**AWS Cloud Practitioner Essentials**

|  |
| --- |
| **Module 1 Introduction to Amazon Web Services**   * Introduction * Cloud Computing * Module 1 Quiz |

|  |
| --- |
| **Module 2 Compute in the cloud**   * Introduction * Amazon EC2 Instance Types * Amazon EC2 Pricing * Scaling Amazon EC2 * Directing traffic with Elastic Load Balancing (ELB) * Messaging and Queuing   + Amazon SNS and SQS * Additional Compute Services   + AWS Lambda   + AWS Container Services (Elastic Kubernetes Service - EKS)   + AWS Fargate * Module 2 Summary * Module 2 Quiz |

|  |
| --- |
| **Module 3 Global Infrastructure and Reliability**   * AWS Global Infrastructure * Edge locations * How to provision AWS resources * Module 3 Summary * Module 3 Quiz |

|  |
| --- |
| **Module 4 Networking**   * Module 4 Introduction * Connectivity to AWS * Subnets and Network control access lists * Global networking * Module 4 Summary * Module 4 Quiz |

|  |
| --- |
| **Module 5 Storage and Databases**   * Instance stores and Amazon Elastic Block Store (Amazon EBS) * Amazon Simple Storage Service (Amazon S3) * Amazon Elastic File System (Amazon EFS) * Amazon Relation Database Service (Amazon RDS) * Amazon DynamoDB * Amazon Redshift * AWS Data Migration Service * Additional Database Services * Module 5 Summary * Module 5 Quiz |
| **Module 6 Security**   * Module 6 Introduction * AWS Shared Responsibility Model * User permissions and access * AWS Organizations * Compliance * Denial-of-Service attacks * Additional Security Services * Module 6 Summary * Module 6 Quiz |

|  |
| --- |
| **Module 7 Monitoring and Analysis**   * Module 7 Introduction * Amazon CloudWatch * AWS CloudTrail * AWS Trusted Advisor * Module 7 Summary * Module 7 Quiz |

|  |
| --- |
| **Module 8 Pricing and Support**   * Module 8 Introduction * AWS free tier * AWS pricing concepts * Billing dashboard * Consolidated billing * AWS budgets * AWS Cost Explorer * AWS Support plans * AWS Market plans * Module 8 Summary * Module 8 Quiz |

|  |
| --- |
| **Module 9 Migration and Innovation**   * Module 9 Introduction * AWS Cloud Adoption Framework (AWS CAF) * Migration Strategies * AWS Snow Family * Module 9 Summary * Module 9 Quiz |

|  |
| --- |
| **Module 10 The cloud Journey**   * Module 10 Introduction * The AWS Well-Architected Framework * Benefits of the AWS cloud * Module 10 Summary * Module 10 Quiz |

|  |
| --- |
| **Module 11 AWS Certified Cloud Practitioner Basics**   * Module 11 Introduction * Exam details * Exam Strategies |

|  |
| --- |
| **Final Assessment** |

**Module 1**

**Introduction to Amazon Web Services**

**Learning objectives**

* Summarize the benefits of AWS.
* Describe differences between on-demand delivery and cloud deployments.
* Summarize the pay-as-you-go pricing model.

**Module 1**

**Introduction to Amazon Web Services**

**What AWS Offers**

AWS is a comprehensive cloud platform providing services for various business needs. These services range from basic computing tools to highly specialized solutions:

**Imagine you’re building city:**

**Basic Services**

**Basic Services are like the city’s essentials**

**Compute**: Virtual servers, container services, and serverless computing to run applications.

These are your construction workers (virtual servers) who build and maintain buildings (applications). They include container services (like temporary teams) and serverless computing (automatic workers ready when needed).

**Storage**: Secure and scalable options like Amazon S3 for file storage and Amazon EBS for block storage.

Think of this as warehouses for your city. Amazon S3 is like a giant public storage facility for files, while Amazon EBS is like personal lockers for specific items.

**Networking and Security**: Tools for secure data transfer, virtual private networks (VPNs), and firewalls.

This is the city’s police and transportation system—secure roads (VPNs) and gates (firewalls) that keep everything running smoothly and safely.

**Advanced Solutions**

**Advanced Solutions** add cool, modern features to your city

**Blockchain**: Platforms for building and managing decentralized applications.

It’s like a secure vault for your city records—unbreakable and decentralized.

**Machine Learning (ML) and Artificial Intelligence (AI)**: Pre-trained models and infrastructure for data-driven insights.

Imagine super-smart city planners who analyze data and predict future trends.

**Robotic Development Platforms**: Tools for building and testing autonomous systems.

These are the city’s engineers who create and test automated machines, like drones or delivery robots.

**Specialized Services**

**Specialized Services** are unique attractions for the city

**Media Services**: Video production and live-streaming platforms.

These are your entertainment hubs—movie theaters or live-streaming stages.

**Orbital Satellites**: Unique services like AWS Ground Station for managing satellite communications.

Picture satellites hovering over the city, managing communication (AWS Ground Station).

Although AWS offers over 200 services.

**Understanding the Cloud Compute Model**

AWS services are based on the **client-server model**, which is fundamental to modern IT architecture.

**Here’s how it works:**

* **Client**: A user or application that makes a request. For example, uploading a file, running a program, or analyzing data.
* **Server**: A virtual machine or physical system that processes and fulfils the client’s request.

**Real-World Analogy**: Consider a coffee shop scenario:

* The **customer (client)** asks for a coffee.
* The **barista (server)** prepares the coffee after validating the request (e.g., payment).

In AWS, the server role is represented by **EC2 instances (Elastic Compute Cloud)**, which are virtual machines used to handle client requests. Each instance is part of a broader architecture, which ensures scalability and stability.

**AWS Key Value: Pay for What You Use**

One of AWS’s most critical benefits is its **pay-as-you-go pricing model**, which allows businesses to only pay for the resources they actively use.

**Traditional IT Challenges**:

* In an on-premises data center, businesses need to buy hardware upfront and maintain capacity for peak demand, even if that capacity is rarely used.
* This leads to wasted resources and higher costs.

**AWS Solution**:

* Businesses can dynamically scale their computing resources.

For example:

* + If demand spikes during a product launch, businesses can add virtual servers (EC2 instances) instantly.
  + When demand decreases, they can reduce resources, stopping the cost.

**Coffee Shop Analogy**:

* A coffee shop doesn’t hire all its employees to work full-time if there’s no demand.
* Instead, the shop hires employees based on customer traffic, paying only for the hours worked.
* Similarly, AWS lets businesses pay for servers and storage only when they’re needed.

This **elasticity** in resource management saves money and ensures operational efficiency.

**Why AWS?**

**Imagine a factory owner trying to modernize their operations:**

**Cost Efficiency**: The owner no longer needs to buy expensive machinery upfront. They rent what they need and scale up or down based on production demand.

**Scalability and Flexibility**: With AWS, adding resources is as simple as clicking a button—no worries about shortages during peak times.

**Reliability and Performance**: AWS has factories (data centers) worldwide, ensuring operations run smoothly, even during unexpected disruptions.

**Broad Range of Services**: AWS offers a wide range of services for businesses of all sizes and industries, including basic computing and advanced AI and machine learning.

AWS empowers businesses to focus on their goals without worrying about infrastructure.

**What is a client-server model?**

You just learned more about AWS and how almost all of modern computing uses a basic client-server model.

Let’s recap what a client-server model is.

In computing, a**client** can be a web browser or desktop application that a person interacts with to make requests to computer servers. A **server** can be services, such as Amazon Elastic Compute Cloud (Amazon EC2) – a type of virtual server.

**For example:**

Suppose that a client makes a request for a news article, the score in an online game, or a funny video.

The server evaluates the details of this request and fulfils it by returning the information to the client.

**Module 1**

**Cloud Computing**

**What is Cloud Computing?**

Cloud computing refers to the **on-demand delivery of IT resources over the internet** with a **pay-as-you-go pricing model**.

Here's a breakdown:

**On-Demand Delivery**:

* + AWS provides the resources when needed—no advance notice required.
  + Example: If you need 300 virtual servers or 2,000 TB of storage, you can instantly launch them with a few clicks. Once you’re done, you can stop using them and stop paying immediately.

**IT Resources**:

* + AWS focuses on providing common IT resources (e.g., databases, servers) so businesses can concentrate on what makes them unique.
  + **Example**: Managing a MySQL database engine isn’t what differentiates your business—what matters is how you structure and use your data.

**Over the Internet**:

* + AWS resources are accessible via a secure webpage, command-line interface, or APIs without additional contracts or lengthy processes.

**Pay-as-You-Go Pricing**:

* + Pay only for what you use, when you use it.
  + Example: Businesses can utilize AWS to avoid paying for resources like developer environments during off-hours (e.g., weekends), similar to how a coffee shop only hires personnel during peak hours.

**Module 1**

**Deployment Models for Cloud Computing**

When selecting a cloud strategy, businesses must consider their application needs, legacy IT requirements, and resource management preferences.

There are **three deployment models**:

**1. Cloud-Based Deployment**

All parts of an application are run in the cloud.

* Run the entire application in the cloud.
* Move existing applications to the cloud.
* Design and build new applications in the cloud.

In a **cloud-based deployment** model allows you to migrate existing applications to the cloud, or you can design and build new applications in the cloud.

You can build those applications on **low-level infrastructure** that your IT staff will need to manage.

You can also build them using **higher-level services**, which reduce the core infrastructure's management, architecture, and scaling requirements.

**Example**: a company might create an application consisting of virtual servers, databases, and networking components that are fully based in the cloud, reducing the need for IT staff to manage the core infrastructure.

**2. On-Premises Deployment (Private Cloud)**

* Utilize virtualization and resource management tools to effectively deploy resources.
* Increase resource utilization through application management and virtualization technologies.

On-premises deployment is also referred to as private cloud deployments. This model deploys resources on-premises using virtualization and resource management tools.

For example, you may have applications that rely on technology that is entirely stored in your on-premises data center. Although this model is similar to legacy IT infrastructure, the incorporation of application management and virtualization technologies improves resource utilization.

**3. Hybrid Deployment**

Combines cloud-based resources with on-premises infrastructure.

* Connect cloud-based resources to on-premises infrastructure.
* Integrate cloud-based resources into older IT applications.

A hybrid deployment combines cloud-based resources with on-premises infrastructure. You might want to use this method in a variety of situations. For example, you may have legacy applications that are best maintained on-site, or government regulations require your company to keep certain records on-site.

For example, suppose a company wants to use cloud services to automate batch data processing and analytics. However, the company has several legacy applications that are better suited to on-premises environments and will not be migrated to the cloud. With a hybrid deployment, the company can keep legacy applications on-premises while leveraging cloud-based data and analytics services.

**Benefits of Cloud Computing**

Businesses choose cloud computing for its flexibility, cost efficiency, and global reach.

**Here are six key benefits:**

**Trade Upfront Expenses for Variable Expenses**

Traditional IT requires heavy upfront investment in data centers and hardware before knowing how they’ll be used.

With cloud computing, you only pay for the resources you consume, enabling innovation without large initial costs.

**Stop Spending Money to run and maintain Data Center**

Managing physical servers and data centers is time-consuming and costly.

Cloud computing allows you to focus on your applications and customers while spending less time on infrastructure.

**Eliminate Guesswork in Capacity Planning**

Cloud computing removes the need to predict how much infrastructure you’ll need.

**Example**: Use Amazon EC2 to scale up when demand increases and scale down when it decreases, paying only for the compute time you use.

**Benefit from Massive Economies of Scale**

Cloud computing reduces variable costs compared to on-premises solutions.

Because hundreds of thousands of customers' usages can be aggregated in the cloud, providers like Amazon Web Services can achieve greater economies of scale. The economy of scale leads to lower pay-as-you-go prices.

**Increase Speed and Agility**

Cloud computing offers greater flexibility for application development and deployment. This flexibility gives you more time to experiment and innovate. When computing in data centers, it may take weeks to obtain the resources you require.

In comparison, cloud computing allows you to access new resources in minutes.

**Go Global in Minutes**

AWS Cloud's global footprint allows for quick and low-latency application deployments to customers worldwide. This means that, even if you are in a different part of the world than your customers, they can still access your applications with minimal delay.

**Module 1**

**QUIZ**

|  |
| --- |
| What is Cloud computing?   * Backing up files that are stored on desktop and mobile devices to prevent data loss * Deploying applications connected to on-premisses infrastructure * Running code without needing to manage or provision servers * **On-demand delivery of IT resources and applications through the internet with pay-as-you-go pricing** |

|  |
| --- |
| What is another name for on-premises deployment?   * **Private cloud deployment** * Cloud-based application * Hybrid deployment * AWS cloud |

|  |
| --- |
| How does the scale of cloud computing help you to save costs?   * You do not have to invest in technology resources before using them. * **The aggregated cloud usage from a large number of customers results in lower pay-as-you-go prices.** * Accessing services on-demand helps to prevent excess or limited capacity. * You can quickly deploy applications to customers and provide them with low latency. |

**Module 2**

**Compute in the cloud**

**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.

**Module 2**

**Introduction**

**What is Amazon EC2?**

Amazon **Elastic Compute Cloud (Amazon EC2)** is a cloud computing service that offers **secure, scalable virtual servers** (known as EC2 instances) to meet your business requirements. It allows you to run applications without having to own or manage physical servers.

EC2 allows you to:

* Quickly provision and launch servers in minutes.
* Pay only for the compute capacity you use.
* Avoid upfront hardware costs and maintenance of physical infrastructure.

**Why Use Amazon EC2 Instead of On-Premises Servers?**

**1. Challenges with On-Premises Servers**:

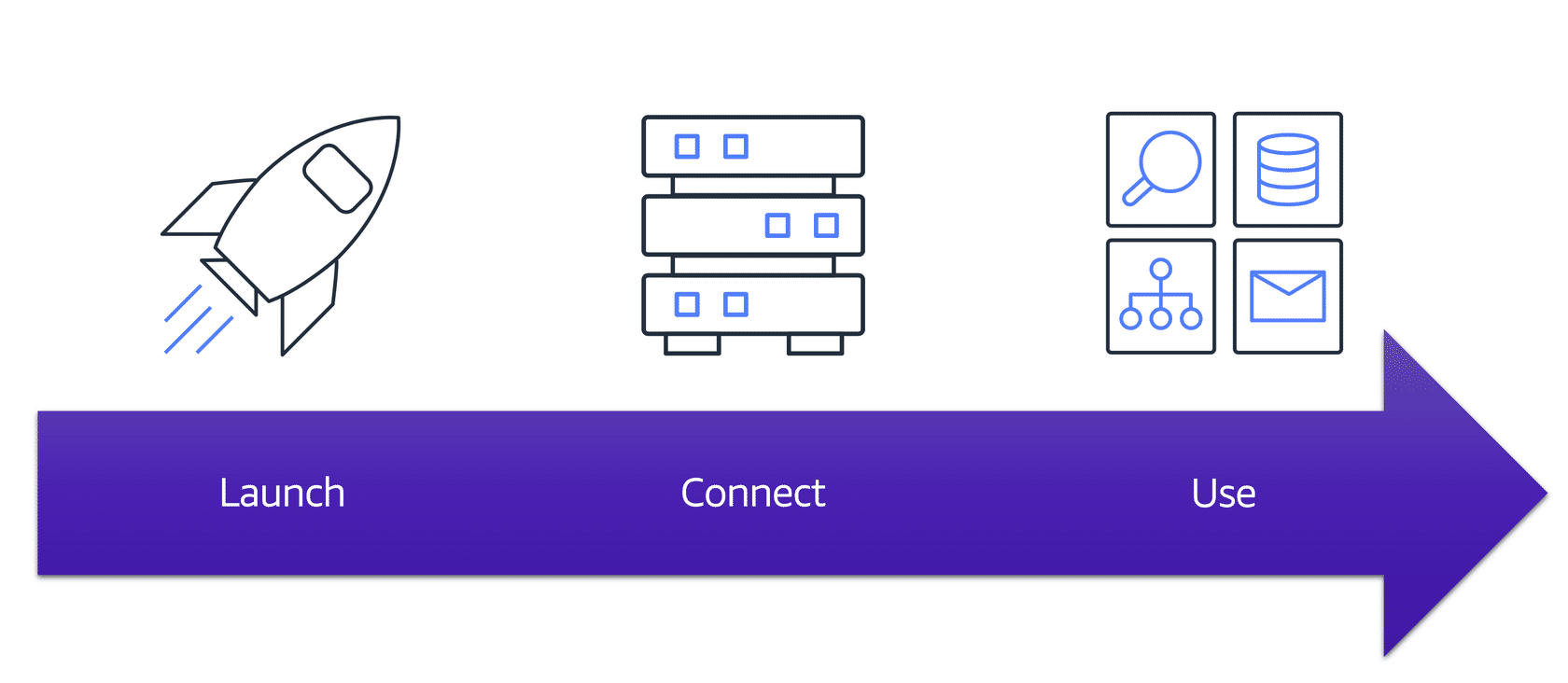
* + **Upfront Costs**: Requires purchasing expensive hardware.
  + **Delivery Delays**: Waiting weeks or months for server delivery.
  + **Setup Effort**: Installation, configuration, and ensuring security take significant time and effort.
  + **Inflexibility**: Once purchased, you're stuck with the servers whether you use them or not. Scaling up or down is difficult and costly.

**2. Benefits of EC2**:

* + **Quick Setup**: AWS has already built and secured the physical infrastructure. You can launch servers (EC2 instances) within minutes.
  + **Pay-As-You-Go Pricing**: Only pay for the time your instances are running, not when stopped or terminated.
  + **Scalability**: Easily adjust capacity as your business needs change.
  + **No Maintenance**: AWS handles the physical data center operations, server management, and resource provisioning.

**How EC2 Works**

1. **Launch**: Request virtual servers (EC2 instances) based on your workload requirements.
2. **Connect**: Use these servers to host applications or services.
3. **Use**: Customize and manage your instances to run business applications efficiently.



**Key Features of EC2**

**1. Virtualization and Multitenancy**:

* + **Virtualization**: EC2 instances are **virtual machines** that **run on physical servers managed by AWS**. A hypervisor enables multiple virtual servers to share the same physical hardware.
  + **Multitenancy**: Multiple instances securely share a host machine’s resources. The hypervisor isolates each instance, ensuring secure and separate environments.

**2. Flexible Operating Systems**:

* + Choose from Linux or Windows-based operating systems.
  + Mix and match OS types for different business applications.

**3. Customizable Software**:

* + Install and run any software you need, including:
  + Web applications (simple or complex).
  + Databases.
  + Enterprise software.

**4. Resizing Instances**:

* + **Vertical Scaling**: Adjust instance size by increasing or decreasing resources like CPU and memory to match application needs.

**5. Networking Options**:

* + Decide which traffic can access your instance:
  + Publicly accessible for websites.
  + Privately accessible for internal applications.

**6. Cost Efficiency**:

* Pay only for the running instances, helping reduce costs during periods of low demand.

**Advantages of Using EC2**

**Flexibility**:

* Quickly scale servers up or down to meet demand.
* Manage multiple configurations and software setups to power different applications.

**Speed and Agility**:

* Launch instances in minutes instead of weeks or months compared to on-premises setups.

**Security**:

* Instances are isolated and secure, even when sharing resources on a host machine.

**Supports Innovation**:

* Developers can focus on building applications rather than managing hardware.

Amazon EC2 provides a cost-effective and scalable way to power your applications with compute resources, allowing businesses to adapt quickly to changing needs without being tied to physical hardware.

**Module 2**

**Amazon EC2 Instance Types**

**What Are EC2 Instance Types?** Amazon EC2 instance types are virtual servers created to fulfil a variety of application requirements. They are optimized for certain workloads by providing various computation, memory, storage, and networking configurations.

Each instance is part of a family, each of which is built for a certain use case. This flexibility enables you to choose the optimum instance type for your workload requirements, ensuring cost efficiency and optimal performance.

**EC2 Instance Types:**

* 1. General Purpose Instances
  2. Compute Optimized Instances
  3. Memory Optimized Instances
  4. Accelerated Computing Instances
  5. Storage Optimized Instances

**1. General Purpose Instances**:

* **Overview**: These instances provide a balance of compute, memory, and networking resources. They are versatile and **can handle a wide range of applications**.
* **Use Cases**:
  + Application servers.
  + Small and medium-sized databases.
  + Backend servers for enterprise applications.
  + Gaming servers.
* **Analogy**: Consider a multi-tasking employee in a coffee shop, such as taking orders, preparing coffee, and managing the cash register. They may not specialize in any one work, but they are adaptable and capable of executing a wide range of duties.
* **When to Use**: Choose these instances **when your workload requires a balanced use of all resources** (compute, memory, and networking) without needing optimization in a specific area.

**2. Compute Optimized Instances**:

* **Overview**: Designed for compute-intensive tasks that require high-performance processors.
* **Use Cases**:
  + High-performance web servers.
  + Gaming servers with complex processing needs.
  + Batch processing workloads (e.g., processing many transactions in one group).
  + Scientific modeling or simulations.
* **Analogy**: These are like **baristas in a coffee shop**—they specialize in making drinks efficiently and quickly, ensuring customers receive their orders promptly.
* **When to Use**: Use these instances for applications that require high processing capacity, such as gaming, real-time simulations, and mathematical computations.

**3. Memory Optimized Instances**:

* **Overview**: Built **to deliver fast performance** for applications **that process large datasets in memory**. These instances are designed for tasks that require a large amount of memory and great performance.
* **Use Cases**:
  + High-performance databases.
  + Real-time analytics and processing of large datasets.
  + In-memory caching and analytics applications.
* **Analogy**: Imagine a **cashier in a coffee shop**—their job requires them to handle customer orders quickly, so they rely on their memory and efficiency to ensure everything is accurate and fast.
* **When to Use**: Choose these instances for workloads that require huge volumes of data to be preloaded into memory before processing, resulting in faster performance.

**4. Accelerated Computing Instances**:

* **Overview**: These instances utilize hardware accelerators (coprocessors) to accomplish specified tasks more effectively than a CPU. They are appropriate for tasks that require specific calculations or processing.
* **Use Cases**:
  + Graphics-intensive applications.
  + Game and application streaming.
  + Machine learning model training.
  + Scientific computations involving floating-point calculations.
* **Analogy**: A latte artist in a coffee shop has a specialized skill set for making beautiful designs in lattes, which requires advanced methods beyond typical barista duties.
* **When to Use**: Use these instances for workloads that require specialist hardware, such as graphics processing or data pattern recognition.

**5. Storage Optimized Instances**:

* **Overview**: Designed for workloads with high sequential read and write access to large datasets stored locally.
* **Use Cases**:
  + Distributed file systems.
  + Data warehousing applications.
  + High-frequency online transaction processing (OLTP) systems.
* **Key Metric**: These instances excel in **Input/Output Operations Per Second (IOPS)**, a measure of how many read/write operations a storage device can handle per second.
* **Analogy**: In a coffee shop, these instances are like the **kitchen staff**, who handle the preparation of food. They need efficient access to ingredients (stored data) to prepare meals quickly and accurately for customers.
* **When to Use**: Use storage-optimized instances for applications that need fast access to huge datasets, such as databases with frequent read/write requests.

**Coffee Shop Analogy for EC2 Instance Types**

**General Purpose Instances**: Like an all-rounder employee who can take orders, make coffee, and help wherever needed.

**Compute Optimized Instances**: Like baristas focused on making drinks efficiently.

**Memory Optimized Instances**: Like cashiers who handle and process customer orders quickly and accurately.

**Accelerated Computing Instances**: Like latte artists who perform specialized tasks requiring unique skills.

**Storage Optimized Instances**: Like kitchen staff who efficiently access ingredients (stored data) to prepare food for customers.

**Quiz**

**Instance Types**

|  |
| --- |
| Which Amazon EC2 instance type is suitable for data warehousing applications?   * Memory Optimized * **Storage Optimized** * General purpose * Compute Optimized |

|  |
| --- |
| Which Amazon EC2 instance type balances compute, memory and networking resource?   * Memory optimized * Storage Optimized * **General purpose** * Compute optimized |

|  |
| --- |
| Which Amazon instance type is ideal for high-performance databases?   * **Memory optimized** * Storage Optimized * General purpose * Compute optimized |

|  |
| --- |
| Which Amazon instance type offers high-performance processors?   * Memory optimized * Storage Optimized * General purpose * **Compute optimized** |

**Module 2**

**Amazon EC2 Pricing**

Amazon EC2 provides a variety of pricing options tailored to different workloads and budgets. The main advantage is flexibility—you pay only for the compute time you use, making it cost-effective for a variety of applications.

**Here are the five main pricing models:**

1. On-demand Instances
2. Reserved Instances
3. EC2 instance saving plans
4. Spot Instances
5. Dedicated hosts

**1. On-Demand Instances**

* **What It Is**: Pay only for the compute time you use, with no upfront costs or long-term commitments.
* **Ideal For**:
  + Short-term, irregular workloads.
  + Testing and developing applications.
  + Workloads with unpredictable usage patterns.
* **Key Features**:
  + Instances run continuously until stopped.
  + Charged per hour or per second, depending on the instance type.
* **Analogy**: On-Demand Instances are like **taking a cab**—you only pay for the time and distance you ride. If you don't need long-term transportation, this is a convenient option.
* **Example Use Case**: A startup testing several server setups for an app prototype.

**2.** **Reserved Instances**

* **What It Is**: A billing discount applied to On-Demand usage in exchange for committing to a 1-year or 3-year term.
* **Types of Reserved Instances**:
  + **Standard Reserved Instances** offer more savings but limited flexibility.
  + **Convertible Reserved Instances** are Flexible, allowing for changes in instance types, size, and tenancy over the term.
* **Suitable** for Steady-state workloads with predictable usage.
* **Payment Options**:
  + **All upfront**: Pay in full at the start.
  + **Partial upfront**: Pay a portion upfront and the rest over time.
  + **No upfront**: Pay as you go.
* **Key Features**:
  + Discounts of up to 75%.
  + Optional capacity reservation in a specific Availability Zone.
* **Analogy**: Reserved Instances are like **a yearly gym membership**—you pay less overall by committing upfront, but you must decide in advance how often you'll use the gym.
* **Example Use Case**: A company running a payroll application that requires consistent compute power every month.

**3. EC2 Instance Savings Plans**

* **What It Is**: A flexible pricing option that provides savings of up to 72% in exchange for committing to an hourly spend for a 1- or 3-year period.
* **Ideal For** workloads that require flexibility across instance types, OS, and tenancy.
* **Key Features**:
  + No need to specify instance type or size upfront.
  + Covers multiple AWS services like EC2, AWS Lambda, and AWS Fargate.
* **Analogy**: Savings Plans are like **a subscription box**—you commit to a monthly amount, and you can pick from a range of items (workloads) as long as it stays within your plan.
* **Example Use Case**: A business expecting variable workloads but wants to save costs by committing to a general usage level.

**4. Spot Instances**

* **What It Is**: Uses spare EC2 capacity at up to 90% off On-Demand prices, but instances can be interrupted if AWS needs the capacity.
  + **Ideal For** workloads with flexibility and resilience to interruptions.
  + Perform batch jobs, background processing, or data analysis.
* **Key Features**:
  + AWS gives a 2-minute warning before reclaiming the instance.
  + Cost savings but requires workload tolerance for interruptions.
* **Analogy**: Spot Instances are like **flying standby**—you pay much less for your ticket, but there's no guarantee you'll get on the plane if it's full.
* **Example Use Case**: A company processing a large survey dataset overnight that doesn’t need real-time results.

**5. Dedicated Hosts**

* **What It Is**: Physical EC2 servers fully dedicated to a single customer, offering control over host placement and compliance with certain regulations.
* **Ideal For**:
  + Workloads requiring strict compliance.
  + Using existing per-core or per-VM software licenses.
* **Key Features**:
  + Exclusive tenancy—no sharing with other customers.
  + Most expensive EC2 pricing option.
* **Analogy**:  
  Dedicated Hosts are like **owning your car**—you have complete control, but it’s more expensive than shared options like a cab or bus.
* **Example Use Case**: A financial organization need complete control over hardware to meet tight rules.

**Conclusion**

* **On-Demand**: Great for testing and unpredictable workloads.
* **Reserved Instances**: Best for predictable, steady workloads.
* **Savings Plans**: Offers flexibility with a term commitment.
* **Spot Instances**: Ideal for cost-efficient, interruption-tolerant jobs.
* **Dedicated Hosts**: Suited for compliance-driven or highly customized workloads.

**Quiz**

**Amazon EC2 pricing**

|  |
| --- |
| Which Amazon EC2 pricing option provides a discount when you specify a number of instances to run specific OS, instance family and size, tenancy in one Region?   * Convertible Reserved instances * EC2 instance saving plans * Spot instances * **Standard Reserved Instances** |

|  |
| --- |
| Which Amazon EC2 pricing option provide a discount when you make an hourly spend commitment to an instance family and region for a 1-year or 3-year term?   * On-demand * **EC2 instance savings plan** * Spot instances * Reserved instances |

**Module 2**

**Scaling Amazon EC2**

**The Problem with On-Premises:**

* **Issue:** Businesses face varying workloads. Buying hardware for peak loads wastes money during low demand, while buying for average loads risks poor customer service during peaks.
* **Analogy:** Running a coffee business with only one barista during peak hours results in long lineups, while hiring extra personnel during low-demand hours wastes money.

**AWS Solution - Scalability and Elasticity:**

* AWS enables you to adjust computing resources dynamically based on demand.
* **Outcome:** You only pay for what you use, ensuring happy customers during peak times and cost savings during slow periods.

**Key Feature: Amazon EC2 Auto Scaling**

* **Function:** Automatically adds or removes EC2 instances based on demand.
* **Real-World Analogy:** Adding more baristas when the line grows (dynamic scaling) or predicting busy hours (predictive scaling) to prepare staff in advance.

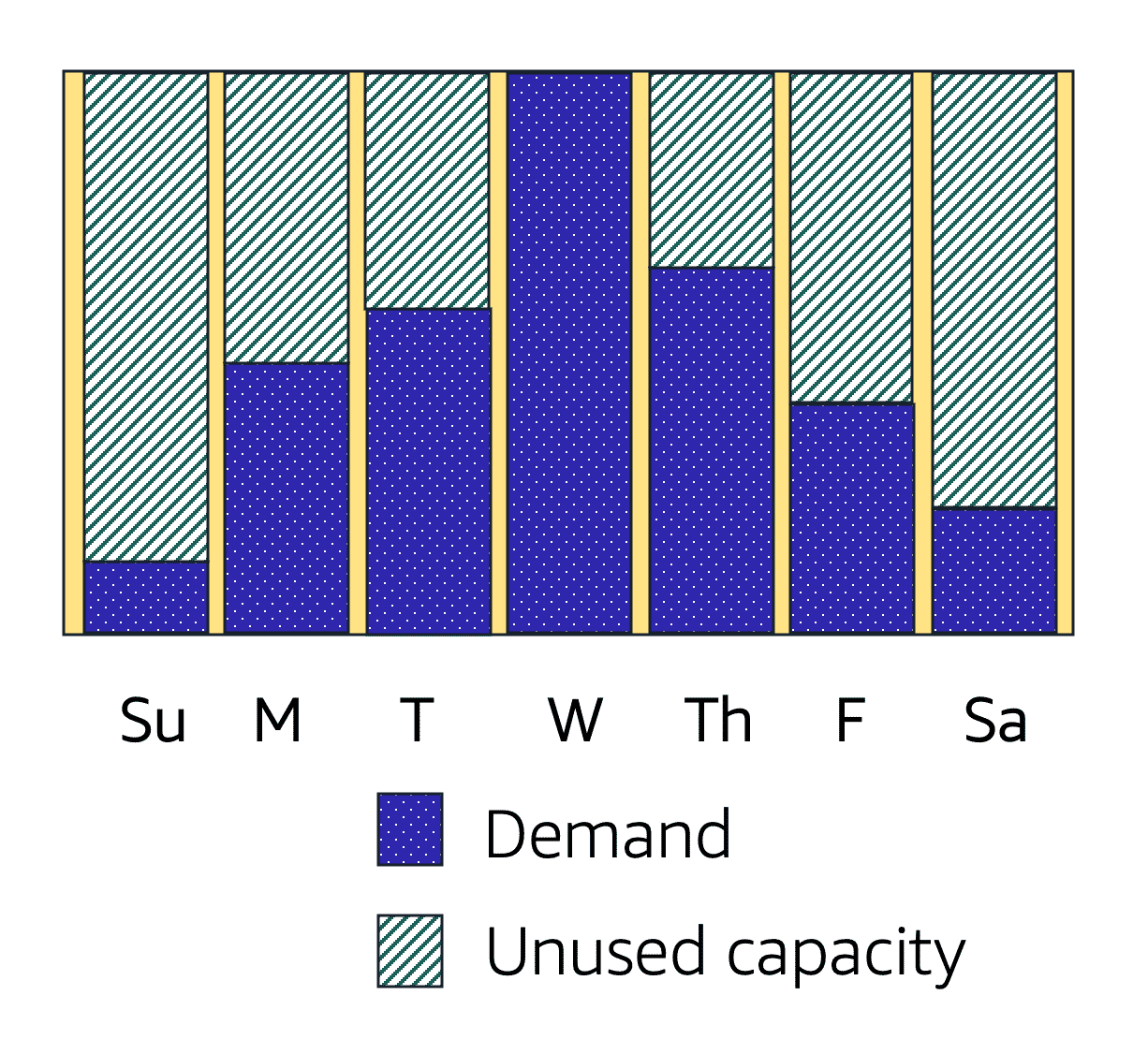
**Two Approaches to Scaling:**

* **Dynamic Scaling:** Responds to immediate changes (e.g., sudden customer rush at a coffee shop).
* **Predictive Scaling:** Prepares resources in advance based on patterns (e.g., knowing morning coffee rush times).

**Benefits of AWS Scaling:**

* Ensures uninterrupted service with no single point of failure.
* Adjusts to demand, reducing idle resources.
* Combines dynamic and predictive scaling for optimized speed and efficiency.

**Amazon EC2 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 can 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.

There are two ways to handle increased demand in a system: **scaling up** and **scaling out**.

**Scaling up** means increasing the power of existing machines. But when demand increases (e.g., more customers), simply making the machines bigger doesn’t necessarily help. For example, Consider Morgan, a café worker who has a larger coffee machine but is unable to serve customers faster simply because the machine is more powerful. The bottleneck is not the equipment, but the number of consumers (workload).

**Analogy**: Imagine Morgan working as a server in a café. Giving Morgan a larger tray will not help him serve clients faster if there are more people waiting in line. Morgan needs more hands (workers) to help with the orders.

**Scaling out**, on the other hand, involves adding more machines (or workers) to distribute the load. If there are more customers, it’s better to have multiple workers (more instances) to serve them quickly.

**Obvious Question**: Why not just add more "order makers" (workers) instead of "order takers" (instances)?

* If the workers are faster than the machines, no need to add more machines. This balance between workers and machines is key to avoiding over-provisioning.

**AWS Advantage**: With **Amazon EC2 Auto Scaling**, you can automatically add or remove workers based on demand. When more workers are needed, new instances are launched, and when demand drops, unnecessary instances are stopped.

**Key Points**

**Flexible Scaling**: Amazon EC2 Auto Scaling adjusts the number of running instances based on your application's needs, which helps in managing fluctuating demand.

**Capacity Settings**:

* **Minimum Capacity**: Set the minimum number of instances needed to run your application (e.g., at least 1 instance).
* **Desired Capacity**: This is the optimal number of instances you want running (e.g., 2 instances) to handle current traffic.
* **Maximum Capacity**: This limits how many instances can be added in peak demand periods (e.g., no more than 4 instances).

**Cost Efficiency**: Auto Scaling helps you avoid paying for excess capacity by scaling out when demand increases and scaling in when it decreases, thus saving costs while still providing a great customer experience.

**Analogy for EC2 Auto Scaling:**

Think of Amazon EC2 Auto Scaling like running a food delivery service.

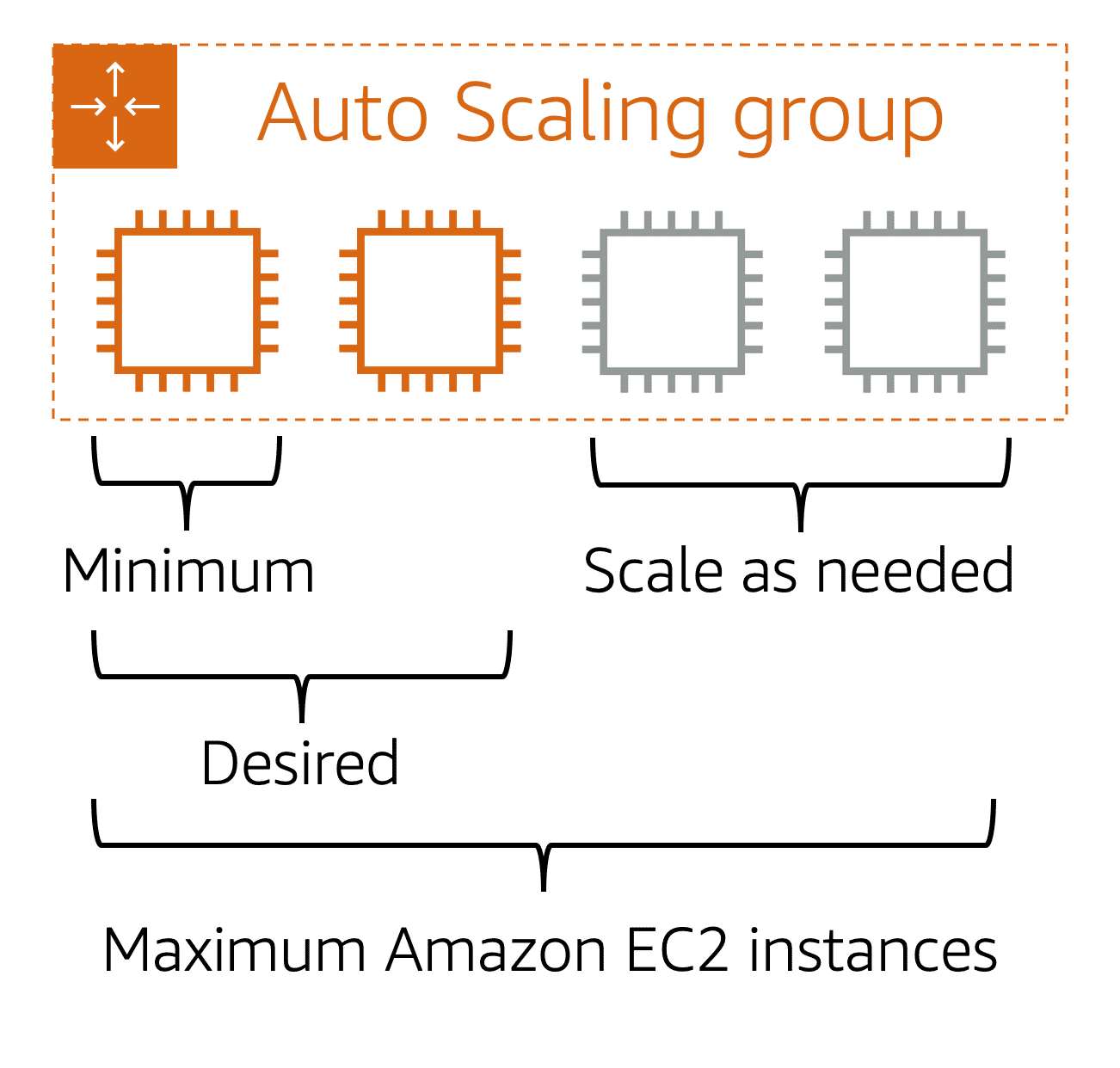
* **Minimum Capacity**: You always need at least one delivery driver available.
* **Desired Capacity**: Ideally, you want two drivers available, as this keeps delivery times short under normal conditions.
* **Maximum Capacity**: But when there’s a sudden surge of orders, you may hire up to four drivers to meet the demand.  
  Once the orders reduce, you can send some drivers home and reduce the cost, ensuring you only pay for the drivers you need.

By breaking the process down like this, it’s easier to understand that scaling isn’t just about adding more resources—it’s about adjusting to the demand and avoiding over-provisioning.

**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.

**Module 2**

**Directing Traffic with Elastic Load Balancing**

While Amazon EC2 Auto Scaling solves the problem of scaling the system by adding or removing EC2 instances, a new problem arises: how to handle traffic distribution across these instances. Without a way to distribute traffic evenly, some instances might be overloaded while others remain idle.

**Scenario with Coffee Shop**: Imagine a coffee shop with three cash registers, but all customers are lining up at just one. This creates a bottleneck, even though other cashiers are available to take orders. To solve this, a host is introduced at the door to direct customers to the shortest line. This ensures the workload is evenly distributed, and customers are served quickly.

**AWS Analogy**: In AWS, when multiple EC2 instances are handling the same application, you need a way to direct incoming requests to the right instance. Without a load balancer, one instance might get overwhelmed with requests, while others sit idle. **Elastic Load Balancing (ELB)** is the service that solves this problem by automatically distributing incoming traffic across multiple EC2 instances, ensuring that no single instance becomes overloaded.

**Elastic Load Balancing**

**What is Elastic Load Balancing (ELB)?** ELB automatically distributes incoming application traffic across multiple resources like Amazon EC2 instances. It acts as a single point of contact for web traffic, directing requests to the most appropriate EC2 instance, ensuring an even workload.

**How ELB and Auto Scaling Work Together**: While Amazon EC2 Auto Scaling adjusts the number of instances based on traffic, ELB ensures that traffic is evenly routed to these instances. As more EC2 instances are added (or removed), ELB ensures that the workload is distributed evenly.

**Scalability and Availability**: ELB is designed to be automatically scalable, meaning as your traffic increases or decreases, ELB adjusts to ensure optimal performance with no extra effort on your part. It works at the regional level, so it provides high availability without you needing to manage it.

**Simplifying Back-End Traffic**: ELB also helps with internal traffic between different application tiers (e.g., front-end and back-end). It directs requests to the backend instances that have the least load, allowing seamless scaling without the front-end needing to know how many backend instances are available.

**Analogy for Elastic Load Balancing:**

Think of a **coffee shop** during different periods of the day:

* **Low-Demand Period**: When only a few customers enter, the shop needs just a few registers (EC2 instances). No need to open extra registers, as the demand is met with the current setup.
* **High-Demand Period**: As more customers enter, more registers are opened (new EC2 instances are added). A host (the **load balancer**) directs customers to the registers with the shortest lines. This ensures that no single register (instance) is overwhelmed, and the workload is evenly distributed.

In this analogy, the **load balancer** is like the host, directing customers (requests) to the appropriate register (EC2 instance), ensuring efficient service and preventing delays.

This makes it easier to understand that **Elastic Load Balancing** works like a host in a busy coffee shop, ensuring that traffic is always distributed efficiently across all available resources (EC2 instances) without manual intervention.

**Module 2**

**Messaging and Queuing**

In the coffee shop, cashiers take orders from customers and pass them to the baristas to prepare. However, if the barista is unavailable (on a break or busy), the cashier gets stuck. To avoid delays and errors, a better approach would be to introduce a buffer where orders are temporarily stored until the barista is ready. This system of storing and processing messages is known as **messaging and queuing**.

**Tightly Coupled vs. Loosely Coupled Systems**: In tightly coupled systems, components are directly dependent on each other. If one component fails, the entire system can break down. For example, if Application A sends messages directly to Application B, and Application B fails, Application A also faces issues.

A more reliable approach is **loosely coupled** architecture. Here, even if one component fails, the other components remain unaffected. In the coffee shop analogy, this is like the cashier placing orders on an order board (queue) instead of directly handing them to the barista. If the barista is unavailable, the orders remain in the queue and are processed when the barista is ready, preventing disruption in the system.

**Analogy for Messaging and Queuing:**

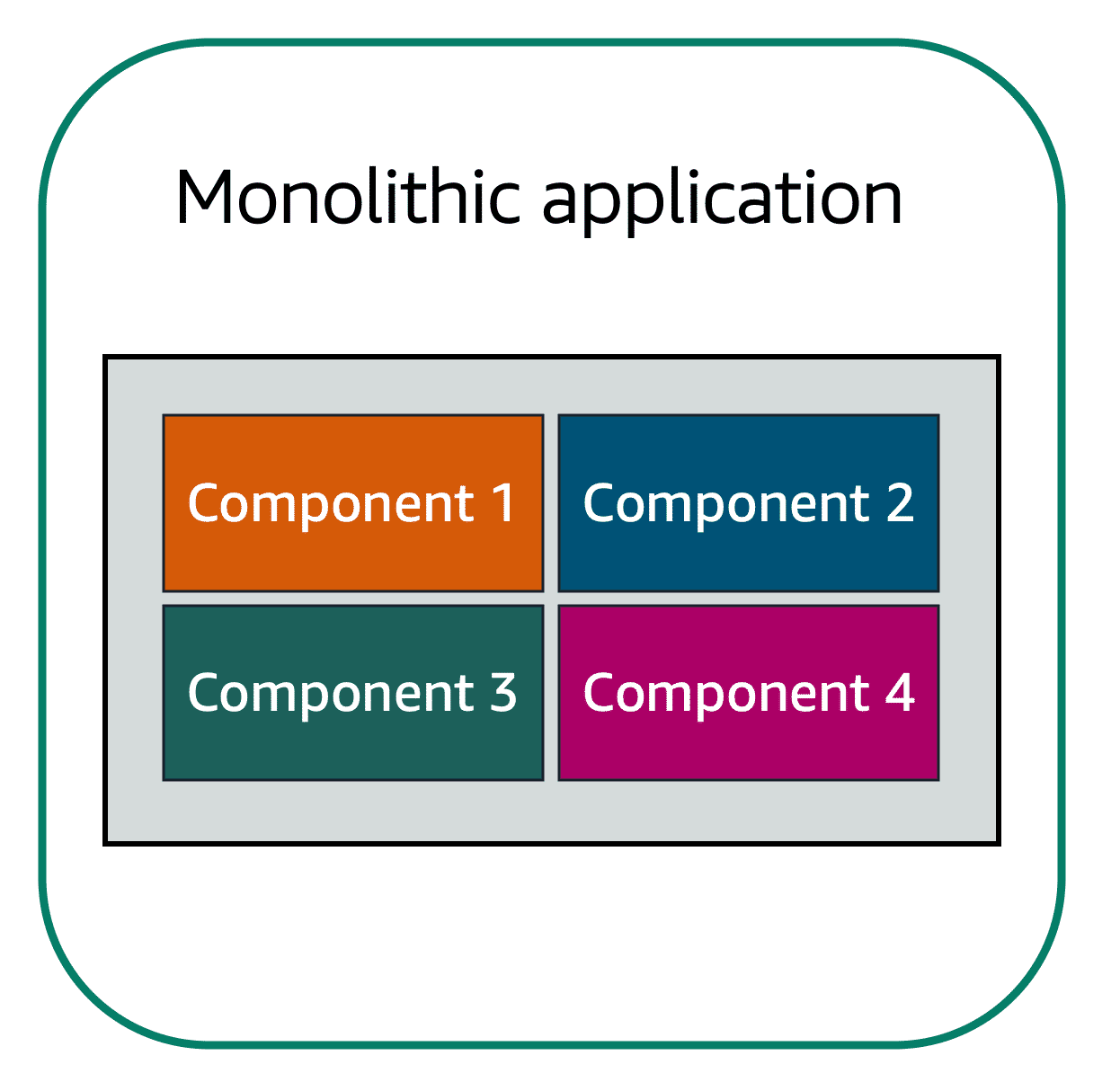
Imagine the **coffee shop** scenario:

* **Tightly Coupled Process**: The cashier takes the order and directly hands it to the barista to make the coffee. If the barista is unavailable, the cashier is stuck, causing delays. Similarly, in tightly coupled applications, if one part fails, everything fails.
* **Loosely Coupled Process (Using Messaging and Queuing)**: Instead of handing orders directly to the barista, the cashier places them on an **order board (queue)**. The barista picks up orders from the board when available, and if the barista is busy, the orders wait in the queue. Even if the barista is unavailable, the cashier can continue taking orders without disruption.

**In AWS**:

* **Amazon SQS** acts like the order board, holding messages until they're ready to be processed.
* **Amazon SNS** works like the cashier sending out updates to customers (subscribers), ensuring that the right people get the right messages at the right time.

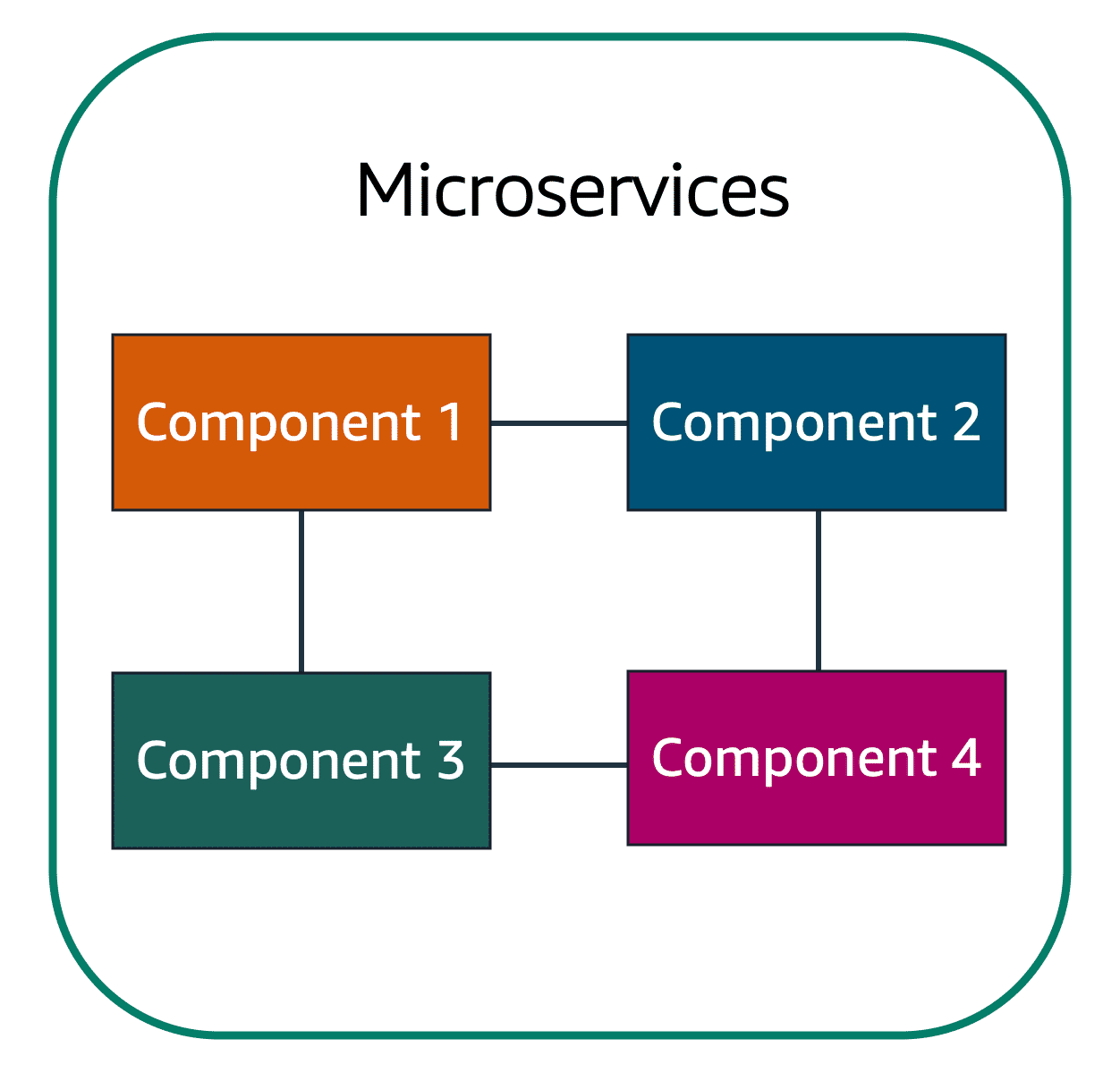
**Monolithic Applications and Microservices**

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)**.

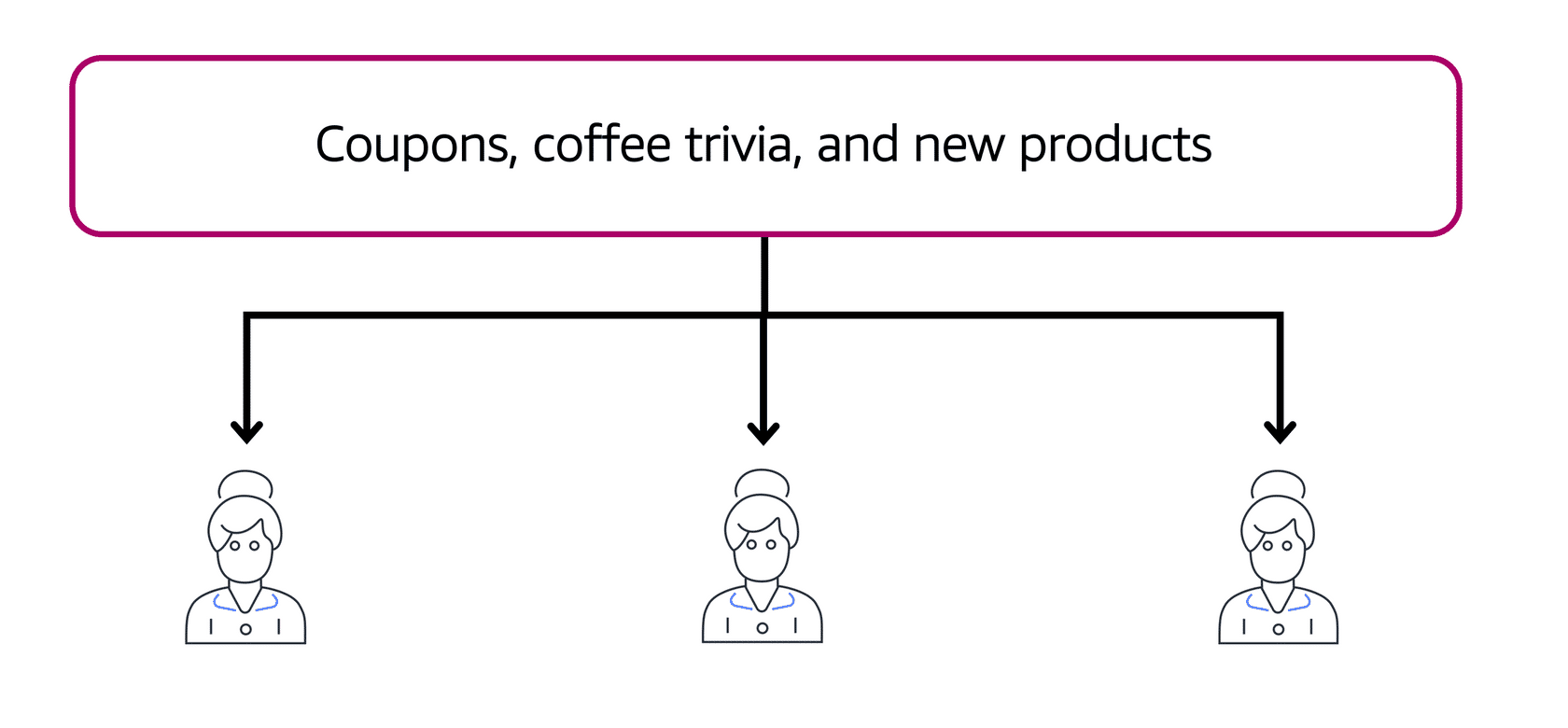
**Module 2**

**Amazon SQS and SNS**

**Amazon Simple Notification Service (SNS)**: SNS follows the **publish/subscribe (pub/sub)** model. In this model, messages are sent to a topic, and subscribers receive the message. For example, in the coffee shop, the cashier can send an update to all customers about new offers via SMS or email. In AWS, SNS sends messages to various endpoints like SQS, Lambda, or even mobile push notifications.

**Two examples of How to use Amazon SNS:**

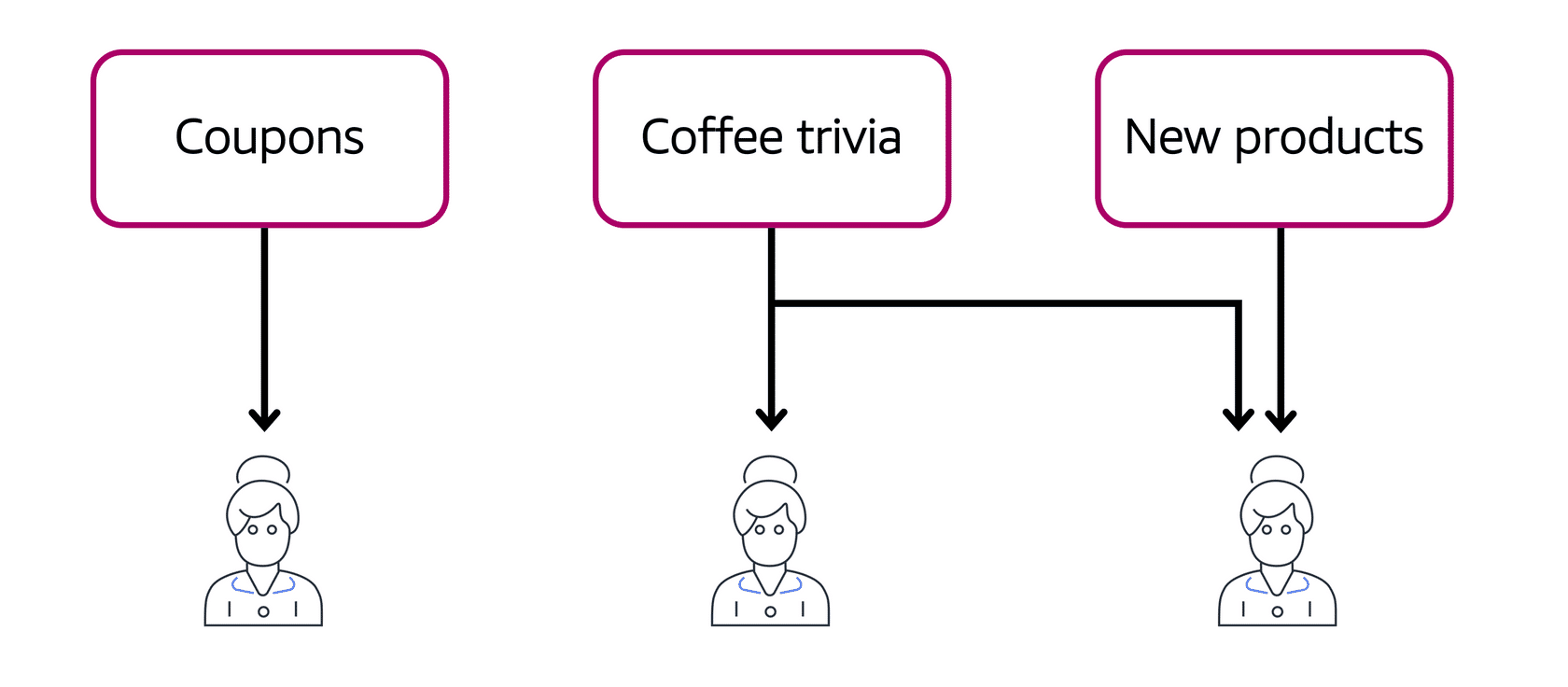
**Step 1: Publishing updates from Single topics**



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.

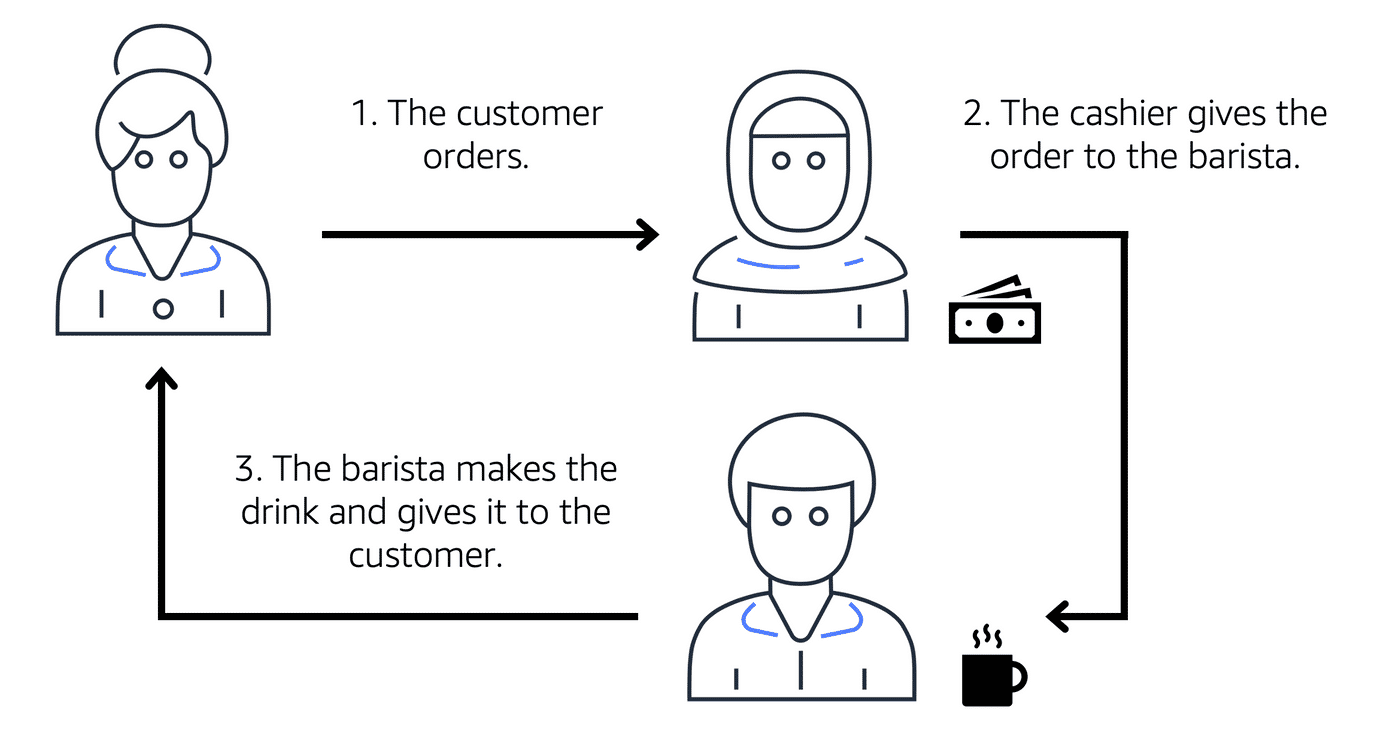
**Amazon Simple Queue Service (SQS)**: SQS allows you to send, store, and receive messages between components without losing any messages, even if services are unavailable. In Amazon SQS, an application sends messages into a queue. A user or service retrieves a message from the queue, processes it, and then deletes it from the queue.

It's like the order board in the coffee shop where messages (orders) are stored until the barista (application) is ready to process them. AWS manages the infrastructure, making it scalable and reliable.

**Two examples of How to use Amazon SQS:**

**Step 1:**

**Example 1: 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.

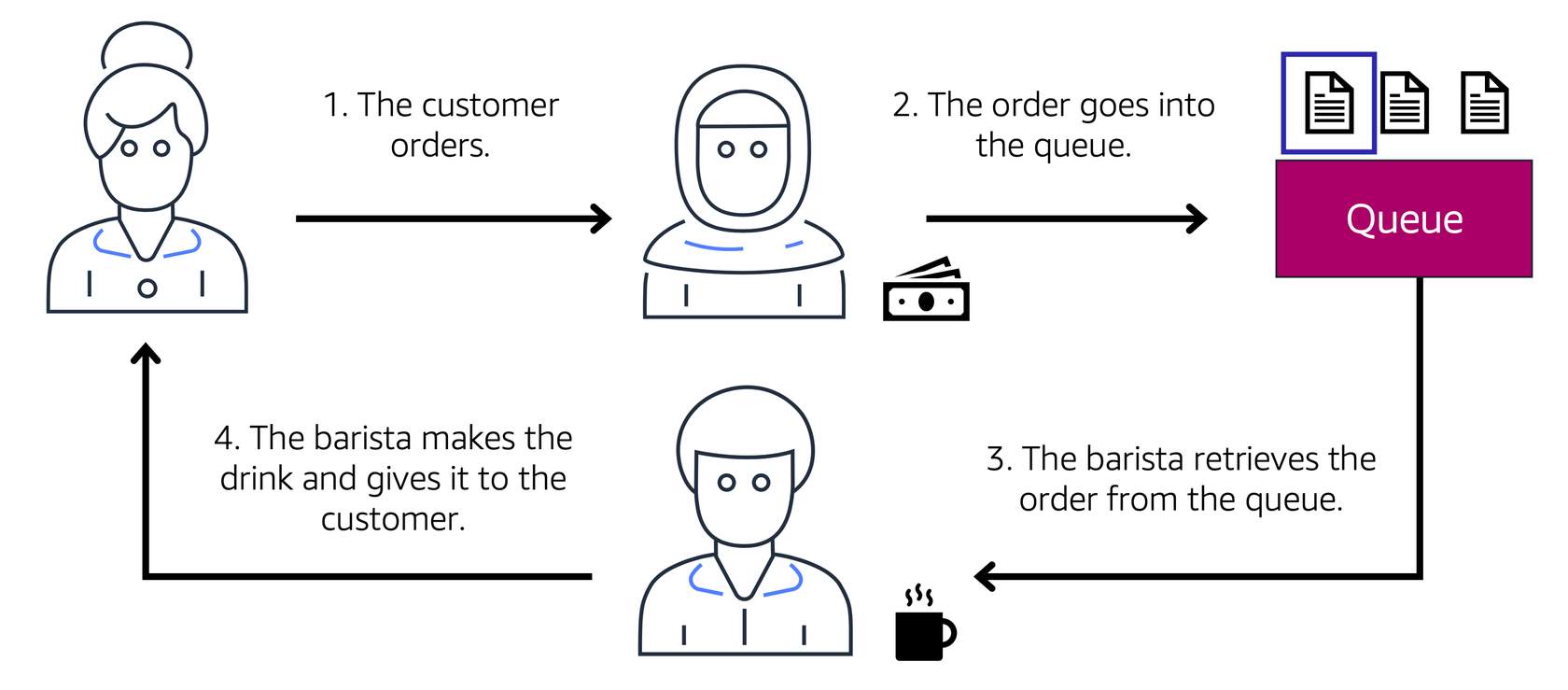
When the next order comes in, the process repeats. This process runs smoothly as long as both the cashier and the barista are coordinated.

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.

**Step 2:**

**Example 2: 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, lets 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.

**Quiz**

**Messaging and Queuing**

|  |
| --- |
| Which AWS service is best choice for publishing messages to subscribers?   * Amazon Simple Queue Service (Amazon SQS) * Amazon EC2 scaling * **Amazon Simple Notification Service (Amazon SNS)** * Elastic Load Balancing |

**Module 2**

**Additional Compute Services**

Amazon EC2 (Elastic Compute Cloud) instances are virtual machines that provide flexibility and scalability for running various types of workloads, from web servers to high-performance computing clusters. However, EC2 requires you to manage the infrastructure, including patching, scaling, and ensuring high availability. For those seeking less management, AWS offers **serverless** options like **AWS Lambda**, which abstracts away the underlying infrastructure, allowing you to focus purely on your application.

If you're not ready for serverless but still want portability and efficiency, AWS provides **container orchestration services** like **Amazon ECS (Elastic Container Service)** and **Amazon EKS (Elastic Kubernetes Service)**. These services help you manage Docker containers on AWS at scale.

For even less management, **AWS Fargate** can be used with ECS or EKS to run containers in a serverless manner, where AWS handles all infrastructure management.

**Compute Services Overview:**

**EC2 (Elastic Compute Cloud)**:

* + Virtual machines you can provision for a wide range of applications.
  + Requires management for scaling, patching, and ensuring availability.
  + Suitable for traditional applications where you need full access to the underlying OS.

**AWS Lambda** (Serverless):

* + A fully managed, serverless compute service for running short tasks.
  + Automatically scales in response to triggers (e.g., events like HTTP requests or file uploads).
  + Limited to tasks that run under 15 minutes, making it suitable for quick backend processes (e.g., web backend, expense reports).
  + AWS handles provisioning, scaling, and maintenance of the infrastructure.

**Container Services** (ECS and EKS):

* + Containers (e.g., Docker) package applications with their dependencies.
  + **ECS**: AWS-managed orchestration service for running containers.
  + **EKS**: Managed Kubernetes service for running containers with Kubernetes orchestration.
  + Both ECS and EKS require you to manage EC2 instances for hosting the containers unless you use **AWS Fargate**.

**AWS Fargate** (Serverless for Containers):

* + A serverless compute platform for ECS or EKS.
  + Manages the infrastructure for running containers, so you don't need to worry about EC2 instances or OS management.
  + Perfect for event-driven or short-running applications that use Docker containers.

**Key Differences:**

* **EC2**: Full control over your virtual machines and underlying infrastructure.
* **Lambda**: Completely serverless, no infrastructure management; focused on short tasks (under 15 minutes).
* **ECS/EKS**: For containerized applications, where you need to choose whether to manage EC2 instances or use Fargate for serverless container management.
* **Fargate**: Serverless platform for containers, simplifying management.

**Analogy for Compute Services:**

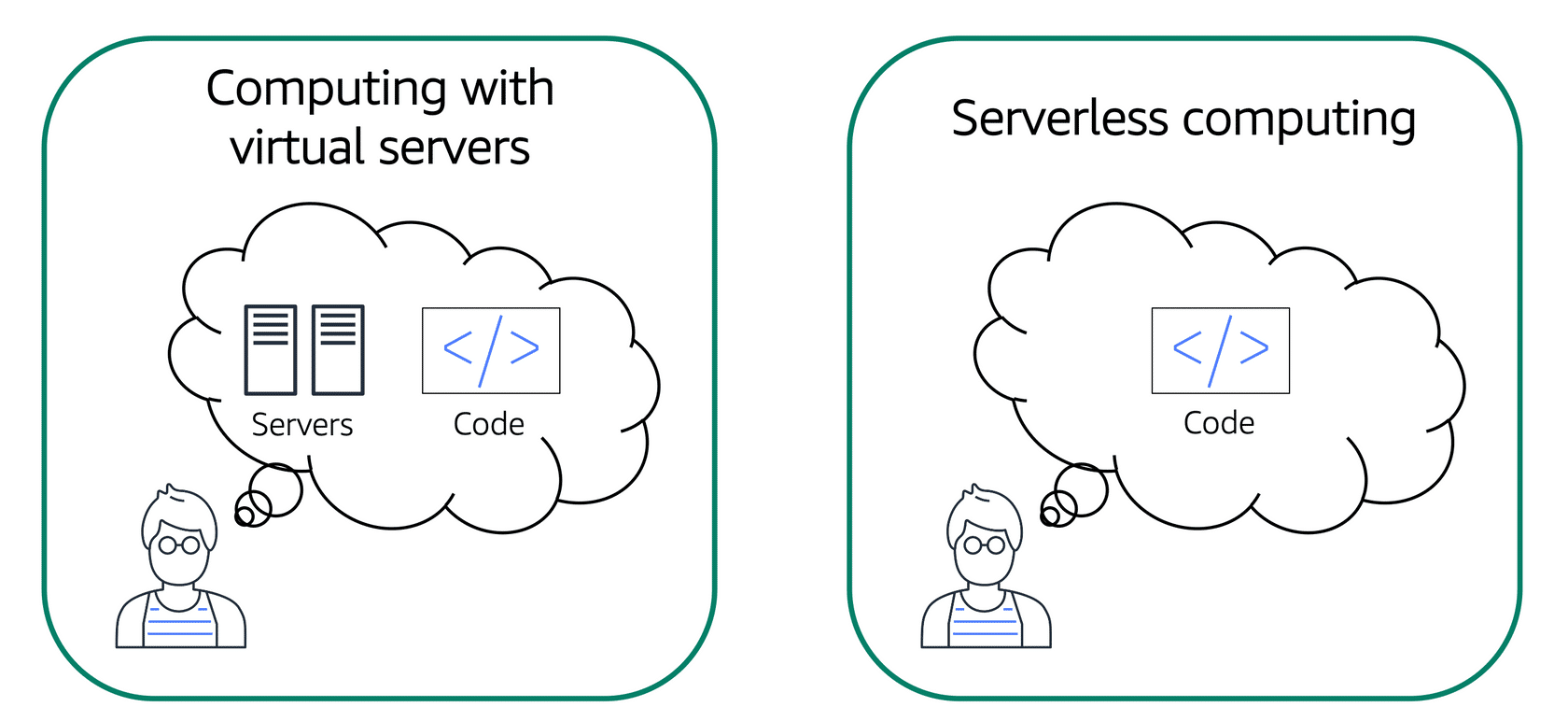
1. **EC2** is like renting an apartment: You manage everything inside (software, scaling, maintenance), but you have full control over the space.
2. **Lambda** is like ordering a meal at a restaurant: You only care about the food (your code), and the restaurant (AWS) handles everything else (scaling, maintenance).
3. **ECS/EKS** are like renting storage units (containers): You store and manage your stuff (applications) but need to manage the infrastructure (EC2 instances) unless you use **Fargate**, which is like renting a fully managed storage unit where the provider handles everything.

This overview highlights the flexibility AWS offers in terms of compute services, from full control with EC2 to completely managed serverless environments with Lambda and Fargate. It gives you the ability to choose the right service depending on your need for control, management, and scalability.

**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.



Comparison between computing with virtual servers (thinking about servers and code) and serverless computing (thinking only about code).

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.

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**.

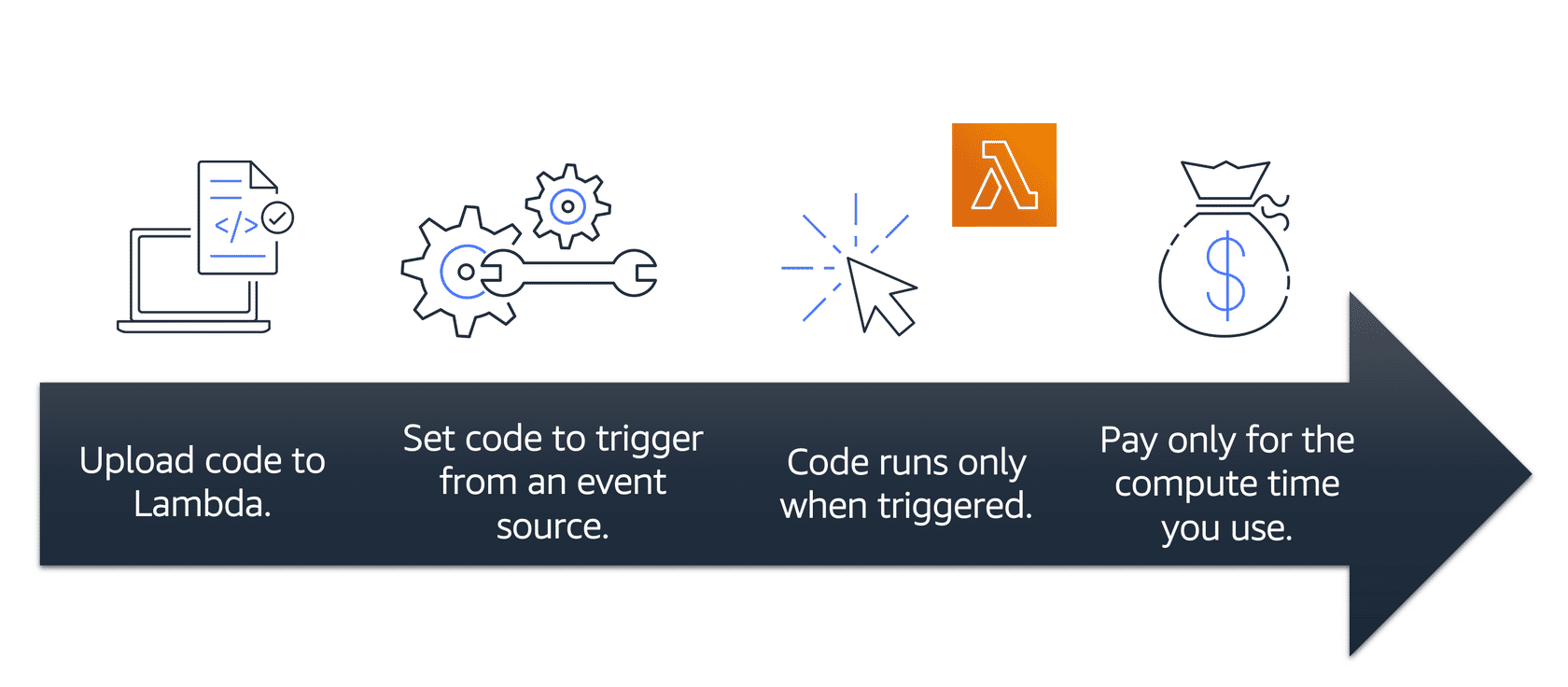
**AWS Lambda**

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.

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**



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.

In AWS, you can also build and run **containerized** applications.

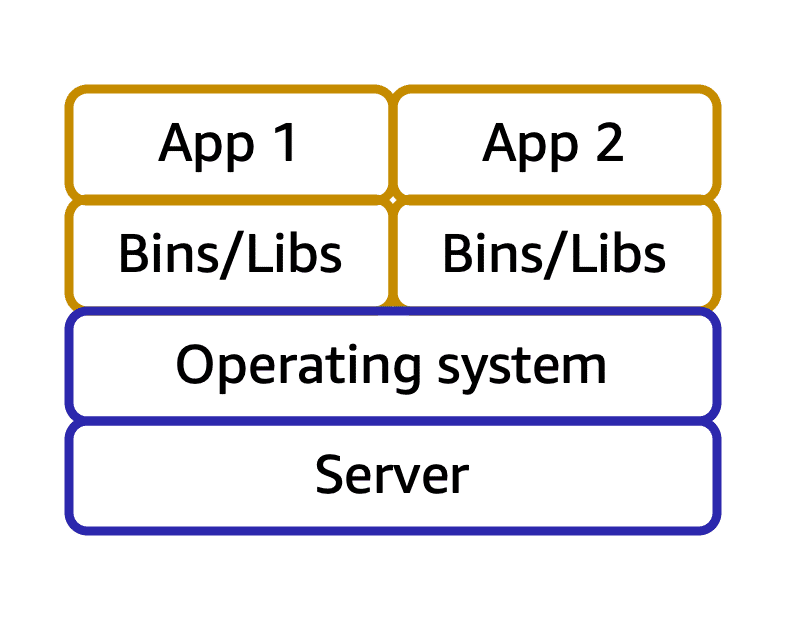
**Containers**

**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.

**An example on how containers work:**

**Step 1:**

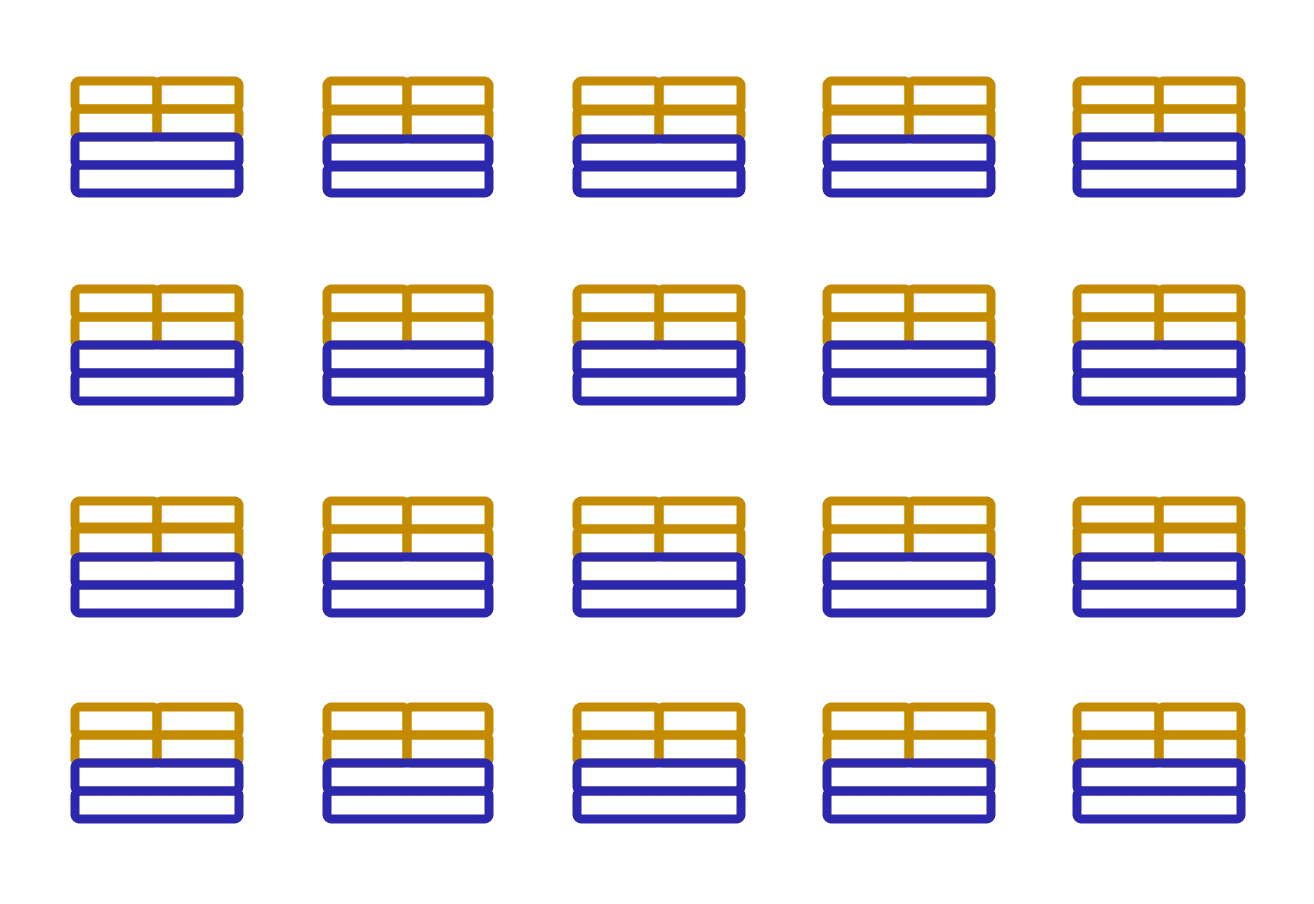
**One host with multiple containers**



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.

**Step 2:**

**Tens of hosts with hundreds of containers**



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.

**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**

**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.

**Module 2**

**Summary**

In Module 2, you learned about the following concepts:

* Amazon EC2 instance types and pricing options
* Amazon EC2 Auto Scaling
* Elastic Load Balancing
* AWS services for messaging, containers, and serverless computing

**Cloud Computing & AWS Overview:**

* **Cloud Computing**: On-demand delivery of IT resources (compute, networking, storage, etc.) with pay-as-you-go pricing.
* **AWS**: Provides various services across multiple categories to build scalable, efficient solutions.

**Amazon EC2 (Elastic Compute Cloud):**

**What it is**: Virtual servers (EC2 instances) that you can launch dynamically based on your needs.

**Instance Types**:

* + **General Purpose**: Balanced resources for diverse workloads.
  + **Compute Optimized**: High-performance computing needs.
  + **Memory Optimized**: For memory-intensive workloads.
  + **Accelerated Computing**: For graphics processing, AI, etc.
  + **Storage Optimized**: For large-scale data storage.

**Scaling Options**:

* + **Vertical Scaling**: Resizing the instance to a larger/smaller size.
  + **Horizontal Scaling**: Adding more instances.
  + **Automated Scaling**: Use **Amazon EC2 Auto Scaling**.

**Traffic Distribution**: Elastic Load Balancer (ELB) distributes traffic across EC2 instances.

**Pricing Models**:

* + **On-Demand**: Pay-as-you-go with no commitment.
  + **Spot Pricing**: Use unused capacity at a discounted rate.
  + **Savings Plans/Reserved Instances**: Commit to usage for discounts.

**Messaging Services:**

**Amazon SQS (Simple Queue Service)**:

* + Decouples system components.
  + Messages stay in the queue until consumed or deleted.

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

* + Sends messages like emails, SMS, push notifications, or HTTP requests.
  + Publishes messages to all subscribers at once.

**Additional Compute Services:**

**Container Services**:

* + **Amazon ECS**: AWS-managed container orchestration for Docker containers.
  + **Amazon EKS**: Kubernetes-based orchestration for containers.
  + **AWS Fargate**: Serverless platform for running containers (no need to manage EC2 instances).

**AWS Lambda**:

* + Serverless compute for event-driven applications.
  + Focused on running short tasks (<15 minutes) triggered by events.
  + Pay only when code runs—no infrastructure to manage.

**Module 2**

**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?   * Memory optimized * Storage Optimized * General purpose * **Compute optimized** |

|  |
| --- |
| What are the contract length options for Amzon EC2 Reserved instances? (Select TWO)   * **1-year** * 2-year * **3-year** * 4-years * 5-years |

|  |
| --- |
| You have a workload that will run for a total 6 of months and can withstand interruptions. What would be the most cost-efficient Amazon EC2 purchasing option?   * **Spot instance** * Reserved instance * Dedicated instance * On-demand instance |

|  |
| --- |
| Which process is an example of Elastic Load Balancing?   * Automatically adjusting the number of EC2 instances to meet demand * Adding a second Amazon EC2 instance during an online store’s popular sale * Removing unneeded Amazon EC2 instances when demand is low * **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 should you use?   * AWS Lambda * Amazon Simple Notification Service (Amazon SNS) * Amazon Simple Queue Service (Amazon SQS) * **Amazon Elastic Kubernetes Service (Amazon EKS)** |

**Module 3**

**Global Infrastructure and Reliability**

**Learning objectives**

In this module, you will learn how to:

* Summarize the **benefits** of the AWS Global Infrastructure.
* Describe the **basic concept of Availability Zones**.
* Describe the **benefits of Amazon CloudFront** and **edge locations**.
* Compare different methods for provisioning AWS services.

**Module 3**

**Introduction to Global Infrastructure and Reliability**

**High Availability Explained Through a Coffee Shop Analogy**:

* Imagine a coffee shop that stays operational despite disruptions like parades blocking the street or a power outage.
* The solution: Multiple shop locations across the city ensure customers can always access their coffee.

**High Availability in AWS**:

* AWS ensures high availability by avoiding dependency on a single data center.
* Instead of one massive center, AWS uses a network of **Regions** and **Availability Zones** around the world to provide reliability and redundancy.

**Key Concept – Regions**:

* **Regions** are collections of data centers strategically placed across the globe.
* If one data center experiences an issue (like a parade or disaster), applications remain operational by relying on resources in another location.

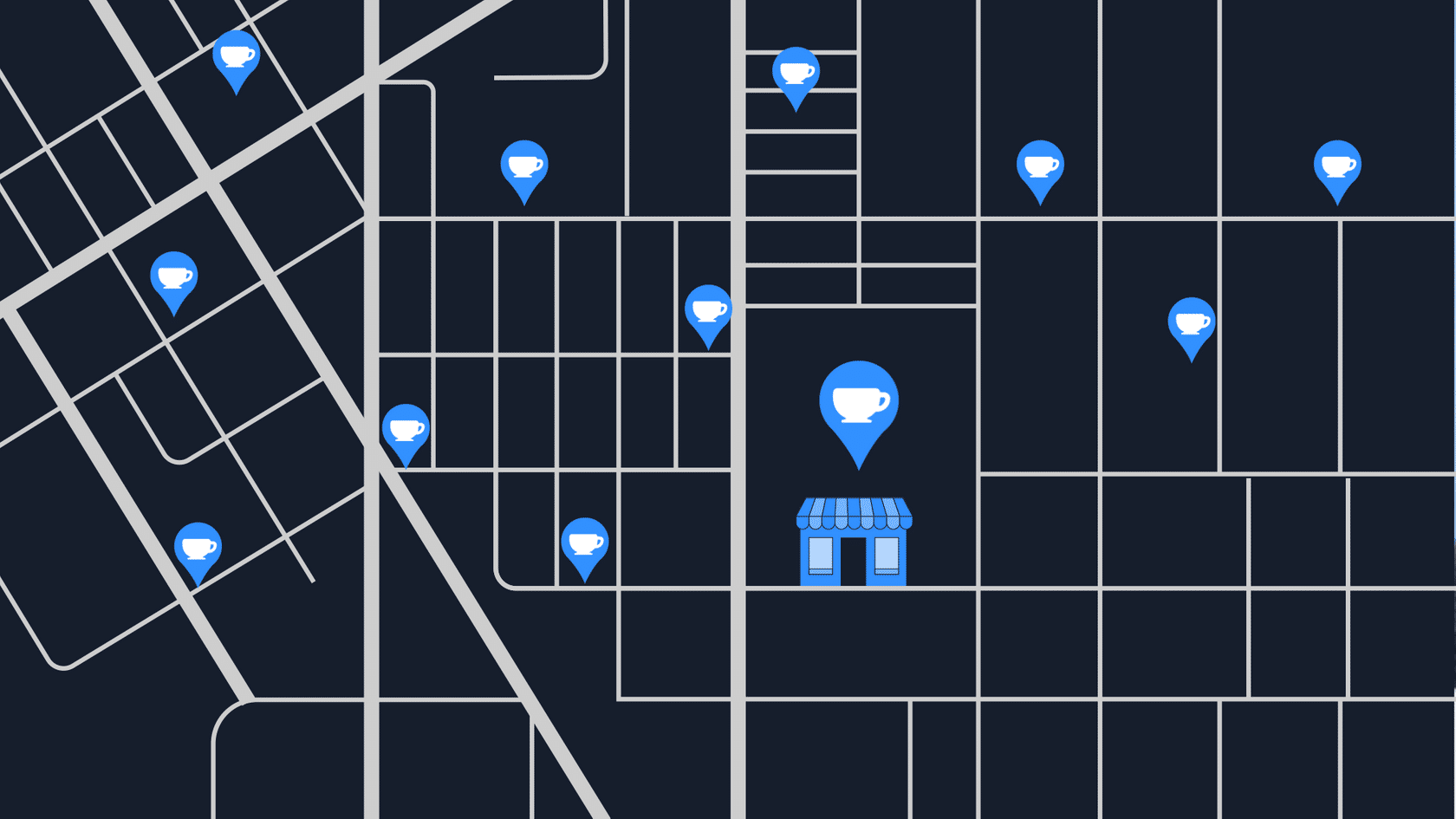
**Benefits of High Availability in AWS**:

* Ensures customer satisfaction and uninterrupted service.
* Minimizes risks of outages from localized disruptions (e.g., power failures, natural disasters).

**Takeaway**:

* Just as a chain of coffee shops ensures business continuity, AWS’s global infrastructure is designed to keep applications running no matter the circumstance.

**Building a Global Footprint**



To understand the AWS global infrastructure, consider the coffee shop. If an event such as a parade, flood, or power outage impacts one location, customers can still get their coffee by visiting a different location only a few blocks away.

This is similar to how the AWS global infrastructure works.

**Module 3**

**AWS Regions**

**The Need for AWS Regions**:

* + Historically, businesses had to run applications and store data in their **own data centers**. This required owning and maintaining expensive infrastructure, including duplicate hardware, staff, and backup solutions.
  + Running a second data center as a disaster recovery solution was often financially unfeasible, leaving many businesses reliant on **backups and hope**.
  + AWS addressed this challenge by creating **Regions**, groups of data centers distributed globally to ensure **high availability**, **fault tolerance**, and **disaster recovery**.

**What Are AWS Regions?**

* + **Regions** are clusters of data centers (multiple facilities) located globally in places like Ohio, Dublin, Paris, Tokyo, and Sydney.
  + Each Region provides compute, storage, and other services for businesses to run their applications.
  + AWS Regions are interconnected using a **high-speed fiber network**, enabling worldwide operations with low latency.

**Data Security and Sovereignty**:

* + Regions are **isolated** from one another, ensuring **no data flows in or out** of a Region unless you explicitly authorize it.
  + This design supports **regional data sovereignty**, meaning data in a specific Region (e.g., Frankfurt) is subject to that Region’s local laws and cannot leave without your permission.
  + Example: If compliance requires financial data to remain within Germany, the **Frankfurt Region** ensures this by default.

**How AWS Regions Support Businesses**:

AWS Regions support businesses by allowing them to select regions that meet their operational and compliance requirements.

**For example:**

* **Proximity**: Host apps near users to reduce latency (e.g., Singapore Region for Southeast Asia).
* **Compliance**: Ensure data complies with local regulations (e.g., use London Region for UK-specific compliance).

AWS operates numerous data centers in each region to provide disaster recovery infrastructure.

**Factors for Choosing a Region**

**Compliance**: Depending on your company and location, you might need to run your data out of specific areas. For example, if your company requires all of its data to reside within the boundaries of the UK, you would choose the London Region.

Not all companies have location-specific data regulations, so you might need to focus more on the other three factors.

**Proximity to your customers**: Data transfer speed (latency) depends on the distance between the Region and your customers.

Selecting a Region that is close to your customers will help you to get content to them faster.

For example, your company is based in Washington, DC, and many of your customers live in Singapore. You might consider running your infrastructure in the Northern Virginia Region to be close to company headquarters, and run your applications from the Singapore Region.

**Available Services withing a Region**:

Sometimes, the closest Region might not have all the features that you want to offer to customers. AWS is frequently innovating by creating new services and expanding on features within existing services. However, making new services available around the world sometimes requires AWS to build out physical hardware one Region at a time.

Suppose that your developers want to build an application that uses Amazon Braket (AWS quantum computing platform). As of this course, Amazon Braket is not yet available in every AWS Region around the world, so your developers would have to run it in one of the Regions that already offers it.

**Pricing**:

Costs vary by region due to factors like taxes, energy costs, and infrastructure expenses.

Suppose that you are considering running applications in both the United States and Brazil. The way Brazil’s tax structure is set up, it might cost 50% more to run the same workload out of the São Paulo Region compared to the Oregon Region.

**Global Scalability and Flexibility**:

AWS Regions offer global scalability and flexibility, eliminating the need for enterprises to manage their own disaster recovery solutions. Additionally, applications and data can run in many Regions for optimal fault tolerance.

AWS's extensive network of regions enables businesses to scale globally.

**Examples and Scenarios**:

* A German business may use the Frankfurt Region to comply with local data requirements.
* A company with primarily Japanese consumers may choose the Tokyo Region to reduce latency.
* A startup experimenting with quantum computing may choose a region that supports Amazon Braket, even if it is further away.

By leveraging AWS Regions, businesses gain the flexibility to meet their compliance, performance, and budget requirements while ensuring global scalability and resilience.

***Clusters*** *refer to a group of interconnected servers or systems that work together to perform tasks efficiently. In AWS Regions, clusters are multiple data centers within a Region that collectively provide services like compute, storage, and networking, ensuring* ***high availability*** *and* ***fault tolerance****.*

**Module 3**

**Availability Zones**

**Definition of Availability Zones (AZs):**

AZs are individual data centers or groups of data centers within an AWS Region.

Each AZ is physically separate and has its own **redundant power, networking, and connectivity** to ensure fault tolerance.

**Why Use Multiple AZs?**

Running applications in a single building or data center is risky because disasters (like power outages or natural events) can cause downtime.

AZs are geographically spread tens of miles apart within a Region, ensuring **low-latency communication** (single-digit milliseconds) while minimizing the risk of a single failure affecting multiple AZs.

**Best Practices for High Availability:**

Always deploy resources across **at least two AZs** to ensure redundancy and fault tolerance.

If one AZ fails, applications can continue running in other AZs, ensuring uninterrupted operations.

**Regional Services:**

Some AWS services, like **Elastic Load Balancers (ELBs)**, are regional by default.

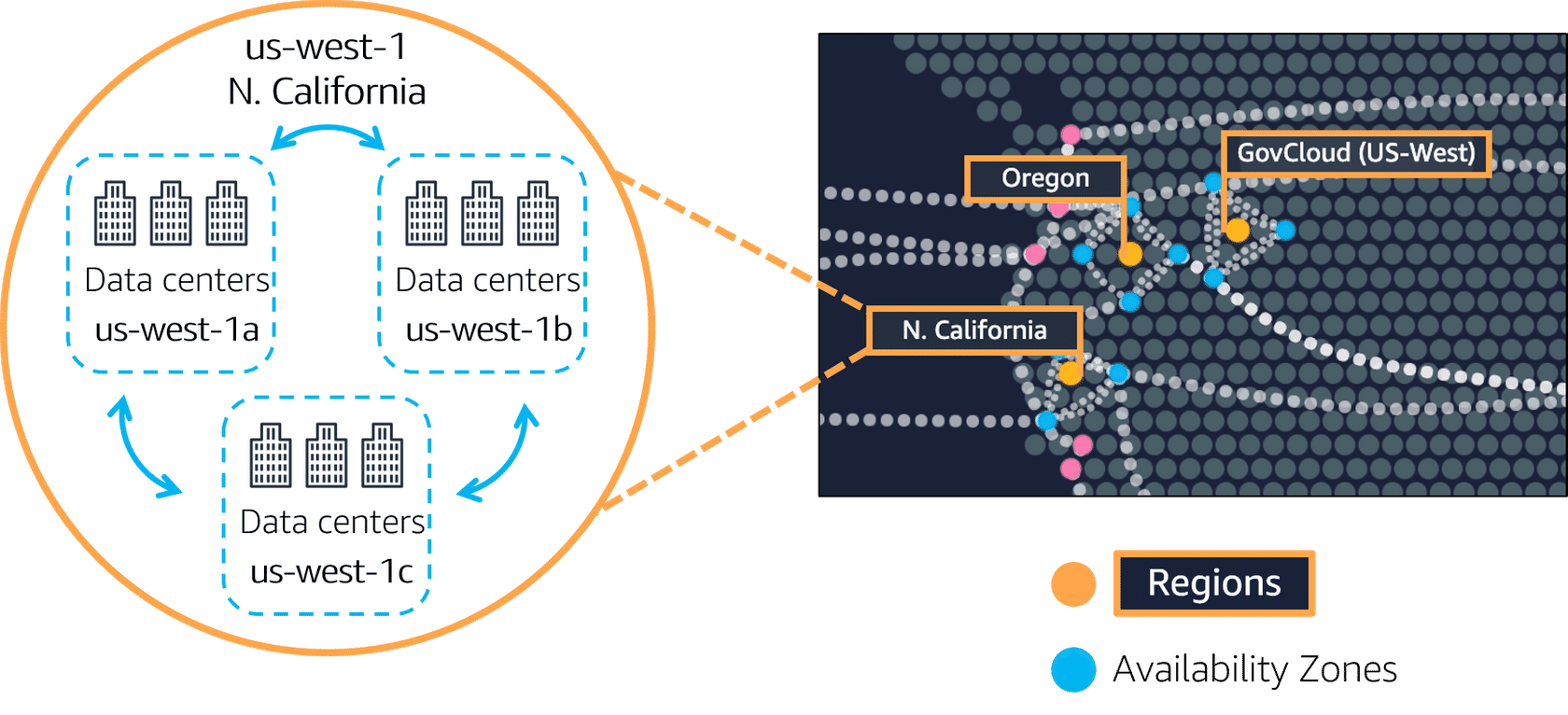
These services automatically operate across multiple AZs, ensuring high availability without additional configuration.

**Disaster Recovery Planning:**

Deploying resources in multiple AZs ensures that even in large-scale disasters, your application can quickly recover and continue to function.

**Analogy:**

Think of AZs as stores in a shopping mall. If one store shuts down due to maintenance or flooding, customers can still shop at other stores nearby without much delay, ensuring the mall's business continues smoothly.

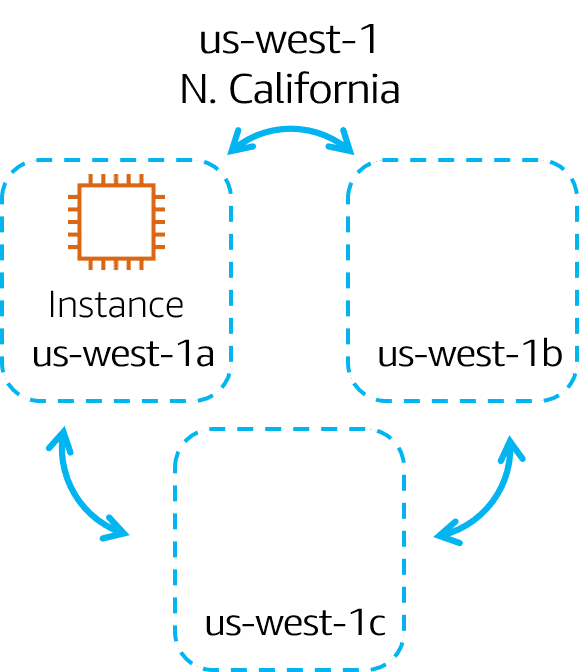


Spotlight on the us-west-1 Region. Northern California, Oregon, and GovCloud (US-West) are separate Regions. The Northern California Region is called us-west-1, and this Region contains three AZs (1a, 1b, and 1c). Then, within each AZ there are three data centers.

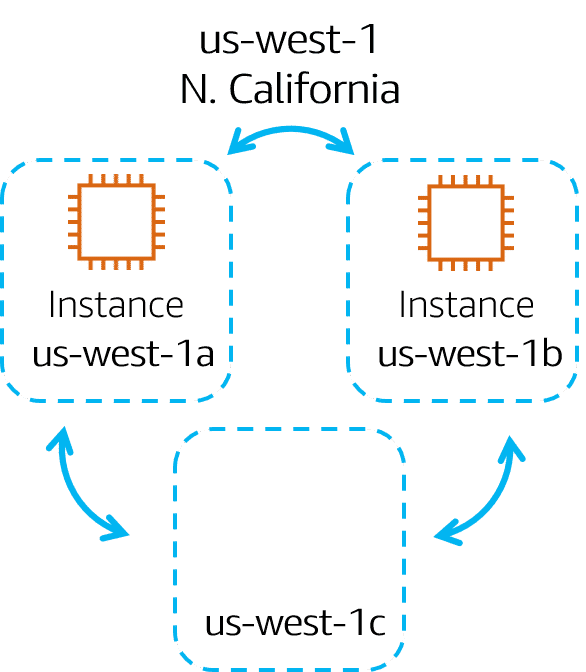
An **Availability Zone** is a single data center or a group of data centers within a Region. Availability Zones are located tens of miles apart from each other. This is close enough to have low latency (the time between when content requested and received) between Availability Zones. However, if a disaster occurs in one part of the Region, they are distant enough to reduce the chance that multiple Availability Zones are affected.

An example of running Amazon EC2 instances in multiple Availability Zones.

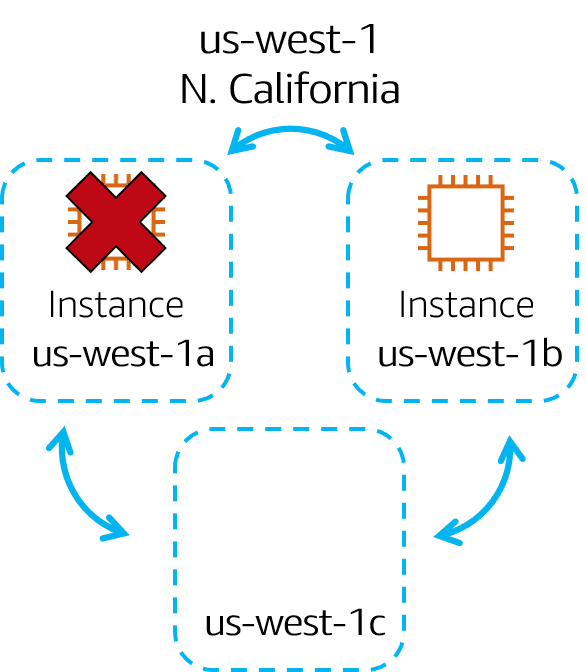
**Amazon EC2 instance in a single Availability Zone**



Suppose that you’re running an application on a single Amazon EC2 instance in the Northern California Region. The instance is running in the us-west-1a Availability Zone. If us-west-1a were to fail, you would lose your instance.

**Amazon EC2 instances in multiple Availability Zones**

A best practice is to run applications across at least two Availability Zones in a Region. In this example, you might choose to run a second Amazon EC2 instance in us-west-1b.

**Availability Zone failure**

If us-west-1a were to fail, your application would still be running in us-west-1b.

**Quiz**

**Availability Zone**

|  |
| --- |
| Which statement best describes an Availability Zone?   * A geographical area that contains AWS resources * **A single data center or group of data centers within a Region** * A data center that an AWS service uses to perform service-specific operations * A service that you can use to run AWS infrastructure within your own on-premises data center in a hybrid approach |

**Module 3**

**Edge Locations**

**Concept of Edge Locations**

* **Definition**: Edge locations are AWS data centers placed worldwide **to cache and deliver content closer to users**, reducing latency.
* **Purpose**:
  + Enhance communication speed and content delivery for users far from AWS Regions.
  + Provide low latency and high transfer speeds for global customers.

**How Edge Locations Work**

* **Analogy**: Like satellite coffee shops for customers in a new city, Edge locations cache copies of data closer to users rather than requiring them to interact with distant Regions.
* **Use Case Example**:
  + If your data is hosted in Tokyo but accessed by users in Mumbai, a cached copy of the data can be placed in Mumbai to reduce delays.

**Content Delivery Networks (CDNs)**

* **AWS CDN**: Amazon CloudFront.
  + **Function**: Delivers data, videos, applications, and APIs globally.
  + **Technology**: Uses Edge locations for low latency communication.

**4. Additional Functions of Edge Locations**

* **Amazon Route 53**:
  + A DNS service running on Edge locations.
  + Directs users to the correct web locations with low latency and reliability.

**AWS Outposts**

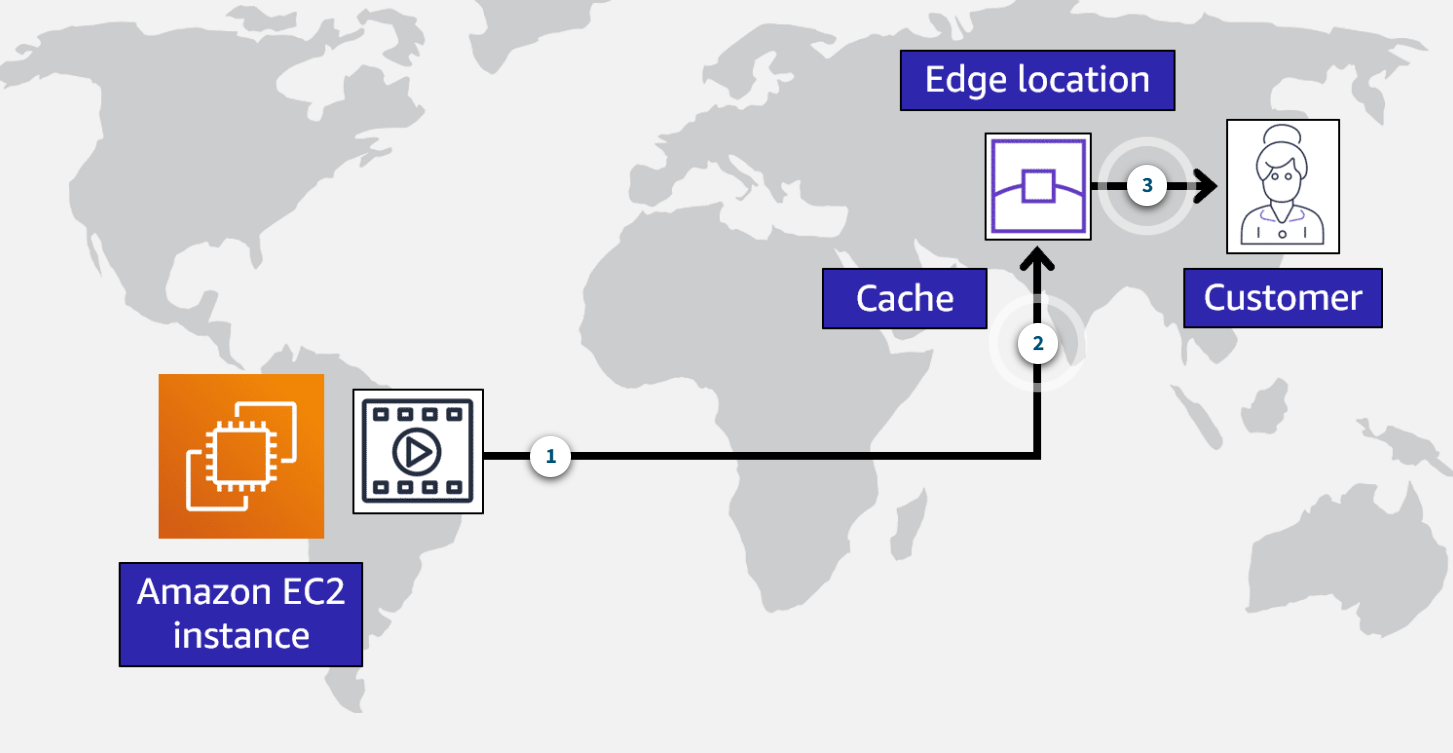
* **Purpose**: For organizations who require AWS services within their own buildings.
* **What It Is**:
  + A fully operating mini-AWS Region is installed directly in a customer's data center.
  + Offers full AWS functionality while remaining isolated in the customer’s premises.
* **Target Audience**: Businesses with specific needs that demand staying in their own facilities.

**Key Takeaways**

1. **Regions**: Geographically isolated areas providing AWS services.
2. **Availability Zones**: Ensure high availability and disaster recovery across physically separated locations.
3. **Edge Locations**: Bring content closer to customers worldwide, powered by Amazon CloudFront and Amazon Route 53.

**Edge Locations:**

An **edge location** is a site that Amazon CloudFront uses to store cached copies of your content closer to your customers for faster delivery.



**1. Origin:** Suppose that your company’s data is stored in Brazil, and you have customers who live in China. To provide content to these customers, you don’t need to move all the content to one of the Chinese Regions.

**2. Edge Location:** Instead of requiring your customers to get their data from Brazil, you can cache a copy locally at an edge location that is close to your customers in China.

**3. Customer:** When a customer in China requests one of your files, Amazon CloudFront retrieves the file from the cache in the edge location and delivers the file to the customer. The file is delivered to the customer faster because it came from the edge location near China instead of the original source in Brazil.

**Module 3**

**How to Provision AWS Resources**

**Interacting with AWS Services**

* **API as the Core**:
  + AWS services are managed via **API calls** (Application Programming Interfaces).
  + APIs allow you to **provision, configure, and manage resources** like EC2 instances or Lambda functions.
* **Tools to Access AWS APIs**:
  + **AWS Management Console** (GUI-based).
  + **AWS Command Line Interface (CLI)**.
  + **AWS Software Development Kits (SDKs)**.
  + **Other tools**, like AWS CloudFormation.

**AWS Management Console**

* **What It Is**:
  + A **browser-based, visual interface** for managing AWS resources.
* **Use Cases**:
  + Ideal for beginners or non-technical users.
  + Great for:
    - Viewing and managing AWS bills.
    - Monitoring resources.
    - Building test environments.
* **Limitations**:
  + **Manual Process**: Requires clicking through screens to configure and launch resources (e.g., EC2 instances).
  + **Error-Prone**: Human errors, like forgetting configurations, are more likely in manual workflows.
  + **Inefficient**: Not suitable for production environments or repetitive tasks.

**AWS Command Line Interface (CLI)**

* **What It Is**:
  + A terminal-based tool for **making API calls** to AWS services.
* **Advantages**:
  + **Scriptable & Repeatable**: Commands can be reused, reducing human error.
  + **Automation**: Allows running scripts automatically (e.g., scheduled tasks).
* **Example**:
  + Launching an EC2 instance via CLI saves time compared to manually repeating the process in the console.

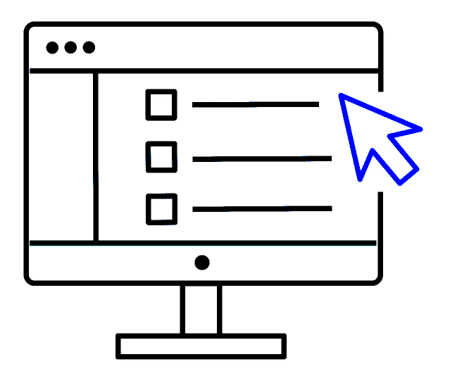
**AWS Software Development Kits (SDKs)**

* **What It Is**:
  + Tools for interacting with AWS using various **programming languages**.
* **Key Benefits**:
  + Developers can **integrate AWS functionality into their applications** without needing low-level API calls.
  + Reduces manual resource creation.

**Key Takeaways**

1. APIs are the backbone of provisioning and managing AWS resources.
2. Tools like the **Management Console** are beginner-friendly but manual and error-prone.
3. The **CLI** enables scriptable and repeatable commands for automation.
4. **SDKs** allow developers to seamlessly integrate AWS functionality into their applications.

**Ways to interact with AWS services**

**AWS Management Console:**

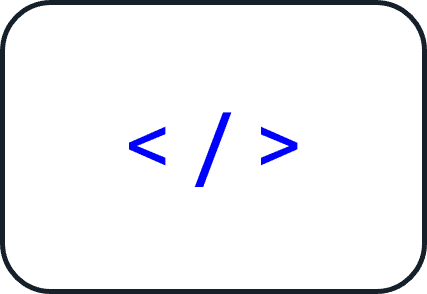
The **AWS Management Console** is a web-based interface for accessing and managing AWS services. You can quickly access recently used services and search for other services by name, keyword, or acronym. The console includes wizards and automated workflows that can simplify the process of completing tasks.

You can also use the AWS Console mobile application to perform tasks such as monitoring resources, viewing alarms, and accessing billing information. Multiple identities can stay logged into the AWS Console mobile app at the same time.

**AWS Command-Line Interface:**

To save time when making API requests, you can use the **AWS Command Line Interface (AWS CLI)**. AWS CLI enables you to control multiple AWS services directly from the command line within one tool. AWS CLI is available for users on Windows, macOS, and Linux.

By using AWS CLI, you can automate the actions that your services and applications perform through scripts. For example, you can use commands to launch an Amazon EC2 instance, connect an Amazon EC2 instance to a specific Auto Scaling group, and more.

**Software Development Kits:**

Another option for accessing and managing AWS services is the **software development kits (SDKs)**. SDKs make it easier for you to use AWS services through an API designed for your programming language or platform. SDKs enable you to use AWS services with your existing applications or create entirely new applications that will run on AWS.

To help you get started with using SDKs, AWS provides documentation and sample code for each supported programming language. Supported programming languages include C++, Java, .NET, and more.

**Advanced Tools for AWS Resource Management**

AWS provides managed tools like **Elastic Beanstalk** and **CloudFormation** to simplify, automate, and standardize the provisioning of resources.

**AWS Elastic Beanstalk: Simplified Application Deployment**

* **What It Is**: A service designed to help deploy and manage Amazon EC2-based environments easily.
* **How It Works**:
  + You provide your **application code** and configuration preferences.
  + Elastic Beanstalk builds the required environment (e.g., networks, EC2 instances, scaling, and Elastic Load Balancers) for you.
* **Key Features**:
  + **Focus on Applications**: You concentrate on your business logic rather than infrastructure management.
  + **Save Configurations**: Makes it easy to save and redeploy environment configurations.
  + **Visibility & Control**: While automated, it still allows you to see and manage underlying resources.

**AWS CloudFormation: Infrastructure as Code**

* **What It Is**: A tool for defining and managing AWS resources through **declarative code** using **JSON** or **YAML** templates.
* **How It Works**:
  + You define the resources you need in a **CloudFormation template** (e.g., storage, compute, databases).
  + CloudFormation interprets the template and provisions the resources by automating the necessary API calls.
* **Key Features**:
  + **Declarative Approach**: You specify **what** to build, not **how** to build it.
  + **Wide Resource Support**: Works with storage, analytics, machine learning, and more.
  + **Consistency Across Environments**: Reuse the same template across regions or accounts for identical setups.
  + **Error Reduction**: Eliminates manual errors with automated deployments.
  + **Parallel Provisioning**: Provisions multiple resources simultaneously for efficiency.

**Comparing Provisioning Tools**

* **AWS Management Console**:
  + **Strengths**: Easy for beginners, visual, and great for learning.
  + **Limitations**: Manual and unsuitable for automation or large-scale environments.
* **AWS Command Line Interface (CLI)**:
  + **Strengths**: Scriptable, repeatable, and enables automation.
  + **Use Case**: Ideal for creating and managing resources via terminal commands.
* **AWS Software Development Kits (SDKs)**:
  + **Strengths**: Integrates AWS into applications using programming languages.
  + **Use Case**: Perfect for developers building applications with AWS services.
* **AWS Elastic Beanstalk**:
  + **Strengths**: Simplifies EC2-based application deployments.
  + **Use Case**: Best for managing application environments without infrastructure complexity.
* **AWS CloudFormation**:
  + **Strengths**: Automates and standardizes infrastructure provisioning as code.
  + **Use Case**: Ideal for large-scale, repeatable, and consistent deployments across multiple regions or accounts.

**Key Takeaways**

1. The **Management Console** is great for learning but not suited for automation.
2. The **CLI** and **SDKs** allow scripting and programming interactions with AWS.
3. **Elastic Beanstalk** simplifies EC2-based application deployment by automating infrastructure setup.
4. **CloudFormation** uses declarative templates to automate resource provisioning with consistency and reduced error.

**AWS Elastic Beanstalk**

With **AWS Elastic Beanstalk**, you provide code and configuration settings, and Elastic Beanstalk deploys the resources necessary to perform the following tasks:

* Adjust capacity
* Load balancing
* Automatic scaling
* Application health monitoring

**AWS CloudFormation**

With **AWS CloudFormation**, you can treat your infrastructure as code. This means that you can build an environment by writing lines of code instead of using the AWS Management Console to individually provision resources.

AWS CloudFormation provisions your resources in a safe, repeatable manner, enabling you to frequently build your infrastructure and applications without having to perform manual actions. It determines the right operations to perform when managing your stack and rolls back changes automatically if it detects errors.

**Module 3**

**Summary**

In Module 3, you learned about the following concepts:

* AWS Regions and Availability Zones
* Edge locations and Amazon CloudFront
* The AWS Management Console, AWS CLI, and SDKs
* AWS Elastic Beanstalk
* AWS CloudFormation

**AWS Global Infrastructure Overview**:

* AWS operates with **logical clusters of data centers** known as **Availability Zones (AZs)**.
* **Availability Zones** are grouped into **Regions**, which are spread across the globe.
* **Best Practice**: Always deploy infrastructure across **at least two Availability Zones** for better reliability and fault tolerance.

**AWS Services and Multi-AZ Deployments**:

* Some AWS services automatically distribute traffic across multiple Availability Zones. These include:
  + **Elastic Load Balancing**
  + **Amazon SQS** (Simple Queue Service)
  + **Amazon SNS** (Simple Notification Service)

**Edge Locations and Content Delivery**:

* **Edge locations** are used to **speed up content delivery** to users globally, reducing latency.
* Services like **Amazon CloudFront** use these locations to cache content and deliver it closer to customers.

**Edge Devices**:

* **AWS Outposts** allows you to run AWS infrastructure directly in your own data center, extending AWS services to on-premise environments.

**Provisioning AWS Resources**:

* AWS provides various methods to provision resources, including:
  + **AWS Management Console** (Web interface)
  + **AWS SDK** (Software Development Kit)
  + **AWS CLI** (Command Line Interface)
  + **AWS Elastic Beanstalk** (Platform-as-a-Service)
  + **AWS CloudFormation** (Infrastructure-as-Code, allows automating and managing infrastructure setup)

**Takeaway**:

* AWS offers a **globally available** infrastructure, and setting up resources is **easy** through multiple provisioning options. Always aim for **multi-AZ deployments** to ensure high availability and reliability.

**Module 3**

**Quiz**

|  |
| --- |
| Which statement is TRUE for the AWS global infrastructure?   * A Region consists of a single Availability Zone. * An Availability Zone consists of two or more Regions. * **A Region consists of three or more Availability Zones.** * An Availability Zone consists of a single Region. |

|  |
| --- |
| Which factors should be considered when selecting a Region? (Select TWO.)   * **Compliance with data governance and legal requirements** * **Proximity to your customers** * Access to 24/7 technical support * Ability to assign custom permissions to different users * Access to the AWS Command Line Interface (AWS CLI) |

|  |
| --- |
| Which statement best describes Amazon CloudFront?   * A service that enables you to run infrastructure in a hybrid cloud approach * A serverless compute engine for containers * A service that enables you to send and receive messages between software components through a queue * **A global content delivery service** |

|  |
| --- |
| Which site does Amazon CloudFront use to cache copies of content for faster delivery to users at any location?   * Region * Availability Zone * **Edge location** * Origin |

|  |
| --- |
| Which action can you perform with AWS Outposts?   * Automate actions for AWS services and applications through scripts. * Access wizards and automated workflows to perform tasks in AWS services. * Develop AWS applications in supported programming languages. * **Extend AWS infrastructure and services to different locations including your on-premises data center.** |

**Module 4**

**Networking**

**Learning objectives**

In this module, you will learn how to:

* Describe the **basic concepts of networking**.
* Describe the **difference between public and private networking** resources.
* Explain a virtual private gateway using a real life scenario.
* Explain a **virtual private network** (VPN) using a real life scenario.
* Describe the **benefit of AWS Direct Connect**.
* Describe the **benefit of hybrid deployments**.
* Describe the **layers of security** used in an IT strategy.
* Describe the services customers use to interact with the AWS global network.

**Module 4**

**Introduction to Networking**

**Amazon Virtual Private Cloud (VPC)**:

* **VPC** is a logically isolated section of the AWS Cloud where you can launch and manage AWS resources within a defined virtual network.
* It provides control over networking, including resource accessibility and internet connectivity.

**Subnets**:

* **Subnets** are subdivisions of a VPC, representing ranges of IP addresses.
* Resources in a VPC can be grouped into:
  + **Public Subnets**: Public subnets allow for internet access, such as for web servers or front-facing apps.
  + **Private Subnets**: Private subnets are used for backend resources, such as databases or application servers, and do not have direct internet access.

**Analogy - Coffee Shop**:

* **Cashier** = **Public Subnet**: Like a cashier who interacts with customers, public-facing resources in a VPC communicate with the internet.
* **Barista** = **Private Subnet**: Like baristas focusing on brewing coffee, private resources handle backend tasks without external interference.

**Key Concept**:

* VPCs provide a **secure and flexible** environment to manage public and private resources, ensuring proper isolation and controlled access within your cloud infrastructure.

**Module 4**

**Connectivity to AWS**

**Amazon VPC (Virtual Private Cloud)**:

* Amazon VPC (Virtual Private Cloud) is a private network in AWS that allows you to define IP address ranges for resources such as EC2 instances and Elastic Load Balancers.
* In a VPC, resources are divided into **subnets** (groups of IP addresses) to define whether they are **publicly** or **privately** accessible.

**Internet Gateway (IGW)**:

* An Internet Gateway (IGW) is a "doorway" that allows public internet traffic to enter and exit your virtual private cloud (VPC).
* **Analogy**: Like a front door in a coffee shop that lets customers in and out to access public services (e.g., websites).

**Virtual Private Gateway (VGW)**:

* A Virtual Private Gateway (VGW) is a private gateway that securely connects your on-premises network to your VPC via a VPN.
* **Analogy**: Like a private bus route for employees only, where authentication (e.g., badge access) ensures that only approved users can access private resources.

**Challenges with VPN Connections**:

* VPN connections, while **secure and private**, rely on the **public internet**, which may result in b**andwidth sharing** with other users.
* **Traffic slowdowns** and potential reliability issues.

**AWS Direct Connect**:

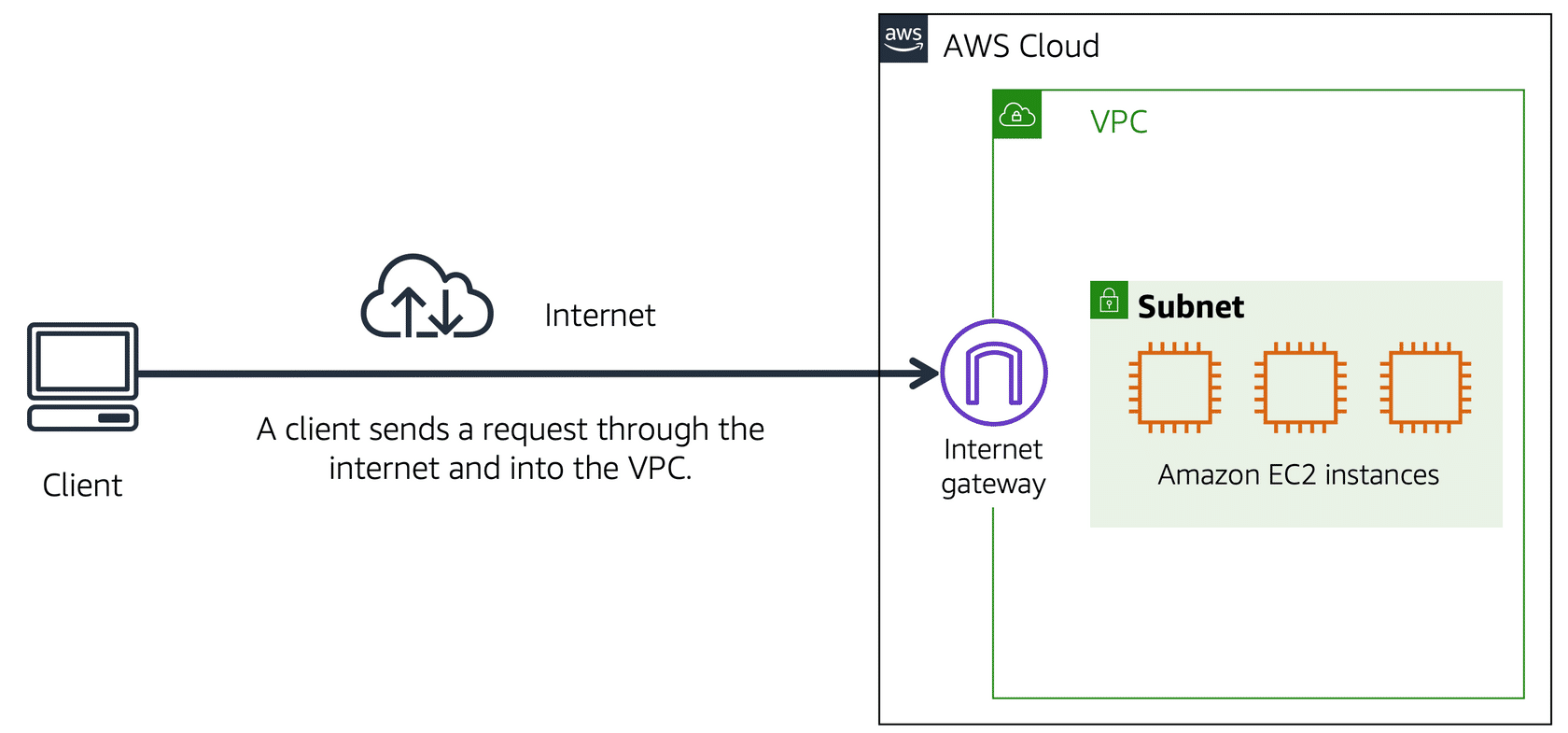
* AWS Direct Connect enables a private fiber connection from your data center to AWS.
* Benefits include low latency and high security.
* No reliance on shared internet bandwidth.
* Suitable for regulatory compliance and high performance requirements.
* **Analogy**: Imagine a private doorway connecting your studio to a coffee shop, bypassing traffic and providing exclusive access.

**Key Point**:

* A single VPC can have **multiple gateways** (e.g., IGW, VGW, Direct Connect) to manage different types of resources across **public and private subnets**.

**Internet gateway**

To allow public traffic from the internet to access your VPC, you attach an **internet gateway** to the VPC.



Internet gateway icon attached to a VPC that holds three EC2 instances. An arrow connects the client to the gateway over the internet indicating that the client's request has gained access to the VPC.

An internet gateway is a **connection between a VPC and the internet**. You can think of an internet gateway as being similar to a doorway that customers use to enter the coffee shop. Without an internet gateway, no one can access the resources within your VPC.

**What if you have a VPC that includes only private resources?**

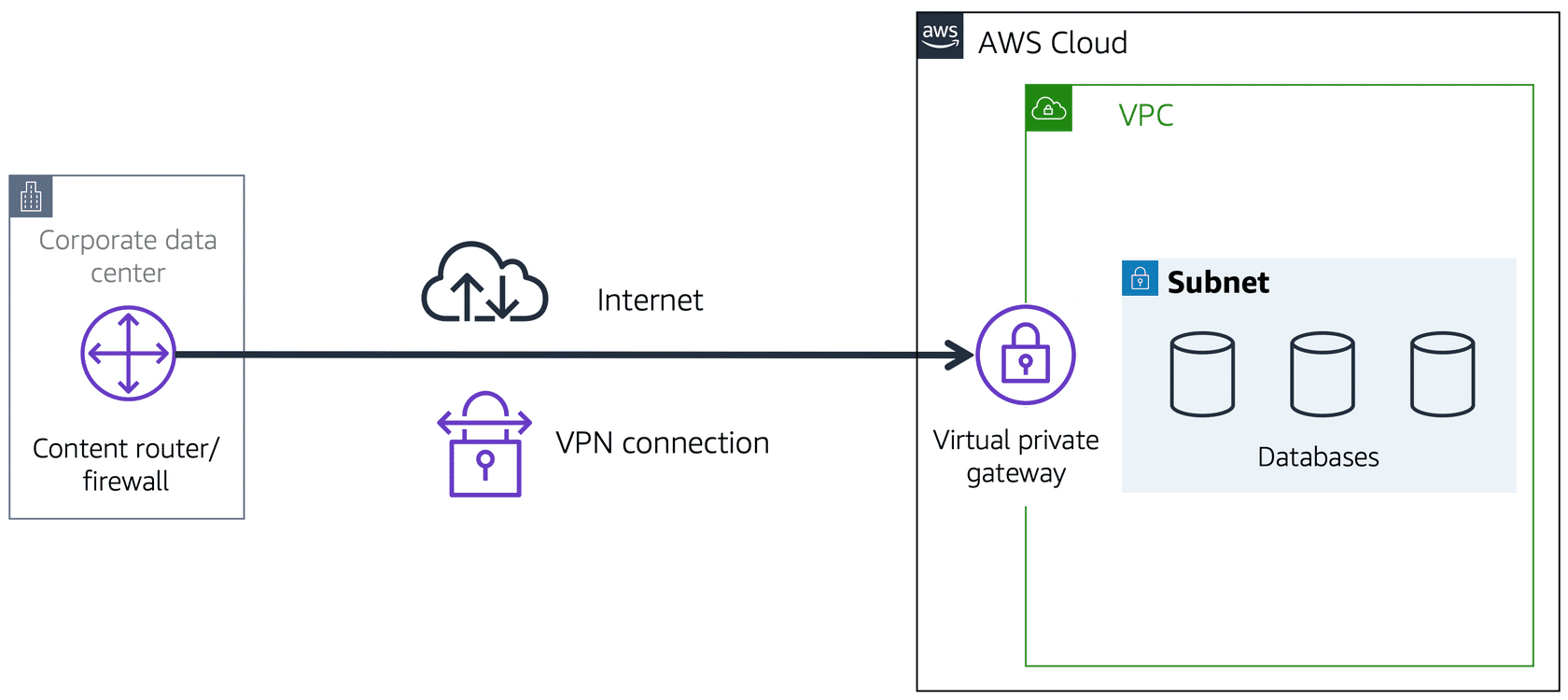
**Virtual private gateway**

To access private resources in a VPC, you can use a **virtual private gateway**.

Here’s an example of how a virtual private gateway works. You can think of the internet as the road between your home and the coffee shop. Suppose that you are traveling on this road with a bodyguard to protect you. You are still using the same road as other customers, but with an extra layer of protection.

The bodyguard is like a virtual private network (VPN) connection that encrypts (or protects) your internet traffic from all the other requests around it.

The virtual private gateway is the component that allows protected internet traffic to enter into the VPC. Even though your connection to the coffee shop has extra protection, traffic jams are possible because you’re using the same road as other customers.



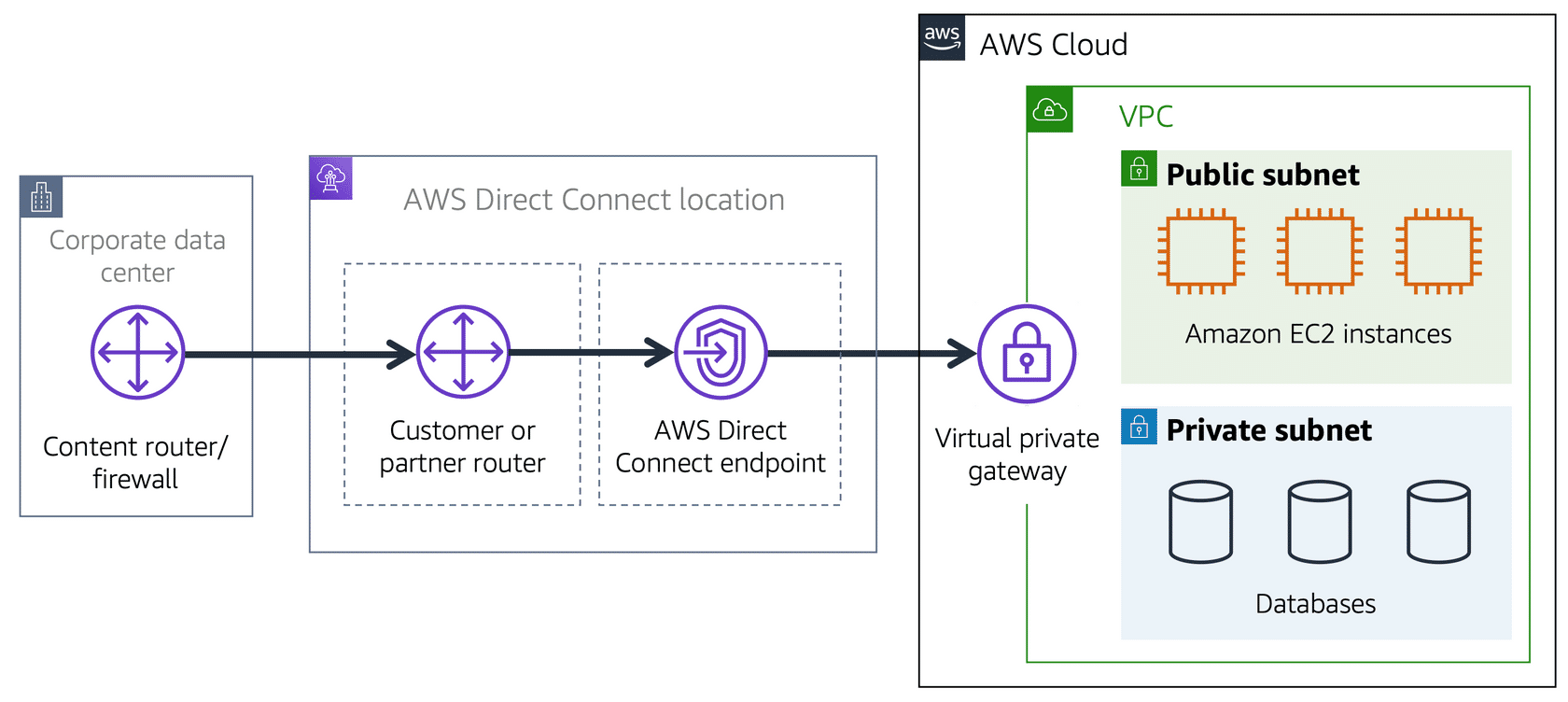
A virtual private gateway enables you to establish a virtual private network (VPN) connection between your VPC and a private network, such as an on-premises data center or internal corporate network. A virtual private gateway allows traffic into the VPC only if it is coming from an approved network.

**AWS Direct Connect**

**AWS Direct Connect** is a service that lets you to establish a dedicated private connection between your data center and a VPC.

Suppose that there is an apartment building with a hallway directly linking the building to the coffee shop. Only the residents of the apartment building can travel through this hallway.

This private hallway provides the same type of dedicated connection as AWS Direct Connect. Residents are able to get into the coffee shop without needing to use the public road shared with other customers.



A corporate data center routes network traffic to an AWS Direct Connect location. That traffic is then routed to a VPC through a virtual private gateway. All network traffic between the corporate data center and VPC flows through this dedicated private connection.

The private connection that AWS Direct Connect provides helps you to reduce network costs and increase the amount of bandwidth that can travel through your network.

**Module 4**

**Subnets and Network Access Control Lists**

**Overview of VPC Security**:

* AWS offers numerous layers of security for VPCs, including network hardening, application security, identity and access management (IAM), DDoS protection, encryption, and more. Consider a VPC to be a fortress, with no traffic entering or exiting without permission.

**Subnets and Traffic Control**:

* **Subnets** divide a VPC into smaller segments, with public subnets connecting to the internet gateway (IGW) and private subnets remaining isolated.
* NACLs are used to restrict traffic permissions within subnets.

**Network Access Control Lists (NACLs)**:

* NACLs control **inbound and outbound traffic** at the **subnet level**.
  + **Analogy**: NACLs act like **passport control officers** inspecting every packet entering or leaving a subnet.
  + Packets are accepted or rejected based on defined rules.
* **Stateless**: NACLs do not remember previous decisions; they check all traffic every time.

**Security Groups**:

* Security groups control **traffic to and from individual EC2 instances**.
* **Analogy**: Security groups act like a **doorman**:
  + Incoming traffic is checked against rules.
  + Outgoing traffic is **allowed by default**.
* **Stateful**: Security groups remember connections, allowing return traffic without re-evaluating.

**Traffic Flow Example**: Sending a packet from **Instance A (Subnet 1)** to **Instance B (Subnet 2)**:

* 1. Packet exits Instance A’s **security group** (outgoing traffic allowed by default).
  2. Packet crosses Subnet 1’s **NACL** (checked against outbound rules).
  3. Packet enters Subnet 2’s **NACL** (checked against inbound rules).
  4. Packet reaches Instance B’s **security group** (checked against incoming traffic rules).
* Return traffic follows the same path:
  1. Security groups (stateful) remember and allow return traffic.
  2. NACLs (stateless) re-check return traffic against rules at every subnet boundary.

**Key Differences Between NACLs and Security Groups**:

* **NACLs**: Stateless, operate at the **subnet level**, inspect every packet.
* **Security Groups**: Stateful, operate at the **instance level**, remember connections.

**Best Practice**:

* Use **both NACLs and security groups** for comprehensive security, following the principle of **defense in depth**.

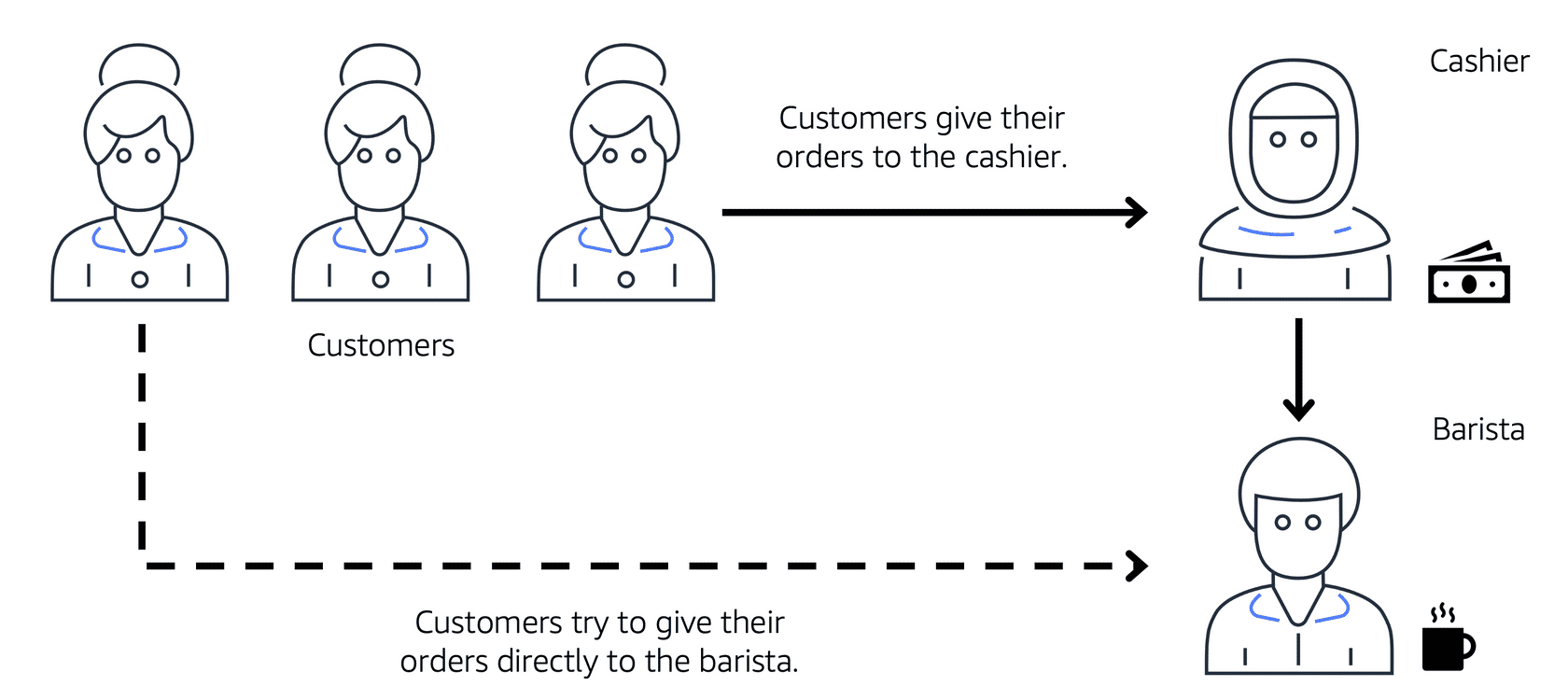
**Conclusion**:

* While the process of packet validation might seem extensive, AWS ensures this happens instantly, maintaining both security and performance.

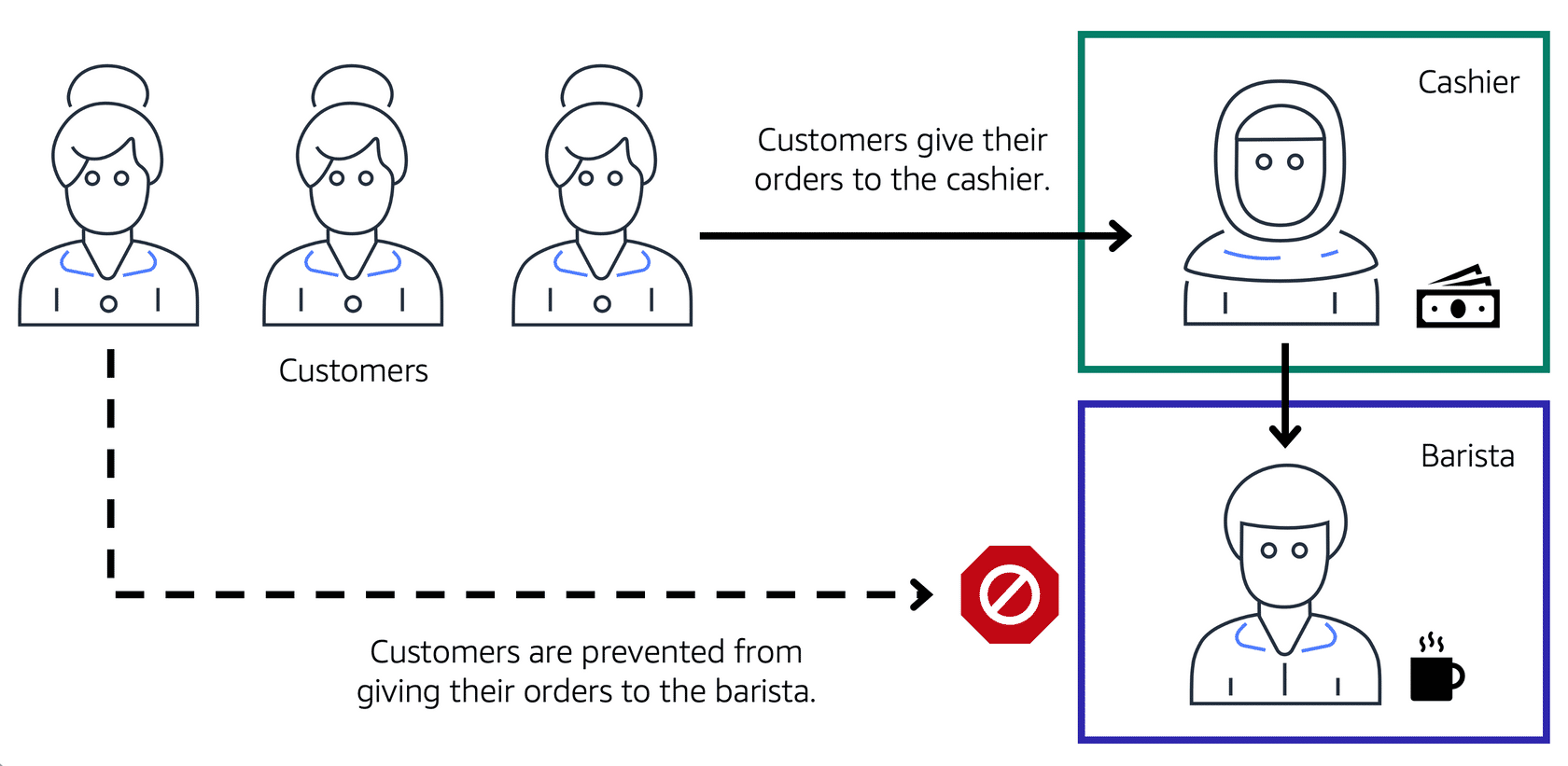
**To learn more about the role of subnets within a VPC, review the following example from the coffee shop.**

First, customers give their orders to the cashier. The cashier then passes the orders to the barista. This process allows the line to keep running smoothly as more customers come in.

Suppose that some customers try to skip the cashier line and give their orders directly to the barista. This disrupts the flow of traffic and results in customers accessing a part of the coffee shop that is restricted to them.



To fix this, the owners of the coffee shop divide the counter area by placing the cashier and the barista in separate workstations. The cashier’s workstation is public facing and designed to receive customers. The barista’s area is private. The barista can still receive orders from the cashier but not directly from customers.



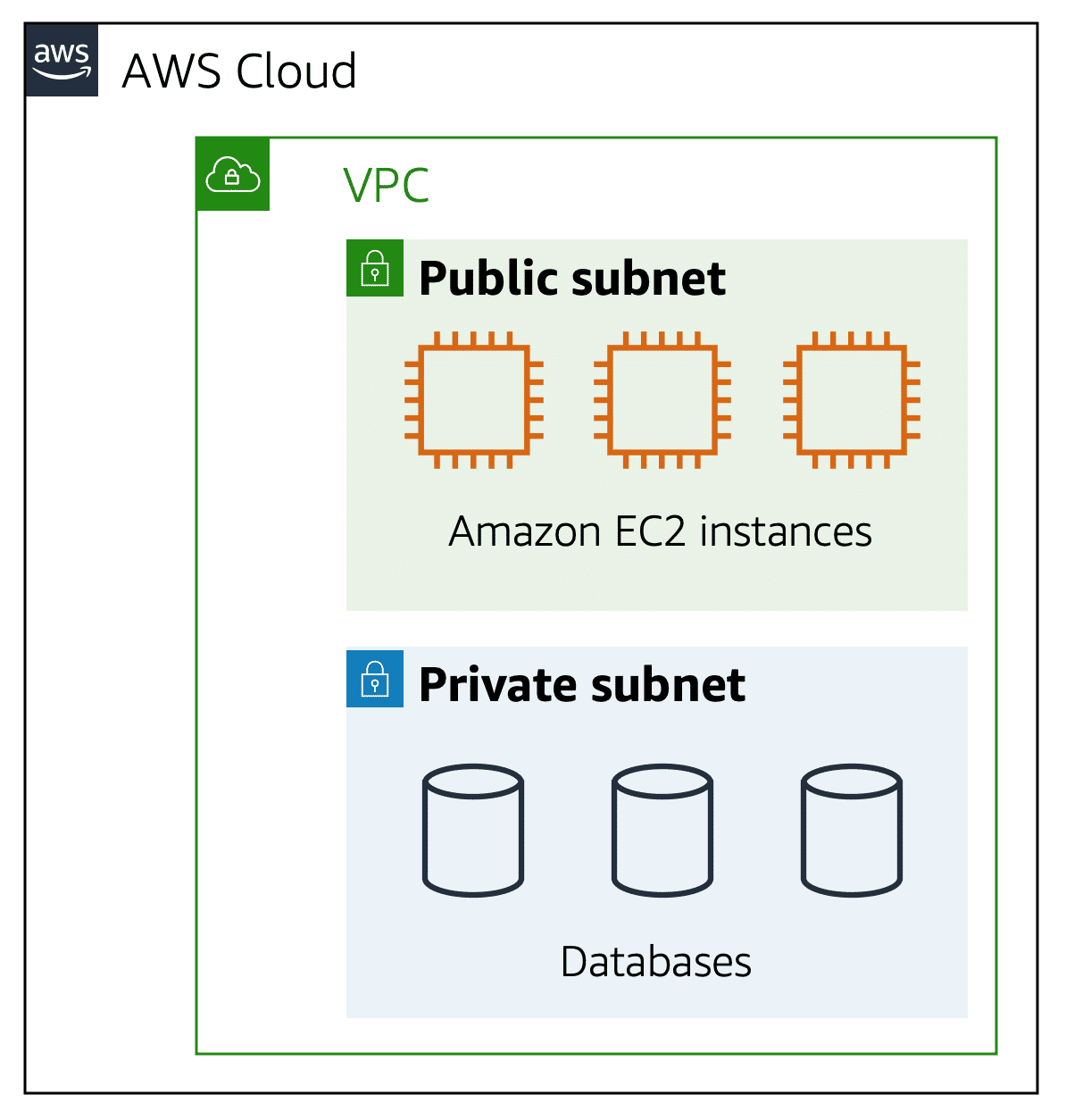
A cashier, a barista, and three customers in line. The icon for the first customer in line has an arrow pointing to cashier showing that the customer gives their order to the cashier. Then the cashier icon has an arrow pointing to barista icon showing that the cashier forwards the customer's order to the barista. The last customer in line tries to give their order directly to the barista, but they're blocked from doing so.

This is similar to how you can use AWS networking services to isolate resources and determine exactly how network traffic flows.

In the coffee shop, you can think of the counter area as a VPC. The counter area divides into two separate areas for the cashier’s workstation and the barista’s workstation. In a VPC, **subnets** are separate areas that are used to group together resources.

**Subnets**

A subnet is a section of a VPC in which **you can group resources based on security or operational needs**. Subnets can be public or private.



**Public subnets** contain resources that need to be **accessible by the public**, such as an online store’s website.

**Private subnets** contain resources that should be accessible only through your private network, such as a database that contains customers’ personal information and order histories.

In a VPC, subnets can communicate with each other. For example, you might have an application that involves Amazon EC2 instances in a public subnet communicating with databases that are located in a private subnet.

**Network traffic in a VPC**

When a customer requests data from an application hosted in the AWS Cloud, the request is sent as a packet. A packet is a piece of data transmitted via the internet or a network.

It enters into a VPC through an internet gateway. Before a packet can enter into a subnet or exit from a subnet, it validates the permissions. These permissions indicate who sent the packet and how the packet is trying to communicate with the resources in a subnet.

The VPC component that checks packet permissions for subnets is a **Network Access Control Lists (ACLs)**.

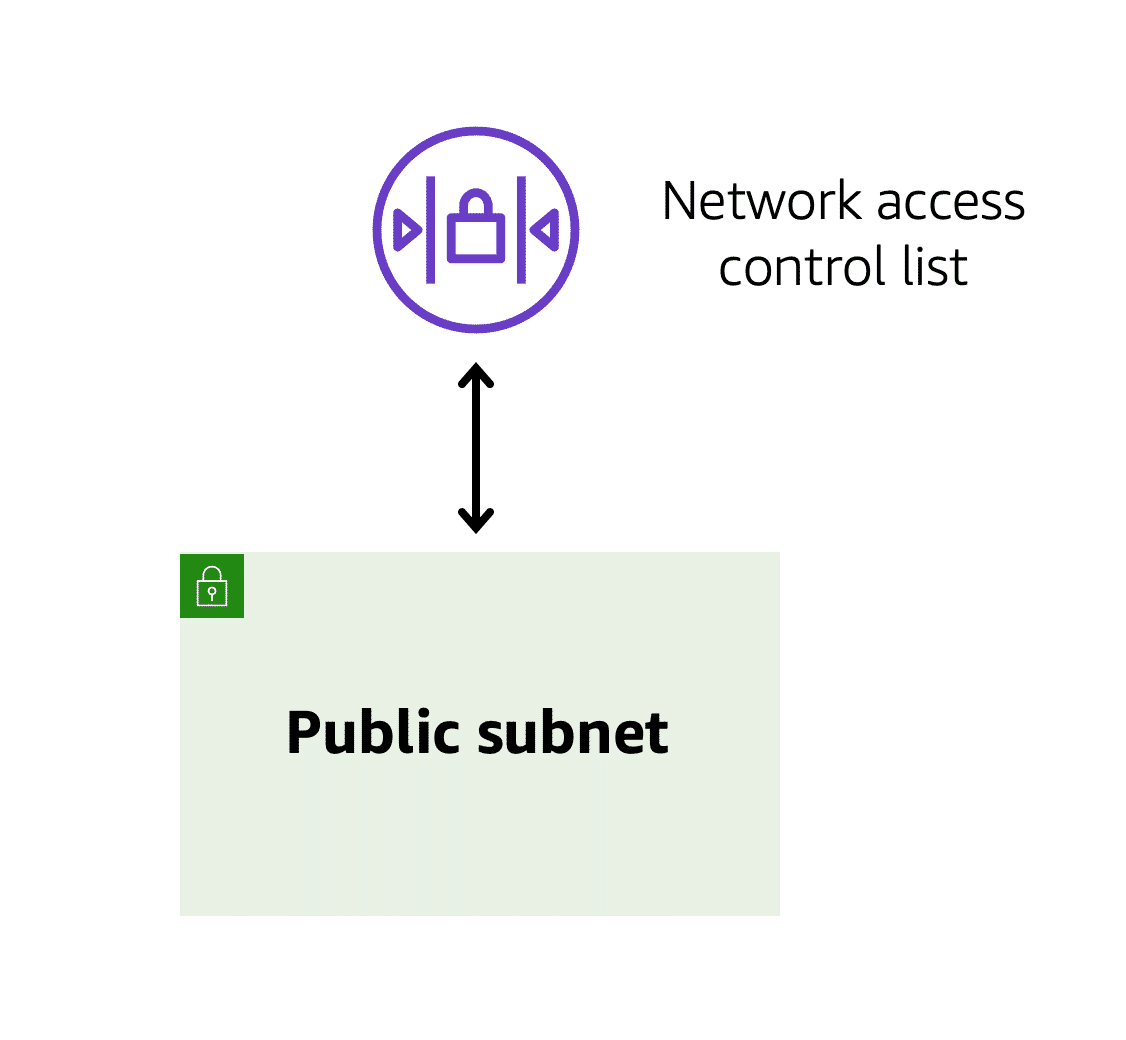
**Network ACLs**

**(Access Control Lists)**

A network ACL is a **virtual firewall** that controls inbound and outbound traffic at the subnet level.

**For example**, step outside of the coffee shop and imagine that you are in an airport. In the airport, travelers are trying to enter into a different country. You can think of the travelers as packets and the passport control officer as a network ACL.

The passport control officer checks travelers’ credentials when they are both entering and exiting out of the country. If a traveler is on an approved list, they are able to get through. However, if they are not on the approved list or are explicitly on a list of banned travelers, they cannot come in.



Each AWS account includes a default network ACL. When configuring your VPC, you can use your account’s default network ACL or create custom network ACLs.

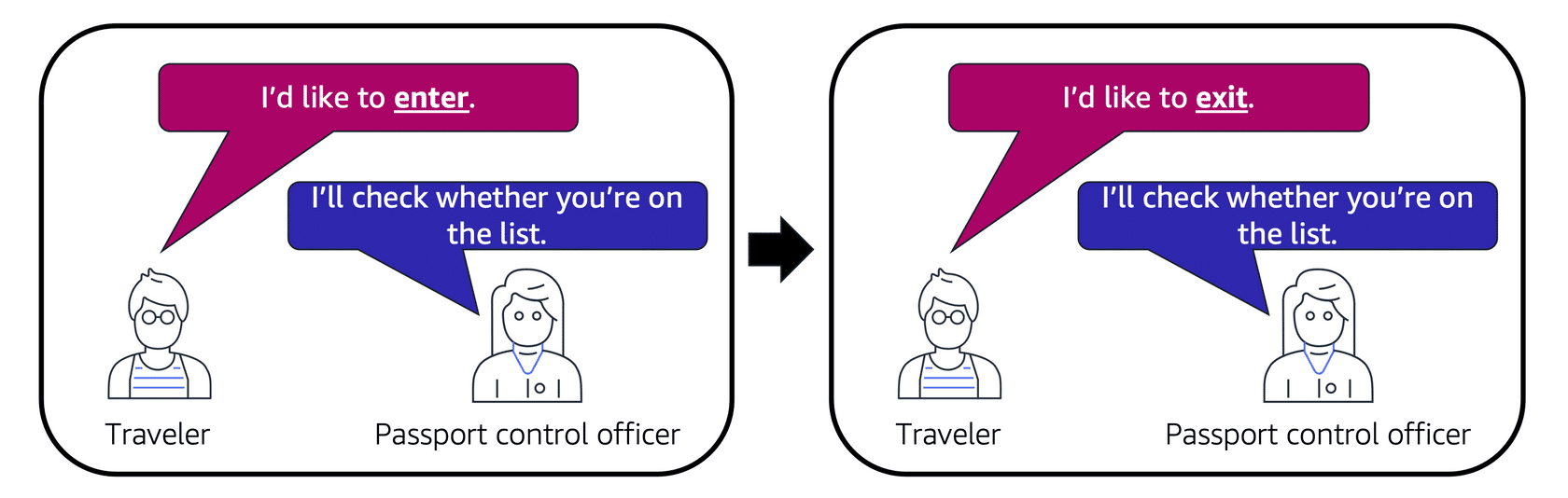
By default, your account’s default network ACL allows all inbound and outbound traffic, but you can modify it by adding your own rules. For custom network ACLs, all inbound and outbound traffic is denied until you add rules to specify which traffic to allow. Additionally, all network ACLs have an explicit deny rule. This rule ensures that if a packet doesn’t match any of the other rules on the list, the packet is denied.

**Stateless packet filtering**

Network ACLs perform **stateless** packet filtering. They remember nothing and check packets that cross the subnet border each way: inbound and outbound.

Recall the previous example of a traveler who wants to enter into a different country. This is similar to sending a request out from an Amazon EC2 instance and to the internet.

When a packet response for that request comes back to the subnet, the network ACL does not remember your previous request. The network ACL checks the packet response against its list of rules to determine whether to allow or deny.

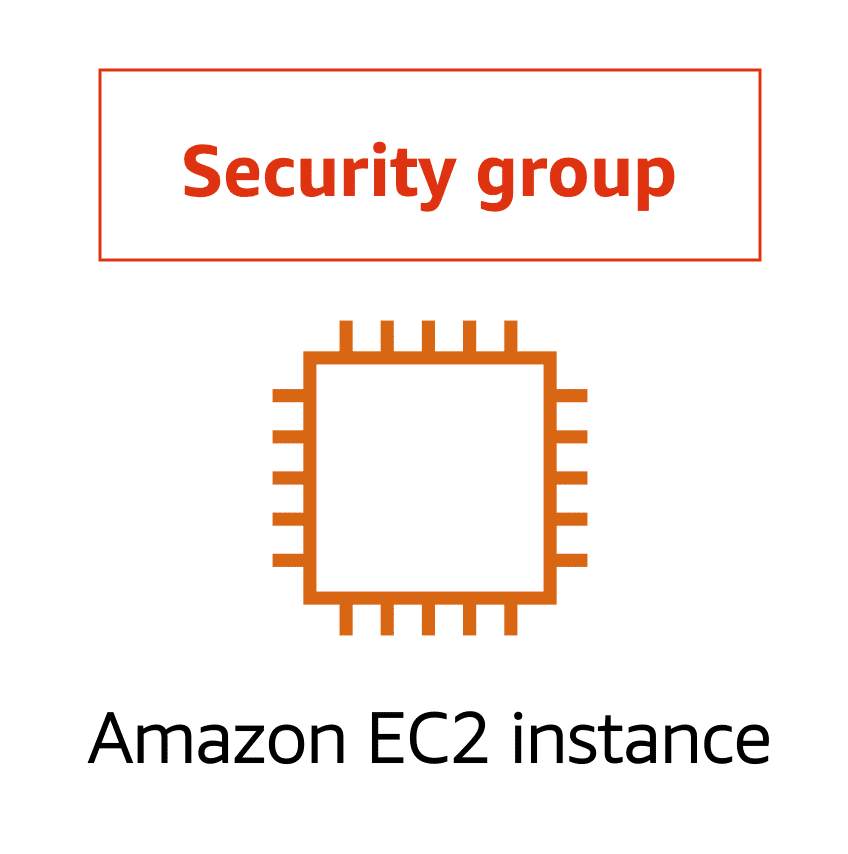


After a packet has entered a subnet, it must have its permissions evaluated for resources within the subnet, such as Amazon EC2 instances.

The VPC component that checks packet permissions for an Amazon EC2 instance is a **Security Group**.

**Security groups**

A security group is a **virtual firewall** that controls inbound and outbound traffic for an Amazon EC2 instance.



By default, a security group denies all inbound traffic and allows all outbound traffic. You can add custom rules to configure which traffic should be allowed; any other traffic would then be denied.

For this **example**, suppose that you are in an apartment building with a door attendant who greets guests in the lobby. You can think of the guests as packets and the door attendant as a security group. As guests arrive, the door attendant checks a list to ensure they can enter the building. However, the door attendant does not check the list again when guests are exiting the building.

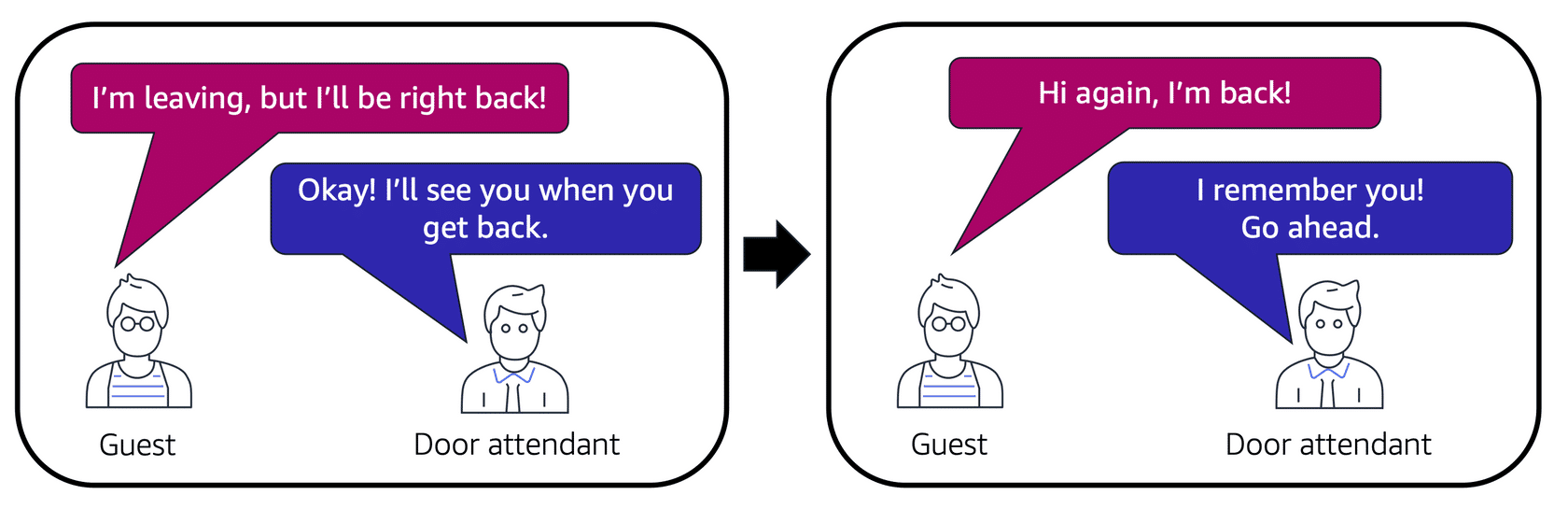
If you have multiple Amazon EC2 instances within the same VPC, you can associate them with the same security group or use different security groups for each instance.

**Stateful packet filtering**

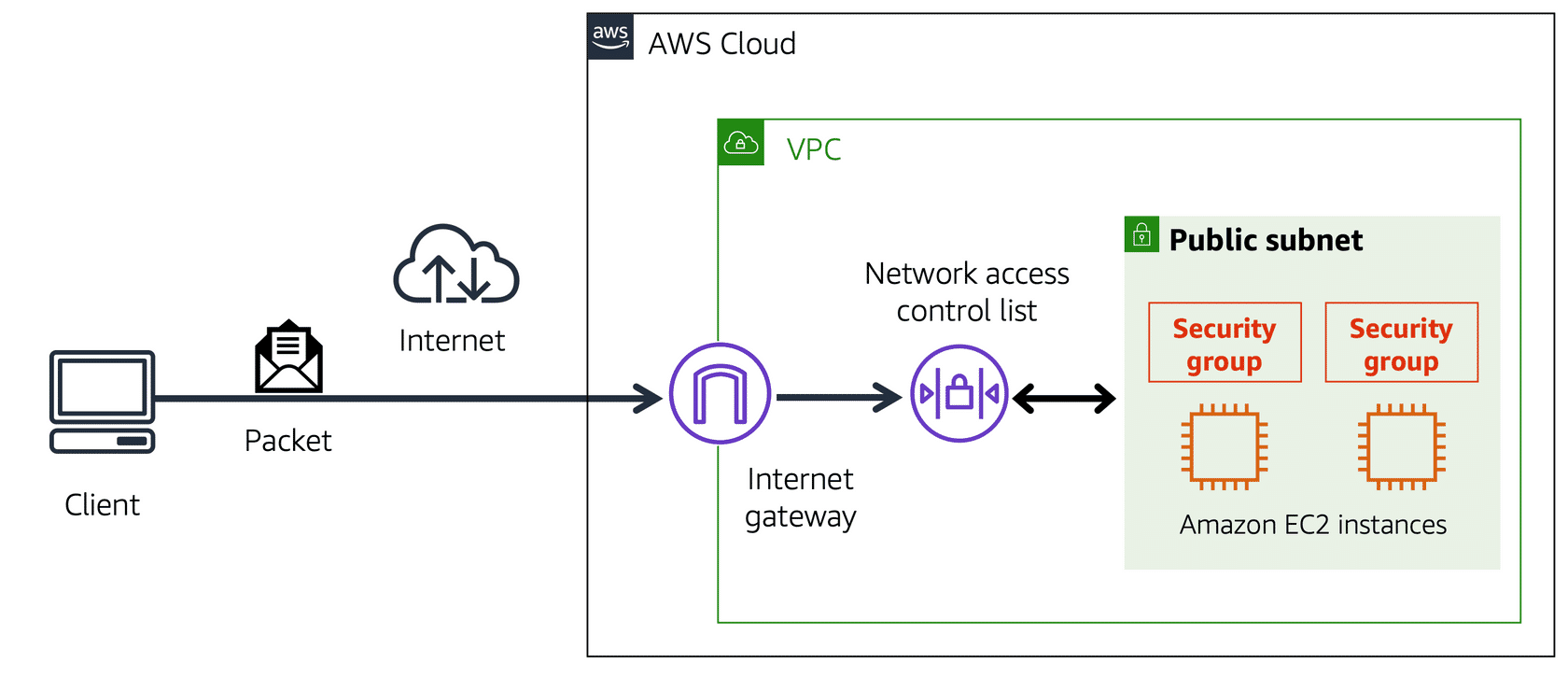
Security groups perform **stateful** packet filtering. They **remember previous decisions** made for incoming packets.

Consider the same example of sending a request out from an Amazon EC2 instance to the internet.

When a packet response to that request returns to the instance, the security group remembers your previous request. The security group allows the response to proceed, regardless of inbound security group rules.



With both network ACLs and security groups, you can configure custom rules for the traffic in your VPC. As you continue to learn more about AWS security and networking, make sure to understand the differences between network ACLs and security groups.



A packet travels over the internet from a client, to the internet gateway and into the VPC. Then the pack goes through the network access control list and accesses the public subnet, where two EC2 instances are located.

**VPC component recall**

**Private Subnet:**

Isolate databases containing customers’ personal information.

**Virtual Private Gateway:**

Create a VPN connection between the VPC and the internal corporate network.

**Public Subnet:**

Support the customer-facing website.

**AWS Direct Connect:**

Establish a dedicated connection between the on-premises data center and the VPC.

**Knowledge Check**

|  |
| --- |
| Which statement best describes an AWS account’s default network access control list?   * It is stateless and denies all inbound and outbound traffic. * It is stateful and allows all inbound and outbound traffic. * **It is stateless and allows all inbound and outbound traffic**. * It is stateful and denies all inbound and outbound traffic. |

**Module 4**

**Global Networking**

**Customer Interaction with AWS Infrastructure**:

* Customers access your AWS-hosted website by entering its domain name in their browser, triggering **DNS services** and **content delivery** mechanisms.

**Route 53 - Domain Name Service (DNS)**:

* **Function**: Translates human-readable website names (e.g., example.com) into machine-readable **IP addresses** (e.g., 192.1.1.1).
* **Capabilities**:
  + **Routing traffic** to different endpoints using policies:
    - **Latency-based routing**: Latency-based routing sends users to the location with the lowest latency.
    - **Geolocation routing**: Geolocation routing directs traffic based on user location (e.g., North American users to Oregon, Irish users to Dublin).
    - **Geoproximity and weighted routing**: Use geoproximity and weighted routing to distribute traffic based on proximity or set weights.
  + **Domain name registration**: Allows users to buy and manage domain names within AWS.
* **Analogy**: DNS is like a **translation service**, converting a website’s name into a location (IP address) that computers understand.

**Amazon CloudFront - Content Delivery Network (CDN)**:

* **Function**: Speeds up the delivery of static and dynamic content by serving it from **Edge locations** close to the user.
* **How it Works**:
  + Static assets (e.g., images, videos, CSS) are stored at multiple **Edge locations**.
  + Requests are routed to the nearest location to reduce **latency**.
  + Example:
    - Users in **Seattle** access content from the Oregon region.
    - Users in **Ireland** access content from the Dublin region.
* **Benefits**:
  + **Reduced latency**: Content is delivered faster by minimizing geographic distance.
  + **Improved user experience**: Faster page load times enhance accessibility.
* **Analogy**: CloudFront acts like a **global warehouse network**—delivering products (content) from the nearest warehouse (Edge location) to the customer.

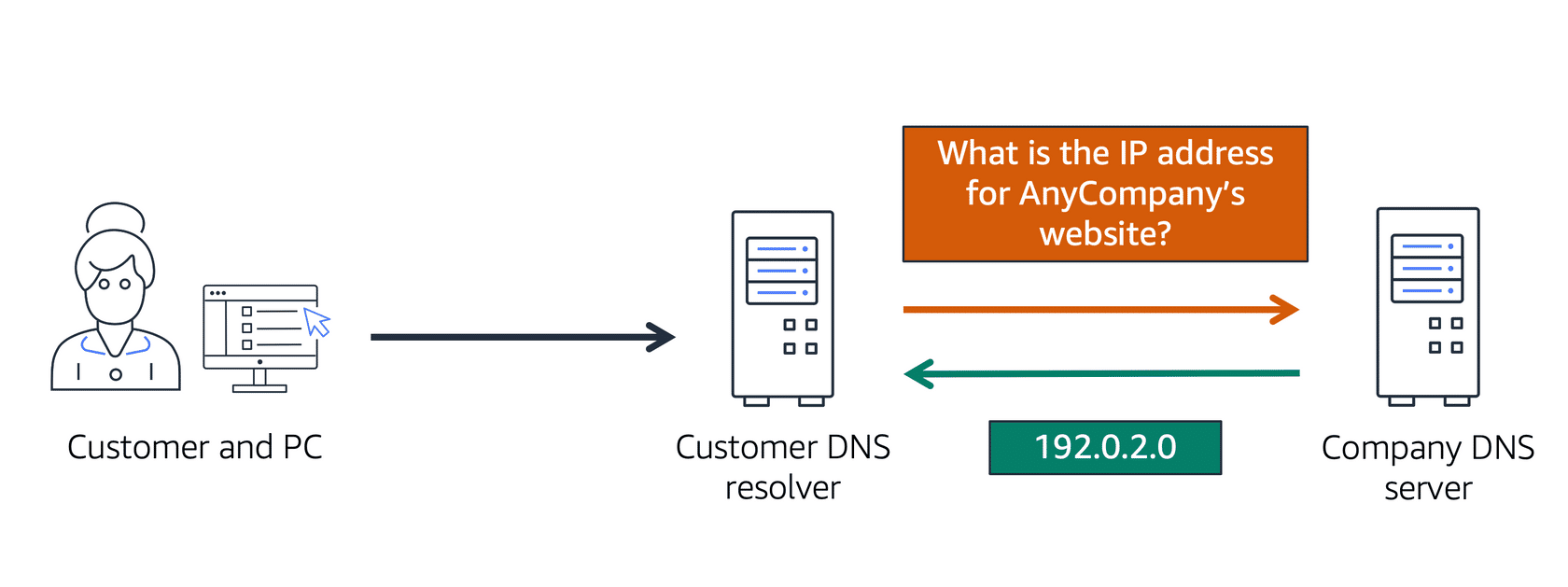
**Summary**:

* **Route 53** ensures users can find your website by resolving domain names to IP addresses.
* **CloudFront** enhances website performance by delivering content from the nearest Edge location.
* Together, they optimize the global accessibility and speed of your website.

**Domain Name System (DNS)**

Suppose that AnyCompany has a website hosted in the AWS Cloud. Customers enter the web address into their browser, and they are able to access the website. This happens because of **Domain Name System (DNS)** resolution. DNS resolution involves a customer DNS resolver communicating with a company DNS server.

You can think of DNS as being the phone book of the internet. DNS resolution is the process of translating a domain name to an IP address.



A client connects to a DNS resolver looking for a domain. The resolver forwards the request to the DNS server, which returns the IP address to the resolver.

**For example, suppose that you want to visit AnyCompany’s website.**

When you enter the domain name into your browser, this request is sent to a customer DNS resolver.

The customer DNS resolver asks the company DNS server for the IP address that corresponds to AnyCompany’s website.

The company DNS server responds by providing the IP address for AnyCompany’s website, 192.0.2.0.

**Amazon Route 53**

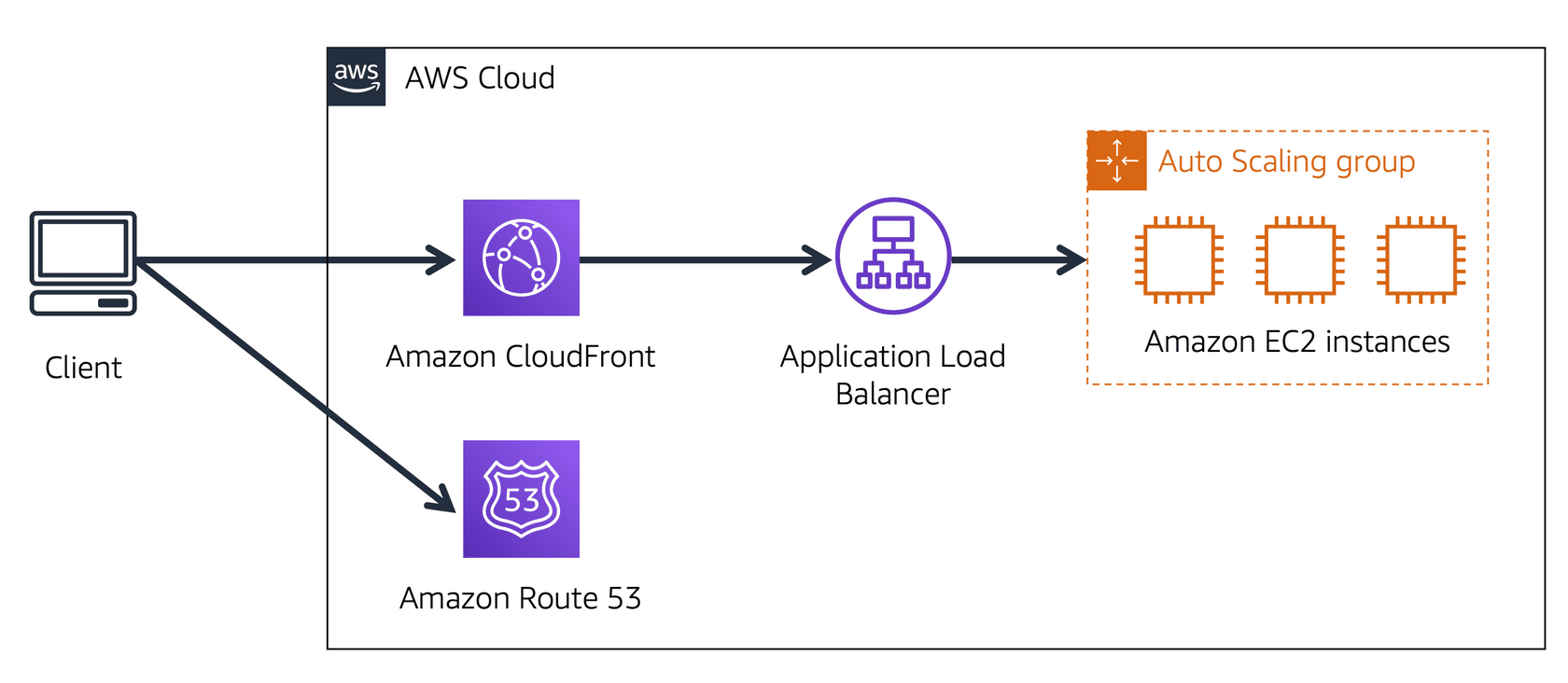
Amazon Route 53 is a **DNS web service**. It gives developers and businesses a reliable way to route end users to internet applications hosted in AWS.

Amazon Route 53 connects user requests to infrastructure running in AWS (such as Amazon EC2 instances and load balancers). It can route users to infrastructure outside of AWS.

Another feature of Route 53 is the **ability to manage the DNS records** **for domain names**. You can register new domain names directly in Route 53. You can also transfer DNS records for existing domain names managed by other domain registrars. This enables you to manage all of your domain names within a single location.

In the previous module, you learned about Amazon CloudFront, a content delivery service. The following example describes how Route 53 and Amazon CloudFront work together to deliver content to customers.

**Example: How Amazon Route 53 and Amazon CloudFront deliver content**



**Suppose that AnyCompany’s application is running on several Amazon EC2 instances. These instances are in an Auto Scaling group that attaches to an Application Load Balancer.**

A customer requests data from the application by going to AnyCompany’s website.

Amazon Route 53 uses DNS resolution to identify AnyCompany.com’s corresponding IP address, 192.0.2.0. This information is sent back to the customer.

The customer’s request is sent to the nearest edge location through Amazon CloudFront.

Amazon CloudFront connects to the Application Load Balancer, which sends the incoming packet to an Amazon EC2 instance.

**Knowledge check**

|  |
| --- |
| Which statement best describes DNS resolution?   * Launching resources in a virtual network that you define * Storing local copies of content at edge locations around the world * Connecting a VPC to the internet * **Translating a domain name to an IP address** |

**Module 4**

**Summary**

**In Module 4, you learned about the following concepts:**

* Structuring and connecting to a VPC
* Securing VPC resources with network access control lists and security groups
* Using Amazon Route 53 and Amazon CloudFront to deliver content

**Simplified Networking on AWS**:

* AWS abstracts complex networking, focusing on answering a single question: **Who should communicate with whom?**
* Once defined, setting up a network becomes straightforward.

**Key Networking Concepts**:

* **VPC (Virtual Private Cloud)**:
  + Provides isolated, private network environments for workloads.
* **Network Security**:
  + **Gateways**: Control how traffic enters/exits the VPC.
  + **Network ACLs (Access Control Lists)**: Stateless, subnet-level traffic control.
  + **Security Groups**: Stateful, instance-level traffic control to manage communication permissions.

**AWS Connectivity**:

* **VPN (Virtual Private Network)**:
  + Encrypted connections over the public internet for secure communication.
* **Direct Connect**:
  + Dedicated fiber connections for exclusive, high-performance network pipelines.

**Global Networking**:

* **Route 53**: DNS service translating domain names into IP addresses and enabling location-based routing.
* **CloudFront**: A CDN that caches content at Edge locations to minimize latency and enhance user experience.

**Final Note**:

* While this overview introduces fundamental concepts, AWS Networking offers many advanced features to explore as your business grows.

**Module 4**

**Quiz**

|  |
| --- |
| Your company has an application that uses Amazon EC2 instances to run the customer-facing website and Amazon RDS database instances to store customers’ personal information. How should the developer configure the VPC according to best practices?   * Place the Amazon EC2 instances in a private subnet and the Amazon RDS database instances in a public subnet. * **Place the Amazon EC2 instances in a public subnet and the Amazon RDS database instances in a private subnet.** * Place the Amazon EC2 instances and the Amazon RDS database instances in a public subnet. * Place the Amazon EC2 instances and the Amazon RDS database instances in a private subnet. |

|  |
| --- |
| Which component can be used to establish a private dedicated connection between your company’s data center and AWS?   * Private subnet * DNS * **AWS Direct Connect** * Virtual private gateway |

|  |
| --- |
| Which statement best describes security groups?   * **They are stateful and deny all inbound traffic by default.** * They are stateful and allow all inbound traffic by default. * They are stateless and deny all inbound traffic by default. * They are stateless and allow all inbound traffic by default. |

|  |
| --- |
| Which component is used to connect a VPC to the internet?   * Public subnet * Edge location * Security group * **Internet gateway** |

|  |
| --- |
| Which service is used to manage the DNS records for domain names?   * Amazon Virtual Private Cloud * AWS Direct Connect * Amazon CloudFront * **Amazon Route 53** |

**Module 5**

**Storage and Databases**

**Learning objectives**

In this module, you will learn how to:

* Summarize the **basic concept of storage and databases**.
* Describe the **benefits of Amazon Elastic Block Store (Amazon EBS)**.
* Describe the **benefits of Amazon Simple Storage Service (Amazon S3).**
* Describe the **benefits of Amazon Elastic File System (Amazon EFS).**
* Summarize various **storage solutions**.
* Describe the **benefits of Amazon Relational Database Service (Amazon RDS).**
* Describe the **benefits of Amazon DynamoDB**.
* Summarize various database services.

**Module 5**

**Introduction to Storage and Databases**

**Current Setup**

* The coffee business now has a robust architecture:
  + **Elastic & Scalable**: Can handle fluctuating demand.
  + **Disaster Resistant**: Reliable even in unexpected situations.
  + **Cost-Optimized**: Efficiently uses resources.
  + **Global & Secure**: Operates worldwide with high security.
  + **Programmatically Deployed**: Entirely managed via code.

**Customer Loyalty Program - The Need for Databases**

* To appreciate loyal customers, a **frequent drinker loyalty program** is proposed.
* Analogy: **Punch Cards vs. Digital Cards**
  + Old Method: Punch cards (hard to track and inefficient).
  + Better Method: Digital cards (trackable, helps in understanding customer preferences).
* Digital cards require **data storage** to keep track of:
  + Customer details.
  + Purchase history.
  + Rewards earned.

**Why Databases and Storage Are Essential**

* Databases are needed to efficiently manage and analyze customer data.
* Choosing the right type of database and storage solution is critical for:
  + Handling different **data types** (e.g., structured, unstructured).
  + Meeting **specific use cases** (e.g., performance, cost, scalability).

**AWS Solutions for Data Management**

* AWS offers various **storage and database services** to build a tailored data solution that meets diverse needs.

**Module 5**

**Instance Stores and Amazon Elastic Block Store (Amazon EBS)**

**The role of Storage in EC2 Instances**

Imagine you’ve just set up a powerful virtual workstation using Amazon EC2. It’s like running your own business machine in the cloud. This machine has everything—CPU for processing, memory for quick thinking, networking to connect with the world, and, of course, storage to save your work.

As your applications run, they need a place to store and update files. Think of this storage as your laptop’s hard drive. When you edit a file on your computer, only the parts you change get rewritten—this is **block-level storage**, and it’s perfect for handling tasks like running a database or storing critical files.

**The Whiteboard of Instance Store Volumes**

Now imagine you’re working on a whiteboard. It’s right there in your room, attached to the wall. You jot down ideas, scribble notes, and sketch plans. This is like an instance store volume—local storage physically attached to the machine hosting your EC2 instance.

But here’s the catch: what happens if you leave the room and someone wipes the board clean? All your notes are gone. That’s how **instance store volumes work—they’re temporary**. If **you stop or terminate your EC2 instance, all the data written to that storage is erased**. Why? **Because** when you **restart** the **instance**, it **might be on a different host machine where that whiteboard doesn’t exist anymore**.

Instance store volumes **are great for temporary tasks**, like storing scratch data or files you can easily recreate. But you wouldn’t want to rely on it for important work.

**Persistent Storage with Amazon EBS**

Now, let’s say you have a USB drive instead of a whiteboard. You can plug it into your laptop, save your files, and unplug it when you’re done. No matter where you take it or which computer you plug it into, your files are safe. This is how **Amazon Elastic Block Store (EBS)** works. EBS are virtual hard drives for EC2 instances.

EBS allows you to create **persistent storage** that isn’t tied to the machine hosting your EC2 instance. You can define how big or fast you want your "USB drive" to be, attach it to your virtual workstation, and start saving data. Even if you stop or restart your EC2 instance, the data on your EBS volume remains intact.

**Backup and Data Protection with Snapshots**

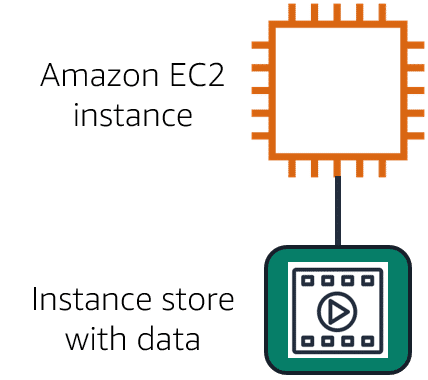
Imagine you’re playing a challenging video game. Before you face the final boss, you save your progress. That way, if something goes wrong, you can go back to that safe point and try again. This is exactly what **EBS snapshots** do.

Snapshots are **incremental backups** of your EBS volume. You take regular snapshots to protect your data so that if your storage ever becomes corrupted, you can restore it to a previous state. It’s like having a safety net for your persistent storage.

**Instance stores**

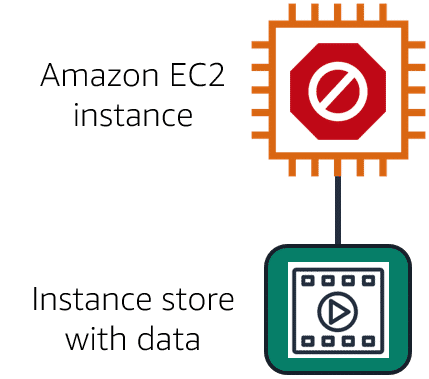
Block-level storage volumes behave like physical hard drives.

An instance stores provides **temporary block-level storage** for an Amazon EC2 instance. An instance store is **disk storage** that is physically attached to the host computer for an EC2 instance, and therefore has the same lifespan as the instance. When the instance is terminated, you lose any data in the instance store.

**An Example of How Instance Stores works:**

**Step 1**

**An Amazon EC2 instance with an attached instance store is running.**

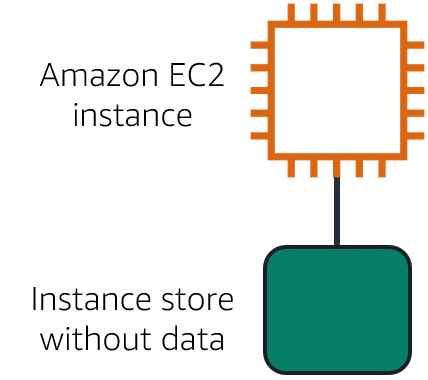


**Step 2**

**The instance is stopped or terminated**

**Step 3**

**All data on the attached instance store is deleted.**

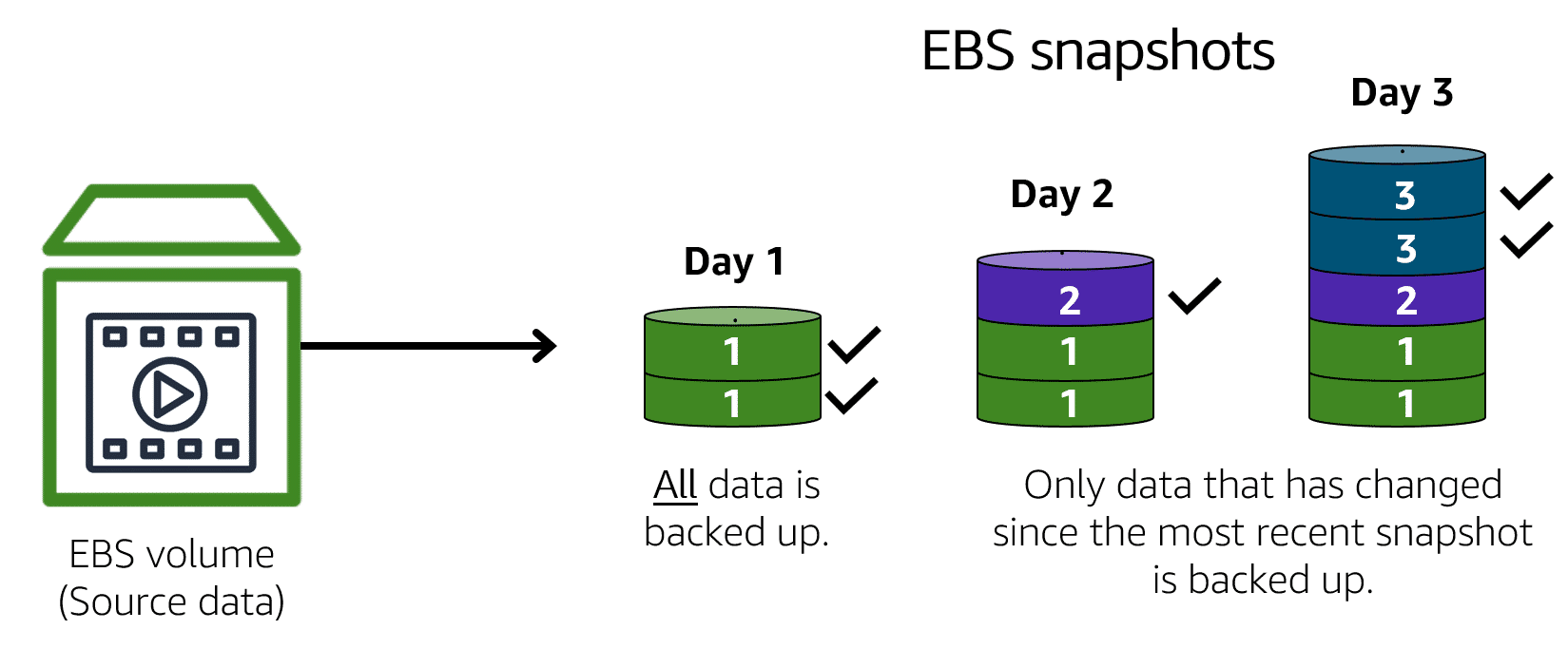


Amazon Elastic Block Store (Amazon EBS) is a service that **provides block-level storage volumes** that you can **use with Amazon EC2 instances**. If you stop or terminate an Amazon EC2 instance, all the **data on the attached EBS volume remains available**.

To create an EBS volume, you define the configuration (such as volume size and type) and provision it. After you create an EBS volume, it can attach to an Amazon EC2 instance.

Because EBS volumes are for data that needs to persist, it’s important to back up the data. You can take **incremental backups of EBS volumes** **by creating Amazon EBS snapshots**.

**Amazon EBS snapshots**



*Incremental backups of EBS volumes with Amazon EBS snapshots. On Day 1, two volumes are backed up. Day 2 adds one new volume and the new volume is backed up. Day 3 adds two more volumes for a total of five volumes. Only the two new volumes are backed up.*

An **EBS snapshot is an incremental backup**. This means that the first backup taken of a volume copies all the data. For subsequent backups, only the blocks of data that have changed since the most recent snapshot are saved.

Incremental backups are **different from full backups**, in which all the data in a storage volume copies each time a backup occurs. The full backup includes data that has not changed since the most recent backup.

**Knowledge check**

|  |
| --- |
| Which of the following are characteristics of the Amazon EBS service? (Select TWO.)   * **Best for data that requires retention** * Best for temporary data that is not kept long term * Separa**te drives from the host computer of an EC2 instance** * Physically attached to the host computer of an EC2 instance * Data is deleted when an EC2 instance is stopped |

**Module 5**

**Amazon Simple Storage Service (Amazon S3)**

**The Coffee Shop’s Storage Challenge**

Imagine the coffee shop is buzzing with business, and now it has a lot of data—receipts, employee training videos, spreadsheets, and even images of those fancy coffee designs. All this data needs a reliable place to live. Enter **Amazon S3**, a storage service that can store and retrieve virtually unlimited data at any scale.

**Buckets and Objects: The Digital Filing Cabinet**

Think of Amazon S3 like a filing cabinet:

* **Objects**: These are your individual files (like receipts or videos).
* **Buckets**: These are the folders in your cabinet where you organize and store your files.

You can **upload files as large as 5 TB**, and S3 even has a built-in **versioning system**—like keeping a paper trail of every version of a document to protect against accidental deletions. You can also create multiple buckets and control who has access to them.

**Different Storage Tiers for Different Needs**

Amazon S3 offers various storage "classes," just like choosing the right coffee roast for different tastes:

**S3 Standard**

* **Durability**: 11 nines (99.999999999%)—imagine your files surviving even if two storage facilities fail.
* **Use Case**: For frequently accessed data.
* **Fun Feature**: You can host static websites (like a coffee blog) by uploading HTML files into a bucket and enabling website hosting.

**S3 Standard-Infrequent Access (S3 Standard-IA)**

* **Perfect for data accessed less** often but still needed quickly, like backups or disaster recovery files.

**S3 Glacier Flexible Retrieval**

* Designed for long-term archives, like audit data you need to keep for years but don’t need frequently.
* **Vault Lock**: Protects data with policies like WORM (Write Once, Read Many) to meet compliance requirements.
* **Retrieval Options**: Retrieve data in minutes or hours, depending on your needs.

**Other Tiers**

* **S3 One Zone-IA**: Lower cost but stored in a single availability zone.
* **S3 Glacier Instant Retrieval**: Quick access to archived data.
* **S3 Glacier Deep Archive**: The cheapest option for data rarely accessed (like old audit logs).

**Lifecycle Policies: Automating Your Storage Journey**

Imagine you have a plan for your data, like where it should go and when. For example:

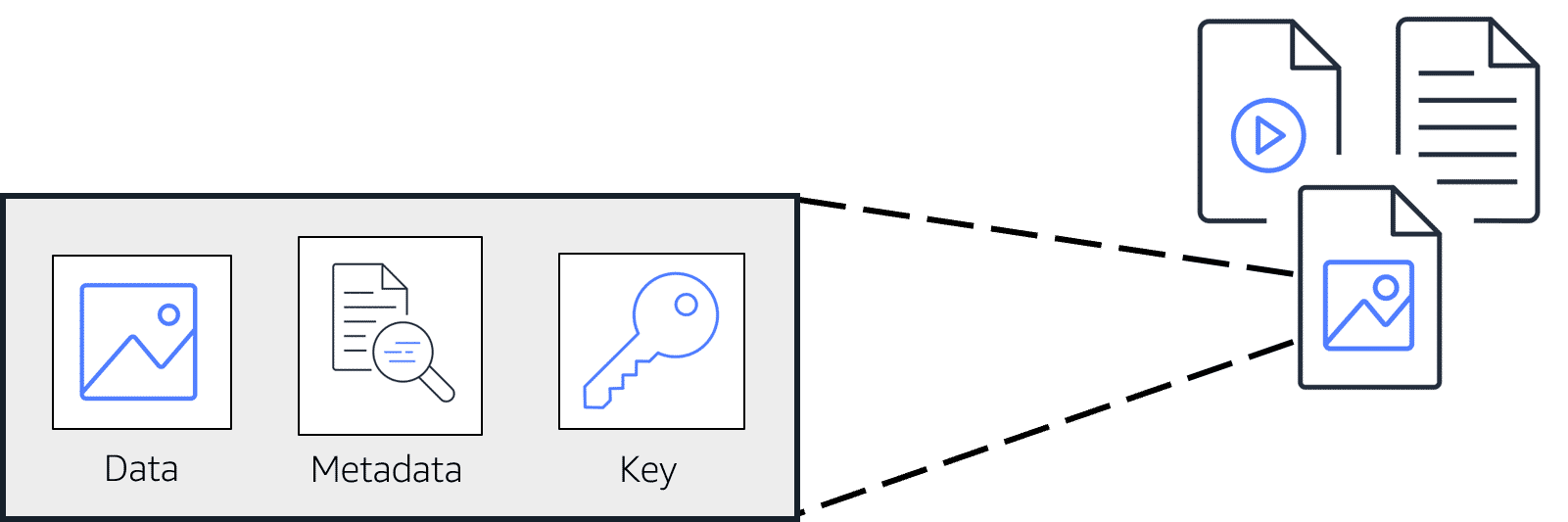
* **Day 1-90**: Keep it in S3 Standard for frequent access.
* **Day 91-120**: Move it to S3 Standard-IA for less frequent access.
* **Day 121+**: Archive it in S3 Glacier Flexible Retrieval.

With **Lifecycle Policies**, you can automate this process—no need to update your application manually. It’s like setting up a delivery schedule for your coffee beans and letting it run on autopilot.

**A Managed Service That Works for You**

Amazon S3 is like a well-trained barista—it handles the hard work of storing and managing data so you can focus on growing your business. Whether it’s frequent data access, long-term storage, or automated data movement, S3 has you covered.

**Object Storage**



In **object storage**, each object consists of data, metadata, and a key.

The **data** might be an image, video, text document, or any other type of file. **Metadata** contains information about what the data is, how it is used, the object size, and so on. An object’s **key** is its unique identifier.

Recall that when you modify a file in block storage, only the pieces that are changed are updated. **When a file in object storage is modified, the entire object is updated.**

**Amazon Simple Storage Service (Amazon S3)**

**Amazon Simple Storage Service (Amazon S3)** is a service that **provides object-level storage**. Amazon S3 **stores data as objects in buckets**.

You **can upload any type of file to Amazon S3**, such as images, videos, text files, and so on. For example, you might use Amazon S3 to store backup files, media files for a website, or archived documents. **Amazon S3 offers unlimited storage space**. The **maximum file size for an object in Amazon S3 is 5 TB**.

When you upload a file to Amazon S3, you can set permissions to control visibility and access to it. You can also use the Amazon S3 versioning feature to track changes to your objects over time.

**Amazon S3 storage classes**

With Amazon S3, you pay only for what you use. You can choose from a range of storage classes to select a fit for your business and cost needs. When selecting an Amazon S3 storage class, consider these two factors:

* How often you plan to retrieve your data
* How available you need your data to be

**Amazon S3 Storage classes**

**S3 Standard**

* Designed for frequently accessed data
* Stores data in a minimum of three Availability Zones

Amazon S3 Standard **provides high availability for objects**. This makes it a good choice for a wide range of use cases, such as websites, content distribution, and data analytics. Amazon S3 Standard **has a higher cost** than other storage classes intended for infrequently accessed data and archival storage.

**S3 Standard-Infrequent Access (S3 Standard-IA)**

* Ideal for infrequently accessed data
* Similar to Amazon S3 Standard but has a lower storage price and higher retrieval price

Amazon S3 Standard-IA is **ideal for data infrequently accessed** but requires high availability when needed. Both Amazon S3 Standard and Amazon S3 Standard-IA **store data in a minimum of three Availability Zones**. Amazon S3 Standard-IA provides the same level of availability as Amazon S3 Standard but with a **lower storage price** and a **higher retrieval price.**

**S3 One Zone-Infrequent Access (S3 One Zone-IA)**

* Stores data in a **single Availability Zone**
* Has a **lower storage price** than Amazon S3 Standard-IA

Compared to S3 Standard and S3 Standard-IA, which store data in a minimum of three Availability Zones, S3 One Zone-IA **stores data in a single Availability Zone**. This makes it a good storage class to consider if the following conditions apply:

* You want to save costs on storage.
* You can easily reproduce your data in the event of an Availability Zone failure.

**S3 Intelligent Tiering**

* Ideal for data with unknown or changing access patterns
* Requires a small monthly monitoring and automation fee per object

In the S3 Intelligent-Tiering storage class, Amazon S3 monitors objects’ access patterns. If you haven’t accessed an object for 30 consecutive days, Amazon S3 automatically moves it to the infrequent access tier, S3 Standard-IA. If you access an object in the infrequent access tier, Amazon S3 automatically moves it to the frequent access tier, S3 Standard.

**S3 Glacier Instant Retrieval**

* Works well for archived data that requires immediate access
* Can retrieve objects within a few milliseconds

When you decide between the options for archival storage, consider how quickly you must retrieve the archived objects. You can retrieve objects stored in the S3 Glacier Instant Retrieval storage class within milliseconds, with the same performance as S3 Standard.

**S3 Glacier Flexible Retrieval**

* Low-cost storage designed for data archiving
* Able to retrieve objects within a few minutes to hours

S3 Glacier Flexible Retrieval is a **low-cost storage class** that is **ideal for data archiving**. For example, you might use this storage class to store archived customer records or older photos and video files. You can retrieve your data from S3 Glacier Flexible Retrieval from 1 minute to 12 hours.

**S3 Glacier Deep Archive**

* Lowest-cost object storage class ideal for archiving
* Able to retrieve objects within 12 hours

S3 Deep Archive supports long-term retention and digital preservation for data that might be accessed once or twice in a year. This storage class is the lowest-cost storage in the AWS Cloud, with data retrieval from 12 to 48 hours. All objects from this storage class are replicated and stored across at least three geographically dispersed Availability Zones.

**S3 Outposts**

* Creates S3 buckets on Amazon S3 Outposts
* Makes it easier to retrieve, store, and access data on AWS Outposts

Amazon S3 Outposts delivers object storage to your on-premises AWS Outposts environment. Amazon S3 Outposts is designed to store data durably and redundantly across multiple devices and servers on your Outposts. It works well for workloads with local data residency requirements that must satisfy demanding performance needs by keeping data close to on-premises applications.

**Knowledge check**

|  |
| --- |
| You want to store data that is infrequently accessed but must be immediately available when needed. Which Amazon S3 storage class should you use?   * S3 Intelligent-Tiering * S3 Glacier Deep Archive * **S3 Standard-IA** * S3 Glacier Flexible Retrieval |

**Comparing Amazon EBS and Amazon S3**

**Module 5: Comparing Amazon EBS and Amazon S3**

Welcome to the **Clash of the Storage Titans!** In this corner, we have the **block storage heavyweight**, **Amazon Elastic Block Store (EBS)**, with its solid-state drives and block-level efficiency. And in the other corner, the **object storage superstar**, **Amazon Simple Storage Service (S3)**, with its limitless capacity and regional durability. Let’s see which champion suits your needs!

**Round 1: Storing Millions of Photos – The S3 Victory**

Imagine you’re running a fun photo analysis website. Users upload selfies, and your app finds animals that resemble them. Behind the scenes, you’re storing **millions of animal images** that need to be accessed quickly and possibly viewed by thousands at the same time.

Here’s why **Amazon S3** wins this round:

* Each image gets its own **URL** for easy access, and you can control who sees or manages the files.
* With **11 nines of durability**, your images are practically indestructible—S3 is your backup strategy!
* No need for EC2 instances—it’s **serverless**, saving you money and reducing complexity.

For handling this massive photo library, S3’s web-enabled, cost-effective design takes the win!

**Round 2: Editing an 80-GB Video – The EBS Knockout**

Now imagine you’re editing a giant 80-gigabyte video file. You need to tweak a few scenes here and there. Let’s compare the two storage options:

1. **Amazon S3 (Object Storage)**
   * Treats the video as one large **object**.
   * Any edit means re-uploading the entire 80 GB file.
2. **Amazon EBS (Block Storage)**
   * Breaks the video into **blocks**.
   * When you edit one scene, only the affected blocks get updated.

Clearly, for frequent edits, **Amazon EBS** wins. It’s efficient, saving time and bandwidth, making it perfect for applications with complex **read/write/change** requirements.

**The Final Verdict**

* If your workload involves **complete objects** (like documents, images, or rarely updated files), **Amazon S3** is your champion.
* If your workload requires **frequent changes or edits**, **Amazon EBS** takes the belt.

In this storage slugfest, the **real winner** depends on your unique use case. Once you understand what your workload demands, you’ll know which storage hero to choose!

**Module 5**

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

Imagine you’re running a business that depends on analysing massive amounts of data. Multiple servers work together, crunching numbers, running analytics, and processing shared data. Traditionally, this shared data would be stored on a **file system in an on-premises data center**.

But with that comes a lot of responsibilities:

* Ensuring the storage can handle all the data.
* Taking regular backups.
* Setting up redundant copies to avoid data loss.
* Managing all the servers to keep the system running.

It’s like running a bakery with a single oven—you have to constantly manage how much dough goes in, how long it bakes, and ensure nothing burns or runs out.

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

Now imagine switching to an **automated, scalable bakery** where ovens adjust automatically based on how much bread you need to bake. That’s what **Amazon EFS** does for your data.

With **EFS**, you:

1. **Skip the hardware worries**: AWS handles all the scaling, replication, and backups.
2. **Share data effortlessly**: Multiple servers (EC2 instances) can read and write to the file system simultaneously, just like a team of bakers working on the same orders.
3. **Scale automatically**: EFS grows and shrinks with your data needs. No need to "provision extra ovens" (storage volumes)—AWS adjusts for you.

**The EBS vs. EFS**

But wait—what’s the difference between EFS and **Amazon EBS**, another storage service? Let’s compare:

* **Amazon EBS** is like a personal hard drive for an EC2 instance. It’s tied to one instance (or server) in the same **Availability Zone (AZ)**. You manually manage its size, and if it fills up, you have to provision more space.
* **Amazon EFS**, on the other hand, is a **regional resource**. Any EC2 instance in the same Region can access the file system. Plus, EFS acts as a true Linux-based **file system** and automatically scales with your data needs.

Think of it this way:

* **EBS** is like a single baking tray for one oven—it’s great for personal-sized projects.
* **EFS** is like a shared dough counter for an entire bakery, where everyone can access and contribute at the same time.

**Why EFS is Awesome**

With Amazon EFS, you can focus on baking (running your business), and AWS takes care of all the heavy lifting, ensuring your "kitchen" (data system) is always ready to handle any workload. Whether it’s scaling, sharing, or managing your storage, EFS has your back!

**File storage**

In **file storage**, multiple clients (such as users, applications, servers, and so on) can access data that is stored in shared file folders. In this approach, a storage server uses block storage with a local file system to organize files. Clients access data through file paths.

Compared to block storage and object storage, file storage is ideal for use cases in which a large number of services and resources need to access the same data at the same time.

**Amazon Elastic File System (Amazon EFS)** is a scalable file system used with AWS Cloud services and on-premises resources. As you add and remove files, Amazon EFS grows and shrinks automatically. It can scale on demand to petabytes without disrupting applications.

**Comparing Amazon EBS and Amazon EFS**

|  |  |
| --- | --- |
| **Amazon EBS**  An Amazon EBS volume stores data in a **single** Availability Zone.  To attach an Amazon EC2 instance to an EBS volume, both the Amazon EC2 instance and the EBS volume must reside within the same Availability Zone. | **Amazon EFS**  Amazon EFS is a regional service. It stores data in and across **multiple** Availability Zones.  The duplicate storage enables you to access data concurrently from all the Availability Zones in the Region where a file system is located. Additionally, on-premises servers can access Amazon EFS using AWS Direct Connect. |

**Module 5**

**Amazon Relational Database Service (Amazon RDS)**

Imagine you run a bustling coffee shop, and you want to track everything: customer orders, favorite drinks, and loyalty points. Now, let's say you want to offer a discount to a loyal customer who always orders the same latte. To make this happen, you need a system that not only stores data but also links it in meaningful ways.

**Relationships in Data: The Coffee Shop Connection**

In your shop, each customer has their own "profile" stored in a **customer table.** Their addresses? Those are stored in a separate **address table.** But here's the magic: both tables are connected by a common piece of information, like the customer’s ID. This connection allows you to easily find customer details and their addresses when you need them.

To make sense of all this data, you’d write commands in **SQL (Structured Query Language)** to retrieve exactly what you need, such as, “Show me all customers who’ve ordered a latte more than five times.”

**Moving Databases to the Cloud**

Traditionally, you might store this database in your shop’s back office, on physical servers. But maintaining those servers—handling software updates, backups, and hardware issues—can become a nightmare.

Thankfully, AWS offers two ways to move these databases to the cloud:

**Lift-and-Shift**: Think of this like picking up your coffee shop’s counter and moving it to a new location, without changing how you run things. In this case, you migrate your database to an **Amazon EC2 instance**, where you manage everything—just like before.

**Amazon RDS (Relational Database Service)**: This is like hiring an expert barista to handle the technical side of things. With **RDS**, AWS takes care of the tough stuff: automatic updates, backups, redundancy, and disaster recovery. You still get to serve coffee (focus on your business), but without worrying about maintaining the machines.

**Introducing Amazon Aurora: The Cloud’s Superstar**

For those who want the best of the best, AWS offers **Amazon Aurora**—its most powerful and cost-effective database option. Think of Aurora as the high-tech, fully automated coffee machine of the cloud.

Aurora offers:

1. **Unbeatable reliability**: Your data is automatically copied six times across different locations, ensuring nothing gets lost.
2. **Scalable performance**: You can add up to 15 read replicas (imagine 15 additional baristas taking orders) to handle heavy traffic.
3. **Continuous backups**: It automatically backs up your data to Amazon S3, ensuring you always have a way to recover lost information.
4. **Point-in-time recovery**: Accidentally deleted yesterday’s sales data? No worries—Aurora lets you restore data to a specific moment in time.

And the cherry on top? Aurora costs **1/10th the price** of traditional commercial-grade databases.

**Why RDS and Aurora Matter**

In short, **Amazon RDS** and **Amazon Aurora** make managing your coffee shop’s “data menu” simple and efficient. Whether you’re tracking loyalty rewards or scaling up for a new branch, AWS helps you focus on brewing the perfect cup—while they handle the data behind the scenes.

Now, you’re ready to conquer the coffee shop’s digital world with relational databases! ☕

**Relational databases**

In a **relational database**, data is stored in a way that relates it to other pieces of data.

An example of a relational database might be the coffee shop’s inventory management system. Each record in the database would include data for a single item, such as product name, size, price, and so on.

Relational databases use **structured query language (SQL)** to store and query data. This approach allows data to be stored in an easily understandable, consistent, and scalable way. For example, the coffee shop owners can write a SQL query to identify all the customers whose most frequently purchased drink is a medium latte.

**Example of data in a relational database:**

| **ID** | **Product name** | **Size** | **Price** |
| --- | --- | --- | --- |
| 1 | Medium roast ground coffee | 12 oz. | $5.30 |
| 2 | Dark roast ground coffee | 20 oz. | $9.27 |

**Amazon Relational Database Service (Amazon RDS)**

**Amazon Relational Database Service (Amazon RDS)** is a service that enables you to run relational databases in the AWS Cloud.

Amazon RDS is a managed service that automates tasks such as hardware provisioning, database setup, patching, and backups. With these capabilities, you can spend less time completing administrative tasks and more time using data to innovate your applications. You can integrate Amazon RDS with other services to fulfill your business and operational needs, such as using AWS Lambda to query your database from a serverless application.

Amazon RDS provides a number of different security options. Many Amazon RDS database engines offer encryption at rest (protecting data while it is stored) and encryption in transit (protecting data while it is being sent and received).

**Amazon RDS database engines**

Amazon RDS is available on six database engines, which optimize for memory, performance, or input/output (I/O).

Supported database engines include:

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

**Amazon Aurora**

**Amazon Aurora** is an enterprise-class relational database. It is compatible with MySQL and PostgreSQL relational databases. It is up to five times faster than standard MySQL databases and up to three times faster than standard PostgreSQL databases.

Amazon Aurora helps to reduce your database costs by reducing unnecessary input/output (I/O) operations, while ensuring that your database resources remain reliable and available.

Consider Amazon Aurora if your workloads require high availability. It replicates six copies of your data across three Availability Zones and continuously backs up your data to Amazon S3.

**Module 5**

**Amazon Dynamo DB**

Imagine you’re building a mobile app for a popular online store. Millions of users are browsing, adding items to their carts, and placing orders at the same time. You need a database that’s fast, flexible, and capable of scaling without breaking a sweat. Enter **Amazon DynamoDB**, a NoSQL, serverless database that’s built for speed and scalability.

**How DynamoDB Works**

**Tables, Items, and Attributes**:

* In DynamoDB, data is organized into **tables**.
* Each **table** contains **items** (like rows in a spreadsheet).
* Each **item** has **attributes** (like columns in a spreadsheet).

**For example:**

* **Table**: Orders
* **Item**: Order #123
* **Attributes**: Customer Name, Product, Quantity

**Flexibility**:

* Unlike traditional databases, not every item in DynamoDB needs the same attributes. For instance, one order might have a "Promo Code" attribute, while another doesn’t.

**Scaling and Availability**:

* DynamoDB automatically handles scaling—whether you have 1 item or 2 million items.
* It stores data across multiple **Availability Zones** and mirrors it across multiple drives, ensuring high availability and durability.

**Why DynamoDB?**

**Lightning Fast**: DynamoDB delivers millisecond response times, even for applications with millions of users.

**Serverless**: AWS takes care of the infrastructure, so you don’t need to worry about managing servers.

**Massive Scalability**: On **Prime Day 2019**, DynamoDB handled **7.11 trillion API calls**, peaking at **45.4 million requests per second**—without breaking a sweat!

**NoSQL vs. SQL**

DynamoDB is a **NoSQL database**, which makes it different from traditional **SQL (relational) databases**:

1. **Relational Databases** (e.g., MySQL):
   * Use **SQL** for complex queries.
   * Have rigid schemas (fixed structure).
   * Tables are related to each other.
   * Great for structured data but can struggle with high traffic or unstructured data.
2. **Non-Relational Databases** (e.g., DynamoDB):
   * Flexible schema—items can have different attributes.
   * Optimized for simple, high-speed queries.
   * Focus on a single table rather than relationships across tables.

**Example**: In DynamoDB, you query based on specific **keys** (attributes like "Order ID" or "Customer Name") instead of running complex, multi-table queries.

**When to Use DynamoDB**

DynamoDB shines in scenarios like:

* Apps with millions of users (e.g., gaming, e-commerce).
* Workloads with unstructured or semi-structured data.
* Use cases requiring lightning-fast response times.

However, it’s not ideal for workloads that require complex queries or strict relationships between datasets.

**Key Features of DynamoDB**

**NoSQL Database**: Flexible, non-relational data storage.

**Serverless**: AWS handles scaling, backups, and infrastructure for you.

**Millisecond Performance**: Fast response times for even the largest workloads.

**Highly Scalable**: Automatically adjusts to handle traffic spikes.

**Nonrelational databases**

In a **nonrelational database**, you create tables. A table is a place where you can store and query data.

Nonrelational databases are sometimes referred to as “NoSQL databases” because they use structures other than rows and columns to organize data. One type of structural approach for nonrelational databases is key-value pairs. With key-value pairs, data is organized into items (keys), and items have attributes (values). You can think of attributes as being different features of your data.

In a key-value database, you can add or remove attributes from items in the table at any time. Additionally, not every item in the table has to have the same attributes.

**Example of data in a nonrelational database:**

| **Key** | **Value** |
| --- | --- |
| 1 | **Name**: John Doe  **Address**: 123 Any Street  **Favorite drink**: Medium latte |
| 2 | **Name**: Mary Major  **Address**: 100 Main Street  **Birthday**: July 5, 1994 |

**Amazon DynamoDB**

**Amazon DynamoDB** is a key-value database service. It delivers single-digit millisecond performance at any scale.

**Features of DynamoDB**

|  |  |
| --- | --- |
| **Serverless**  DynamoDB is serverless, which means that you do not have to provision, patch, or manage servers.  You also do not have to install, maintain, or operate software. | **Automatic Scaling**  As the size of your database shrinks or grows, DynamoDB automatically scales to adjust for changes in capacity while maintaining consistent performance.  This makes it a suitable choice for use cases that require high performance while scaling. |

**Module 5**

**Comparing Amazon RDS and Amazon DynamoDB**

Let’s step into the ring for a showdown between two powerful database champions:

* **Amazon RDS** (Relational Database Service): The heavyweight champion for structured, relational data with complex relationships.
* **Amazon DynamoDB**: The lightning-fast NoSQL database built for speed, scalability, and flexibility.

**Round 1: Complex Relationships and Analytics**

**Use Case**: Analyzing a sales supply chain for weak spots.

**Winner**: **Amazon RDS**

* RDS is built for relational data, making it perfect for complex queries and joins across multiple tables.
* Relational databases (like MySQL, PostgreSQL, or Oracle) allow you to create schemas and run powerful SQL queries for deep data analysis.

**Round 2: Simpler, High-Performance Needs**

**Use Case**: A simple employee contact list with names, phone numbers, and emails.

**Winner**: **Amazon DynamoDB**

* DynamoDB is purpose-built for single-table use cases without the overhead of relational databases.
* It delivers **ultra-fast performance** and eliminates unnecessary complexity.
* Perfect for scalable, high-performance apps that don’t require complex relationships between data.

**Key Differences**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Amazon RDS** | **Amazon DynamoDB** |
| **Type** | Relational Database (SQL) | Non-Relational Database (NoSQL) |
| **Best For** | Complex relationships, joins, and business analytics. | Simple, high-speed queries without relationships. |
| **Schema** | Fixed schema with strict structure. | Flexible schema; items can vary. |
| **Performance** | Ideal for structured, relational workloads. | Lightning-fast for massive traffic. |
| **Scalability** | Scales vertically (upgrading resources). | Scales horizontally (adds capacity automatically). |
| **Use Cases** | Business analytics, ERP, CRM. | Gaming, e-commerce, real-time data. |

**Summary**

* **Choose Amazon RDS** for workloads that rely on complex relationships, structured data, and SQL-based analytics.
* **Choose DynamoDB** for flexible, high-speed workloads with simpler queries and no need for complex joins.

Each database is a champion in its own category. Your use case determines the ultimate winner! 🏆

**Knowledge check**

|  |
| --- |
| What are the scenarios in which you should use Amazon Relational Database Service (Amazon RDS)? (Select TWO.)   * Running a serverless database * **Using SQL to organize data** * Storing data in a key-value database * Scaling up to 10 trillion requests per day * **Storing data in an Amazon Aurora database** |

**Module 5**

**Amazon RedShift**

Amazon Redshift is AWS's **data warehousing service**, designed for **big data analytics**. It's the go-to solution when you need to analyze historical data and gain insights for business decisions.

**Why Use a Data Warehouse?**

**Operational Analysis vs. Historical Analysis**

* Operational: Tracks live data (e.g., "How many bags of coffee are in stock right now?").
* Historical: Analyzes past data (e.g., "How have sales improved over the past year?").

**Challenges with Traditional Databases**

* Struggle to handle massive amounts of historical data.
* Can't easily integrate data from multiple sources (e.g., inventory, financials, and sales).

**What is Amazon Redshift?**

Amazon Redshift is **data warehousing as a service**, purpose-built for big data analytics.

**Key Features:**

1. **Massive Scalability**
   * Handles petabytes of structured data and integrates with Redshift Spectrum to query **exabytes** of unstructured data in **data lakes**.
2. **Performance**
   * Up to **10x faster** than traditional databases for business intelligence (BI) queries.
3. **Simplified Management**
   * Fully managed: Reduces the time spent on tasks like tuning, scaling, and maintaining the system.
4. **SQL Support**
   * Allows you to run SQL queries across both structured and unstructured data.
5. **Fast Deployment**
   * Start with a single API call and focus on insights instead of setup and maintenance.

**When to Use Amazon Redshift?**

* **For Business Intelligence (BI):**
  + Answering questions like "What were last hour’s sales across all stores?"
  + Consolidating data from multiple systems like inventory, financials, and retail sales for analytics.
* **For Big Data:**
  + If traditional databases struggle with your data's **volume** (size), **velocity** (speed), or **variety** (different types).

**How is Amazon Redshift Different?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Traditional Relational Databases** | **Amazon Redshift** |
| **Focus** | Real-time operational data. | Historical data and big data analytics. |
| **Scalability** | Limited to smaller data sets. | Handles petabytes to exabytes. |
| **Performance** | Slower with large datasets. | Optimized for faster analytics. |
| **Management** | Requires manual tuning and scaling. | Fully managed by AWS. |

**Why Amazon Redshift?**

* Provides quick and actionable insights from historical data.
* Handles **big data workloads** effortlessly, letting you focus on analyzing data, not managing the system.
* Perfect for organizations needing **fast, scalable, and powerful** business intelligence solutions.

**Amazon Redshift**

**Amazon Redshift** is a data warehousing service that you can use for big data analytics. It offers the ability to collect data from many sources and helps you to understand relationships and trends across your data.

**Module 5**

**AWS Data Migration Service**

Imagine you’re the captain of a ship called "Data Explorer," and your mission is to help your data migrate safely to a new home on AWS. With storms of downtime and compatibility issues looming, you need a trusty guide. Enter **AWS Database Migration Service (DMS)**—the magical navigator that ensures a smooth journey for your data, no matter how complex the voyage.

**The Journey Begins: Smooth Sailing with DMS**

Your database, "Old Data Harbor," is currently anchored either on-premises or in the cloud. You’ve decided it’s time to move to AWS for a brighter future. The big question: **Do you need to start from scratch?** Thankfully, no! DMS takes the wheel and starts the migration without disrupting your current operations. Your source database keeps running while DMS charts a secure and seamless route to AWS.

**The Two Migration Routes**

**The Homogeneous Path** (Same Database Type)

* You’re moving from **MySQL to Amazon RDS for MySQL** or **Oracle to Amazon RDS for Oracle.**
* The waters are calm since the source and target databases speak the same "language."
* You simply connect the source and target databases, press a button, and let DMS handle the rest.

**The Heterogeneous Path** (Different Database Types)

* You’re moving from **Oracle to Amazon Aurora** or **SQL Server to Amazon DynamoDB.**
* This route is trickier because the source and target have different "dialects."
* **Step 1:** Use the **AWS Schema Conversion Tool** to translate the schema and database code.
* **Step 2:** Set sail with DMS to migrate the data itself.

**Side Quests on the Data Seas**

DMS isn’t just about big migrations; it helps you conquer other challenges too:

**Testing New Waters:** Want to test changes without disturbing the production database? DMS creates a copy of your live database in a dev or test environment.

**Merging Islands of Data:** Need to combine multiple databases into one central hub? DMS helps you consolidate them effortlessly.

**Guarding Against Disaster:** With **continuous replication,** DMS keeps your data synced across locations, preparing you for unexpected events or ensuring data is available globally.

**Why Choose DMS as Your Navigator?**

**Keeps the Lights On:** Your source database stays operational during the move, so your apps don’t skip a beat.

**Flexible Explorer:** Handles both same-type and different-type migrations with ease.

**Handles Any Terrain:** Works with on-premises, Amazon EC2, or Amazon RDS as both the source and destination.

**Always Ready for the Future:** Whether it’s a one-time migration or ongoing replication, DMS is up for the challenge.

With **AWS DMS**, your data’s journey to the cloud becomes an exciting adventure rather than a treacherous expedition. Whether you’re modernizing your database, testing new horizons, or fortifying against disasters, DMS is the hero that gets your data home safely and efficiently. 🌊✨

**AWS Database Migration Service (AWS DMS)**

**AWS Data Migration Service** enables you to migrate relational databases, nonrelational databases, and other types of data stores.

With AWS DMS, you move data between a source database and a target database. The source and target database can be of the same type or different types. During the migration, your source database remains operational, reducing downtime for any applications that rely on the database.

For example, suppose that you have a MySQL database that is stored on premises in an Amazon EC2 instance or in Amazon RDS. Consider the MySQL database to be your source database. Using AWS DMS, you could migrate your data to a target database, such as an Amazon Aurora database.

**Other use cases for AWS DMS**

**Development and test database migrations:** Enabling developers to test applications against production data without affecting production users**.**

**Database consolidation:** Combining several databases into a single database.

**Continuous replication:** Sending ongoing copies of your data to other target sources instead of doing a one-time migration

**Module 5**

**Additional Database Services and Database Accelerators**

In the land of AWS, there’s no such thing as a "one-size-fits-all" database. Different businesses have different needs, and forcing data to fit into a rigid mold is like trying to shove a square peg into a round hole. So, AWS offers a wide array of specialized databases and tools to ensure every business can pick the perfect fit for their unique use case.

**The Specialized Database Heroes**

**Amazon DocumentDB** – The Content Keeper: Imagine you’re managing a library of user profiles, product catalogs, or even content management systems. You need a database that can handle structured documents like JSON. Enter **Amazon DocumentDB**, built for handling rich, document-based data while staying scalable and efficient.

**Amazon Neptune** – The Connection Tracker: Picture a vast web of connections, like a social network or a recommendation engine. Traditional databases struggle to map "who knows who" or "who bought what." **Amazon Neptune**, a graph database, swoops in to handle these intricate relationships effortlessly. It’s also a fraud detective’s dream for tracking suspicious connections.

**Amazon QLDB (Quantum Ledger Database)** – The Record Keeper: Some businesses, like banks or supply chains, need a tamper-proof system where every action is permanently recorded. That’s where **Amazon QLDB** shines. Unlike blockchain, which decentralizes data, QLDB offers an **immutable ledger**—perfect for industries that demand auditability and 100% data integrity.

**Amazon Managed Blockchain** – The Decentralization Expert: If your business needs a **blockchain solution**, such as managing decentralized processes, AWS has you covered with **Amazon Managed Blockchain**. It’s perfect for specific use cases where transparency and decentralization are critical.

**The Speed Demons: Database Accelerators**

Databases are great, but who doesn’t love things to run faster? AWS offers **accelerators** to turbocharge your database performance:

**Amazon ElastiCache** – The Speed Booster

* Think of **ElastiCache** as a turbocharged memory layer that handles repetitive database requests in **microseconds** instead of milliseconds.
* Available in two flavors: **Memcached** for simple caching and **Redis** for advanced features like data structures and replication.

**DynamoDB Accelerator (DAX)** – The DynamoDB Supercharger

* If you’re using **DynamoDB**, add **DAX** for a **native caching layer** that slashes read times for non-relational data.
* It’s like strapping a rocket to your database for near-instant responses.

**Key Lesson: The Right Tool for the Job**

AWS isn’t about forcing you into a specific database. It’s about providing a toolbox where every tool has a purpose:

* Use **DocumentDB** for flexible document storage.
* Choose **Neptune** for mapping relationships and social graphs.
* Rely on **QLDB** for immutable and auditable records.
* Add **ElastiCache** or **DAX** to accelerate your database for ultra-fast responses.

With AWS, you’re the craftsman, and the cloud is your workshop—filled with tools to build whatever your business needs.

**Additional database services**

**Amazon DocumentDB**

**Amazon DocumentDB** is a document database service that supports MongoDB workloads. (MongoDB is a document database program.)

**Amazon Neptune**

**Amazon Neptune** is a graph database service.

You can use Amazon Neptune to build and run applications that work with highly connected datasets, such as recommendation engines, fraud detection, and knowledge graphs.

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

Amazon Quantum Ledger Database (Amazon QLDB) is a ledger database service.

You can use Amazon QLDB to review a complete history of all the changes that have been made to your application data.

**Amazon Managed Blockchain**

**Amazon Managed Blockchain** is a service that you can use to create and manage blockchain networks with open-source frameworks.

Blockchain is a distributed ledger system that lets multiple parties run transactions and share data without a central authority.

**Amazon ElastiCache**

**Amazon ElastiCache** is a service that adds caching layers on top of your databases to help improve the read times of common requests.

It supports two types of data stores: Redis and Memcached.

**Amazon DynamoDB Accelerator**

**Amazon DynamoDB Accelerator** is an in-memory cache for DynamoDB.

It helps improve response times from single-digit milliseconds to microseconds.

**Module 5**

**Summary**

In Module 5, you learned about the following concepts:

* Amazon EC2 instance store and Amazon EBS
* Amazon S3
* Amazon EFS
* Relational databases and Amazon RDS
* Nonrelational databases and DynamoDB
* Amazon Redshift
* AWS DMS
* Additional database services and accelerators

**Elastic Block Store (EBS)**: Provides persistent, non-ephemeral storage attached to EC2 instances.

**Amazon S3**: A simple yet powerful object storage solution accessible via a click or API.

**Amazon RDS and DynamoDB**:

* **Relational Databases**: For structured data needing complex relationships and queries.
* **DynamoDB**: A non-relational database for workloads requiring key-value pairs with high scalability.

**Amazon EFS**: Ideal for scalable and distributed **file storage** use cases.

**Amazon Redshift**: Purpose-built for **data warehousing** and analytics involving massive datasets.

**Database Migration Service (DMS)**: Facilitates secure and seamless migration of databases to AWS, supporting both homogeneous and heterogeneous migrations.

**Specialized Database Services**:

* **DocumentDB**: For content management and catalogs.
* **Neptune**: A graph database for social networks and recommendation engines.
* **QLDB**: An immutable ledger for secure financial and supply chain records.
* **Amazon Managed Blockchain**: For decentralized blockchain applications.

**Database Accelerators**:

* **Amazon ElastiCache**: Adds a caching layer for faster read operations.
* **DynamoDB Accelerator (DAX)**: Enhances read performance for DynamoDB workloads.

**Module 5**

**Quiz**

|  |
| --- |
| Which Amazon S3 storage classes are optimized for archival data? (Select TWO.)   * Amazon S3 Standard * **Amazon S3 Glacier Flexible Retrieval** * Amazon S3 Intelligent-Tiering * Amazon S3 Standard-IA * **Amazon S3 Glacier Deep Archive** |

|  |
| --- |
| Which statement or statements are TRUE about Amazon EBS volumes and Amazon EFS file systems?   * **EBS volumes store data within a single Availability Zone. Amazon EFS file systems store data across multiple Availability Zones.** * EBS volumes store data across multiple Availability Zones. Amazon EFS file systems store data within a single Availability Zone. * EBS volumes and Amazon EFS file systems both store data within a single Availability Zone. * EBS volumes and Amazon EFS file systems both store data across multiple Availability Zones. |

|  |
| --- |
| You want to store data in an object storage service. Which AWS service is best for this type of storage?   * Amazon Managed Blockchain * Amazon Elastic File System (Amazon EFS) * Amazon Elastic Block Store (Amazon EBS) * **Amazon Simple Storage Service (Amazon S3)** |

|  |
| --- |
| Which statement best describes Amazon DynamoDB?   * A service that enables you to run relational databases in the AWS Cloud * **A serverless key-value database service** * A service that you can use to migrate relational databases, nonrelational databases, and other types of data stores * An enterprise-class relational database |

|  |
| --- |
| Which service is used to query and analyze data across a data warehouse?   * **Amazon Redshift** * Amazon Neptune * Amazon DocumentDB * Amazon ElastiCache |

**Module 6**

**Security**

**Learning objectives**

In this module, you will learn how to:

* Explain the **benefits of the shared responsibility model**.
* Describe **multi-factor authentication (MFA**).
* Differentiate between the **AWS Identity and Access Management (IAM) security levels.**
* Explain the main **benefits of AWS Organizations**.
* Describe **security policies** at a basic level.
* Summarize the **benefits of compliance with AWS**.
* Explain **additional AWS security services** at a basic level.

**Module 6**

**Introduction to Security**

Security is a critical aspect of AWS, and this module introduces the foundational concepts and measures in place to keep your workloads safe. Here's a summary:

1. **Shared Responsibility Model**:
   * **AWS's Responsibility**: Security **of** the cloud, including physical security of data centers, service infrastructure, and foundational layers.
   * **Customer's Responsibility**: Security **in** the cloud, including securing their workloads, applications, and data.
2. **AWS Security Mechanisms**:
   * AWS provides robust security services, tools, and features to help customers secure their cloud environments.

**Module 6**

**AWS Shared Responsibility Model**

Once upon a time, a business decided to move its operations to the AWS Cloud. As the team embarked on this journey, they were greeted by a wise AWS guide who posed a thought-provoking question: “Who is ultimately responsible for the security of your cloud environment? Is it *you*, the customer, or *AWS*?”

The team hesitated, unsure of the answer. The guide smiled and replied,  
“The answer is both. Security is a shared responsibility, much like protecting a home.”

**The House Analogy**

The guide painted a vivid picture: “Imagine AWS is the builder of a house. We are responsible for constructing strong walls, solid doors, and a secure foundation. But once the house is handed over, it’s your job to lock the doors and set up your security systems.”

This shared approach to responsibility was called the **AWS Shared Responsibility Model**. The guide explained that the cloud was not a single object but a collection of layers, each with distinct roles:

* AWS is responsible **for the cloud**, ensuring the security of its infrastructure, such as data centers, hardware, and networking layers.
* Customers are responsible **in the cloud**, safeguarding their data, operating systems, and applications.

**The Layers of Responsibility**

**The Foundation: Physical Security:** Deep within the cloud’s core lay the physical layer, guarded by fences, cameras, and security personnel.  
“AWS takes full ownership here,” said the guide. “Our data centers are fortresses, impervious to intruders.”

**The Middle Layer: Networks and Hypervisors:** Above the physical layer, AWS fortified its networks and hypervisors—technologies that ensure EC2 instances run smoothly.

“We’ve engineered these layers to be robust and tamper-proof,” the guide added.

**The Magic Line: Operating System:** Then came a crucial dividing line—the operating system.

“Here, the responsibility shifts to you,” said the guide. “You choose the OS, patch it, and control access. AWS has no backdoor to your system, just as a builder doesn’t keep copies of your house keys.”

**Your Creations: Applications and Data:** On top of the operating system, customers installed applications and stored their data. This layer was entirely their responsibility.

“Your data is your crown jewel,” the guide emphasized. “Whether you want it public, private, or encrypted, AWS provides tools, but you make the decisions.”

To illustrate the importance of encryption, the guide shared a tip:  
“Think of encryption as locking valuables in a safe. Even if someone sneaks into your house, all they’ll find are unreadable files.”

**The Power of Collaboration**

The AWS Shared Responsibility Model wasn’t just about dividing tasks. It was about collaboration, ensuring both sides worked together to create a secure and trustworthy environment.

“AWS protects the **of** the cloud,” said the guide, “while you secure what’s **in** the cloud. Together, we can achieve unparalleled security.”

And so, the business team left the guide, empowered by their newfound understanding of shared responsibility. They knew their role in the cloud and trusted AWS to handle its part. With both sides working in harmony, the team built a secure, resilient environment, ready to face any challenge.

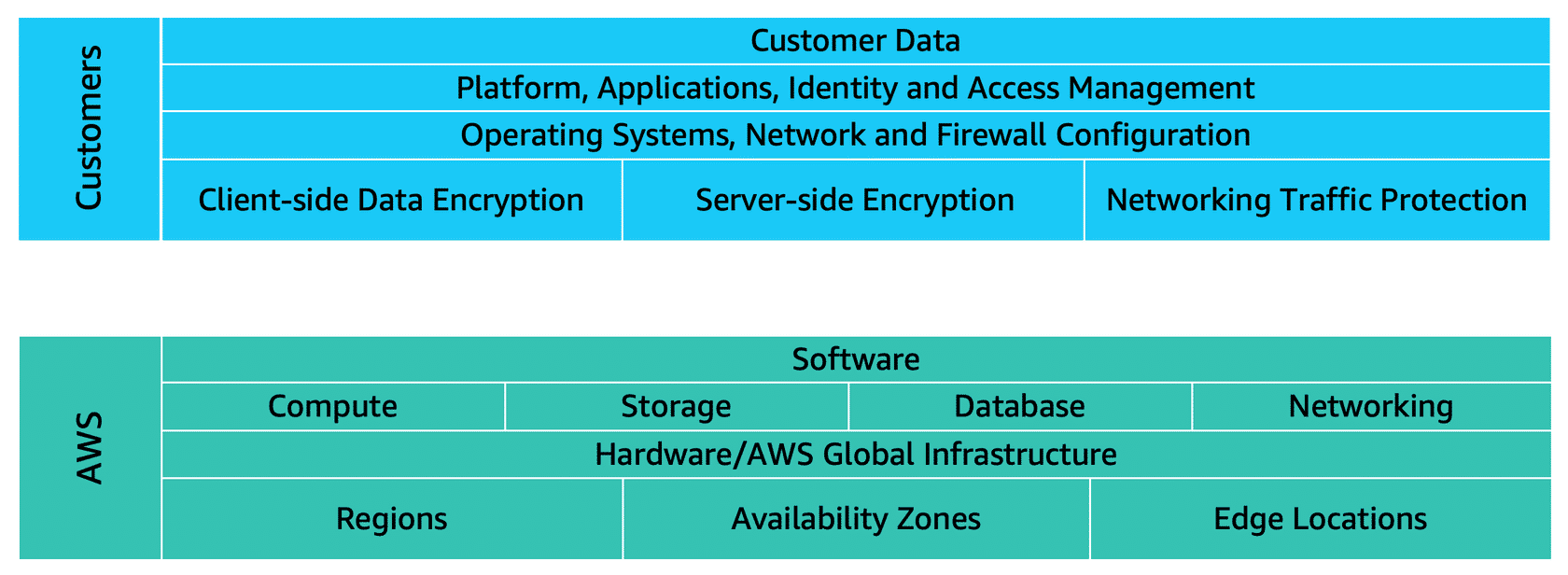
**Moral of the story**: When you share responsibilities wisely, security becomes a partnership that inspires trust.

**The AWS shared responsibility model**

Throughout this course, you have learned about a variety of resources that you can create in the AWS Cloud. These resources include Amazon EC2 instances, Amazon S3 buckets, and Amazon RDS databases. Who is responsible for keeping these resources secure: you (the customer) or AWS?

The answer is both. The reason is that you do not treat your AWS environment as a single object. Rather, you treat the environment as a collection of parts that build upon each other. AWS is responsible for some parts of your environment and you (the customer) are responsible for other parts. This concept is known as the **Shared Responsibility Model**.

The shared responsibility model divides into customer responsibilities (commonly referred to as “security in the cloud”) and AWS responsibilities (commonly referred to as “security of the cloud”).



You can think of this model as being similar to the division of responsibilities between a homeowner and a homebuilder. The builder (AWS) is responsible for constructing your house and ensuring that it is solidly built. As the homeowner (the customer), it is your responsibility to secure everything in the house by ensuring that the doors are closed and locked.

**Customers: Security in the cloud**

Customers are responsible for the security of everything that they create and put *in*the AWS Cloud.

When using AWS services, you, the customer, maintain complete control over your content. You are responsible for managing security requirements for your content, including which content you choose to store on AWS, which AWS services you use, and who has access to that content. You also control how access rights are granted, managed, and revoked.

The security steps that you take will depend on factors such as the services that you use, the complexity of your systems, and your company’s specific operational and security needs. Steps include selecting, configuring, and patching the operating systems that will run on Amazon EC2 instances, configuring security groups, and managing user accounts.

**AWS: Security of the cloud**

AWS is responsible for security *of*the cloud.

AWS operates, manages, and controls the components at all layers of infrastructure. This includes areas such as the host operating system, the virtualization layer, and even the physical security of the data centers from which services operate.

AWS is responsible for protecting the global infrastructure that runs all of the services offered in the AWS Cloud. This infrastructure includes AWS Regions, Availability Zones, and edge locations.

AWS manages the security of the cloud, specifically the physical infrastructure that hosts your resources, which include:

* Physical security of data centers
* Hardware and software infrastructure
* Network infrastructure
* Virtualization infrastructure

Although you cannot visit AWS data centers to see this protection firsthand, AWS provides several reports from third-party auditors. These auditors have verified its compliance with a variety of computer security standards and regulations.

**Knowledge Check**

|  |
| --- |
| Which tasks are the responsibilities of customers? (Select TWO.)   * Maintaining network infrastructure * **Patching software on Amazon EC2 instances** * Implementing physical security controls at data centers * **Setting permissions for Amazon S3 objects** * Maintaining servers that run Amazon EC2 instances |

**Module 6**

**User Permissions and Access**

Imagine a bustling coffee shop. Every employee has a unique role, and the shop’s systems are designed to keep things running smoothly. Each worker only has access to the tools they need to do their job. This coffee shop serves as the perfect metaphor for understanding how AWS handles user permissions and access.

**The Coffee Shop Owner: The Root User**

In our coffee shop, the owner has ultimate authority. They can use the register, check inventory, or make major business decisions. In AWS, this level of access is represented by the **root user**.

* The root user is the most powerful identity in an AWS account.
* It can control everything, from spinning up databases to creating S3 buckets.
* To keep it safe, AWS recommends enabling **Multi-Factor Authentication (MFA)** for the root user. This adds an extra layer of security by requiring a token along with the password.

But just like the owner doesn’t run every aspect of the coffee shop alone, the root user should only be used sparingly in AWS.

**The Cashier and the Inventory Manager: IAM Users and Least Privilege**

Now, meet Rudy, the cashier, and Blaine, the inventory manager. Each of them has a unique login and access to only the systems they need:

* Rudy can log into the register but **cannot** access the inventory system.
* Blaine can check inventory but **cannot** process customer payments.

In AWS, this concept is managed through **IAM Users**.

* When you create an IAM user, they start with **no permissions**—they can’t do anything until explicitly granted access.
* Access is granted using the **least privilege principle**: users only get the permissions they need, and nothing more.

For instance, to let a user view the contents of a specific S3 bucket, you’d create a **policy**. Policies are like instruction manuals written in JSON that define what actions a user can perform.

* Example: A policy might allow a user to list the files in a specific bucket but deny all other actions.

**Groups: Simplifying Permissions**

Let’s say the coffee shop hires several new cashiers. Instead of granting each cashier access to the register individually, the owner creates a “Cashiers” group.

* Everyone in the group automatically inherits the same permissions.
* Similarly, AWS allows you to organize IAM users into **groups** and assign policies to the group, simplifying permission management.

**Blaine’s Changing Roles: AWS Roles**

One day, Blaine’s role at the coffee shop changes. Instead of managing inventory, he’s assigned to clean up the shop. Tomorrow, he might be back at the register. His responsibilities vary day by day.

In AWS, this flexibility is mirrored by **IAM Roles**.

* A role grants **temporary permissions** to perform specific tasks.
* Unlike IAM users, roles don’t have usernames or passwords. They are assumed by users, applications, or even other AWS services to gain temporary access.
* For example, Blaine could assume a “Cleaner” role today and an “Inventory Manager” role tomorrow.

**Federated Access: Logging in with Corporate Credentials**

What if employees at the coffee shop could use their existing corporate ID cards to clock in? In AWS, this is possible through **federated access**.

* Instead of creating IAM users for everyone, you can map corporate identities to IAM roles.
* Employees can log in using their regular credentials, streamlining the process and reducing administrative overhead.

**Key Takeaways from the Coffee Shop**

1. **Root User:** The coffee shop owner has unlimited access, just like the AWS root user. Use it sparingly and secure it with MFA.
2. **IAM Users:** Each employee has their own login with permissions tailored to their job.
3. **Least Privilege:** Employees only access what they need—no more, no less.
4. **Policies:** JSON-based instructions define what users can and cannot do.
5. **Groups:** Like creating a “Cashiers” group, AWS groups simplify permission management for similar users.
6. **Roles:** Blaine’s temporary responsibilities mirror the flexibility of IAM roles, granting access for specific tasks.
7. **Federated Access:** Corporate credentials can be linked to AWS accounts, making authentication seamless.

Just like running a successful coffee shop, managing permissions in AWS requires assigning the right roles and responsibilities. With these tools, AWS ensures your environment is secure while staying efficient and organized.

**AWS Identity and Access Management (IAM)**

**AWS Identity and Access Management** enables you to manage access to AWS services and resources securely.

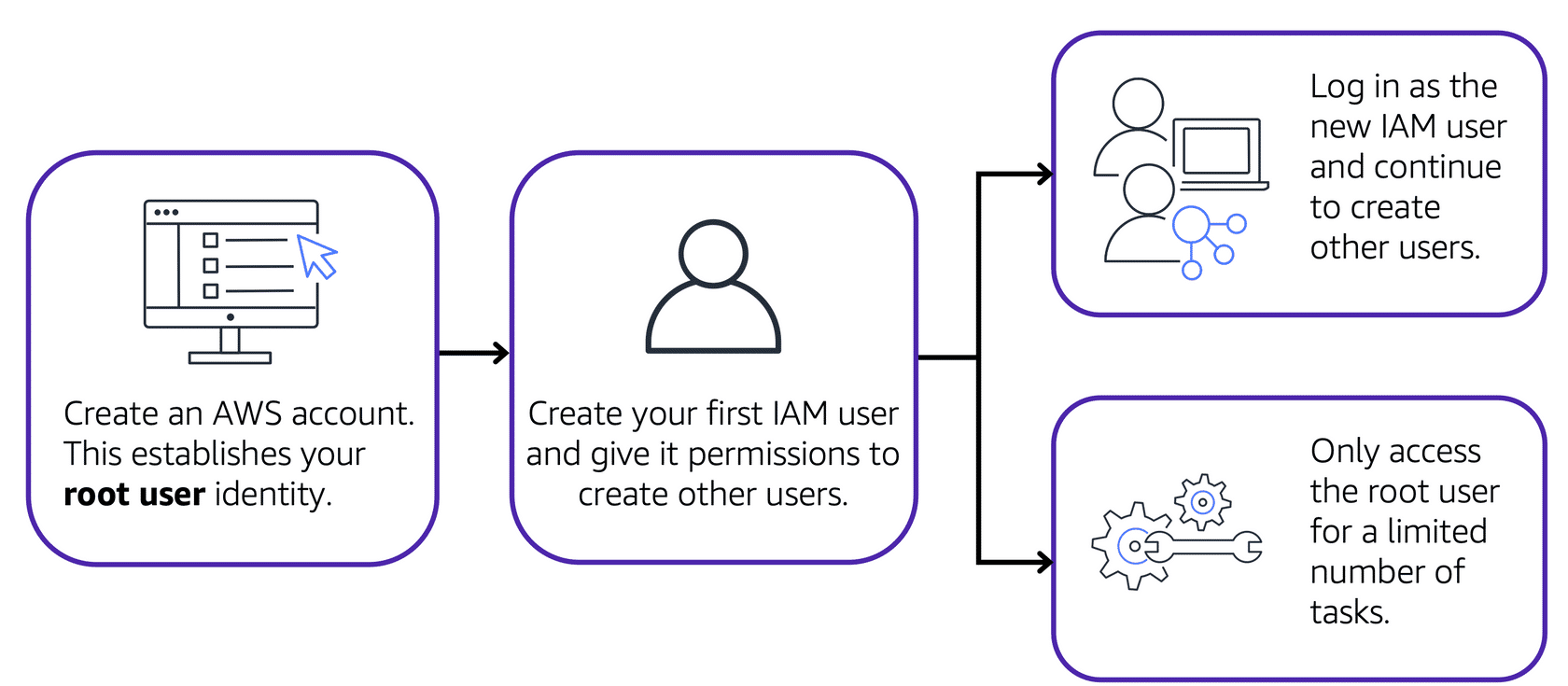
IAM gives you the flexibility to configure access based on your company’s specific operational and security needs. You do this by using a combination of IAM features.

* IAM users, groups, and roles
* IAM policies
* Multi-factor authentication

**AWS account root user**

When you first create an AWS account, you begin with an identity known as the **root user**.

The root user is accessed by signing in with the email address and password that you used to create your AWS account. You can think of the root user as being similar to the owner of the coffee shop. It has complete access to all the AWS services and resources in the account.



Best practice:

Do **not** use the root user for everyday tasks.

Instead, use the root user to create your first IAM user and assign it permissions to create other users.

Then, continue to create other IAM users, and access those identities for performing regular tasks throughout AWS. Only use the root user when you need to perform a limited number of tasks that are only available to the root user. Examples of these tasks include changing your root user email address and changing your AWS support plan.

**IAM users**

An **IAM user** is an identity that you create in AWS. It represents the person or application that interacts with AWS services and resources. It consists of a **name** and **credentials**.

By default, when you create a new IAM user in AWS, it has no permissions associated with it. To allow the IAM user to perform specific actions in AWS, such as launching an Amazon EC2 instance or creating an Amazon S3 bucket, you must grant the IAM user the necessary permissions.

**Best practice:**

We recommend that you create individual IAM users for each person who needs to access AWS.

Even if you have multiple employees who require the same level of access, you should create individual IAM users for each of them. This provides additional security by allowing each IAM user to have a unique set of security credentials.

**IAM policies**

An **IAM policy** is a document that allows or denies permissions to AWS services and resources.

IAM policies enable you to customize users’ levels of access to resources. **For example**, you can allow users to access all of the Amazon S3 buckets within your AWS account, or only a specific bucket.

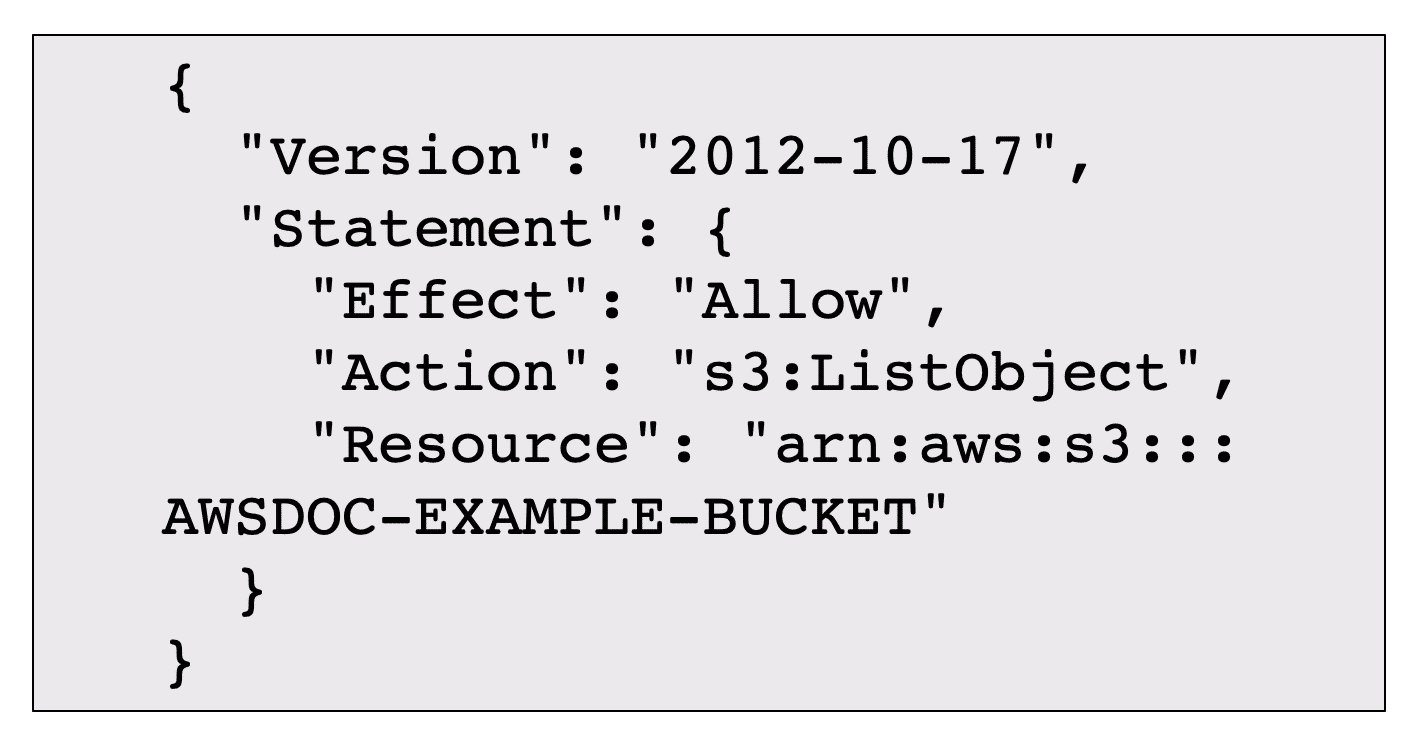
**Best practice:**

Follow the security principle of **least privilege** when granting permissions.

By following this principle, you help to prevent users or roles from having more permissions than needed to perform their tasks.

**For example**, if an employee needs access to only a specific bucket, specify the bucket in the IAM policy. Do this instead of granting the employee access to all of the buckets in your AWS account.

**Example: IAM policy**

Here’s an example of how IAM policies work. Suppose that the coffee shop owner has to create an IAM user for a newly hired cashier. The cashier needs access to the receipts kept in an Amazon S3 bucket with the ID: AWSDOC-EXAMPLE-BUCKET.

*This example IAM policy allows permission to access the objects in the Amazon S3 bucket with ID: AWSDOC-EXAMPLE-BUCKET.*

In this example, the IAM policy is allowing a specific action within Amazon S3: ListObject. The policy also mentions a specific bucket ID: AWSDOC-EXAMPLE-BUCKET. When the owner attaches this policy to the cashier’s IAM user, it will allow the cashier to view all of the objects in the AWSDOC-EXAMPLE-BUCKET bucket.

If the owner wants the cashier to be able to access other services and perform other actions in AWS, the owner must attach additional policies to specify these services and actions.

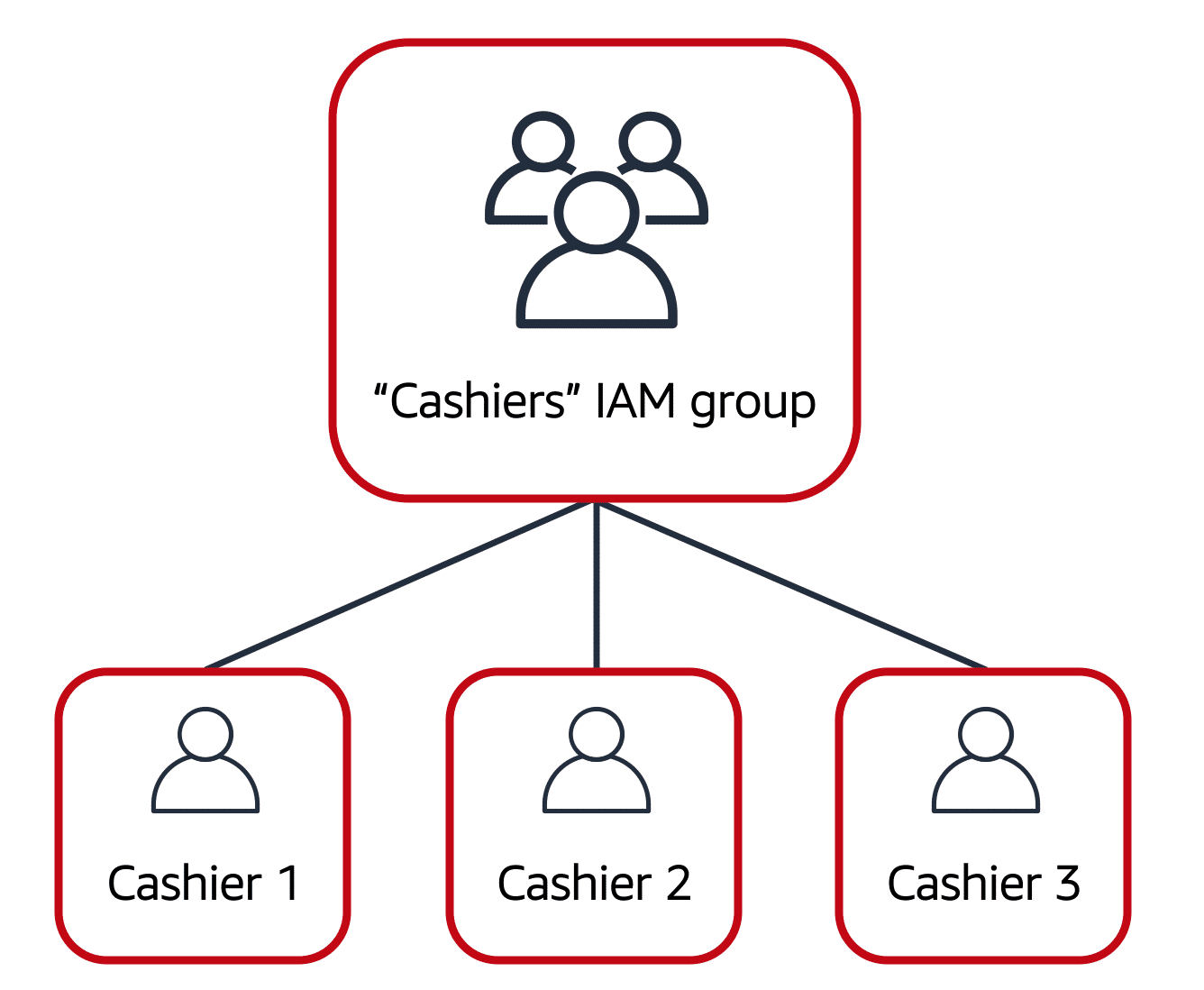
Now, suppose that the coffee shop has hired a few more cashiers. Instead of assigning permissions to each individual IAM user, the owner places the users into an **IAM Groups**.

**IAM Groups**

**IAM groups**

An IAM group is a collection of IAM users. When you assign an IAM policy to a group, all users in the group are granted permissions specified by the policy.

Here’s an example of how this might work in the coffee shop. Instead of assigning permissions to cashiers one at a time, the owner can create a “Cashiers” IAM group. The owner can then add IAM users to the group and then attach permissions at the group level.

Assigning IAM policies at the group level also makes it easier to adjust permissions when an employee transfers to a different job.

**For example**, if a cashier becomes an inventory specialist, the coffee shop owner removes them from the “Cashiers” IAM group and adds them into the “Inventory Specialists” IAM group. This ensures that employees have only the permissions that are required for their current role.

What if a coffee shop employee hasn’t switched jobs permanently, but instead, rotates to different workstations throughout the day? This employee can get the access they need through **IAM Roles**.

**IAM roles**

In the coffee shop, an employee rotates to different workstations throughout the day. Depending on the staffing of the coffee shop, this employee might perform several duties: work at the cash register, update the inventory system, process online orders, and so on.

When the employee needs to switch to a different task, they give up their access to one workstation and gain access to the next workstation. The employee can easily switch between workstations, but at any given point in time, they can have access to only a single workstation. This same concept exists in AWS with IAM roles.

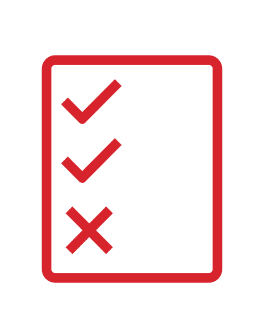
An IAM role is an identity that you can assume to gain temporary access to permissions.

Before an IAM user, application, or service can assume an IAM role, they must be granted permissions to switch to the role. When someone assumes an IAM role, they abandon all previous permissions that they had under a previous role and assume the permissions of the new role.

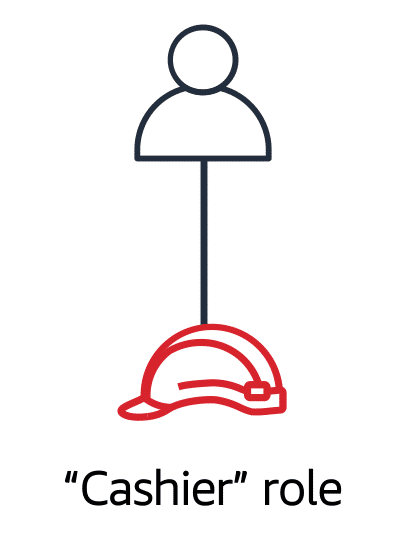
**Best practice:**

IAM roles are ideal for situations in which access to services or resources needs to be granted temporarily, instead of long-term.

An example of how IAM roles could be used in the coffee shop example.



First, the coffee shop owner grants the employee permissions to the "Cashier" and "Inventory" roles so they can switch between these two workstations.



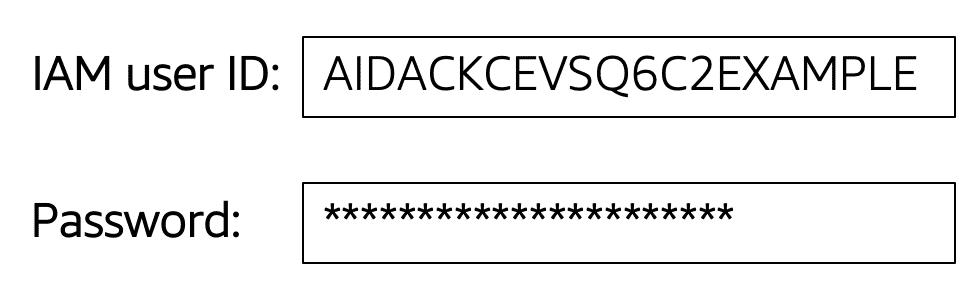
The employee begins their day by assuming the “Cashier” role. This grants them access to the cash register system.

**Multi-factor authentication**

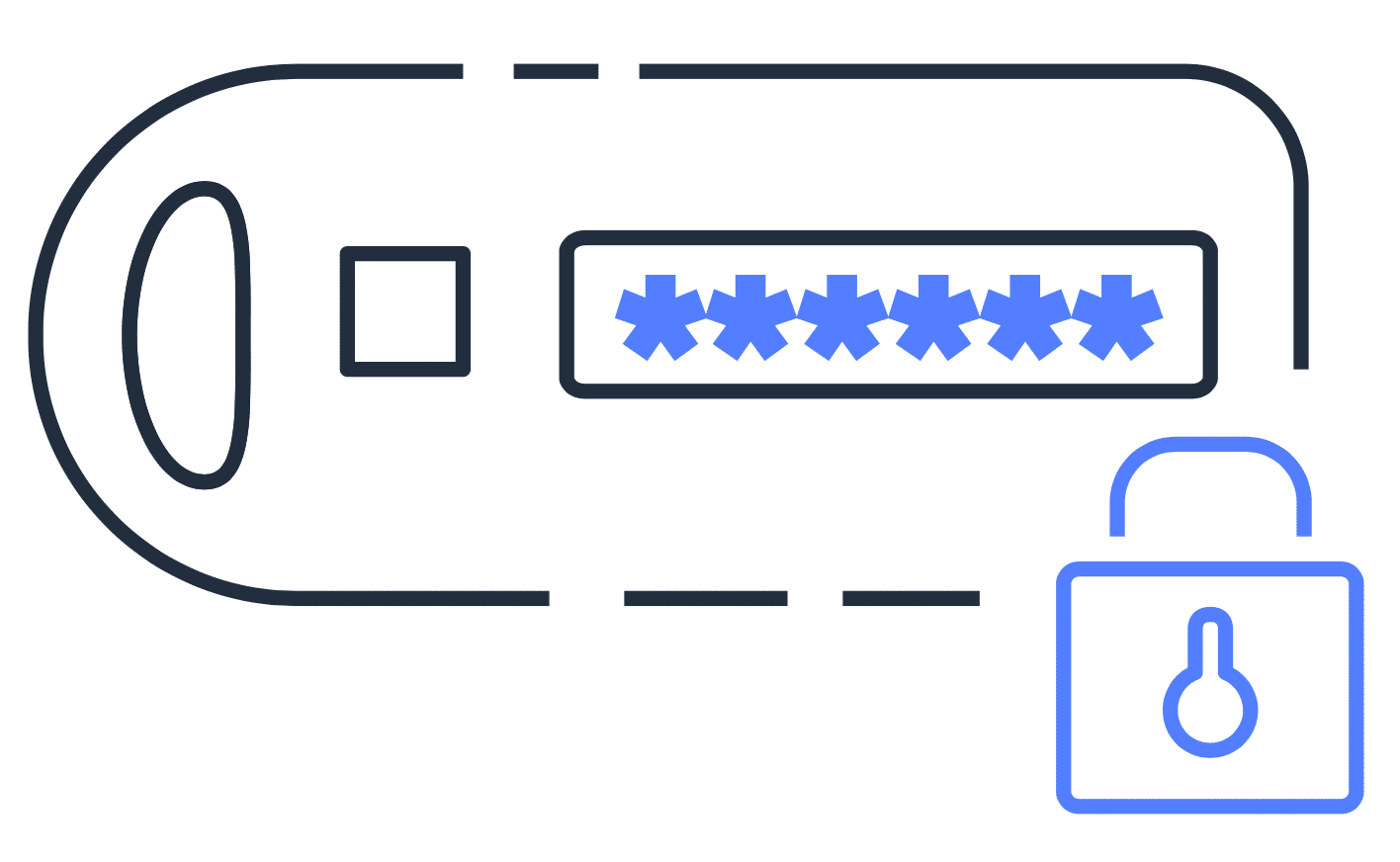
Have you ever signed in to a website that required you to provide multiple pieces of information to verify your identity? You might have needed to provide your password and then a second form of authentication, such as a random code sent to your phone. This is an example of **multi-factor authentication**.

In IAM, multi-factor authentication (MFA) provides an extra layer of security for your AWS account.

**How MFA works:**



First, a user enters their IAM user ID and password to sign in to an AWS website.



Next, the user is prompted for an authentication response from their AWS MFA device. This device could be a hardware security key, a hardware device, or an MFA application on a device such as a smartphone.

**Module 6**

**AWS Organizations**

Once upon a time, a small startup ventured into the AWS Cloud with a single AWS account. Everything the company needed—development resources, billing information, and experiments—resided in this one account. It worked well at first, like a small team sharing one office.

But as the company grew, things got complicated. Developers needed isolated spaces to test new features. The accounting team wanted access to billing without sifting through technical data. Business units demanded independence to experiment without worrying about breaking something critical.

So, the company decided to create separate AWS accounts for everyone: one for developers, one for accounting, one for each business unit. However, this led to a chaotic mess—a tangled bowl of AWS account spaghetti! Each account had its own set of permissions, billing details, and resources. Keeping track of which account did what became a nightmare.

**The Solution: AWS Organizations**

Just as the chaos reached its peak, someone introduced the team to **AWS Organizations**, a service that could bring order to the madness.

**"Think of AWS Organizations as the central hub for managing all your accounts,"** the team lead explained.

Here’s how AWS Organizations transformed their operations:

**Centralized Management**

The startup had AWS accounts scattered everywhere, but with AWS Organizations, they grouped them together under one umbrella.

* All accounts—Account A, B, C, and even the forgotten D and E—were now part of a single organization.
* This central management made it easy to oversee and control every account from one place.

**Consolidated Billing**

Previously, each account had its own billing process, which was like keeping track of receipts from a hundred different shops. AWS Organizations solved this by enabling **consolidated billing**.

* Now, all accounts’ bills flowed into one primary account.
* They even got **bulk discounts** on AWS services, saving the company money.

It was like switching from individual grocery bills to a family membership card with discounts!

**Hierarchical Groupings: Organizational Units (OUs)**

The team then organized their accounts into groups, called **Organizational Units (OUs)**.

* Developers were grouped into a **Developer OU**, where they could experiment with AWS services freely.
* Accounts requiring strict compliance were placed in a **Compliance OU** to ensure they only accessed regulated AWS services.
* Business units got their own OUs to keep their experiments separate from core company resources.

This hierarchy brought structure to the chaos, like organizing files into labeled folders instead of dumping everything into one drawer.

**Service Control Policies (SCPs)**

The company’s admin realized they needed a way to enforce rules across accounts. What if someone in the Developer OU accidentally spun up a service outside the company’s approved list?

Enter **Service Control Policies (SCPs)**:

* SCPs acted like guardrails, specifying which AWS services and API actions each account could access.
* For example, the Compliance OU could only use services that met regulatory requirements, while the Developer OU could play around with non-critical services.

It was like giving each group in the company a customized set of tools tailored to their needs.

**Happily, Organized Ever After**

With AWS Organizations, the startup transformed its tangled mess of accounts into a well-structured system. Centralized management, consolidated billing, hierarchical groupings, and SCPs ensured every team had what they needed while staying within company policies.

The company could now grow confidently, knowing their AWS accounts were as organized and efficient as their expanding team. And just like that, they turned their AWS account spaghetti into a perfectly plated masterpiece.

**Moral of the Story**

AWS Organizations is your go-to tool for managing multiple AWS accounts. Whether you’re a small startup or a growing enterprise, it brings order, saves money, and ensures compliance—helping you focus on innovation without worrying about chaos.

**AWS Organizations**

Suppose that your company has multiple AWS accounts. You can use **AWS Organizations**to consolidate and manage multiple AWS accounts within a central location.

When you create an organization, AWS Organizations automatically creates a **root**, which is the parent container for all the accounts in your organization.

In AWS Organizations, you can centrally control permissions for the accounts in your organization by using **service control policies**. SCPs enable you to place restrictions on the AWS services, resources, and individual API actions that users and roles in each account can access.

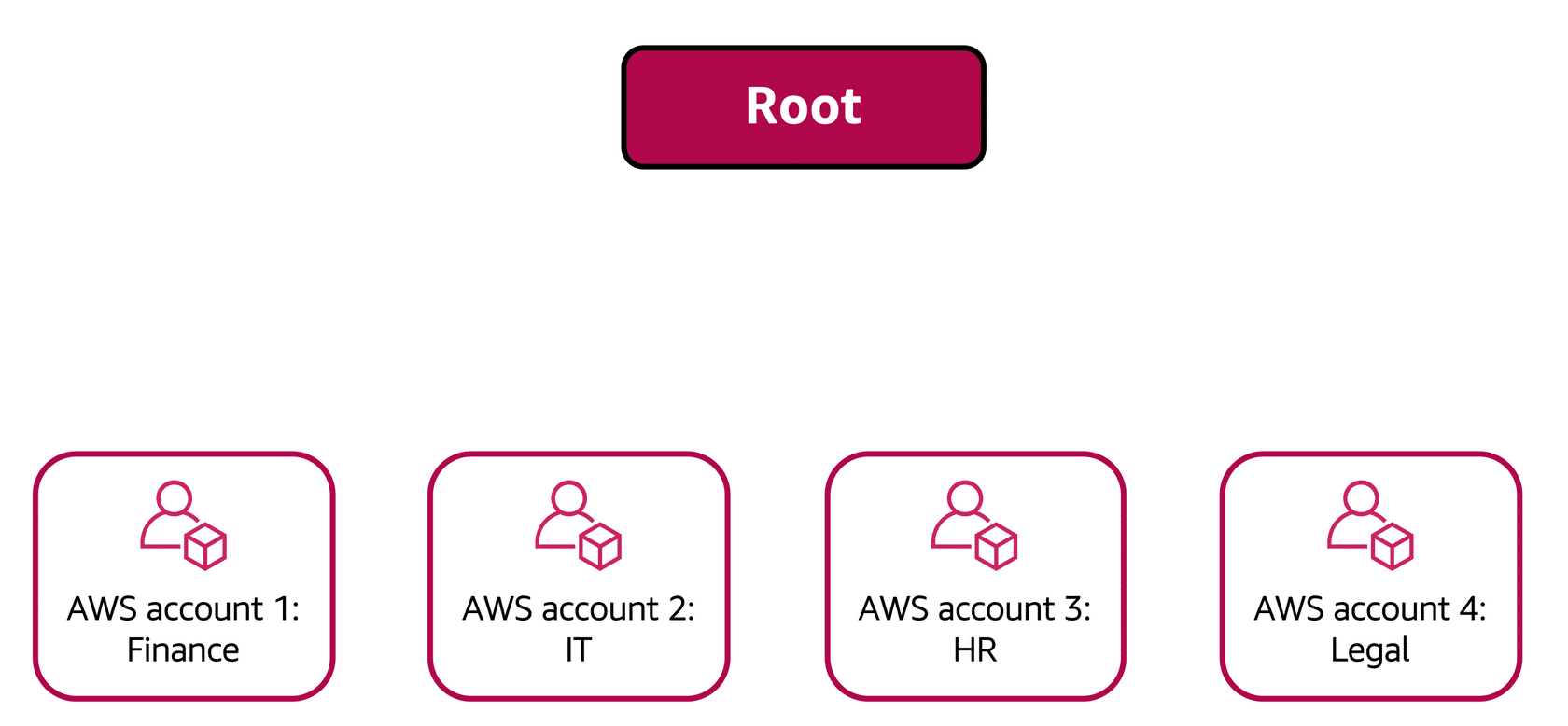
Consolidated billing is another feature of AWS Organizations. You will learn about consolidated billing in a later module.

**Organizational units**

In AWS Organizations, you can group accounts into organizational units (OUs) to make it easier to manage accounts with similar business or security requirements. When you apply a policy to an OU, all the accounts in the OU automatically inherit the permissions specified in the policy.

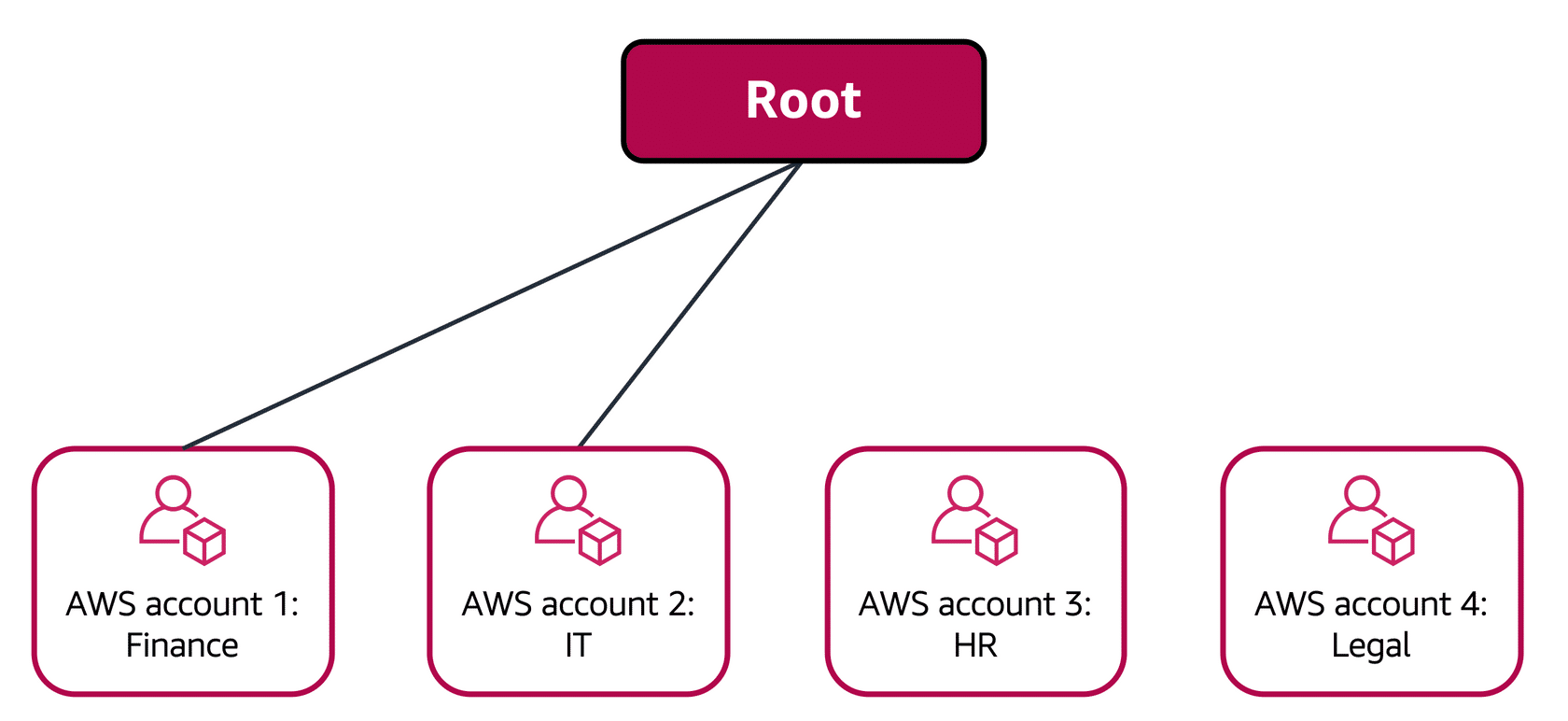
By organizing separate accounts into OUs, you can more easily isolate workloads or applications that have specific security requirements. For instance, if your company has accounts that can access only the AWS services that meet certain regulatory requirements, you can put these accounts into one OU. Then, you can attach a policy to the OU that blocks access to all other AWS services that do not meet the regulatory requirements.

**An example of how a company might use AWS Organizations:**

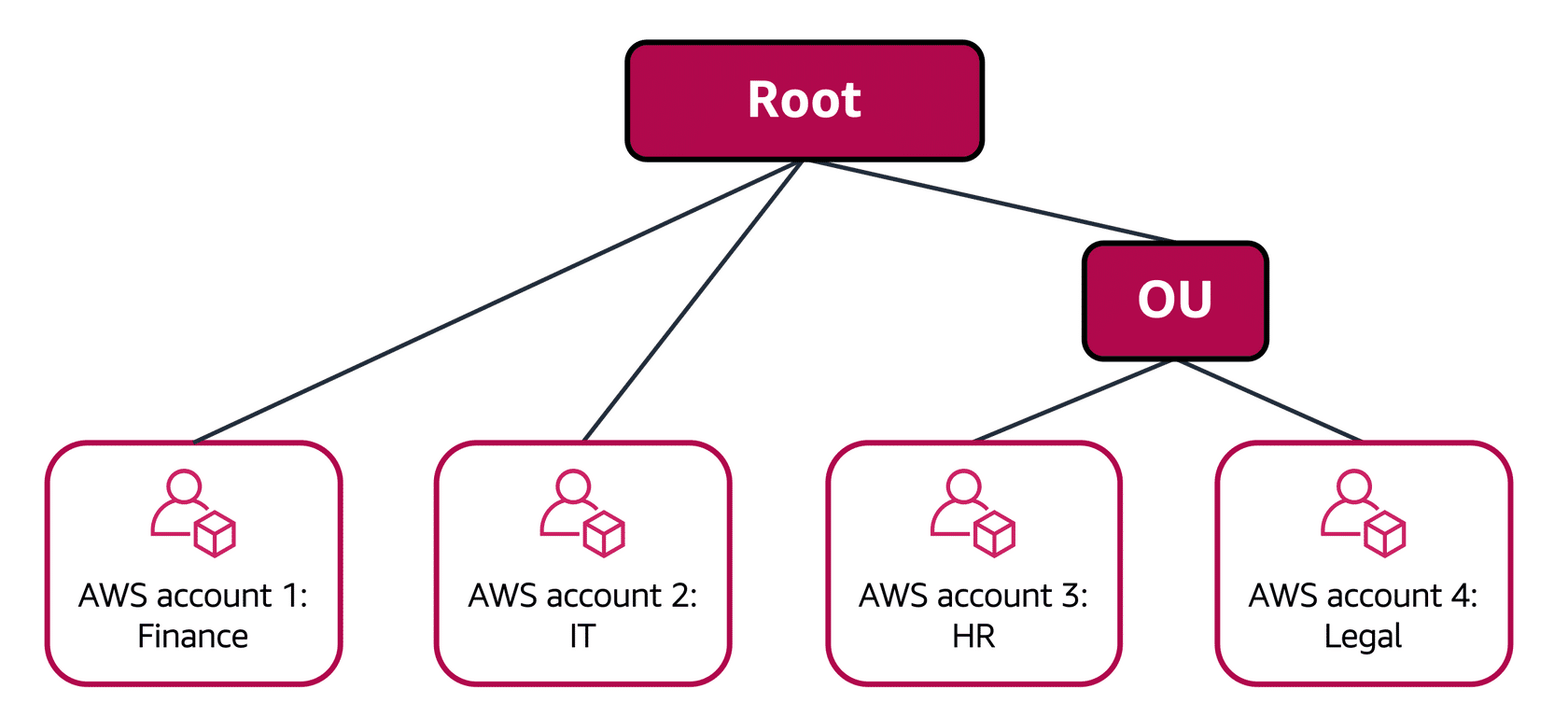


Imagine that your company has separate AWS accounts for the finance, information technology (IT), human resources (HR), and legal departments. You decide to consolidate these accounts into a single organization so that you can administer them from a central location. When you create the organization, this establishes the root.

In designing your organization, you consider the business, security, and regulatory needs of each department. You use this information to decide which departments group together in OUs.



The finance and IT departments have requirements that do not overlap with those of any other department. You bring these accounts into your organization to take advantage of benefits such as consolidated billing, but you do not place them into any OUs.



The HR and legal departments need to access the same AWS services and resources, so you place them into an OU together. Placing them into an OU empowers you to attach policies that apply to both the HR and legal departments’ AWS accounts.

**Knowledge check**

|  |
| --- |
| You are configuring service control policies (SCPs) in AWS Organizations. Which identities and resources can SCPs be applied to? (Select TWO.)   * IAM users * IAM groups * **An individual member account** * IAM roles * **An organizational unit (OU)** |

**Module 6**

**Compliance**

Imagine you're the owner of a coffee shop. To keep your business running smoothly, you need to follow strict rules—health inspectors check for cleanliness, tax auditors review your records, and maybe even a fire marshal ensures safety standards. To pass these inspections, you rely on proper documentation, records, and processes.

Now, let’s bring this analogy to the world of AWS. If your business uses AWS to store data or run applications, you’ll also need to follow industry-specific compliance standards.

For example:

* **GDPR** for consumer data protection in the EU.
* **HIPAA** for healthcare data in the US.

Just like in your coffee shop, documentation, checks, and proper processes are key to staying compliant.

**AWS Compliance: Shared Responsibility**

AWS works like your building's landlord. They’ve already designed the space (data centers and networks) to meet industry best practices for security and compliance. As a customer, you inherit these robust foundations.

But just as you, the coffee shop owner, are responsible for running your shop according to health codes, in AWS, you’re responsible for making sure your applications and data meet compliance standards. This is called the **Shared Responsibility Model**:

* **AWS's responsibility:** The security and compliance of the infrastructure (the building).
* **Your responsibility:** The security and compliance of your data and applications (how you use the building).

**Choosing the Right Location (Region)**

In some cases, laws dictate where you can store your data.

**For example:**

* A coffee shop in Paris must follow French food safety laws.
* Similarly, data collected in the EU must comply with GDPR and often remain within EU borders.

In AWS, this means you can choose a **Region** where your data is stored. AWS ensures that data in one Region isn’t automatically replicated to another.

**Keeping Your Data Secure**

AWS gives you tools to safeguard your data, just like you’d lock your shop doors at night or install a safe for cash.

* **Encryption:** AWS offers encryption mechanisms to secure data, and for many services, you can enable encryption with a simple configuration setting.
* **Ownership:** You always own and control your data in AWS, giving you the freedom to meet specific compliance requirements.

**Accessing Compliance Documents**

Sometimes, inspectors (auditors) want proof that your shop follows the rules. In AWS, you can get these "inspection reports" through a service called **AWS Artifact**.

* AWS Artifact provides compliance reports validated by third-party auditors.
* You can also find additional resources in the **AWS Compliance Center**, including guides like the **AWS Risk and Security Whitepaper**.

**Passing Compliance Audits with AWS**

AWS simplifies compliance in many ways:

1. **Best Practices Built In:** AWS data centers and networks follow industry-leading standards, so parts of your compliance are already handled.
2. **Flexible Tools:** AWS offers features like encryption and access controls to help you meet compliance for your specific use case.
3. **Documentation Support:** Services like AWS Artifact provide the proof you need to show auditors that AWS infrastructure is compliant.

**The Moral of the Story**

Just like running a clean, well-documented coffee shop helps you pass health inspections, following AWS best practices helps you stay compliant in the cloud.

Remember:

* AWS secures the infrastructure (the building).
* You’re responsible for securing your applications and data (the coffee shop operations).

By working together, you can confidently meet compliance standards and keep your cloud operations running smoothly.

**AWS Artifact**

Depending on your company’s industry, you may need to uphold specific standards. An audit or inspection will ensure that the company has met those standards.

**AWS Artifacts** is a service that provides on-demand access to AWS security and compliance reports and select online agreements. AWS Artifact consists of two main sections: **AWS Artifact Agreements and AWS Artifact Reports.**

**AWS Artifact Agreements**

Suppose that your company needs to sign an agreement with AWS regarding your use of certain types of information throughout AWS services. You can do this through **AWS Artifact Agreements**.

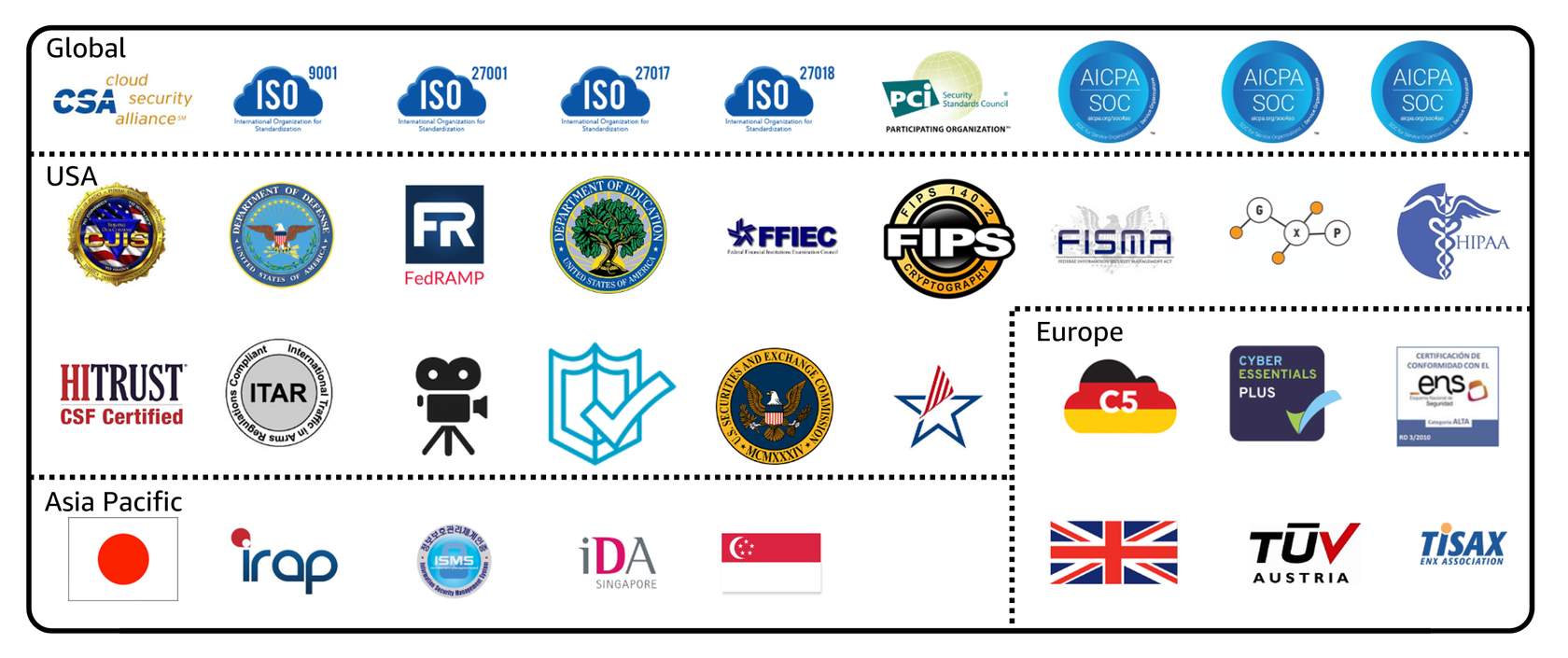
In AWS Artifact Agreements, you can review, accept, and manage agreements for an individual account and for all your accounts in AWS Organizations. Different types of agreements are offered to address the needs of customers who are subject to specific regulations, such as the Health Insurance Portability and Accountability Act (HIPAA).

**AWS Artifact Reports**

Next, suppose that a member of your company’s development team is building an application and needs more information about their responsibility for complying with certain regulatory standards. You can advise them to access this information in **AWS Artifact Reports**.

 AWS Artifact Reports provide compliance reports from third-party auditors. These auditors have tested and verified that AWS is compliant with a variety of global, regional, and industry-specific security standards and regulations. AWS Artifact Reports remains up to date with the latest reports released. You can provide the AWS audit artifacts to your auditors or regulators as evidence of AWS security controls.

The following are some of the compliance reports and regulations that you can find within AWS Artifact. Each report includes a description of its contents and the reporting period for which the document is valid.



AWS Artifact provides access to AWS security and compliance documents, such as AWS ISO certifications, Payment Card Industry (PCI) reports, and Service Organization Control (SOC) reports.  To learn more about the available compliance reports, visit AWS Compliance Programs.

**Customer Compliance Center**

The **Customer Compliance Center** contains resources to help you learn more about AWS compliance.

In the Customer Compliance Center, you can read customer compliance stories to discover how companies in regulated industries have solved various compliance, governance, and audit challenges.

You can also access compliance whitepapers and documentation on topics such as:

* AWS answers to key compliance questions
* An overview of AWS risk and compliance
* An auditing security checklist

Additionally, the Customer Compliance Center includes an auditor learning path. This learning path is designed for individuals in auditing, compliance, and legal roles who want to learn more about how their internal operations can demonstrate compliance using the AWS Cloud.

**Knowledge check**

|  |
| --- |
| Which tasks can you complete in AWS Artifact? (Select TWO.)   * **Access AWS compliance reports on-demand.** * Consolidate and manage multiple AWS accounts within a central location. * Create users to enable people and applications to interact with AWS services and resources. * Set permissions for accounts by configuring service control policies (SCPs). * **Review, accept, and manage agreements with AWS.** |

**Module 6**

**DDOS attacks**

Once upon a time, there was an online bakery that sold delicious cupcakes to customers worldwide. People loved ordering their treats, and everything ran smoothly... until one day, the bakery was under attack.

An evil villain, known as the "DDoS Commander," had a sinister plan to shut down the bakery’s website. This attack wasn’t just from one computer—it was a **Distributed Denial-of-Service (DDoS)** attack, where an army of zombie-like machines, known as "bots," were tricked into overwhelming the bakery’s website with fake requests.

**What Is a DDoS Attack?**

Under normal conditions, the bakery’s website would handle customer orders:

* A request comes in ("I’d like 6 cupcakes!"), and the system processes it and sends back the results.

But during a DDoS attack, the villain uses their zombie bot army to flood the website with so many fake requests that it crashes. Real customers can't get through to order cupcakes anymore.

**How Do They Do It?**

1. **Zombie Army:** The villain tricks other computers on the internet to unknowingly join the attack.
2. **Flooding Tactics:** Bots send so many requests that the bakery’s website runs out of resources to function.

Some of the attack methods include:

* **UDP Floods:** Imagine the villain asking a weather station for a report, but giving *your* address as the return location. Now, your bakery’s website is bombarded with endless weather data it never asked for.
* **HTTP Attacks:** Bots repeatedly request complicated searches from the website, overwhelming its ability to serve real customers.
* **Slowloris Attack:** This is like a customer at the front of the line at your bakery who takes *forever* to place an order, blocking everyone else from getting served.

**How AWS Defends the Bakery**

Luckily, the bakery was hosted on AWS, and the AWS defenders were ready to protect it with a mix of smart strategies and powerful tools.

**The Mighty Security Groups:** These act like bouncers at the door of the bakery. They only let in valid customers and block requests from suspicious sources like UDP floods.

Since AWS operates at a massive scale, attacks like UDP floods are absorbed at the AWS Region level, making it nearly impossible to overwhelm individual servers.

**Elastic Load Balancer (ELB):** The ELB acts like a super-efficient cashier who handles customer requests. Even when faced with a Slowloris attack, the ELB calmly waits for the complete order before passing it to the bakery’s web server.

And since the ELB is scalable across the entire AWS Region, it’s nearly impossible for the villain to overwhelm it.

**Specialized Defense Tools:**

* **AWS Shield Advanced:** A powerful knight that defends against the sharpest and most sophisticated attacks.
* **AWS WAF (Web Application Firewall):** Like a guard dog, it uses machine learning to sniff out bad actors and stop them before they reach the bakery’s website.

**The Takeaway**

The bakery’s website survived the attack thanks to AWS’s **Well-Architected Framework** and defense tools.

* The infrastructure was designed to handle massive attacks.
* Security groups, ELB, and AWS Shield worked together to stop the villain in their tracks.

The villain learned an important lesson: overwhelming AWS would be *way too expensive*. And the bakery? It went on to deliver cupcakes to happy customers worldwide, knowing that AWS had its back.

Moral of the story: With AWS’s powerful defenses, your business can stand strong against even the evilest DDoS attacks. 🍰

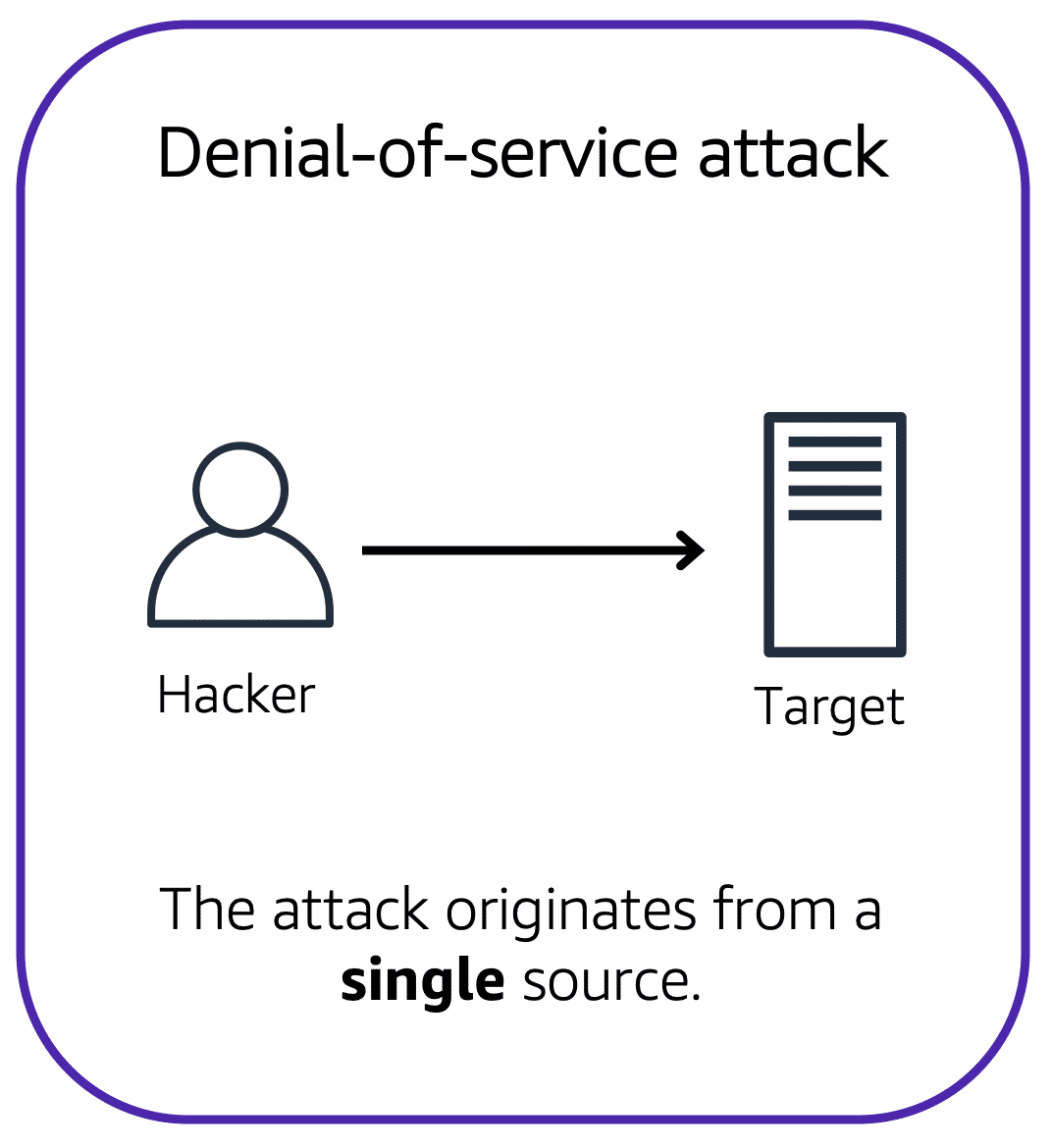
Customers can call the coffee shop to place their orders. After answering each call, a cashier takes the order and gives it to the barista.

However, suppose that a prankster is calling in multiple times to place orders but is never picking up their drinks. This causes the cashier to be unavailable to take other customers’ calls. The coffee shop can attempt to stop the false requests by blocking the phone number that the prankster is using.

In this scenario, the prankster’s actions are similar to a **denial-of-service attack**.

**Denial-of-service attacks**

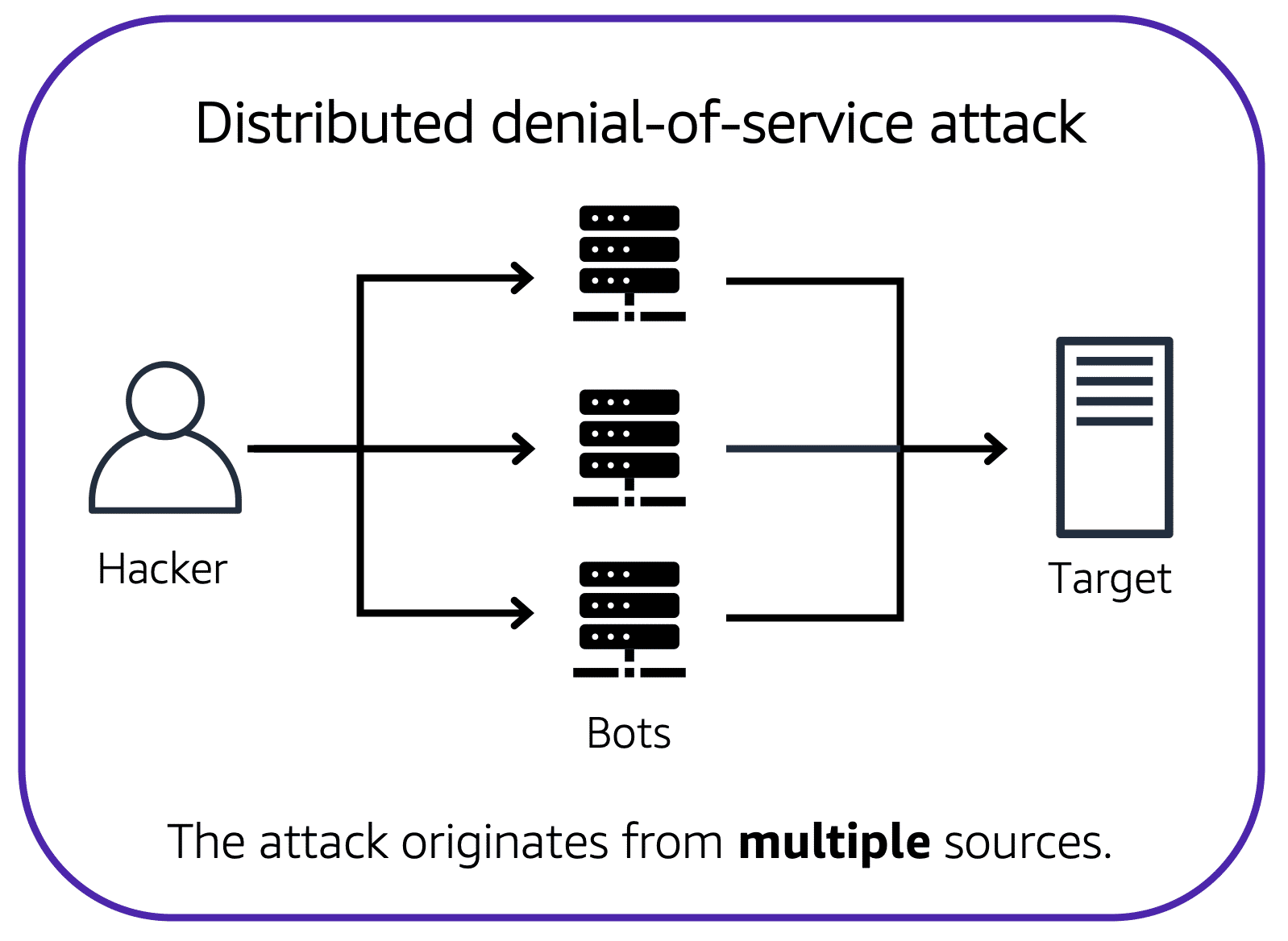
A **denial-of-service (DoS) attack** is a deliberate attempt to make a website or application unavailable to users.



For example, an attacker might flood a website or application with excessive network traffic until the targeted website or application becomes overloaded and is no longer able to respond. If the website or application becomes unavailable, this denies service to users who are trying to make legitimate requests.

**Distributed denial-of-service attacks**

Now, suppose that the prankster has enlisted the help of friends.

The prankster and their friends repeatedly call the coffee shop with requests to place orders, even though they do not intend to pick them up. These requests are coming in from different phone numbers, and it’s impossible for the coffee shop to block them all. Additionally, the influx of calls has made it increasingly difficult for customers to be able to get their calls through. This is similar to a **distributed denial-of-service attack**.

In a distributed denial-of-service (DDoS) attack, multiple sources are used to start an attack that aims to make a website or application unavailable. This can come from a group of attackers, or even a single attacker. The single attacker can use multiple infected computers (also known as “bots”) to send excessive traffic to a website or application.

To help minimize the effect of DoS and DDoS attacks on your applications, you can use **AWS Shield**.

**AWS Shield**

AWS Shield is a service that protects applications against DDoS attacks. AWS Shield provides two levels of protection: **Standard and Advanced.**

**AWS Shield Standard**

**AWS Shield Standard** automatically protects all AWS customers at no cost. It protects your AWS resources from the most common, frequently occurring types of DDoS attacks.

As network traffic comes into your applications, AWS Shield Standard uses a variety of analysis techniques to detect malicious traffic in real time and automatically mitigates it.

**AWS Shield Advanced**

**AWS Shield Advanced** is a paid service that provides detailed attack diagnostics and the ability to detect and mitigate sophisticated DDoS attacks.

It also integrates with other services such as Amazon CloudFront, Amazon Route 53, and Elastic Load Balancing. Additionally, you can integrate AWS Shield with AWS WAF by writing custom rules to mitigate complex DDoS attacks.

**Module 6**

**Additional Security Services**

Once upon a time, there was a bustling coffee shop that prided itself on serving the best beans in town. But with the hustle and bustle of customers coming in and out, the shop owner realized it was time to step up security—not just for the beans, but for everything: the equipment, the money, and even the shop’s operations.

Just like securing a coffee shop, businesses on AWS need additional security measures to protect their precious resources, like data and applications. Let’s dive into how the coffee shop solved its problems—and how AWS can help your business do the same.

**Protecting the Beans: Encryption**

The first thing the shop owner did was to secure the coffee beans. At night, the storeroom door was locked with a special key. Only authorized staff could access it.

In the world of AWS, this concept is called **encryption**—a way to lock your data so only authorized parties can access it.

**Encryption at Rest:**

* Imagine the coffee beans sitting idle in the storeroom. Similarly, when your data is being stored but not actively used, it’s considered "at rest."
* AWS offers **server-side encryption** for services like DynamoDB to keep stored data secure. The key to unlock this data is managed by **AWS Key Management Service (KMS)**, just like the storeroom key.

**Encryption in Transit:**

* Now imagine the coffee beans being transported from one shop to another. Similarly, when data is moving from one place to another—like between a database and a client—it’s called "in transit."
* AWS protects data in transit using protocols like **SSL (Secure Sockets Layer)** and service certificates. Whether it's S3, RDS, or Redshift, AWS ensures the data stays secure during the journey.

**Inspecting the Shop: Amazon Inspector**

Next, the shop owner decided to do a full security inspection to check for weak locks, potential intruders, and other risks.

AWS has a similar tool called **Amazon Inspector**, which acts like a security inspector for your applications.

**Here’s how it works:**

* It automatically checks your infrastructure for vulnerabilities, misconfigurations, and deviations from security best practices.
* For example, Inspector might find an EC2 instance that’s not properly secured and provide recommendations on how to fix it.

Just like a detailed inspection report, the findings are displayed in the **Inspector console**, complete with steps to improve security.

**Detecting Threats: Amazon GuardDuty**

One day, the shop owner noticed suspicious activity around the shop—a customer loitering too long near the cash register. To stay vigilant, the shop installed cameras and motion detectors to monitor everything.

In AWS, this job is handled by **Amazon GuardDuty**, a powerful tool for **threat detection**.

* GuardDuty analyzes logs and metadata from services like **CloudTrail**, **VPC Flow Logs**, and **DNS Logs** to detect suspicious activity.
* It uses machine learning and threat intelligence to spot malicious IP addresses or unusual patterns.

The best part? GuardDuty works independently of your main services, so it doesn’t slow down your operations.

**More Tools in the Security Toolbox**

The shop owner didn’t stop there. They explored other tools to ensure top-notch security, like safes for money and alarms for emergencies. Similarly, AWS offers a variety of other security services:

* **AWS Shield Advanced:** A powerful tool for protecting against DDoS attacks.
* **AWS Security Hub:** A central place to monitor and manage all your security findings.

**The Coffee Shop’s Happy Ending**

With all these measures in place, the coffee shop owner could sleep peacefully, knowing the beans, equipment, and money were secure. And just like that, businesses using AWS can rest easy, knowing their data and applications are safe with tools like encryption, Amazon Inspector, and GuardDuty.

The moral of the story? Just like a coffee shop, every business needs strong security measures. With AWS, you have the right tools to protect your operations and keep your business running smoothly. ☕✨

**AWS Key Management Service (AWS KMS)**

The coffee shop has many items, such as coffee machines, pastries, money in the cash registers, and so on. You can think of these items as data. The coffee shop owners want to ensure that all of these items are secure, whether they’re sitting in the storage room or being transported between shop locations.

In the same way, you must ensure that your applications’ data is secure while in storage **(encryption at rest)** and while it is transmitted, known as **encryption in transit**.

**AWS Key Management Service** enables you to perform encryption operations through the use of **cryptographic keys**. A cryptographic key is a random string of digits used for locking (encrypting) and unlocking (decrypting) data. You can use AWS KMS to create, manage, and use cryptographic keys. You can also control the use of keys across a wide range of services and in your applications.

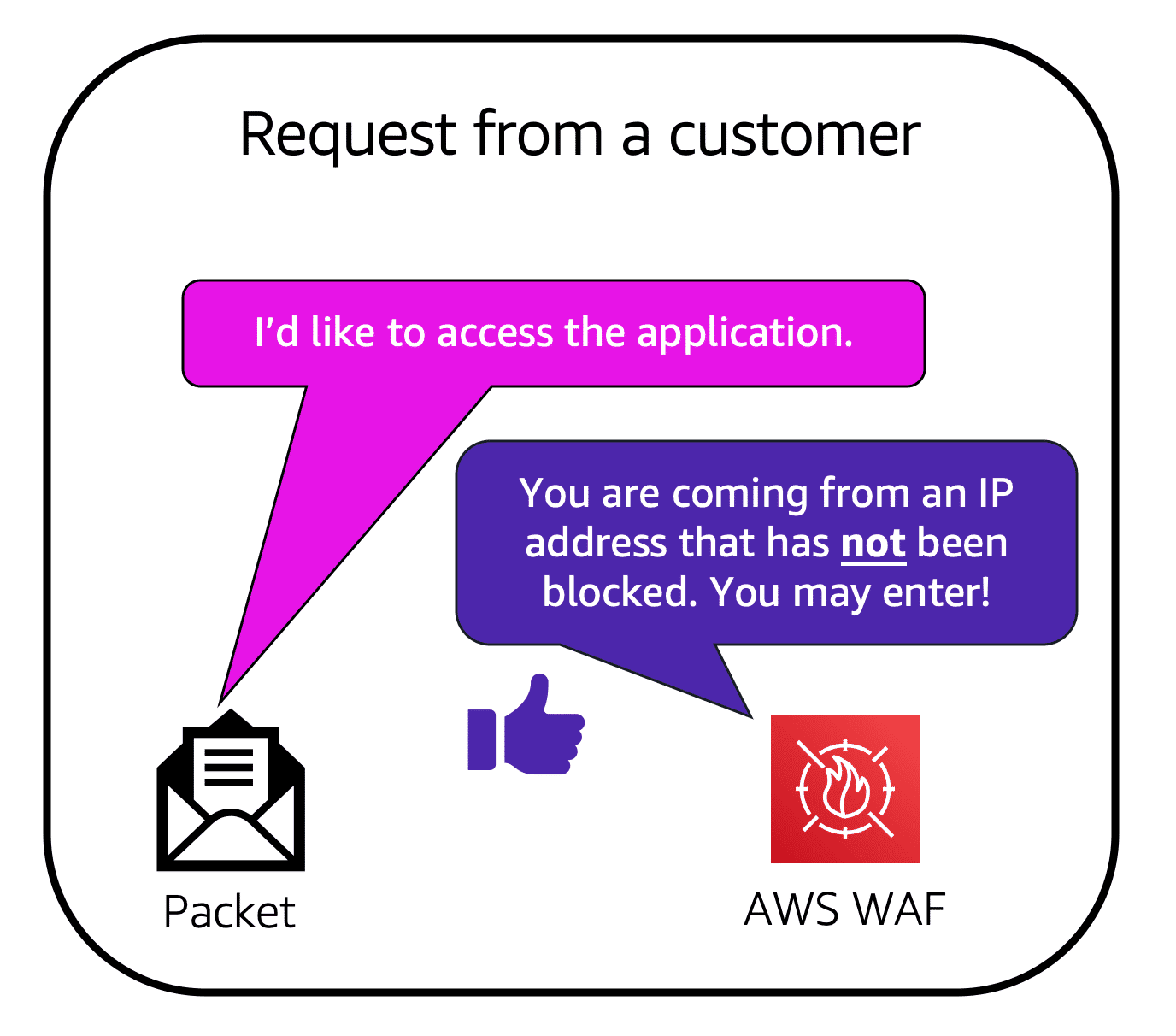
With AWS KMS, you can choose the specific levels of access control that you need for your keys. For example, you can specify which IAM users and roles are able to manage keys. Alternatively, you can temporarily disable keys so that they are no longer in use by anyone. Your keys never leave AWS KMS, and you are always in control of them.

**AWS WAF**

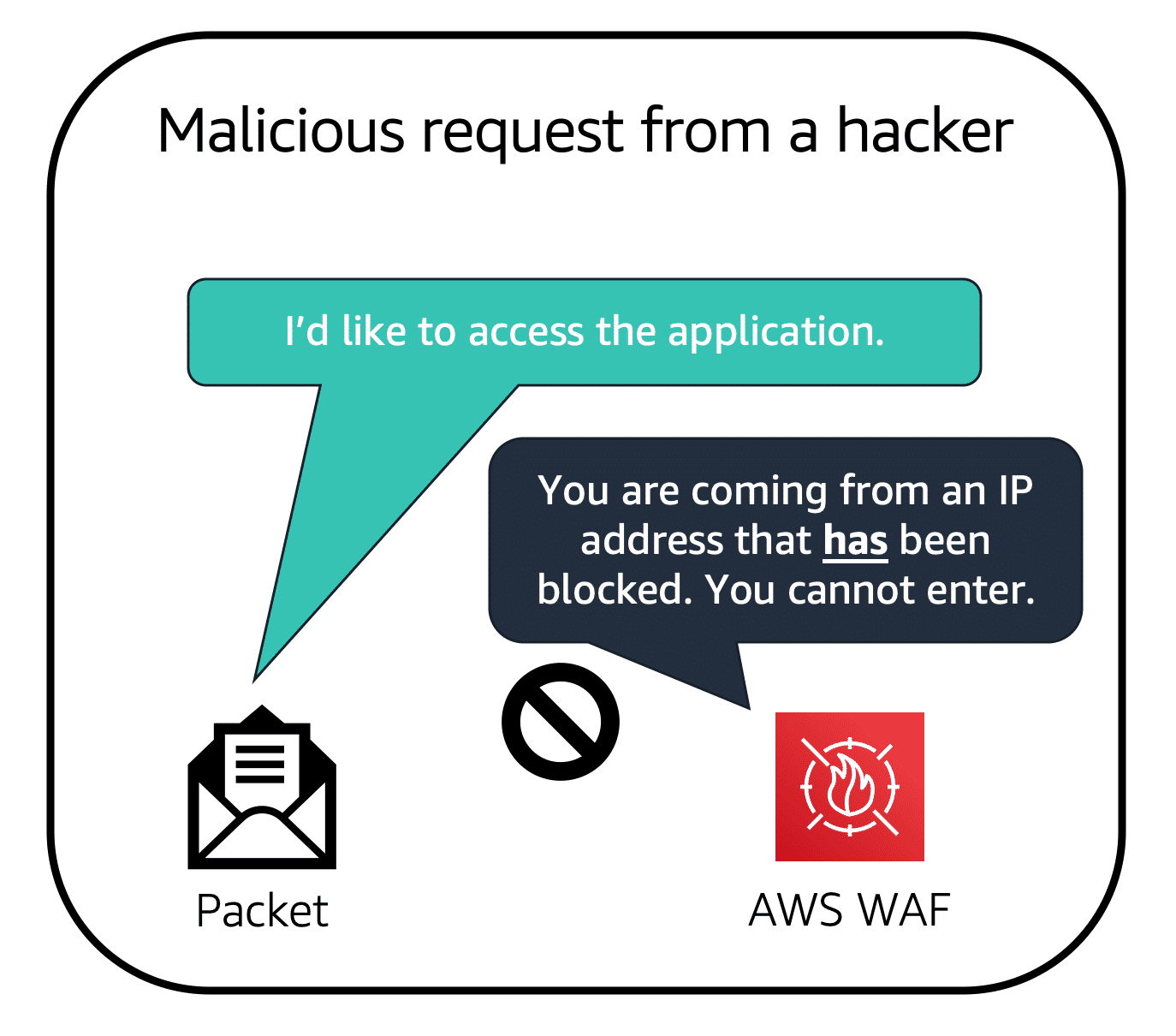
**AWS WAF** is a web application firewall that lets you monitor network requests that come into your web applications.

AWS WAF works together with Amazon CloudFront and an Application Load Balancer. Recall the network access control lists that you learned about in an earlier module. AWS WAF works in a similar way to block or allow traffic. However, it does this by using a **web access control list (ACL)** to protect your AWS resources.

**Here’s an example of how you can use AWS WAF to allow and block specific requests.**

Suppose that your application has been receiving malicious network requests from several IP addresses. You want to prevent these requests from continuing to access your application, but you also want to ensure that legitimate users can still access it. You configure the web ACL to allow all requests except those from the IP addresses that you have specified.

When a request comes into AWS WAF, it checks against the list of rules that you have configured in the web ACL. If a request does not come from one of the blocked IP addresses, it allows access to the application.



However, if a request comes from one of the blocked IP addresses that you have specified in the web ACL, AWS WAF denies access.

**Amazon Inspector**

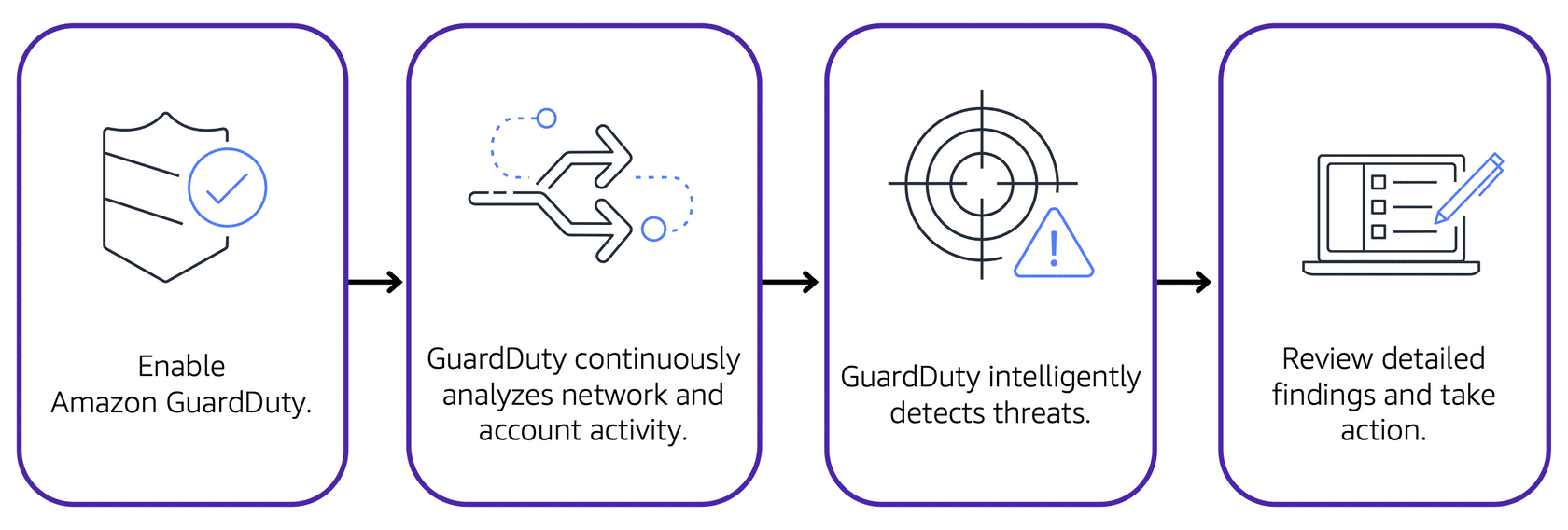
Suppose that the developers at the coffee shop are developing and testing a new ordering application. They want to make sure that they are designing the application in accordance with security best practices. However, they have several other applications to develop, so they cannot spend much time conducting manual assessments. To perform automated security assessments, they decide to use Amazon Inspector.

Amazon Inspector helps to improve the security and compliance of applications by running automated security assessments. It checks applications for security vulnerabilities and deviations from security best practices, such as open access to Amazon EC2 instances and installations of vulnerable software versions.

After Amazon Inspector has performed an assessment, it provides you with a list of security findings. The list prioritizes by severity level, including a detailed description of each security issue and a recommendation for how to fix it. However, AWS does not guarantee that following the provided recommendations resolves every potential security issue. Under the shared responsibility model, customers are responsible for the security of their applications, processes, and tools that run on AWS services.

**Amazon GuardDuty**

**Amazon GuardDuty** is a service that provides intelligent threat detection for your AWS infrastructure and resources. It identifies threats by continuously monitoring the network activity and account behavior within your AWS environment.



After you have enabled GuardDuty for your AWS account, GuardDuty begins monitoring your network and account activity. You do not have to deploy or manage any additional security software. GuardDuty then continuously analyzes data from multiple AWS sources, including VPC Flow Logs and DNS logs.

If GuardDuty detects any threats, you can review detailed findings about them from the AWS Management Console. Findings include recommended steps for remediation. You can also configure AWS Lambda functions to take remediation steps automatically in response to GuardDuty’s security findings.

**Module 6**

**Summary**

In Module 6, you learned about the following concepts:

* The shared responsibility model
* Features of AWS Identity and Access Management
* Methods of managing multiple accounts in AWS Organizations
* AWS compliance resources
* AWS services for application security and encryption

Once upon a time, there was an ambitious entrepreneur who decided to take their thriving coffee shop business to the cloud. The journey wasn’t just about brewing the perfect digital espresso—it was also about keeping everything secure. So, here’s how they mastered the art of cloud security with AWS, step by step.

**The Shared Responsibility Partnership**

Imagine the coffee shop’s building. The landlord ensures the roof, walls, and locks are sturdy (just like AWS ensures the security *of* the cloud infrastructure). But inside the shop, it’s up to the owner to protect the beans, the cash register, and the recipes (similar to securing resources *in* the cloud).

This is the **shared responsibility model**: AWS secures the cloud infrastructure, and you secure everything you build and store in it.

**The Security Guards: IAM**

To manage access, the shop owner hired security guards with specific roles:

1. **Users** are like employees logging in for their shifts—they have usernames and passwords. By default, they can’t touch anything until they’re granted permissions.
2. **Groups** are teams, like baristas or cashiers, with similar permissions.
3. **Roles** are temporary uniforms employees can "wear" to do specialized tasks (like fixing a machine).

Permissions are set through **policies**, which are like rules that either allow or deny specific actions.

But wait—there’s more! The shop owner wanted to allow their staff to use one badge to access both the shop and the delivery warehouse. This is **identity federation**, where existing credentials (like corporate logins) can be used across systems, including AWS.

Pro Tip: **Multi-factor authentication (MFA)** is like adding a second lock to the shop’s door, ensuring the keys alone aren’t enough to break in. Especially important for the "root user," which has the keys to everything.

**Expanding the Franchise: AWS Organizations**

As the business grew, the owner opened multiple coffee shops in different areas. Each shop had its own manager, but they were all part of one parent company. This setup mirrored **AWS Organizations**, which helps businesses manage multiple accounts. These accounts could represent different workloads, teams, or applications, all organized hierarchically for better control.

**Staying Compliant: Following the Rules**

The shop owner had to follow local health and safety laws. Similarly, businesses using AWS must comply with industry regulations, like GDPR for data protection or HIPAA for healthcare.

AWS makes compliance easier by:

* Offering **AWS Artifact**, a tool to access compliance documents.
* Providing information in the **AWS Compliance Center** to ensure businesses meet their specific requirements.

**Defending Against Attacks: DDoS Protection**

One day, troublemakers tried to flood the shop with fake customers, blocking real ones from getting in. This was a **DDoS attack** in action.

Thankfully, the shop owner had some powerful defenses:

* **Security Groups**: Only real customers with proper requests could enter.
* **Elastic Load Balancer (ELB)**: Handled customer requests efficiently, even during a rush.
* **AWS Shield and WAF**: Advanced tools to detect and block bad actors.

**Protecting the Beans: Encryption**

The coffee shop secured its beans both in the storeroom (data **at rest**) and during delivery between shops (data **in transit**). In AWS, encryption works similarly:

* **At Rest**: Data is locked up using encryption keys managed by services like **AWS KMS**.
* **In Transit**: Data traveling between systems is secured with protocols like **SSL**.

**The Final Takeaway: Security First**

At the end of the day, the shop owner learned a few golden rules for security:

1. **Use least privilege**: Only give employees (or users) access to what they need, nothing more.
2. **Encrypt everything**: Keep your data secure, whether it’s sitting still or on the move.
3. **Leverage AWS tools**: From IAM to Shield, AWS offers a full toolkit for protecting your environment.

And just like the shop owner, you can build a thriving, secure business in the cloud by following AWS’s best practices. Security isn’t just a priority—it’s the foundation of success! ☕✨

**Module 6**

**Quiz**

|  |
| --- |
| Which statement best describes an IAM policy?   * An authentication process that provides an extra layer of protection for your AWS account * **A document that grants or denies permissions to AWS services and resources** * An identity that you can assume to gain temporary access to permissions * The identity that is established when you first create an AWS account |

|  |
| --- |
| An employee requires temporary access to create several Amazon S3 buckets. Which option would be the best choice for this task?   * AWS account root user * IAM group * **IAM role** * Service control policy (SCP) |

|  |
| --- |
| Which statement best describes the principle of least privilege?   * Adding an IAM user into at least one IAM group * Checking a packet’s permissions against an access control list * **Granting only the permissions that are needed to perform specific tasks** * Performing a denial of service attack that originates from at least one device |

|  |
| --- |
| Which service helps protect your applications against distributed denial-of-service (DDoS) attacks?   * Amazon GuardDuty * Amazon Inspector * AWS Artifact * **AWS Shield** |

|  |
| --- |
| Which task can AWS Key Management Service (AWS KMS) perform?   * Configure multi-factor authentication (MFA). * Update the AWS account root user password. * **Create cryptographic keys.** * Assign permissions to users and groups. |