:** A **serverless** compute engine for containers. You don't need to provision or manage any servers; AWS handles all the underlying infrastructure for you.
  - **Best for:** Workloads where you want to focus only on your containers and not the servers. It offers maximum convenience and efficiency, as you only pay for the resources your containers consume.

The Complete Workflow:

1. **Build** your application and package it into a container image.
2. **Push** the container image to **Amazon ECR** for storage.
3. **Choose** an orchestration service (**Amazon ECS** or **Amazon EKS**) to define how your application should run.
4. **Choose** a compute option (**AWS Fargate** or **Amazon EC2**) to provide the power to run your containers.

Summary: Additional Compute Services

This lesson provides an overview of four purpose-built AWS compute services, each designed to streamline specific tasks and address particular use cases beyond general-purpose compute like Amazon EC2.

1. AWS Elastic Beanstalk

- **What it is:** A fully managed **Platform-as-a-Service (PaaS)** that simplifies the deployment, management, and scaling of web applications.
- **How it Works:** Developers simply upload their application code, and Elastic Beanstalk automatically handles the entire infrastructure deployment. This includes provisioning EC2 instances, configuring load balancing and Auto Scaling, and monitoring application health. While it automates management, you retain full control over the underlying AWS resources. It supports numerous platforms like Java, .NET, Python, Node.js, and Docker.
- **Good for:** Quickly deploying and managing web applications, APIs, mobile backends,

and microservices without needing to be an expert in infrastructure management.

2. AWS Batch

- **What it is:** A fully managed service designed to efficiently run hundreds of thousands of batch computing jobs on AWS.
- **How it Works:** AWS Batch dynamically provisions the optimal quantity and type of compute resources (like EC2 instances) based on the volume and specific requirements of the batch jobs submitted. It manages job scheduling and resource scaling, allowing you to focus on analysis rather than infrastructure.
- **Good for:** Large-scale, parallel workloads such as scientific computing, financial risk analysis, media transcoding, big data processing, and machine learning model training.

3. Amazon Lightsail

- **What it is:** A service designed to be the easiest way to get started with AWS for developers, small businesses, or anyone needing a simple virtual private server (VPS) solution.
- **How it Works:** Lightsail bundles everything you need to launch a project quickly—a virtual machine, SSD-based storage, data transfer, DNS management, and a static IP—into a simple, predictable, low-cost monthly plan. It provides a simplified management console separate from the main AWS console.
- **Good for:** Basic web applications, low-traffic websites, development and testing environments, blogs, and for users who are new to the cloud and want a straightforward experience.

4. AWS Outposts

- **What it is:** A fully managed service that offers a **hybrid cloud** solution by extending AWS infrastructure, services, APIs, and tools to virtually any on-premises data center or co-location space.
- **How it Works:** AWS delivers and installs a physical rack of AWS-designed hardware in your data center, which is then managed by AWS. This allows you to run AWS compute and storage services locally.
- **Good for:** Workloads that require very low latency to on-premises systems, local data processing, or need to meet strict data residency and sovereignty requirements. It provides a consistent development and operations experience across your on-premises and AWS cloud environments.