E-LEARNING

AWS Cloud Practitioner Essentials

Module 2 – Compute in the Cloud

Contents

Learning objectives

In this module, you will learn how to:

- Describe the benefits of Amazon EC2 at a basic level.
- Identify the different Amazon EC2 instance types.
- Differentiate between the various billing options for Amazon EC2.
- Summarize the benefits of Amazon EC2 Auto Scaling.
- Summarize the benefits of Elastic Load Balancing.
- Give an example of the uses for Elastic Load Balancing.
- Summarize the differences between Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Queue Service (Amazon SQS).
- Summarize additional AWS compute options.

Amazon Elastic Compute Cloud (Amazon EC2)

Amazon Elastic Compute Cloud (Amazon EC2) provides secure, resizable compute capacity in the cloud as Amazon EC2 instances.

Imagine you are responsible for the architecture of your company's resources and need to support new websites. With traditional on-premises resources, you have to do the following:

- Spend money upfront to purchase hardware.
- Wait for the servers to be delivered to you.
- Install the servers in your physical data centre.
- Make all the necessary configurations.

The time and money it takes to get up and running with on-premises resources is fairly high. When you own your own fleet of physical servers, you first have to do a bunch of research to see what type of servers you want to buy and how many you'll need. Then you purchase that hardware up front. You'll wait for multiple weeks or months for a vendor to deliver those servers to you. You then take them to a data centre that you own or rent to install them, rack and stack them, and wire them all up. Then you make sure that they are secure and powered up and then they're ready to be used. Only then can you begin to host your applications on top of these servers. The worst part is, once you buy these servers you are stuck with them whether you use them or not.

By comparison, with an Amazon EC2 instance you can use a virtual server to run applications in the AWS Cloud.

- You can provision and launch an Amazon EC2 instance within minutes.
- You can stop using it when you have finished running a workload.
- You pay only for the compute time you use when an instance is running, not when it is stopped or terminated.

- You can save costs by paying only for server capacity that you need or want.

How Amazon EC2 works

(1) Launch

First, you launch an instance. Begin by selecting a template with basic configurations for your instance. These configurations include the operating system, application server, or applications. You also select the instance type, which is the specific hardware configuration of your instance.

As you are preparing to launch an instance, you specify security settings to control the network traffic that can flow into and out of your instance. Later in this course, we will explore Amazon EC2 security features in greater detail.

(2) Connect

Next, connect to the instance. You can connect to the instance in several ways. Your programs and applications have multiple different methods to connect directly to the instance and exchange data. Users can also connect to the instance by logging in and accessing the computer desktop.

(3) Use

After you have connected to the instance, you can begin using it. You can run commands to install software, add storage, copy and organize files, and more.

Multi-tenancy

EC2 runs on top of physical host machines managed by AWS using virtualization technology. When you spin up an EC2 instance, you aren't necessarily taking an entire host to yourself. Instead, you are sharing the host with multiple other instances, otherwise known as virtual machines. And a **hypervisor running on the host machine** is responsible for **sharing the underlying physical resources between the virtual machines**.

This idea of **sharing underlying hardware** is called **multi-tenancy**. The hypervisor is responsible for coordinating this **multi-tenancy** and it is managed by AWS. The hypervisor is responsible for isolating the virtual machines from each other as they share resources from the host. This means EC2 instances are secure. Even though they may be sharing resources, one EC2 instance is not aware of any other EC2 instances also on that host. They are secure and separate from each other.

Amazon EC2 instance types

Amazon EC2 instance types are **optimized for different tasks**. When selecting an instance type, consider the specific needs of your workloads and applications. This might include requirements for compute, memory, or storage capabilities.

General purpose instances provide a balance of compute, memory, and networking resources. You can use them for a variety of workloads, such as:

- application servers
- gaming servers
- backend servers for enterprise applications
- small and medium databases

Suppose that you have an application in which the resource needs for compute, memory, and networking are roughly equivalent. You might consider running it on a general purpose instance because the application does not require optimization in any single resource area.

Compute optimized instances are ideal for compute-bound applications that benefit from **high-performance processors**. Like general purpose instances, you can use compute optimized instances for workloads such as web, application, and gaming servers.

However, the difference is compute optimized applications are ideal for high-performance web servers, **compute-intensive applications servers, and dedicated gaming servers**. You can also use compute optimized instances for **batch processing workloads** that require **processing many transactions** in a single group.

Memory optimized instances are designed to deliver fast performance for workloads that **process large datasets in memory**. In computing, memory is a temporary storage area. It holds all the data and instructions that a central processing unit (CPU) needs to be able to complete actions. Before a computer program or application is able to run, it is loaded from storage into memory. This preloading process gives the CPU direct access to the computer program.

Suppose that you have a workload that requires **large amounts of data to be preloaded** before running an application. This scenario might be a high-performance database or a workload that involves performing real-time processing of a large amount of unstructured data. In these types of use cases, consider using a memory optimized instance. Memory optimized instances enable you to run workloads with high memory needs and receive great performance.

Accelerated computing instances use hardware accelerators, or coprocessors, to perform some functions more efficiently than is possible in software running on CPUs. Examples of these functions include **floating-point number calculations, graphics processing, and data pattern matching.**

In computing, a **hardware accelerator** is a component that can expedite data processing. Accelerated computing instances are ideal for workloads such as **graphics applications, game streaming, and application streaming.**

Storage optimized instances are designed for workloads that require **high, sequential read and write access to large datasets on local storage**. Examples of workloads suitable for storage optimized instances include **distributed file systems, data warehousing applications, and high-frequency online transaction processing (OLTP)** systems.

In computing, the term input/output operations per second (**IOPS**) is a metric that measures the performance of a storage device. It indicates how many different input or output operations a device can perform in one second. Storage optimized instances are designed to deliver tens of thousands of low-latency, random IOPS to applications.

You can think of input operations as data put into a system, **such as records entered into a database**. An output operation is data generated by a server. An example of output might be the analytics performed on the records in a database. If you have an application that has a high IOPS requirement, a storage optimized instance can provide better performance over other instance types not optimized for this kind of use case.

Amazon EC2 pricing

With Amazon EC2, you pay only for the compute time that you use. Amazon EC2 offers a variety of pricing options for different use cases. For example, if your use case can withstand interruptions, you can save with Spot Instances. You can also save by committing early and locking in a minimum level of use with Reserved Instances.

On-Demand Instances are ideal for short-term, irregular workloads that cannot be interrupted. **No upfront costs or minimum contracts** apply. The instances run continuously until you stop them, and you pay for only the compute time you use.

Sample use cases for On-Demand Instances include developing and testing applications and running applications that have unpredictable usage patterns. On-Demand Instances are not recommended for workloads that last a year or longer because these workloads can experience greater cost savings using Reserved Instances.

This type of on-demand pricing is usually for when you get started and want to spin up servers to test out workloads and play around. You don't need any prior contracts or communication with AWS to use On-Demand pricing. You can also use them to get a **baseline for your average usage**, which leads us to our next pricing option, Savings Plan.

AWS offers **Savings Plans** for several compute services, including Amazon EC2. **Amazon EC2 Savings Plans** enable you to reduce your compute costs by **committing to a consistent amount of compute usage** (measured in dollars per hour) for a 1-year or 3-year term. This term commitment results in savings of up to 72% over On-Demand costs.

Any usage up to the commitment is charged at the discounted Savings Plan rate (for example, $10 an hour). Any usage beyond the commitment is charged at regular On-Demand rates.

Later in this course, you will review AWS Cost Explorer, a tool that enables you to visualize, understand, and manage your AWS costs and usage over time.

Reserved Instances are a billing discount applied to the use of **On-Demand Instances** in your account, and these are suited for **steady-state workloads or ones with predictable usage**. You can purchase Standard Reserved and Convertible Reserved Instances for a 1-year or 3-year term, and Scheduled Reserved Instances for a 1-year term. You realize greater cost savings with the 3-year option.

At the end of a Reserved Instance term, you can continue using the Amazon EC2 instance without interruption. However, you are charged On-Demand rates until you do one of the following:

- Terminate the instance.
- Purchase a new Reserved Instance that matches the instance attributes (instance type, Region, tenancy, and platform).

Spot Instances are ideal for workloads with **flexible start and end times**, or that can withstand interruptions. They allow you to request spare Amazon EC2 computing capacity for up to 90% off of the On-Demand price. The catch here is that **AWS can reclaim the instance at any time** they need it, giving you a two-minute warning to finish up work and save state. You can always resume later if needed. So when choosing Spot Instances, **make sure your workloads can tolerate being interrupted.** A good example of those are **batch workloads**.

Suppose that you have a background processing job that can start and stop as needed (such as the data processing job for a customer survey). You want to start and stop the processing job without affecting the overall operations of your business. If you make a Spot request and Amazon EC2 capacity is available, your Spot Instance launches. However, if you make a Spot request and Amazon EC2 capacity is unavailable, the request is not successful until capacity becomes available. The unavailable capacity might delay the launch of your background processing job.

After you have launched a Spot Instance, if capacity is no longer available or demand for Spot Instances increases, your instance may be interrupted. This might not pose any issues for your background processing job. However, in the earlier example of developing and testing applications, you would most likely want to avoid unexpected interruptions. Therefore, choose a different EC2 instance type that is ideal for those tasks.

Dedicated Hosts are physical servers with Amazon EC2 instance capacity that is fully dedicated to your use. These are usually for meeting certain compliance requirements and **nobody else will share tenancy of that host**.

You can use your existing per-socket, per-core, or per-VM software licenses to help maintain license compliance. You can purchase On-Demand Dedicated Hosts and Dedicated Hosts Reservations. Of all the Amazon EC2 options that were covered, Dedicated Hosts are the most expensive.

Scalability

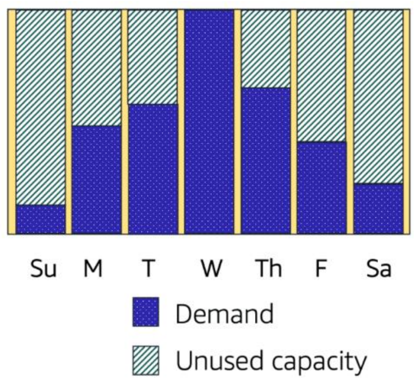
Scalability involves beginning with only the resources you need and designing your architecture to **automatically respond to changing demand by scaling out or in**. As a result, you pay for only the resources you use. You don’t have to worry about a lack of computing capacity to meet your customers’ needs.

Here is the on-premise data centre dilemma. If your business is like 99% of all businesses out in the world, your customer workloads vary over time: perhaps over a simple 24 hour period, or you might have seasons where you're busy, and weeks that are not in demand. If you're building out a data centre, the question is, what is the right amount of hardware to purchase? If you buy for the average amount, the average usage, you won't be wasting money on average. But when the peak loads come in, you won't have the hardware to service the customers, especially during the critical moments to expect to be making all your results. Now, if you buy for the top max load, you might have happy customers, but for most of the year, you'll have idle resources, which means your average utilization is very low.

If you wanted the scaling process to happen automatically, which AWS service would you use? The AWS service that provides this functionality for Amazon EC2 instances is **Amazon EC2 Auto Scaling**.

Amazon EC2 Auto Scaling

If you’ve tried to access a website that wouldn’t load and frequently timed out, the website might have received more requests than it was able to handle. This situation is similar to waiting in a long line at a coffee shop, when there is only one barista present to take orders from customers.



Amazon EC2 **Auto Scaling enables you to automatically add or remove Amazon EC2 instances in response to changing application demand**. By automatically scaling your instances in and out as needed, you are able to maintain a greater sense of application availability.

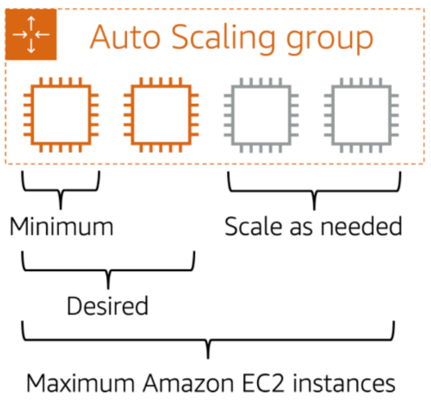
Within Amazon EC2 Auto Scaling, you can use two approaches: dynamic scaling and predictive scaling.

- *Dynamic scaling* responds to changing demand.
- *Predictive scaling* automatically schedules the right number of Amazon EC2 instances based on predicted demand.

Example: Amazon EC2 Auto Scaling

In the cloud, computing power is a programmatic resource, so you can take a more flexible approach to the issue of scaling. By adding Amazon EC2 Auto Scaling to an application, you can **add new instances to the application when necessary** and **terminate them when no longer needed.**

Suppose that you are preparing to launch an application on Amazon EC2 instances. When configuring the size of your Auto Scaling group, you might set the minimum number of Amazon EC2 instances at one. This means that at all times, there must be at least one Amazon EC2 instance running.



When you create an **Auto Scaling group**, you can set the minimum number of Amazon EC2 instances. The **minimum capacity** is the number of Amazon EC2 instances that launch immediately after you have created the Auto Scaling group. In this example, the Auto Scaling group has a minimum capacity of one Amazon EC2 instance.

Next, you can set the **desired capacity** at two Amazon EC2 instances even though your application needs a minimum of a single Amazon EC2 instance to run. If you do not specify the desired number of Amazon EC2 instances in an Auto Scaling group, the desired capacity defaults to your minimum capacity.

The third configuration that you can set in an Auto Scaling group is the **maximum capacity**. For example, you might configure the Auto Scaling group to scale out in response to increased demand, but only to a maximum of four Amazon EC2 instances.

Because Amazon EC2 Auto Scaling uses Amazon EC2 instances, you pay for only the instances you use, when you use them. You now have a cost-effective architecture that provides the best customer experience while reducing expenses.

Elastic Load Balancing

Elastic Load Balancing (ELB) is the AWS service that automatically **distributes incoming application traffic across multiple resources**, such as Amazon EC2 instances.

A **load balancer acts as a single point of contact for all incoming web traffic** to your Auto Scaling group. This means that as you add or remove Amazon EC2 instances in response to the amount of incoming traffic, these requests route to the load balancer first. Then, the requests spread across multiple resources that will handle them. For example, if you have multiple Amazon EC2 instances, Elastic Load Balancing distributes the workload across the multiple instances so that no single instance has to carry the bulk of it.

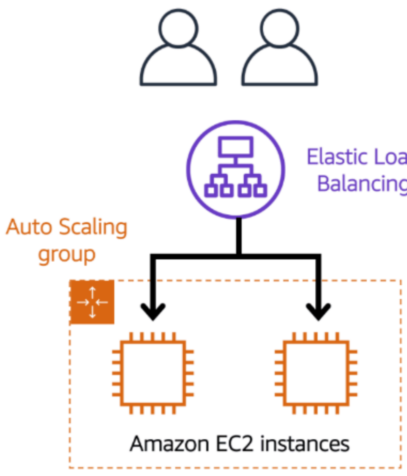
It is engineered to address the undifferentiated heavy lifting of load balancing. **Elastic Load Balancing is a Regional construct**, and we'll explain more of what that means in later videos. But the key value for you is that because it **runs at the Region level** rather than on individual EC2 instances, the service is **automatically highly available** with no additional effort on your part.

Although Elastic Load Balancing and Amazon EC2 Auto Scaling are separate services, they work together to help ensure that applications running in Amazon EC2 can provide high performance and availability.

ELB is automatically scalable. As your traffic grows, ELB is designed to handle the additional throughput with no change to the hourly cost. When your EC2 fleet auto-scales out, as each instance comes online, the auto-scaling service just lets the Elastic Load Balancing service know that it's ready to handle the traffic, and off it goes. Once the fleet scales in, ELB first stops all new traffic, and waits for the existing requests to complete, to drain out. Once they do that, then the auto-scaling engine can terminate the instances without disruption to existing customers.

Because **ELB is regional**, it's a **single URL** that each front end instance uses. Then the ELB directs traffic to the back end that has the least outstanding requests. Now, if the back end scales, once the new instance is ready, it just tells the ELB that it can take traffic and it gets to work. The front end doesn't know and doesn't care how many back end instances are running. This is true **decoupled architecture**.

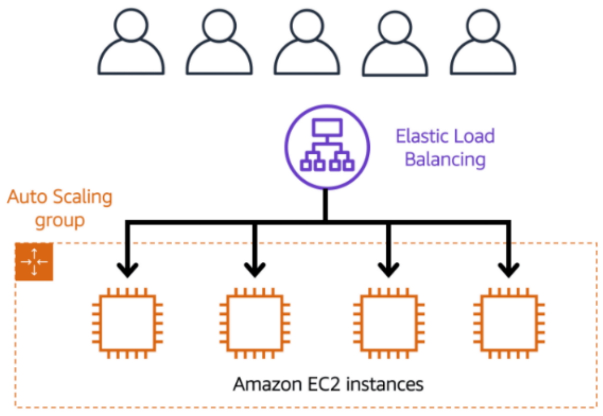
Example: Elastic Load Balancing



Low-demand period

Here’s an example of how Elastic Load Balancing works. Suppose that a few customers have come to the coffee shop and are ready to place their orders.

If only a few registers are open, this matches the demand of customers who need service. The coffee shop is less likely to have open registers with no customers. In this example, you can think of the registers as Amazon EC2 instances.



High-demand period

Throughout the day, as the number of customers increases, the coffee shop opens more registers to accommodate them. In the diagram, the Auto Scaling group represents this.

Additionally, a coffee shop employee directs customers to the most appropriate register so that the number of requests can evenly distribute across the open registers. You can think of this coffee shop employee as a load balancer.

Messaging and Queuing

In the coffee shop, there are cashiers taking orders from the customers and baristas making the orders. Currently, the cashier takes the order, writes it down with a pen and paper, and delivers this order to the barista. The barista then takes the paper and makes the order. When the next order comes in, the process repeats. This works great as long as both the cashier and the barista are in sync. But what would happen if the cashier took the order and turned to pass it to the barista and the barista was out on break or busy with another order? Well, that cashier is stuck until the barista is ready to take the order. And at a certain point, the order will probably be dropped so the cashier can go serve the next customer.

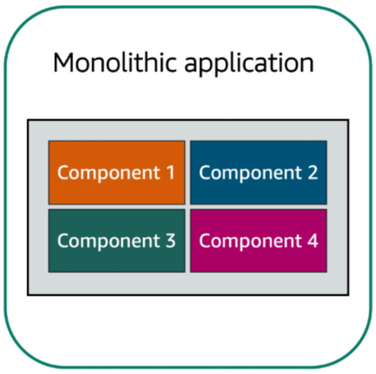
You can see how this is a flawed process, because as soon as either the cashier or barista is out of sync, the process will degrade, causing slowdowns in receiving orders and failures to complete orders at all. A much better process would be to introduce some sort of buffer or queue into the system. Instead of handing the order directly to the barista, the cashier would post the order to some sort of buffer, like **an order board**.

This idea of **placing messages into a buffer is called messaging and queuing**. Just as our cashier sends orders to the barista, applications send messages to each other to communicate. If applications **communicate directly** like our cashier and barista previously, this is called being **tightly coupled**. For example, if we have Application A and it is sending messages directly to Application B, if Application B has a failure and cannot accept those messages, Application A will begin to see errors as well. This is a **tightly coupled architecture.**

A more reliable architecture is **loosely coupled**. This is an architecture where if one component fails, it is isolated and therefore won't cause cascading failures throughout the whole system.

In a message queue, messages are sent into the queue by Application A and they are processed by Application B. If Application B fails, Application A doesn't experience any disruption. Messages being sent can still be **sent to the queue and will remain there until they are eventually processed**. This is **loosely coupled**. This is what we strive to achieve with architectures on AWS. And this brings me to two AWS services that can assist in this regard. Amazon Simple Queue Service or SQS and Amazon Simple Notification Service or SNS.

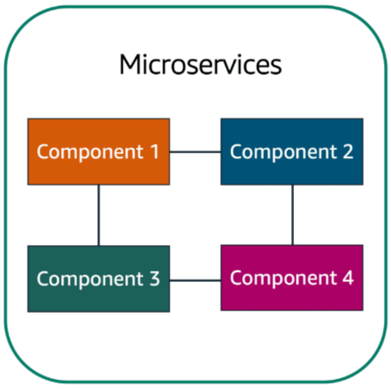
Monolithic applications and microservices



Applications are made of multiple components. The components communicate with each other to transmit data, fulfil requests, and keep the application running.

Suppose that you have an application with tightly coupled components. These components might include databases, servers, the user interface, business logic, and so on. This type of architecture can be considered a **monolithic application**.

In this approach to application architecture, if a single component fails, other components fail, and possibly the entire application fails. To help **maintain application availability** when a single component fails, you can design your application through a **microservices** approach.



In a microservices approach, application components are loosely coupled. In this case, if a single component fails, the other components continue to work because they are communicating with each other. The loose coupling prevents the entire application from failing.

When designing applications on AWS, you can **take a microservices approach** with services and components that fulfil different functions. **Two services facilitate application integration**: Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Queue Service (Amazon SQS).

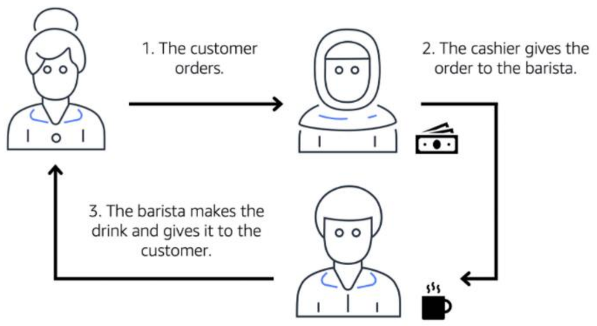
Amazon Simple Queue Service (Amazon SQS)

Amazon Simple Queue Service (Amazon SQS) is a **message queuing service**.

Using Amazon SQS, you can **send, store, and receive messages between software components**, without losing messages or requiring other services to be available. In Amazon SQS, an application sends messages into a queue. A user or service retrieves a message from the queue, processes it, and then deletes it from the queue.

SQS allows you to send, store, and receive messages between software components at any volume. This is without losing messages or requiring other services to be available. Think of messages as our coffee orders and the order board as an SQS queue. Messages have the person's name, coffee order, and time they ordered. The **data contained within a message is called a payload, and it's protected until delivery**. SQS queues are where messages are placed until they are processed. AWS manages the underlying infrastructure for you to host those queues. These scale automatically, are reliable, and are easy to configure and use.

Step 1 – Fulfilling an Order


1. The customer orders.
2. The cashier gives the order to the barista.
3. The barista makes the drink and gives it to the customer.

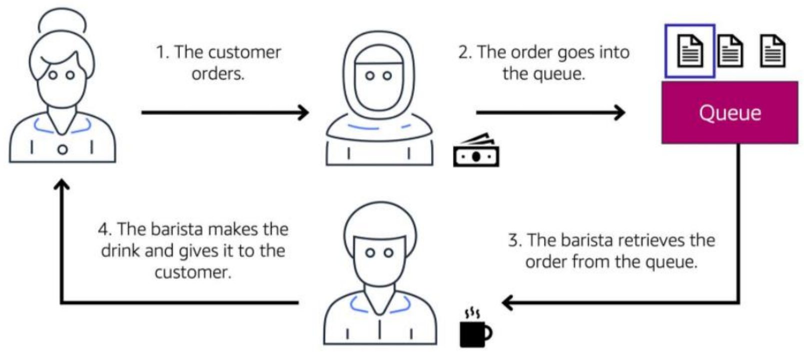
Suppose that the coffee shop has an ordering process in which a cashier takes orders, and a barista makes the orders. Think of the cashier and the barista as two separate components of an application. First, the cashier takes an order and writes it down on a piece of paper. Next, the cashier delivers the paper to the barista. Finally, the barista makes the drink and gives it to the customer.

When the next order comes in, the process repeats. This process runs smoothly as long as both the cashier and the barista are coordinated.

What might happen if the cashier took an order and went to deliver it to the barista, but the barista was out on a break or busy with another order? The cashier would need to wait until the barista is ready to accept the order. This would cause delays in the ordering process and require customers to wait longer to receive their orders.

As the coffee shop has become more popular and the ordering line is moving more slowly, the owners notice that the current ordering process is time consuming and inefficient. They decide to try a different approach that uses a queue.

Step 2 – Orders in a Queue


1. The customer orders.
2. The order goes into the queue.
Queue
3. The barista retrieves the order from the queue.
4. The barista makes the drink and gives it to the customer.

Recall that the cashier and the barista are two separate components of an application. A message queuing service such as Amazon SQS enables messages between decoupled application complements.

In this example, the first step in the process remains the same as before: a customer places an order with the cashier.

The cashier puts the order into a queue. You can think of this as an order board that serves as a buffer between the cashier and the barista. Even if the barista is out on a break or busy with another order, the cashier can continue placing new orders into the queue.

Next, the barista checks the queue and retrieves the order. The barista prepares the drink and gives it to the customer. The barista then removes the completed order from the queue. While the barista is preparing the drink, the cashier is able to continue taking new orders and add them to the queue.

For decoupled applications and microservices, **Amazon SQS enables you to send, store, and retrieve messages between components**. This decoupled approach enables the separate components to work more efficiently and independently.

Note that Amazon SQS is a **message queuing service**, and is therefore not the best choice for **publishing messages to subscribers** since it does **not use the message subscription and topic model that is involved with Amazon SNS**.

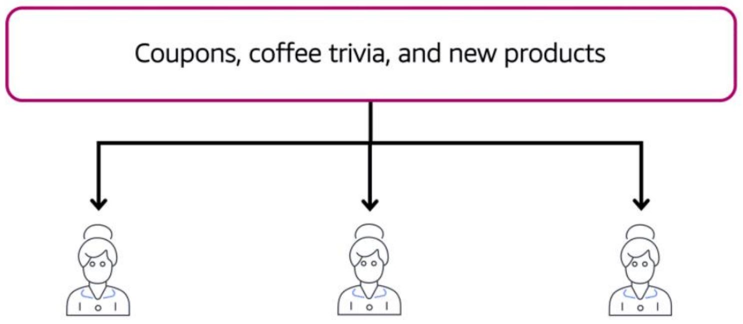
Amazon Simple Notification Service (Amazon SNS)

Amazon Simple Notification Service (Amazon SNS) is a **publish/subscribe** service. Using Amazon SNS **topics**, a **publisher publishes messages to subscribers**. This is similar to the coffee shop; the cashier provides coffee orders to the barista who makes the drinks.

Amazon SNS is similar to SQS in that it is used to send out messages to services, but it can **also send out notifications to end users**. It does this in a different way called a publish/subscribe or **pub/sub model**. This means that you can create something called an **SNS topic which is just a channel for messages** to be delivered.

Additionally, SNS can be used to **fan out notifications to end users using mobile push, SMS, and email.** Taking this back to our coffee shop, we could send out a notification when a customer's order is ready. This could be a simple SMS to let them know to pick it up or even a mobile push.

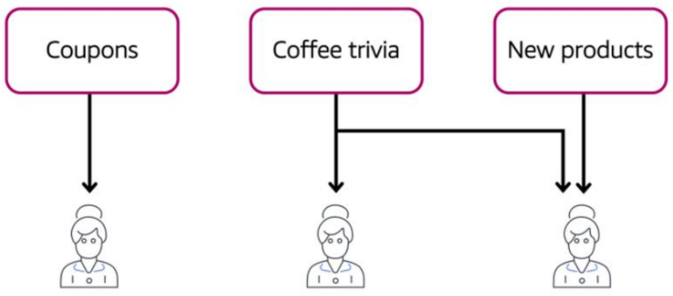
Step 1 – Publishing updates from a single topic



Suppose that the coffee shop has a single newsletter that includes updates from all areas of its business. It includes topics such as coupons, coffee trivia, and new products. All of these topics are grouped because this is a single newsletter. All customers who subscribe to the newsletter receive updates about coupons, coffee trivia, and new products.

After a while, some customers express that they would prefer to receive separate newsletters for only the specific topics that interest them. The coffee shop owners decide to try this approach.

Step 2 – Publishing updates from multiple topics



Now, instead of having a single newsletter for all topics, the coffee shop has broken it up into three separate newsletters. Each newsletter is devoted to a specific topic: coupons, coffee trivia, and new products. Subscribers will now receive updates immediately for only the specific topics to which they have subscribed.

It is possible for subscribers to subscribe to a single topic or to multiple topics. For example, the first customer subscribes to only the coupons topic, and the second subscriber subscribes to only the coffee trivia topic. The third customer subscribes to both the coffee trivia and new products topics.

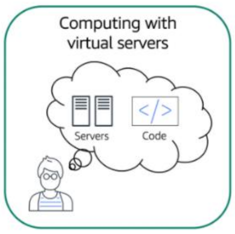
Although this example from the coffee shop involves subscribers who are people, in Amazon SNS, **subscribers can be web servers, email addresses, AWS Lambda functions, or several other options.**

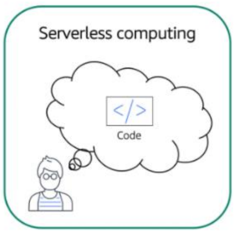
Serverless computing

Earlier in this module, you learned about Amazon EC2, a service that lets you run virtual servers in the cloud. If you have applications that you want to run in Amazon EC2, you must do the following:

1) Provision instances (virtual servers).
2) Upload your code.
3) Continue to manage the instances while your application is running.

EC2 requires that you set up and manage your fleet of instances over time. When you're using EC2, **you are responsible for patching your instances** when new software packages come out, **setting up the scaling** of those instances as well as ensuring that you've architected your solutions to be hosted in a highly available manner. This is still not as much management as you would have if you hosted these on-premises. But management processes will still need to be in place.





The term “serverless” means that your code runs on servers, but you do not need to provision or manage these servers. With serverless computing, you can focus more on innovating new products and features instead of maintaining servers. Serverless means that you **cannot actually see or access the underlying infrastructure** or instances that are hosting your application. Instead, all the **management of the underlying environment from a provisioning, scaling, high availability, and maintenance perspective are taken care of** for you. All you need to do is focus on your application and the rest is taken care of.

Another benefit of serverless computing is the **flexibility** to scale serverless applications automatically. Serverless computing can adjust the applications' capacity by modifying the units of consumptions, such as throughput and memory. An AWS service for **serverless computing** is **AWS Lambda**.

AWS Lambda

AWS Lambda is a service that lets you run code **without needing to provision or manage servers**.

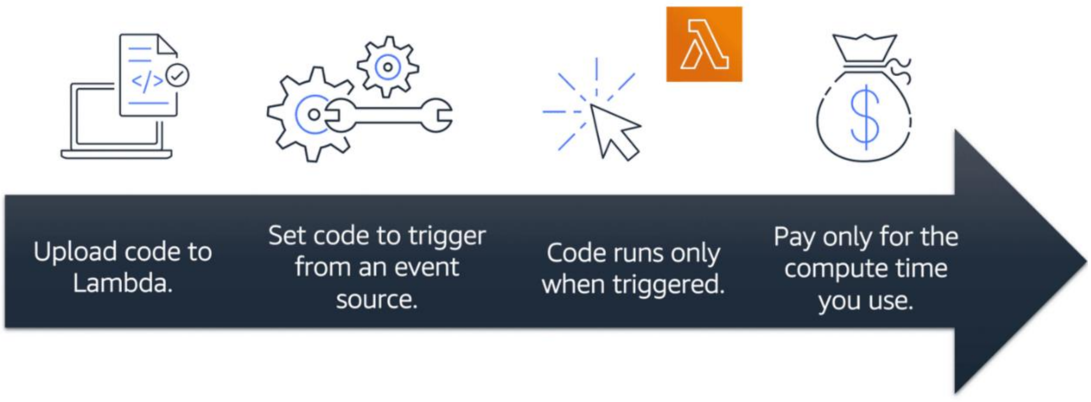
While using AWS Lambda, you pay only for the compute time that you consume. Charges apply only when your code is running. You can also run code for virtually any type of application or backend service, all with zero administration.

Lambda's a service that allows you to upload your code into what's called a **Lambda function**. Configure a trigger and from there, the service waits for the trigger. When the trigger is detected, the code is automatically run in a managed environment, an environment you do not need to worry too much about because it is **automatically scalable, highly available and all of the maintenance in the environment itself is done by AWS**. If you have one or 1,000 incoming triggers, Lambda will scale your function to meet demand.

Lambda is designed to **run code under 15 minutes** so this **isn't for long running processes like deep learning.** It's **more suited for quick processing** like a **web backend, handling requests or a backend expense report processing** service where each invocation takes less than 15 minutes to complete.

For example, a simple Lambda function might involve automatically resizing uploaded images to the AWS Cloud. In this case, the function triggers when uploading a new image.

How AWS Lambda works


Upload code to Lambda.
Set code to trigger from an event source.
Code runs only when triggered.
Pay only for the compute time you use.

1. You upload your code to Lambda.
2. You set your code to trigger from an event source, such as AWS services, mobile applications, or HTTP endpoints.
3. Lambda runs your code only when triggered.
4. You pay only for the compute time that you use. In the previous example of resizing images, you would pay only for the compute time that you use when uploading new images. Uploading the images triggers Lambda to run code for the image resizing function.

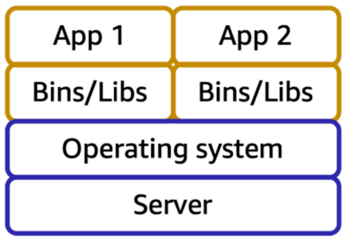
Containers

If you weren't quite ready for serverless yet or you need access to the underlying environment, but still want efficiency and portability, you should look at AWS container services like Amazon Elastic Container Service, otherwise known as ECS. Or Amazon Elastic Kubernetes Service, otherwise known as EKS.

In AWS, you can also build and run **containerized** applications. **Containers** provide you with a standard way to **package your application's code and dependencies into a single object**. You can also use containers for processes and workflows in which there are essential requirements for security, reliability, and scalability.

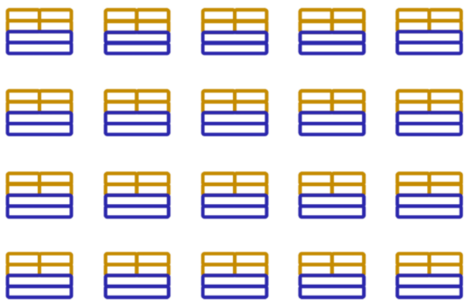
A container in this case is a **Docker container**. Docker is a widely used platform that uses **operating system level virtualization to deliver software in containers**. Now a container is a package for your code where you package up your application, its dependencies as well as any configurations that it needs to run. **These containers run on top of EC2 instances** and run in isolation from each other similar to how virtual machines work, but in this case, the host is an EC2 instance.

<u>Step 1 – One host with multiple containers</u>



Suppose that a company's application developer has an environment on their computer that is different from the environment on the computers used by the IT operations staff. The developer wants to ensure that the application's environment remains consistent regardless of deployment, so they use a containerized approach. This helps to reduce time spent debugging applications and diagnosing differences in computing environments.

<u>Step 2 – Tens of hosts with hundreds of containers</u>



When running containerized applications, it’s important to consider scalability. Suppose that instead of a single host with multiple containers, you have to manage tens of hosts with hundreds of containers. Alternatively, you have to manage possibly hundreds of hosts with thousands of containers. At a large scale, imagine how much time it might take for you to monitor memory usage, security, logging, and so on.

Container orchestration services help you to deploy, manage, and scale your containerized applications. Next, you will learn about two services that provide container orchestration: Amazon Elastic Container Service and Amazon Elastic Kubernetes Service.

When you use Docker containers on AWS, you need processes to start, stop, restart, and monitor containers running across not just one EC2 instance, but a number of containers together which is called a **cluster**.

The process of doing these tasks is called **container orchestration** and it turns out it's really hard to do on your own. **Orchestration tools** were created to help you manage your containers. **ECS** is designed to help you run your containerized applications at scale **without the hassle of managing your own container orchestration software**. **EKS does a similar thing**, but uses different tooling and with different features.

Amazon Elastic Container Service (Amazon ECS)

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance **container management system** that enables you to **run and scale containerized applications** on AWS.

Amazon ECS supports Docker containers. Docker is a software platform that enables you to build, test, and deploy applications quickly. AWS supports the use of open-source Docker Community Edition and subscription-based Docker Enterprise Edition. With Amazon ECS, you can use API calls to launch and stop Docker-enabled applications.

Amazon Elastic Kubernetes Service (Amazon EKS)

Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed service that you can use to run Kubernetes on AWS.

Kubernetes is open-source software that enables you to deploy and manage containerized applications at scale. A large community of volunteers maintains Kubernetes, and AWS actively works together with the Kubernetes community. As new features and functionalities release for Kubernetes applications, you can easily apply these updates to your applications managed by Amazon EKS.

AWS Fargate

Both Amazon ECS and Amazon EKS can run on top of EC2. **But if you don't want to even think about using EC2s to host your containers** because you either don't need access to the underlying OS or you don't want to manage those EC2 instances, you can use a compute platform called AWS Fargate. **AWS Fargate** is a serverless compute engine for containers. It works with both Amazon ECS and Amazon EKS.

When using AWS Fargate, you do not need to provision or manage servers. AWS Fargate manages your server infrastructure for you. You can focus more on innovating and developing your applications, and you pay only for the resources that are required to run your containers.

Choice of Compute Service

If you are trying to host traditional applications and want full access to the underlying operating system like Linux or Windows, you are going to want to use **EC2**.

If you are looking to host short running functions, service-oriented or event driven applications and you don't want to manage the underlying environment at all, look into the serverless **AWS Lambda**.

If you are looking to run Docker container-based workloads on AWS, you first need to choose your orchestration tool. Do you want to use Amazon **ECS** or Amazon **EKS**? After you choose your tool, you then need to choose your platform. Do you want to run your containers on EC2 instances that you manage or in a serverless environment like AWS **Fargate** that is managed for you?

Summary

We define cloud computing as the on-demand delivery of IT resources over the internet with pay-as-you-go pricing. This means that you can make requests for IT resources like compute, networking, storage, analytics, or other types of resources, and then they're made available for you on demand. You don't pay for the resource upfront. Instead, you just provision and pay at the end of the month.

AWS offers services and many categories that you use together to create your solutions. We've only covered a few services so far, you learned about Amazon EC2. With **EC2**, you can **dynamically spin up and spin down virtual servers called EC2 instances**. When you launch an EC2 instance, you choose the instance family. The instance family determines the **hardware** the instance runs on.

And you can have instances that are built for your specific purpose. The categories are **general** purpose, **compute** optimized, **memory** optimized, **accelerated** computing, and **storage** optimized.

You can **scale** your EC2 instances either **vertically by resizing the instance, or horizontally by launching new instances and adding them to the pool**. You can set up automated **horizontal scaling, using Amazon EC2 Auto Scaling**.

Once you've scaled your EC2 instances out horizontally, you need something to **distribute the incoming traffic across those instances**. This is where the Elastic Load Balancer comes into play.

EC2 instances have different pricing models. There is **On-Demand**, which is the most flexible and has no contract, spot pricing, which allows you to utilize unused capacity at a discounted rate, **Savings Plans or Reserved Instances**, which allow you to enter into a contract with AWS to get a discounted rate when you commit to a certain level of usage, and Savings Plans which apply to AWS **Lambda**, and AWS **Fargate**, as well as EC2 instances.

We also covered messaging services. There is Amazon Simple Queue Service or SQS**. This service allows you to decouple system components. Messages remain in the queue until they are either consumed or deleted.** Amazon Simple Notification Service or SNS, is used for sending messages like emails, text messages, push notifications, or even HTTP requests. Once a message is published, it is sent to all of these subscribers.

You also learned that AWS has different types of compute services beyond just virtual servers like EC2. There are **container services** like Amazon Elastic Container Service, or ECS. And there's Amazon Elastic Kubernetes Service, or EKS. **Both ECS and EKS are container orchestration tools.** You can use these tools with EC2 instances, but if you don't want to manage that, you don't need to. You can use AWS **Fargate**, which allows you **to run your containers on top of a serverless compute platform.**

Then there is AWS Lambda, which allows you to just **upload your code, and configure it to run based on triggers.** And you only get charged for when the code is actually running. No containers, no virtual machines. Just code and configuration.
Hopefully that sums it up. Catch you in the next one.

Quiz

You want to use an Amazon EC2 instance for a batch processing workload. What would be the best Amazon EC2 instance type to use?

- Compute optimized instance type

What are the contract length options for Amazon EC2 Reserved Instances?

- 1 year and 3 years

You have a workload that will run for a total of 6 months and can withstand interruptions. What would be the most cost-efficient Amazon EC2 purchasing option?

- Spot instance

Which process is an example of Elastic Load Balancing?

- Ensuring that no single Amazon EC2 instance has to carry the full workload on its own

You want to deploy and manage containerized applications. Which service can you use?

- Amazon Elastic Kubernetes Service (Amazon EKS). Amazon EKS is a fully managed Kubernetes service. Kubernetes is open-source software that enables you to deploy and manage containerized applications at scale.

Additional resources

To learn more about the concepts that were explored in Module 2, review these resources.

- Compute on AWS
- AWS Compute Blog
- AWS Compute Services
- Hands-On Tutorials: Compute
- Category Deep Dive: Serverless
- AWS Customer Stories: Serverless

Compiled by Kenneth Leung – December 2020