**Module 2 introduction**

**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)](https://aws.amazon.com/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 center.
* Make all the necessary configurations.

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**

**Launch – Configurations**

**Connect – Connecting to EC2 instance**

**Use – to install software, storage**

# Amazon EC2 instance types

**Amazon EC2 instance types**

[Amazon EC2 instance types](https://aws.amazon.com/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**

–

**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**

–

**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**

–

**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**

–

**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**

–

**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.

**Knowledge check**

Match each description to an Amazon EC2 instance type.

* Ideal for high-performance databases - Memory optimized
* Suitable for data warehousing applications - Storage optimized
* Balances compute, memory, and networking resources - General purpose
* Offers high-performance processors - Compute optimized

**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**

–

**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.

**Amazon EC2 Savings Plans**

–**{{{** **Amazon EC2 Savings Plans** are ideal for workloads that involve a consistent amount of compute usage over a 1-year or 3-year term.

With Amazon EC2 Savings Plans, you can reduce your compute costs by up to 72% over On-Demand costs.

**}}}**

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 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. If you are considering your options for Savings Plans, AWS Cost Explorer can analyze your Amazon EC2 usage over the past 7, 30, or 60 days. AWS Cost Explorer also provides customized recommendations for Savings Plans. These recommendations estimate how much you could save on your monthly Amazon EC2 costs, based on previous Amazon EC2 usage and the hourly commitment amount in a 1-year or 3-year Savings Plan.

**Reserved Instances**

–

**Reserved Instances** are a billing discount applied to the use of On-Demand Instances in your account. 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**

* **Spot Instances** are ideal for workloads with flexible start and end times, or that can withstand interruptions. With Spot Instances, you can reduce your compute costs by up to 90% over On-Demand costs.
* Unlike Amazon EC2 Savings Plans, Spot Instances do ***not*** require contracts or a commitment to a consistent amount of compute usage.

**Spot Instances** are ideal for workloads with flexible start and end times, or that can withstand interruptions. Spot Instances use unused Amazon EC2 computing capacity and offer you cost savings at up to 90% off of On-Demand prices.  
  
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**

–

**Dedicated Hosts**are physical servers with Amazon EC2 instance capacity that is fully dedicated to your use.

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.

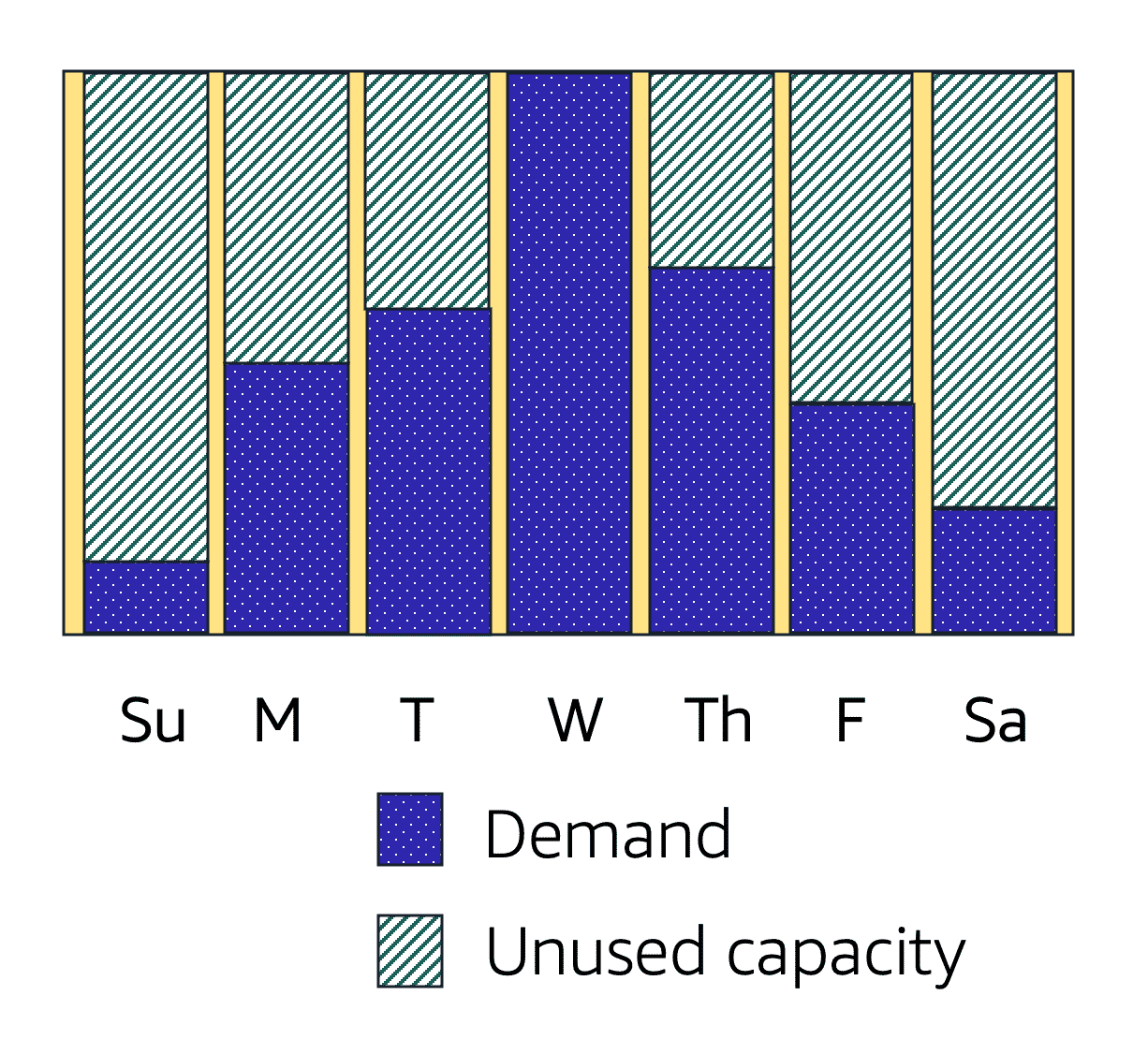
**Scalability**

**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.

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**.

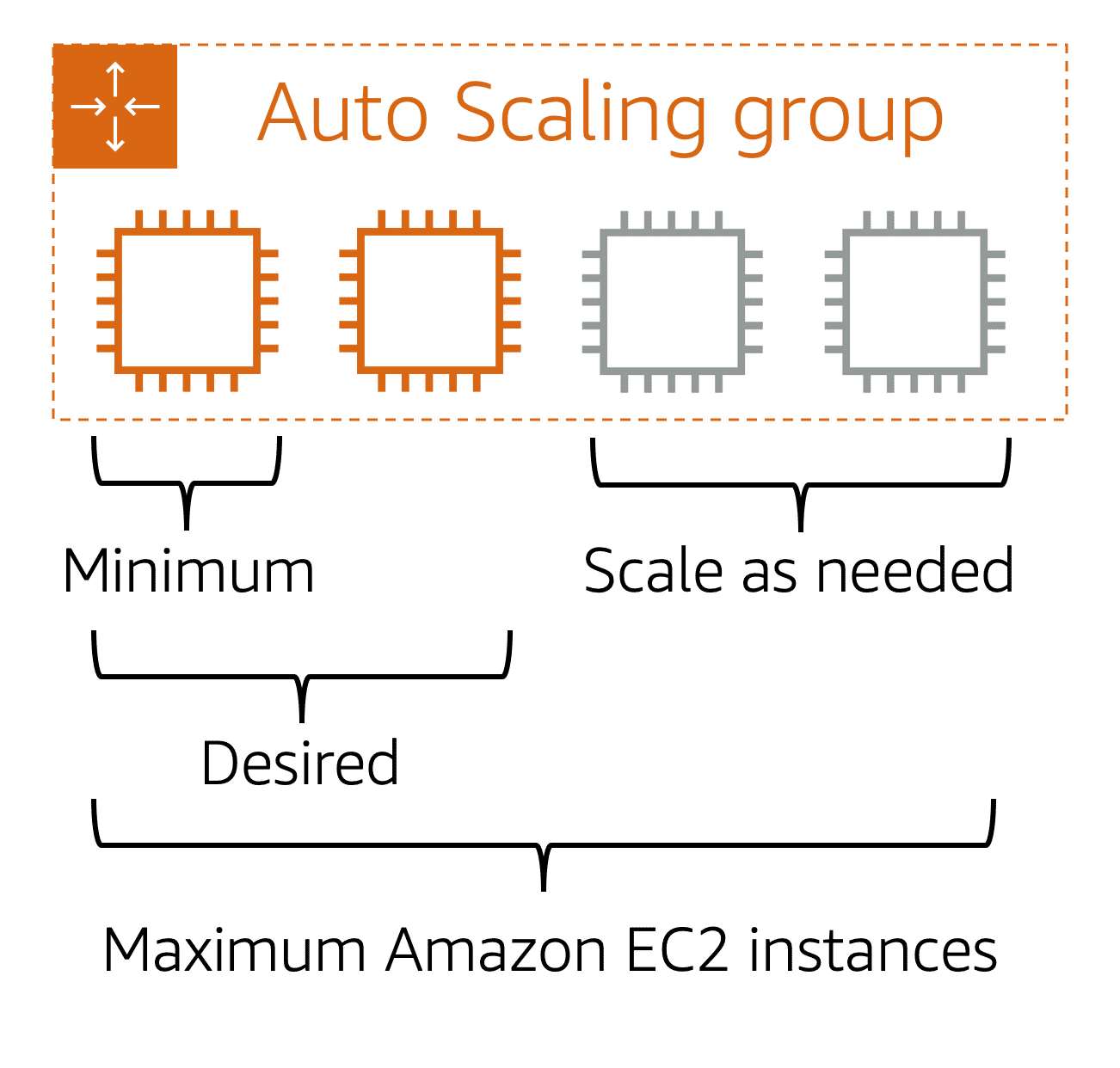
**Amazon EC2 Auto Scaling**

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.

# Directing traffic with Elastic Load Balancing

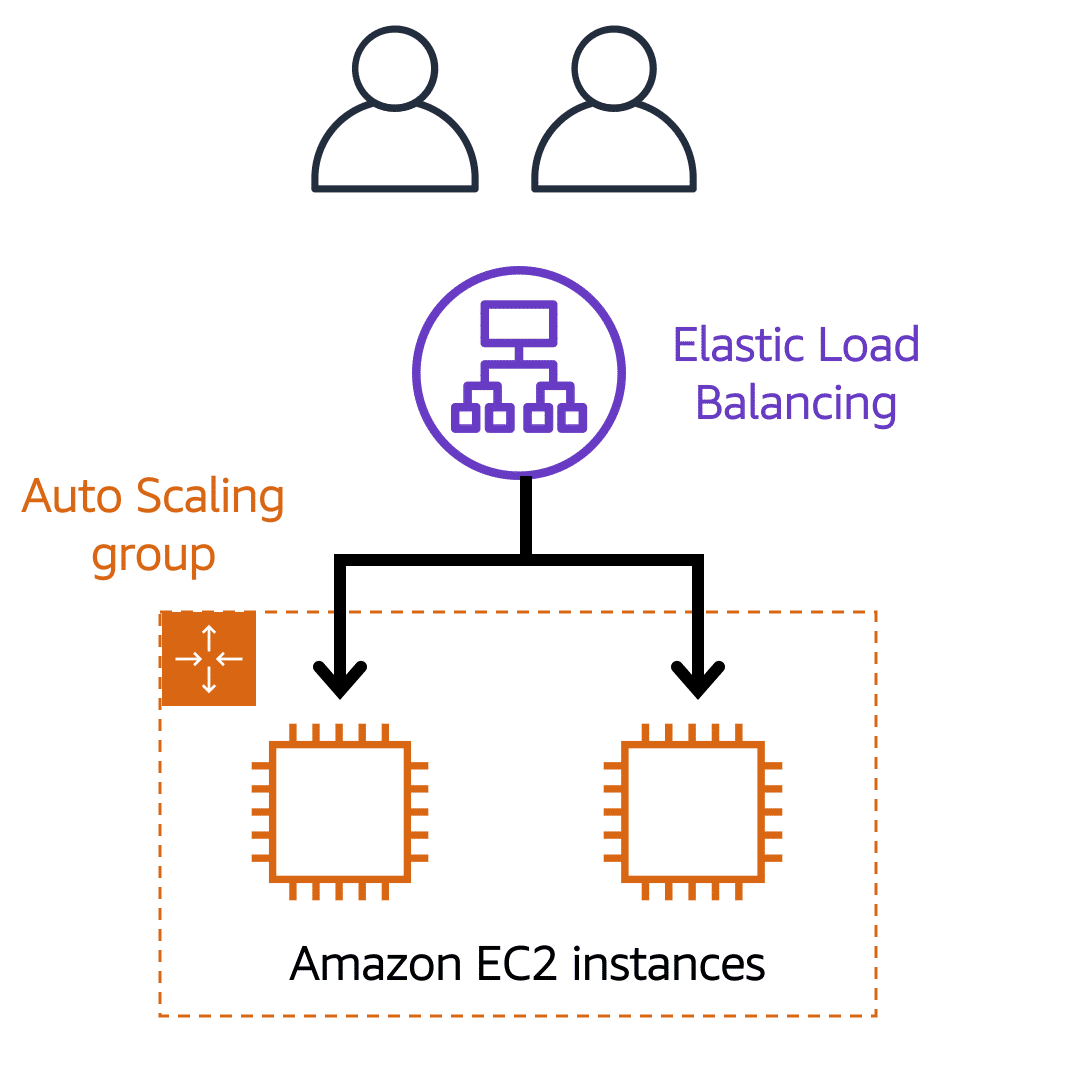
**Elastic Load Balancing**

**Elastic Load Balancing** 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.

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.

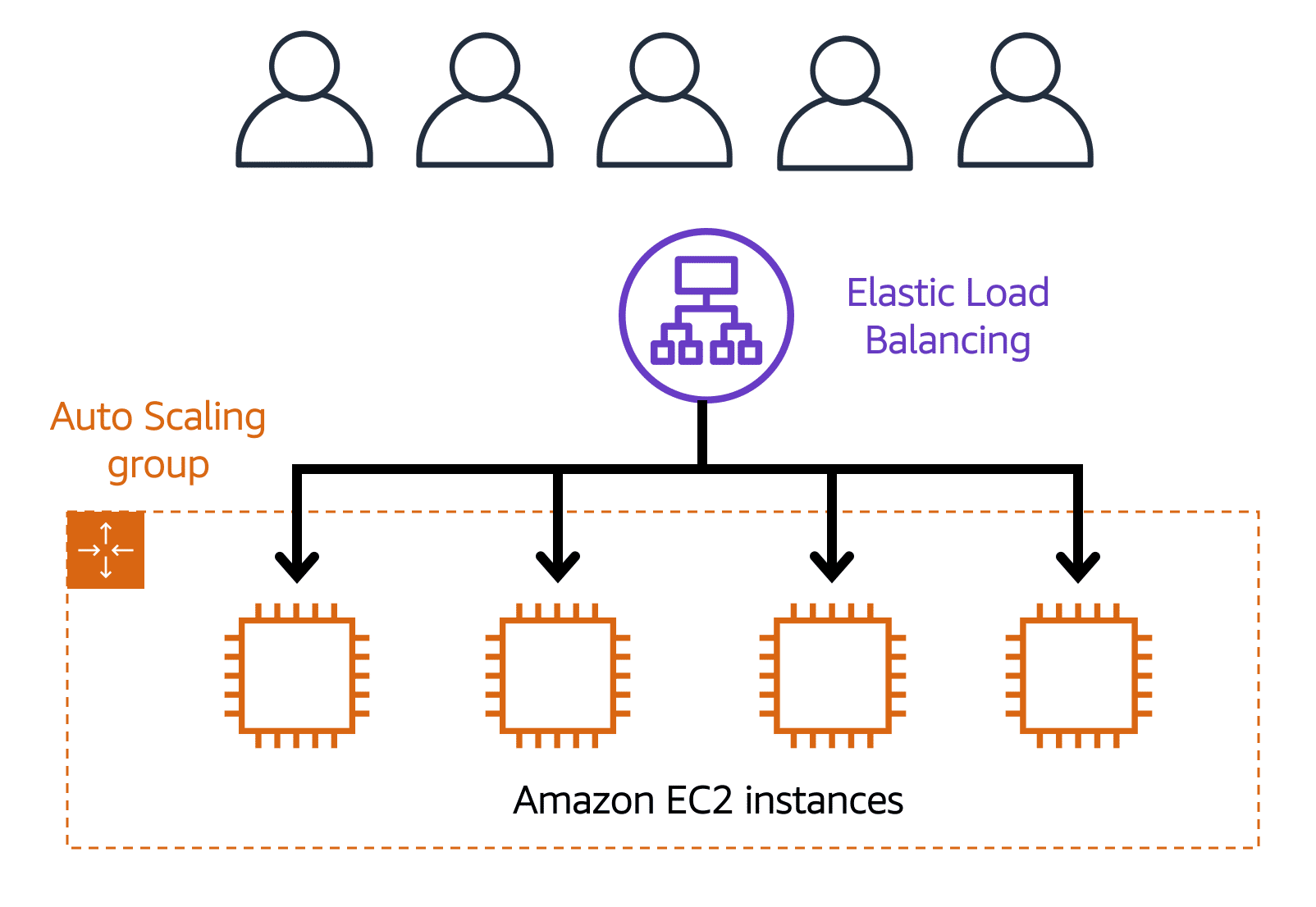
**Example: Elastic Load Balancing**



**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.



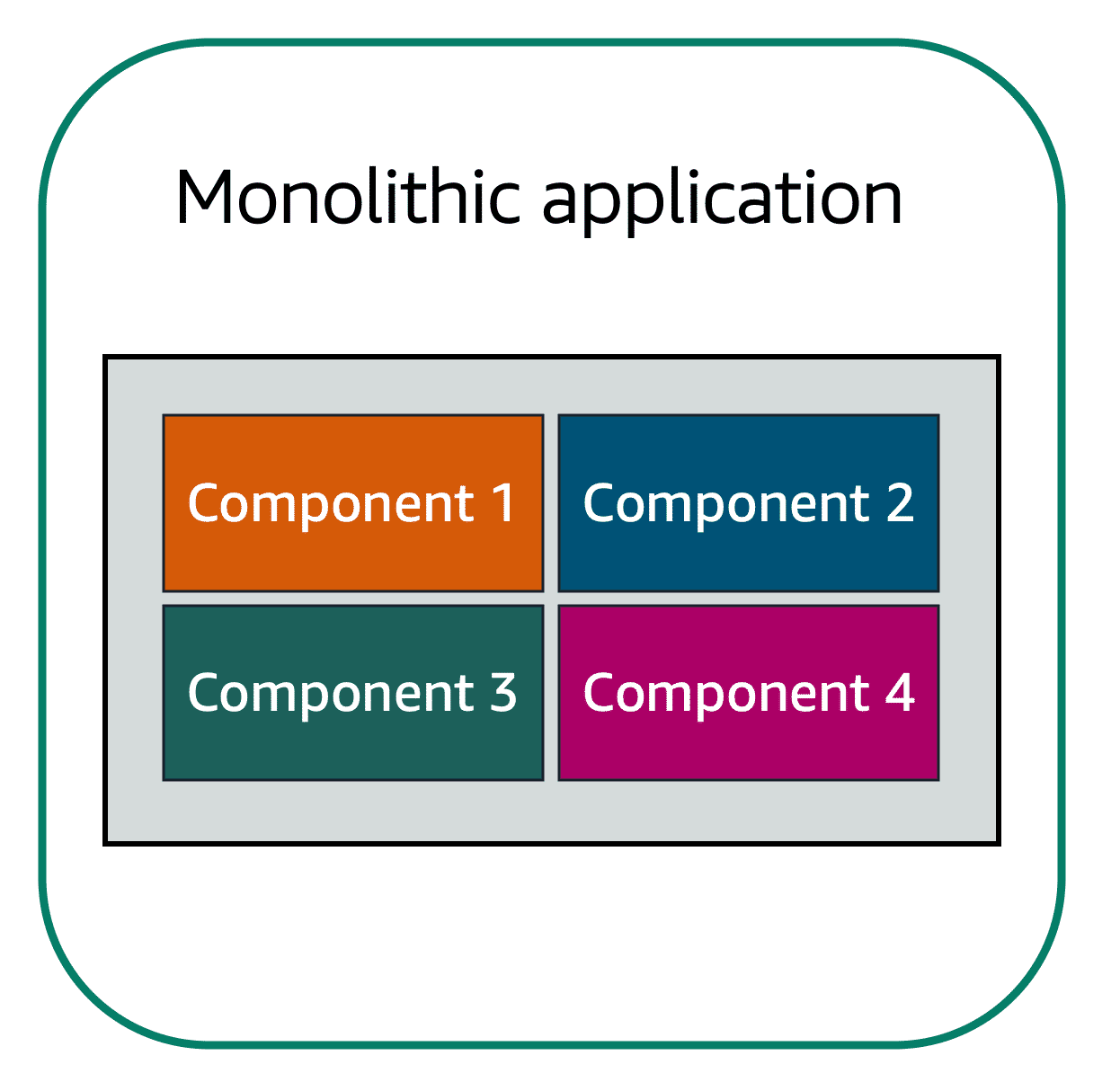
**High-demand period**

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

**Monolithic applications and microservices**

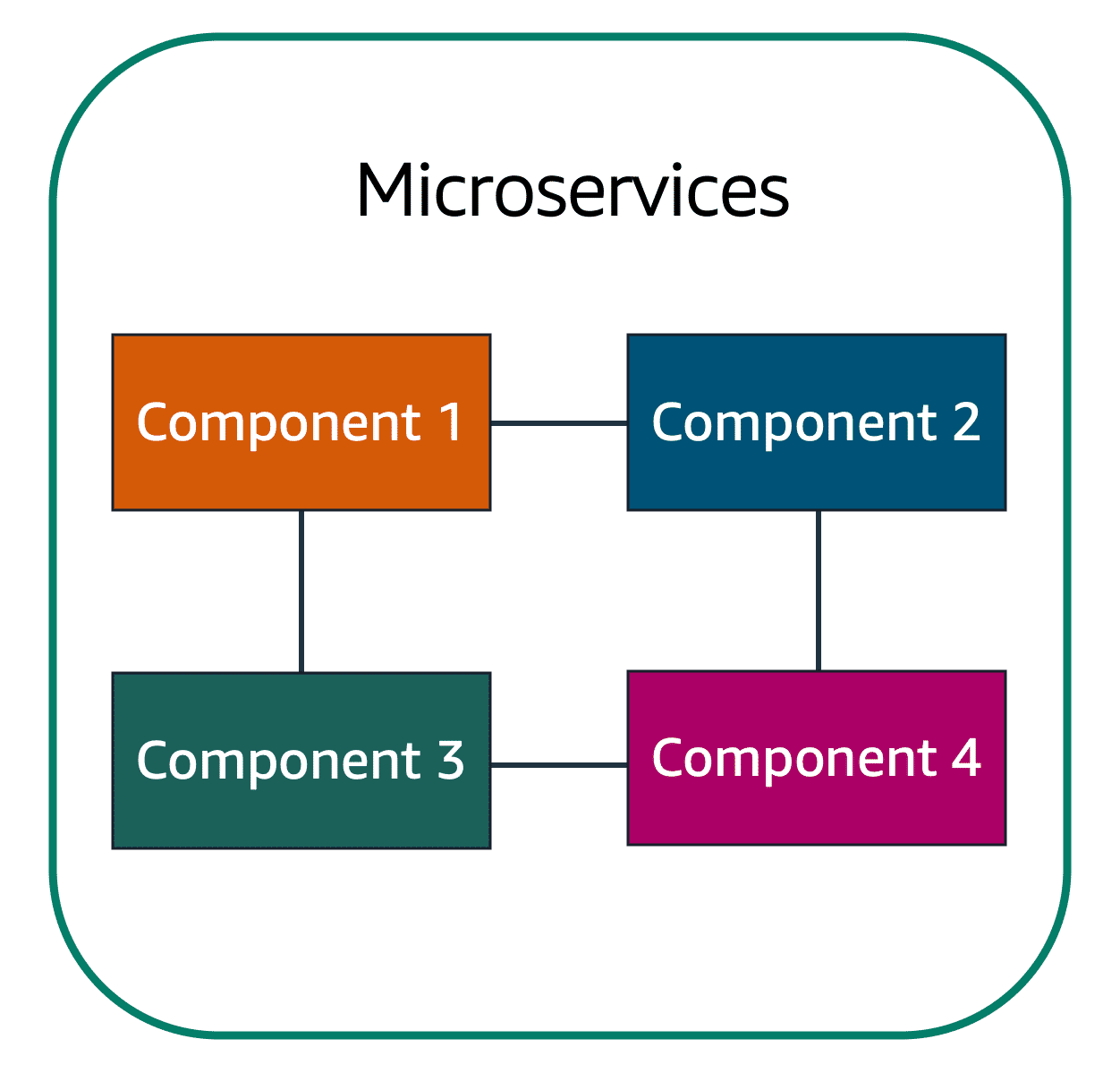


Applications are made of multiple components. The components communicate with each other to transmit data, fulfill requests, and keep the application running.

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 fulfill different functions. Two services facilitate application integration: Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Queue Service (Amazon SQS).

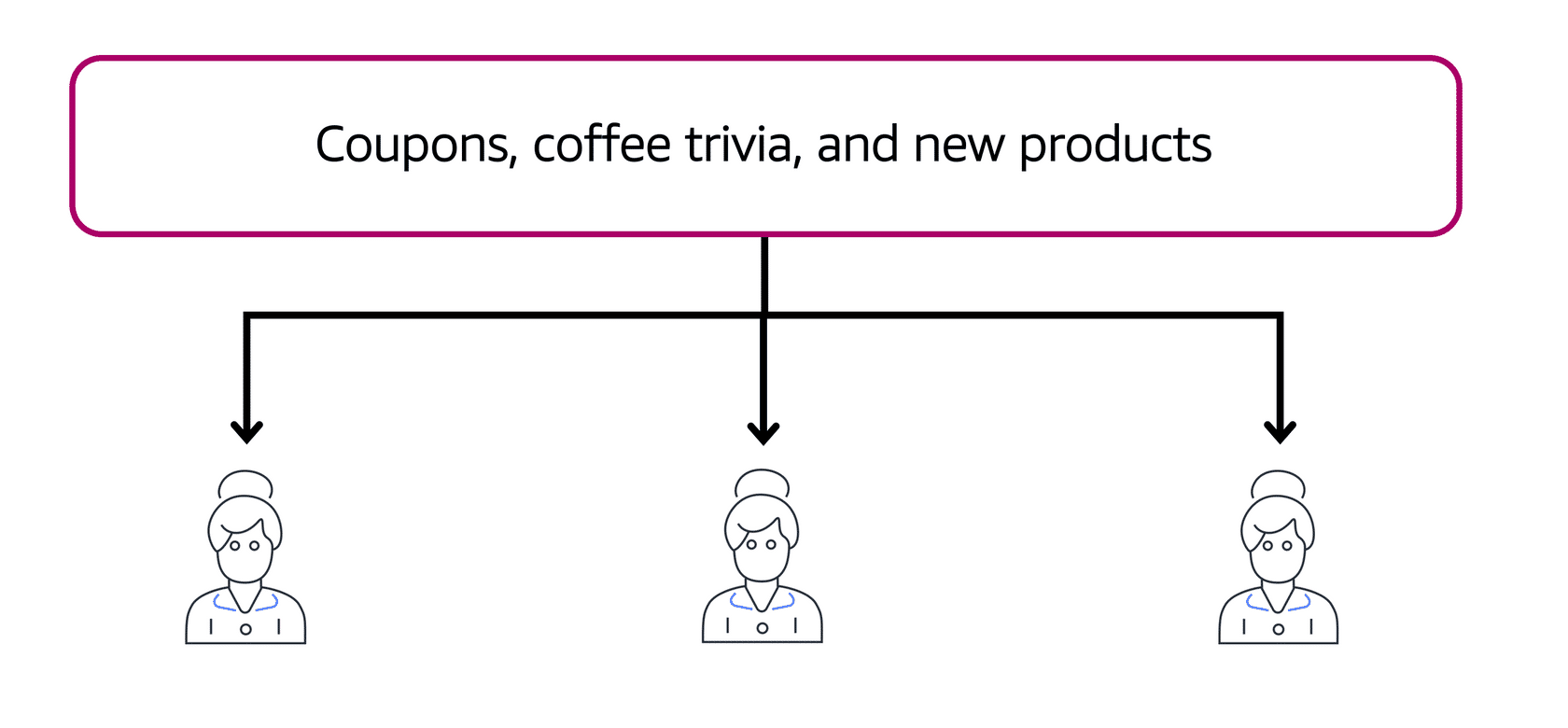
**Amazon Simple Notification Service (Amazon SNS)**

**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.

In Amazon SNS, subscribers can be web servers, email addresses, AWS Lambda functions, or several other options.

**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.

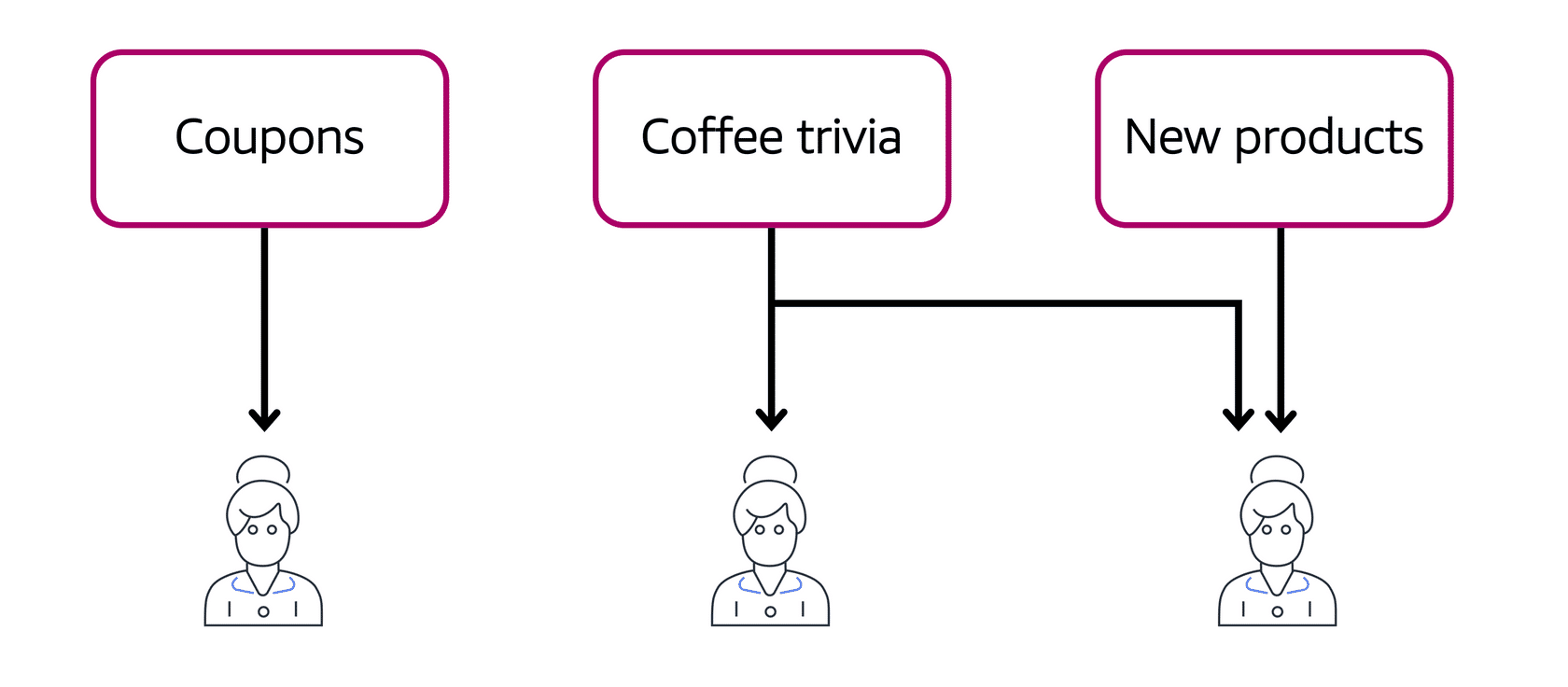
After a while, some customers express that they would prefer to receive separate newsletters for only the specific topics that interest them. The coffee shop owners decide to try this approach.

1

2

**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.

1

2

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.

1

2

**START AGAIN**

**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.

## Example: Amazon SQS

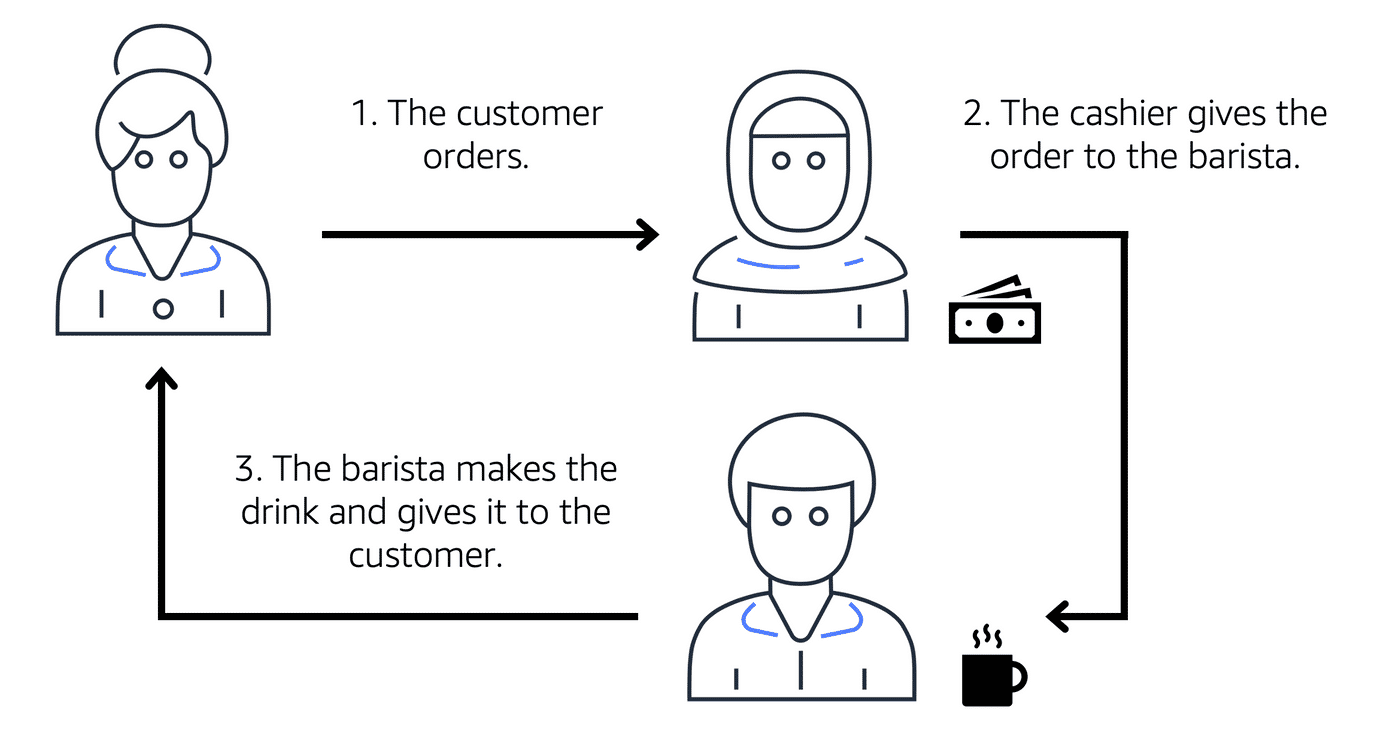
## Example: Amazon SQS

To review an example of how to use Amazon SQS, select **Start**.

START

**Step 1**

## Example: Fulfilling an order



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.

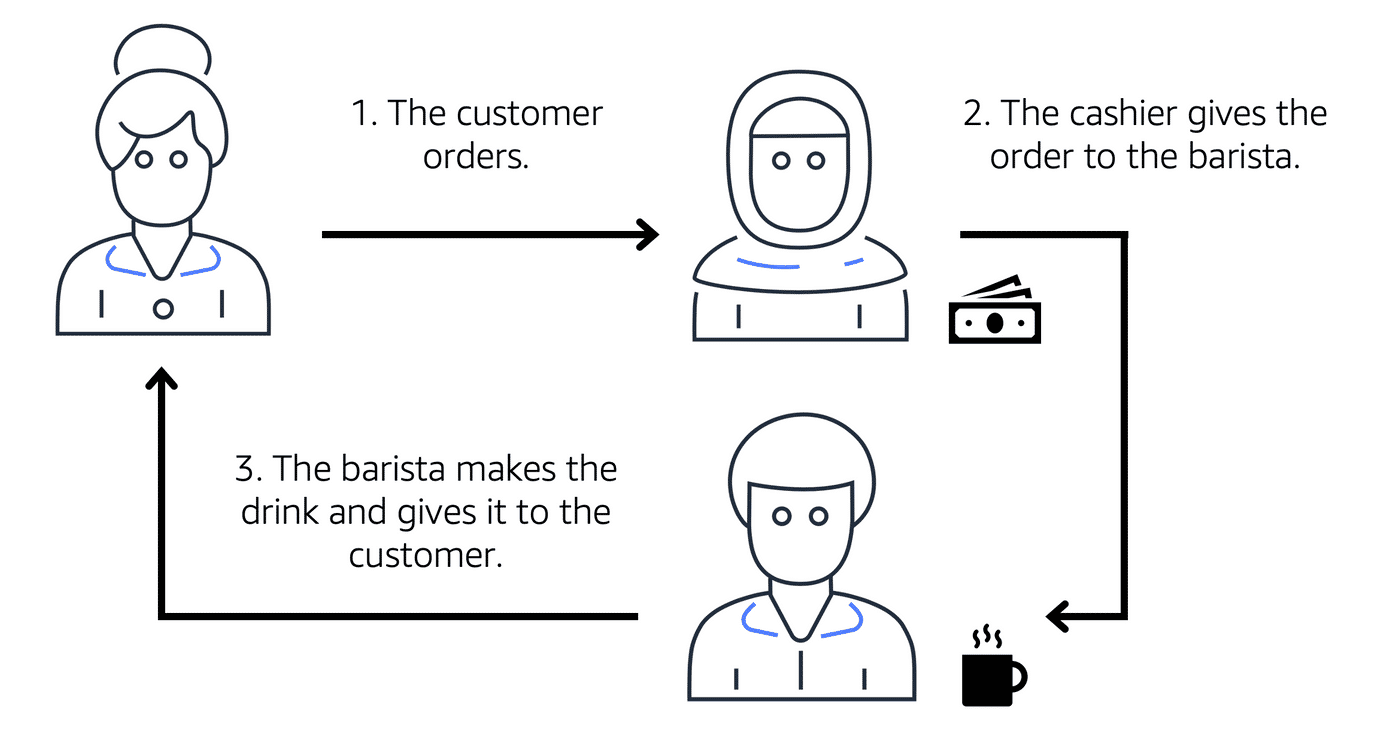
## Example: Amazon SQS

To review an example of how to use Amazon SQS, select **Start**.

START

**Step 1**

## Example: Fulfilling an order



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.

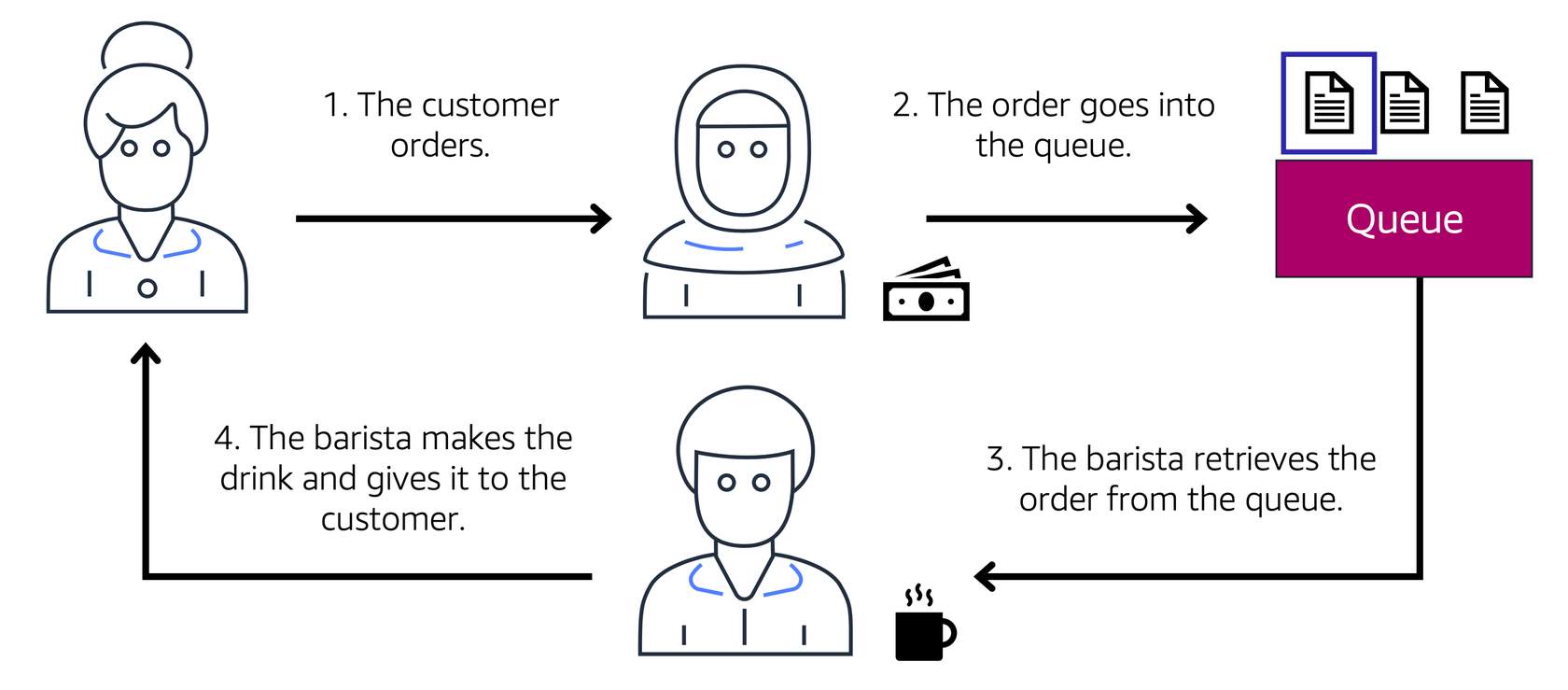
As the coffee shop has become more popular and the ordering line is moving more slowly, the owners notice that the current ordering process is time consuming and inefficient. They decide to try a different approach that uses a queue.

1

2

**Step 2**

## Example: Orders in a queue



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.