

1. The application will be constructed within a secure environment facilitated by Amazon Virtual Private Cloud (VPC), ensuring that the network is isolated and inaccessible from the public internet, thereby enhancing security.
2. **Backend code hosting** will be facilitated by **Amazon Elastic Compute Cloud (EC2),** a service that offers virtual machines on the AWS platform. This allows for flexible and scalable deployment of application logic and processes.
3. Employee data will be **securely stored within a database** using **Amazon Relational Database Service (RDS).** RDS simplifies the setup, operation, and scaling of a relational database, providing high availability and durability for critical data.
4. Images associated with employees will be stored using Amazon **Simple Storage Service (S3).** S3 provides **unlimited storage capacity** for various types of files, ensuring efficient management and retrieval of image data.
5. Monitoring of the application's performance and health will be conducted through Amazon **CloudWatch.** CloudWatch enables **real-time monitoring** and logging of various metrics, allowing for proactive management and troubleshooting.
6. The application will be designed to **handle scalability and fault tolerance** using Amazon **Elastic Load Balancing and Amazon EC2 Auto Scaling**. These services distribute incoming traffic across multiple instances and automatically adjust capacity to maintain steady performance, even during fluctuations in demand.
7. Security and access control will be managed using **Amazon Identity and Access Management (IAM)**. IAM allows for the creation and management of user identities, access policies, and permissions, ensuring that only authorized users can access resources.
8. The application will be developed step by step using the AWS Management Console, providing a user-friendly interface for configuring and managing AWS services.
9. The architecture of the application will be designed with flexibility in mind, allowing for modifications and adaptations to meet evolving requirements and technological advancements.
10. Throughout the course, informational popups will be available to provide additional guidance, tips, best practices, and commentary. These popups will feature Meowzy and Fluffy, offering a friendly and informative touch to the learning experience.

**The Six Benefits of Cloud Computing**

**Pay as you go.** Instead of investing in data centers and hardware before you know how you are going to use them, you pay only when you use computing resources, and pay only for how much you use.

**Benefit from massive economies of scale.** By using cloud computing, you can achieve a lower cost than you can get on your own. Because usage from hundreds of thousands of customers is aggregated in the cloud, AWS can achieve higher economies of scale, which translates into lower pay as-you-go prices.

**Stop guessing capacity**. Eliminate guessing on your infrastructure capacity needs. When you make a capacity decision prior to deploying an application, you often end up either sitting on expensive idle resources or dealing with limited capacity. With cloud computing, these problems go away. You can access as much or as little capacity as you need, and scale up and down as required with only a few minutes notice.

**Increase speed and agility.** IT resources are only a click away, which means that you reduce the time to make those resources available to your developers from weeks to just minutes. This results in a dramatic increase in agility for the organization since the cost and time it takes to experiment and develop is significantly lower.

**Stop spending money running and maintaining data centers.** Focus on projects that differentiate your business, not the infrastructure. Cloud computing lets you focus on your customers, rather than on the heavy lifting of racking, stacking, and powering physical infrastructure. This is often referred to as undifferentiated heavy lifting.

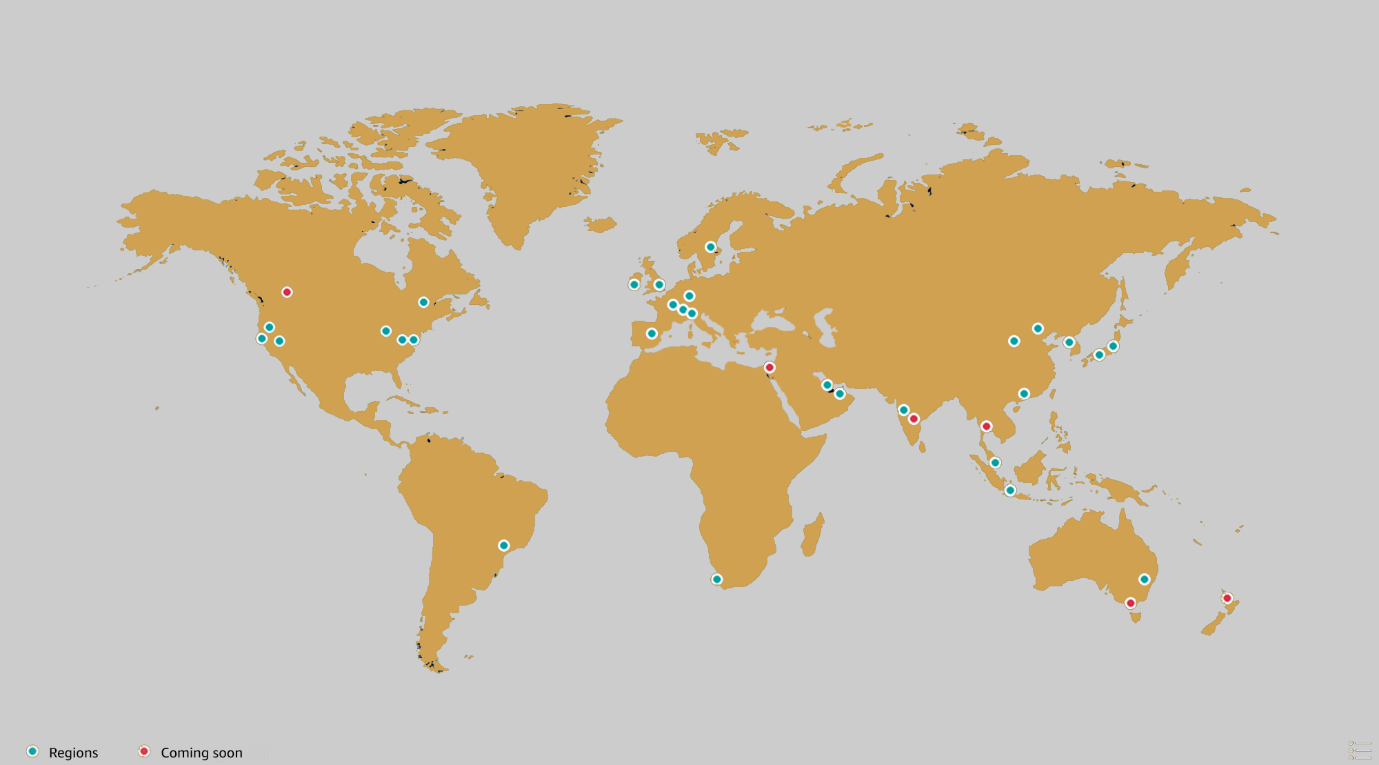
**Go global in minutes.** Easily deploy your application in multiple Regions around the world with just a few clicks. This means you can provide lower latency and a better experience for your customers at a minimal cost.

1. **Importance of storing employee photos securely:** In the passage, the need to securely store employee photos is emphasized due to the risk of losing them if they are stored on a single device, such as a laptop. To mitigate this risk, the decision is made to upload the photos to AWS (Amazon Web Services) for safekeeping. By doing so, the photos can be accessed from anywhere, providing flexibility and ensuring that even if the laptop is damaged or lost, the photos remain safe.
2. **AWS redundancy planning:** AWS ensures the safety and availability of data through redundancy measures. This involves organizing data centers into clusters globally, which are further divided into Availability Zones (AZs) and regions. These redundant structures ensure that if one data center or AZ experiences a failure, there are backups in place to maintain service continuity. This planning includes considerations for natural disasters and other unexpected events, ensuring that data remains accessible even in adverse circumstances.
3. **Factors to consider when choosing an AWS region:** The passage outlines several key factors to consider when selecting an AWS region.
   * **Compliance:** Compliance requirements, such as regulatory obligations or company policies, may dictate the choice of region. For example, data may be required to reside within specific geographical boundaries to comply with legal or regulatory frameworks.
   * **Latency:** The proximity of IT resources to the user base impacts the speed at which data can be accessed. Placing infrastructure closer to users reduces latency, resulting in faster response times and improved user experience.
   * **Pricing:** Costs for AWS services can vary across regions due to factors such as tax structures and operating expenses. Considering pricing implications is important to optimize costs and ensure the chosen region aligns with budgetary constraints.
   * **Service availability:** Not all AWS services may be available in every region immediately. It's essential to ensure that the desired services are supported in the chosen region to avoid compatibility issues and limitations.
4. **Overview of AWS global infrastructure:** AWS maintains a complex global infrastructure with a nested structure comprising data centers within AZs, and AZs within regions. This hierarchical arrangement provides redundancy and scalability, enabling AWS to deliver reliable services to customers worldwide. Additionally, AWS employs optimization techniques, such as the Global Edge Network, which utilizes Edge locations and regional Edge caches to reduce latency and improve performance for global users.
5. **Importance of caching content using Edge locations:** Caching content at Edge locations is crucial for optimizing performance and reducing latency, particularly for users located far from the hosting region. By caching frequently accessed content at Edge locations, AWS can deliver content from the nearest location to users, minimizing the time required for data retrieval. This significantly improves the user experience by reducing delays and ensuring fast access to content. Services like Amazon CloudFront facilitate content caching using Edge locations, further enhancing the efficiency of content delivery across the AWS network.

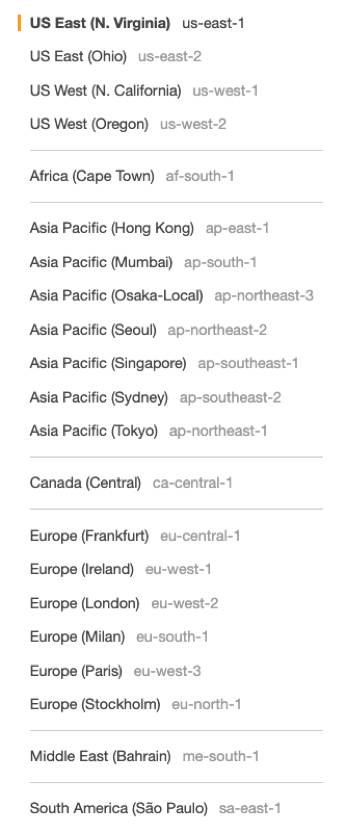
Top of Form

Infrastructure, like data centers and networking connectivity, still exists as the foundation of every cloud application. In AWS, this physical infrastructure makes up the AWS Global Infrastructure, in the form of Availability Zones and Regions.

**REGIONS**



Regions are geographic locations worldwide where AWS hosts its data centers. AWS Regions are named after the location where they reside. For example, in the United States, there is a Region in Northern Virginia called the Northern Virginia Region and a Region in Oregon called the Oregon Region. There are Regions in Asia Pacific, Canada, Europe, the Middle East, and South America, and AWS continues to expand to meet the needs of its customers.Each AWS Region is associated with a geographical name and a Region code.



Here are a few examples of Region codes:

* us-east-1: This is the first Region created in the east of the US. The geographical name for this Region is N. Virginia.
* ap-northeast-1: The first Region created in the northeast of Asia Pacific. The geographical name for this Region is Tokyo.

AWS Regions are independent from one another. This means that your data is not replicated from one Region to another, without your explicit consent and authorization.

**CHOOSE THE RIGHT AWS REGION**

Consider four main aspects when deciding which AWS Region to host your applications and workloads: latency, price, service availability, and compliance.

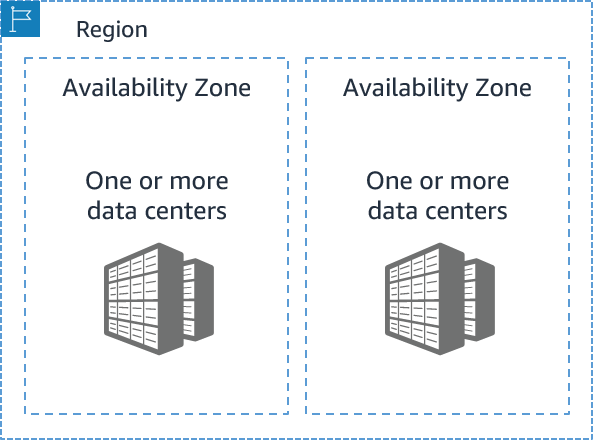
**Latency.** If your application is sensitive to latency, choose a Region that is close to your user base. This helps prevent long wait times for your customers. Synchronous applications such as gaming, telephony, WebSockets, and IoT are significantly affected by higher latency, but even asynchronous workloads, such as ecommerce applications, can suffer from an impact on user connectivity.

**Price.** Due to the local economy and the physical nature of operating data centers, prices may vary from one Region to another. The pricing in a Region can be impacted by internet connectivity, prices of imported pieces of equipment, customs, real estate, and more. Instead of charging a flat rate worldwide, AWS charges based on the financial factors specific to the location.

**Service availability.** Some services may not be available in some Regions. The AWS documentation provides a table containing the Regions and the available services within each one.

**Data compliance.** Enterprise companies often need to comply with regulations that require customer data to be stored in a specific geographic territory. If applicable, you should choose a Region that meets your compliance requirements.

**AVAILABILITY ZONES**



Inside every Region is a cluster of Availability Zones (AZ). An AZ consists of one or more data centers with **redundant power, networking, and connectivity**. These data centers operate in discrete facilities with undisclosed locations. They are connected using redundant high-speed and low-latency links.AZs also have a code name. Since they’re located inside Regions, they can be addressed by appending a letter to the end of the Region code name. For example:

* us-east-1a: an AZ in us-east-1 (Northern Virginia Region)
* sa-east-1b: an AZ in sa-east-1 (São Paulo Region in South America)

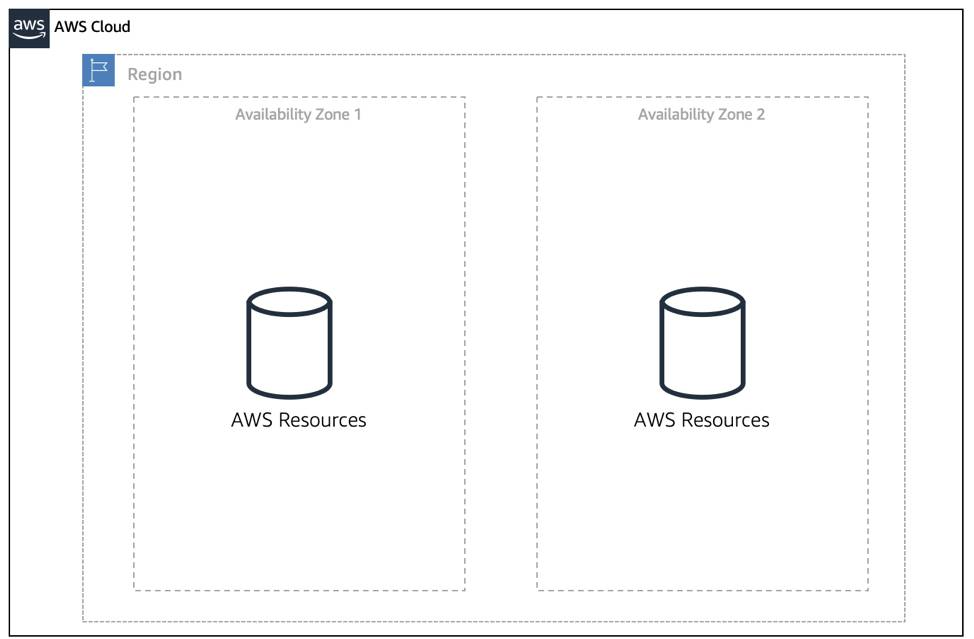
If you see that a resource exists in us-east-1c, you know this resource is located in AZ c of the us-east-1 Region.

**SCOPE AWS SERVICES**

Depending on the AWS Service you use, your resources are either deployed at the AZ, Region, or Global level. Each service is different, so you need to understand how the scope of a service may affect your application architecture.When you operate a Region-scoped service, you only need to select the Region you want to use. If you are not asked to specify an individual AZ to deploy the service in, this is an indicator that the service operates on a Region-scope level. For region-scoped services, AWS **automatically performs actions to increase data durability and availability.On the other hand, some services ask you to specify an AZ**. With these services, you are often responsible for increasing the **data durability and high availability of these resources.**

**MAINTAIN RESILIENCY**

To keep your application available, you need to maintain high availability and resiliency. A well-known best practice for cloud architecture is to use Region-scoped, managed services. These services come with availability and resiliency built in. When that is not possible, make sure the workload is replicated across multiple AZs. At a minimum, you should use two AZs. If one entire AZ fails, your application will have infrastructure up and running in at least a second AZ to take over the traffic.



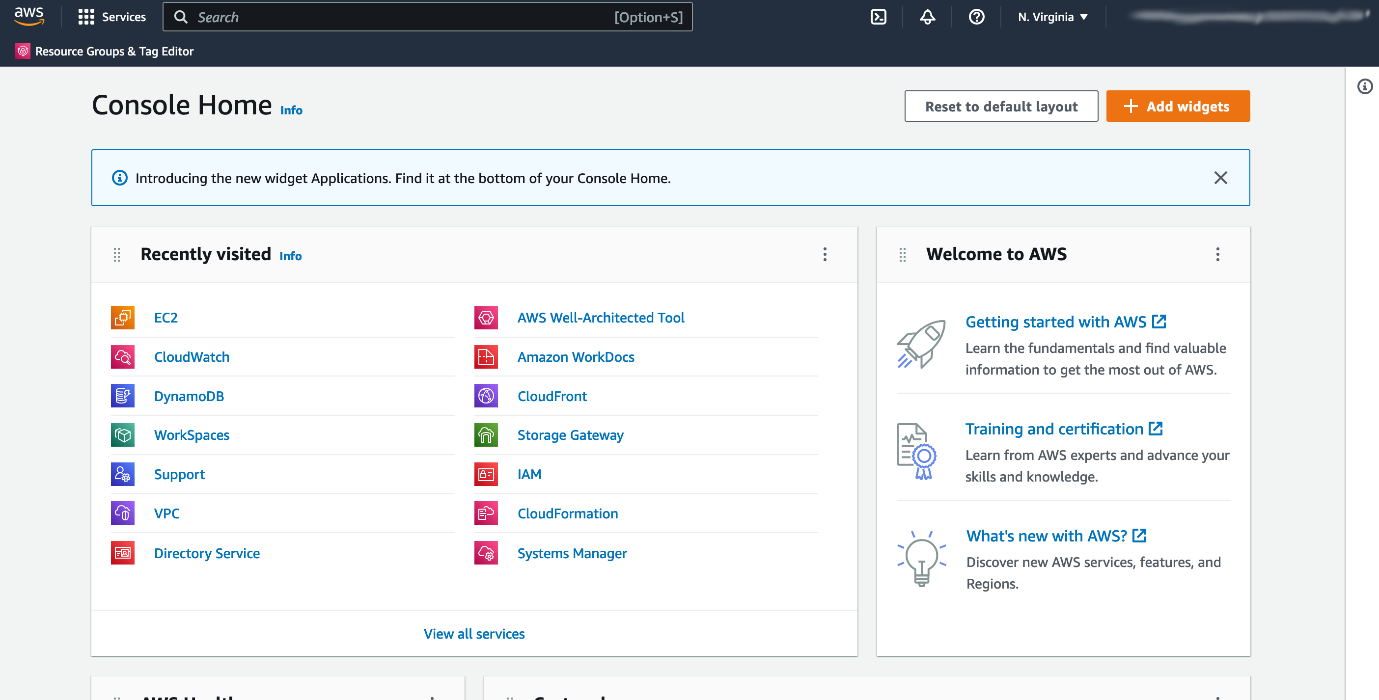
1. **Understanding infrastructure is easier when physically owned and managed:** When you physically own and manage infrastructure, such as having a server in your closet, it's more straightforward to comprehend how it operates because you can directly interact with it. You can physically touch it, see it, and manage it at every level, which provides a tangible understanding of its workings.
2. **Transition to virtual infrastructure requires logical management via APIs:** When infrastructure becomes virtual, it moves away from physical hardware to abstracted resources managed through software. This transition necessitates a shift in management approach towards logical management through Application Programming Interfaces (APIs). Instead of directly interacting with physical components, interactions occur through API calls to control virtual resources.
3. **Three main methods to interact with AWS: Management Console, Command Line Interface (CLI), and Software Development Kits (SDKs):** AWS offers various tools for interacting with its services. The three primary methods are:
   * **Management Console:** A web-based interface that allows users to manage AWS resources through a graphical user interface (GUI). It's user-friendly and **suitable for beginners.**
   * **Command Line Interface (CLI):** A scripting tool that enables users to interact with AWS services through the command line. It provides more control and efficiency, especially for repetitive tasks, but requires knowledge of syntax and commands.
   * **Software Development Kits (SDKs):** SDKs are available for popular programming languages and provide libraries and tools for developers to integrate AWS services into their applications. SDKs offer programmatic control over AWS services, allowing for more flexibility and customization.
4. **Management Console:**
   * **Web-based, point-and-click interface suitable for beginners:** The Management Console provides an intuitive web interface where users can easily navigate AWS services and perform actions through point-and-click interactions. This approach is ideal for users who are new to AWS and prefer a visual interface.
   * **Console organizes services into categories; allows regional selection:** Services within the Console are organized into categories, making it easier for users to find and access the services they need. Additionally, users can select different AWS regions to manage resources in specific geographic locations.
5. **Command Line Interface (CLI):**
   * **Scripting tool for API calls, reduces human error, requires understanding of syntax:** The CLI allows users to interact with AWS services through the command line, enabling automation and scripting of tasks. This reduces the potential for human error compared to manual interactions. However, users need to understand the syntax and structure of CLI commands to use it effectively.
   * **AWS Cloud Shell provides access to CLI via console:** AWS Cloud Shell is an integrated development environment (IDE) within the AWS Management Console that provides a browser-based shell pre-configured with AWS CLI and other tools. Users can access the CLI directly from the Console without installing any additional software.
6. **Software Development Kits (SDKs):**
   * **Available for popular programming languages, enables integration of application code with AWS services:** AWS SDKs are libraries and tools provided by AWS for various programming languages such as **Python, Java, Node.js,ruby etc**. Developers can use SDKs to integrate AWS services seamlessly into their applications, allowing for programmatic management and control of AWS resources.
   * **Example of using Python SDK to interact with AWS storage service for an employee directory application:** For instance, developers can use the Python SDK to write code that interacts with AWS storage services, such as Amazon S3, to store and retrieve data for applications like an employee directory.
7. **Using SDKs allows for programmatic management of AWS services, providing flexibility and power:** SDKs empower developers to programmatically manage AWS services, offering greater flexibility and customization compared to manual interactions or CLI commands. Developers can leverage programming constructs like conditions, loops, arrays, and more to automate tasks and integrate AWS services seamlessly into their applications.
8. **Recap: Three main options to connect with AWS are Console, CLI, and SDKs:** In summary, users have three primary options for interacting with AWS services: **Management Console for a graphical interface, CLI for command-line scripting, and SDKs for programmatic integration into applications.**
9. **Course mainly focuses on Console usage but encourages CLI usage for advanced users:** While the course primarily emphasizes using the Management Console for interacting with AWS services, it also encourages more advanced users to explore and utilize the CLI for greater control and efficiency in managing AWS resources.

**Reading 1.4: Interacting with AWS**

Every action you make in AWS is an API call that is authenticated and authorized. In AWS, you can make API calls to services and resources through the AWS Management Console, the AWS Command Line Interface (CLI), or the AWS Software Development Kits (SDKs).

**THE AWS MANAGEMENT CONSOLE**

One way to manage cloud resources is through the web-based console, where you log in and click on the desired service. This can be the easiest way to create and manage resources when you’re first begin working with the cloud. Below is a screenshot that shows the landing page when you first log into the AWS Management Console.



The services are placed in categories, such as **compute, database, storage and security, identity and compliance**.On the upper right corner is the Region selector. If you click it and change the Region, you will make requests to the services in the chosen Region. The URL changes, too. Changing the Region directs the browser to make requests to a whole different AWS Region, represented by a different subdomain.

**THE AWS COMMAND LINE INTERFACE (CLI)**

Consider the scenario where you run tens of servers on AWS for your application’s frontend. You want to run a report to collect data from all of these servers. You need to do this programmatically every day because the server details may change. Instead of manually logging into the AWS Management Console and copying/pasting information, you can schedule an AWS Command Line Interface (CLI) script with an API call to pull this data for you. The AWS CLI is a unified tool to manage AWS services. With just one tool to download and configure, you control multiple AWS services from the command line and automate them with scripts. The AWS CLI is open-source, and there are installers available for Windows, Linux, and Mac OS. Here is an example of running an API call against a service using the AWS CLI:

You get this response:

{

"Reservations": [

{"Groups": [],

"Instances": [

{"AmiLaunchIndex": 0,

and so on.

**AWS SOFTWARE DEVELOPMENT KITS (SDKS)**

API calls to AWS can also be performed by executing code with programming languages. You can do this by using AWS Software Development Kits (SDKs). SDKs are open-source and maintained by AWS for the most popular programming languages, such as **C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python, and Ruby**. Developers commonly use AWS SDKs to integrate their application source code with AWS services. Let’s say the frontend of the application runs in Python and every time it receives a cat photo, it uploads that **photo to a storage service**. This action can be achieved from within the source code by using the **AWS SDK for Python**.

Here is an example of code you can implement to work with AWS resources using the Python AWS SDK.

import boto3

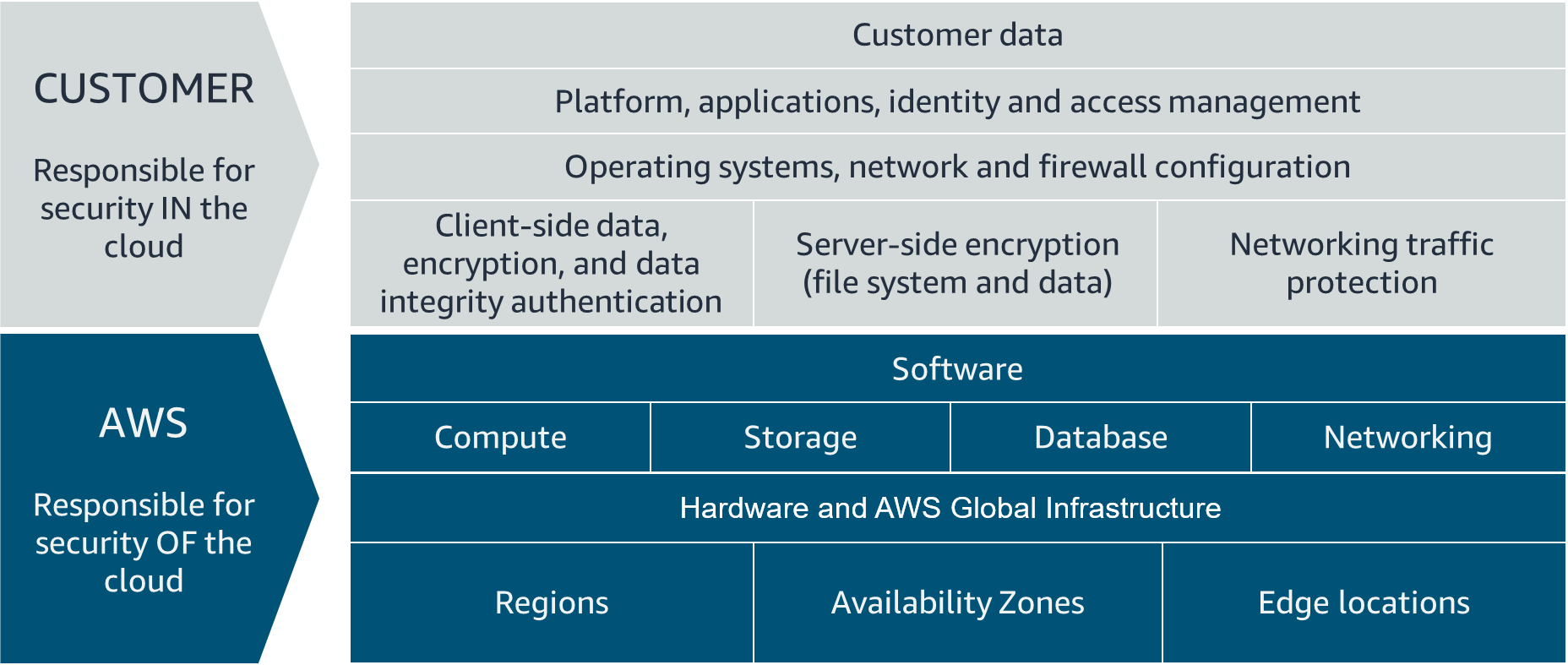
ec2 = boto3.client('ec2')

response = ec2.describe\_instances()

print(response)

**Reading 1.5: Security and the AWS Shared Responsibility Model**

When you begin working with the AWS Cloud, managing security and compliance is a shared responsibility between AWS and you. To depict this shared responsibility, AWS created the shared responsibility model. This distinction of responsibility is commonly referred to as security of the cloud, versus security in the cloud.



**WHAT IS AWS RESPONSIBLE FOR?**

AWS is responsible for security of the cloud. This means AWS is required to protect and secure the infrastructure that runs all the services offered in the AWS Cloud. AWS is responsible for:

* Protecting and securing AWS Regions, Availability Zones, and data centers, down to the physical security of the buildings
* Managing the hardware, software, and networking components that run AWS services, such as the physical server, host operating systems, virtualization layers, and AWS networking components

The level of responsibility AWS has depends on the service. AWS classifies services into three different categories. The following table provides information about each, as well as the AWS responsibility.

| **Category** | **Examples of AWS Services in the Category** | **AWS Responsibility** |
| --- | --- | --- |
| Infrastructure services | Compute services, such as Amazon Elastic Compute Cloud (Amazon EC2) | AWS manages the underlying infrastructure and foundation services. |
| Container services | Services that require less management from the customer, such as Amazon Relational Database Service (Amazon RDS) | AWS manages the underlying infrastructure and foundation services, operating system, and application platform. |
| Abstracted services | Services that require very little management from the customer, such as Amazon Simple Storage Service (Amazon S3) | AWS operates the infrastructure layer, operating system, and platforms, as well as server-side encryption and data protection. |

**Note**

Container services refer to AWS abstracting application containers behind the scenes, not Docker container services. This enables AWS to move the responsibility of managing that platform away from customers.

**WHAT IS THE CUSTOMER RESPONSIBLE FOR?**

You’re responsible for security in the cloud. When using any AWS service, you’re responsible for properly configuring the service and your applications, as well as ensuring your data is secure.The level of responsibility you have depends on the AWS service. Some services require you to perform all the necessary security configuration and management tasks, while other more abstracted services require you to only manage the data and control access to your resources. Using the three categories of AWS services, you can determine your level of responsibility for each AWS service you use.

| **Category** | **AWS Responsibility** | **Customer Responsibility** |
| --- | --- | --- |
| Infrastructure services | AWS manages the infrastructure and foundation services. | You control the operating system and application platform, as well as encrypting, protecting, and managing customer data. |
| Container services | AWS manages the infrastructure and foundation services, operating system, and application platform. | You are responsible for customer data, encrypting that data, and protecting it through network firewalls and backups. |
| Abstracted services | AWS operates the infrastructure layer, operating system, and platforms, as well as server-side encryption and data protection. | You are responsible for managing customer data and protecting it through client-side encryption. |

Due to the varying level of effort, it’s important to consider which AWS service you use and review the level of responsibility required to secure the service. It’s also important to review how the shared security model aligns with the security standards in your IT environment, as well as any applicable laws and regulations.It’s important to note that you maintain complete control of your data and are responsible for managing the security related to your content. Here are some examples of your responsibilities in context.

* Choosing a Region for AWS resources in accordance with data sovereignty regulations
* Implementing data protection mechanisms, such as encryption and managing backups
* Using access control to limit who has access to your data and AWS resources

**What’s the Big Deal About Auth?**

When you’re configuring access to any account, two terms come up frequently: **authentication and authorization**. Though these terms may seem basic, you need to understand them to properly configure access management on AWS. It’s important to keep this mind as you progress in this course. Let’s define both terms.

**Understand Authentication**

When you create your AWS account, you use a combination of an email address and a password to verify your identity. If the user types in the correct email and password, the system assumes the user is allowed to enter and grants them access. This is the process of *authentication*.Authentication ensures that the user is who they say they are. Usernames and passwords are the most common types of authentication, but you may also work with other forms, such as token-based authentication or biometric data like a fingerprint. Authentication simply answers the question, “Are you who you say you are?”

**Understand Authorization**

Once you’re inside your AWS account, you might be curious about what actions you can take. This is where *authorization* comes in. Authorization is the process of giving users permission to access AWS resources and services. Authorization determines whether the user can perform an action—whether it be to read, edit, delete, or create resources. Authorization answers the question, “What actions can you perform?”

**What Is the AWS Root User?**

The email address you sign up with becomes the root user of the AWS account.

This root user has **unrestricted access to everything** in your account in most cases.

**Understand the AWS Root User Credentials**

The AWS root user has two sets of credentials associated with it. One set of credentials is the email address and password used to create the account. This allows you to access the AWS Management Console. The second set of credentials is called access keys, which allow you to **make programmatic requests from the** [**AWS Command Line Interface (AWS CLI) or AWS API**](https://trailhead.salesforce.com/content/learn/modules/aws-cloud-technical-professionals)**.** Access keys consist of two parts:

* **An access key ID, for example, A2lAl5EXAMPLE**
* **A secret access key, for example, wJalrFE/KbEKxE**

Similar to a username and password combination, you need both the access key ID and secret access key to authenticate your requests via the AWS CLI or AWS API. Access keys should be managed with the same security as an email address and password.

**Follow Best Practices When Working with the AWS Root User**

Keep in mind that the root user has complete access to all AWS services and resources in your account, as well as your billing and personal information. Due to this, securely lock away the credentials associated with the root user and do not use the root user for everyday tasks. To ensure the safety of the root user:

* Choose a strong password for the root user.
* Never share your root user password or access keys with anyone.
* Disable or delete the access keys associated with the root user.
* Do not use the root user for administrative tasks or everyday tasks.

When is it OK to use the AWS root user? There are some tasks where it makes sense to use the AWS root user. Check out the links at the end of this section to read about them.

**Delete Your Keys to Stay Safe**

If you don't already have an access key for your AWS account root user, don't create one unless you absolutely need to. If you do have an access key for your AWS account root user and want to delete the keys:

1. Go to the [My Security Credentials page](https://console.aws.amazon.com/iam/home?#security_credential) in the AWS Management Console and sign in with the root user’s email address and password.
2. Open the Access keys section.
3. Under Actions, click **Delete**.
4. Click **Yes**.

**The Case for Multi-Factor Authentication**

When you create an AWS account and first log in to that account, you use single-factor authentication. Single-factor authentication is the simplest and most common form of authentication. It only requires one authentication method. In this case, you use a username and password to authenticate as the AWS root user. Other forms of single-factor authentication include a security pin or a security token.However, sometimes a user’s password is easy to guess.

For example, your coworker Bob’s password, IloveCats222, might be easy for someone who knows Bob personally to guess, because it’s a combination of information that is easy to remember and describes certain things about Bob (1. Bob loves cats, and 2. Bob’s birthday is February 22).

If a bad actor guessed or cracked Bob’s password through social engineering, bots, or scripts, Bob might lose control of his account. Unfortunately, this is a common scenario that users of any website often face.

This is why using MFA has become so important in preventing unwanted account access. MFA requires two or more authentication methods to verify an identity, pulling from three different categories of information.

* Something you know, such as a username and password, or pin number
* Something you have, such as a one-time passcode from a hardware device or mobile app
* Something you are, such as fingerprint or face scanning technology

Using a combination of this information enables systems to provide a layered approach to account access. Even though the first method of authentication, Bob’s password, was cracked by a malicious user, it’s very unlikely that a second method of authentication, such as a fingerprint, would also be cracked. This extra layer of security is needed when protecting your most sacred accounts, which is why it’s important to enable MFA on your AWS root user.

**Use MFA on AWS**

If you enable MFA on your root user, you are required to present **a piece of identifying information from both the something you know category and the something you have category.** e.

**Review Supported MFA Devices**

AWS supports a variety of MFA mechanisms, such as virtual MFA devices, hardware devices, and Universal 2nd Factor (U2F) security keys. For instructions on how to set up each method, check out the Resources section.

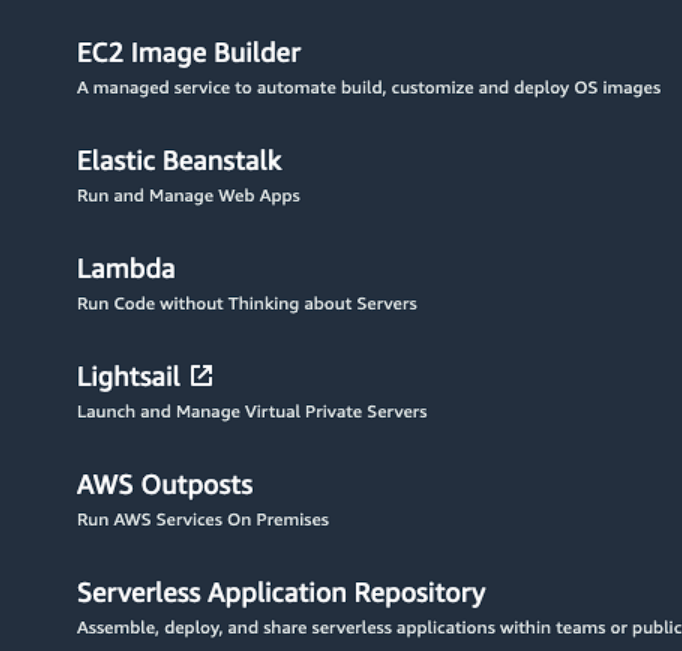
|  |  |  |
| --- | --- | --- |
| Device | Description | Supported Devices |
| Virtual MFA | A software app that runs on a phone or other device that provides a one-time passcode. Keep in mind that these applications can run on unsecured mobile devices, and because of that, may not provide the same level of security as hardware or U2F devices. | Authy, Duo Mobile, LastPass Authenticator, Microsoft Authenticator, Google Authenticator |
| Hardware | A hardware device, generally a key fob or display card device that generates a one-time six-digit numeric code | Key fob, display card |
| U2F | A hardware device that you plug into a USB port on your computer | YubiKey |

Certainly! Let's delve deeper into the key points extracted from the passage:

1. **Access Control and Credential Management**: The passage underscores the criticality of managing access to various components of an application. It highlights the need for controlling who can access what parts of the system and how they authenticate themselves to gain that access.
2. **Levels of Access Control**: Access control is not limited to just user login. It extends to interactions between different services within the application architecture, such as the **communication between the EC2 instance and the S3 storage service in the AWS environment**.
3. **Introduction of AWS IAM**: AWS Identity and Access Management (IAM) is introduced as a central solution for managing access and permissions within an AWS account. IAM enables the creation of unique users, each with their own set of permissions, thereby facilitating secure authentication and authorization processes.
4. **Authentication and Authorization**: The passage distinguishes between authentication and authorization. Authentication verifies the identity of users, ensuring they are who they claim to be, while authorization determines what actions users are permitted to perform within the system based on their permissions.
5. **IAM Policies**: IAM policies are described as **JSON-based documents that specify permissions for actions (API calls) and resources within the AWS environment**. Policies can be attached to **individual users or groups, allowing for fine-grained control over access to AWS resources.**
6. **Best Practices**: Best practices for managing AWS accounts are highlighted, including enabling Multi-Factor Authentication (MFA) for the root user, creating an IAM user with **administrative** permissions, and organizing users into groups for easier management and permission assignment.
7. **Role-Based Access**: Role-based access is introduced as a mechanism for granting temporary permissions to resources. **IAM roles allow entities such as EC2 instances to assume temporary permissions without the need for long-term credentials, enhancing security and reducing the risk of credential exposure**.
8. **You can’t apply policy to root user but you can to iam user**
9. **Future Learning**: The passage sets the stage for further exploration of IAM roles and their application in specific use cases, such as enabling EC2 instances to interact securely with S3 without requiring explicit user credentials.

Overall, the passage emphasizes the importance of robust access control and credential management practices in building and securing applications within the AWS ecosystem, with IAM serving as a foundational tool for implementing these practices effectively.

Top of Form

* Compute capacity is essential for virtually every business operation, encompassing a wide array of applications ranging from basic web servers to more complex tasks such as machine learning algorithms and HR management software.
* Establishing on-premises computing resources is a daunting task, requiring significant investments of time, money, and effort. This process involves meticulous research to determine the appropriate server specifications and quantities needed, followed by the upfront purchase of hardware. Additionally, there's a considerable wait period for server delivery, typically spanning weeks or even months. Subsequently, there's the labor-intensive process of installing and configuring the servers in a data center, ensuring they are properly secured and operational.
* Once the servers are in place, businesses face the challenge of utilization and scalability. Regardless of actual usage, they are committed to the investment, making it difficult to adapt to changing needs. Scaling up requires further investment and time, and recouping costs on unused servers is often impractical.
* Compute as a Service (CaaS) models, exemplified by AWS, alleviate many of these burdens. AWS takes care of the infrastructure, including data center facilities and server provisioning. This enables businesses to focus on their applications rather than the underlying hardware.
* Amazon EC2 stands out as a fundamental service within AWS, providing virtual machine instances for various computing needs. While EC2 is a key component, it's important to recognize that other options exist, such as container services and serverless compute.
* 
* Understanding the different compute options and their respective advantages is crucial for optimizing application architecture and resource allocation. Depending on the specific requirements and objectives of a project, choosing between virtual machines, containers, or serverless computing can significantly impact performance, scalability, and cost-effectiveness.

eading 2.2: Introduction to Amazon Elastic Compute Cloud

What Is Amazon EC2?

Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. It allows you to provision virtual servers called EC2 instances. Although AWS uses the phrase “web service” to describe it, it doesn’t mean that you are limited to running just web servers on your EC2 instances. You can create and manage these instances through the AWS Management Console, the AWS Command Line Interface (CLI), AWS Software Development Kits (SDKs), or through automation tools and infrastructure orchestration services.In order to create an EC2 instance, you need to define:

Hardware specifications, like CPU, memory, network, and storage

Logical configurations, like networking location, firewall rules, authentication, and the operating system of your choice.When launching an EC2 instance, the first setting you configure is which operating system you want by selecting an Amazon Machine Image (AMI).

What Is an AMI?

In the traditional infrastructure world, the process of spinning up a server consists of installing an operating system from installation disks, installation drives, or installation wizards over the network. In the AWS Cloud, this operating system installation is no longer your responsibility, and is instead built into the AMI that you choose.Not only does an AMI let you configure which operating system you want, you can also select storage mappings, the architecture type (such as 32-bit, 64-bit, or 64-bit ARM), and additional software installed.

What Is the Relationship Between AMIs and EC2 Instances?

EC2 instances are live instantiations of what is defined in an AMI, much like a cake is a live instantiation of a cake recipe. If you are familiar with software development, you can also see this kind of relationship between a Class and an Object.

A Class is something you model and define, while an object is something you interact with. In this case, the AMI is how you model and define your instance, while the EC2 instance is the entity you interact with, where you can install your web server, and serve your content to users.When you launch a new instance, AWS allocates a virtual machine that runs on a hypervisor. Then the AMI you selected is copied to the root device volume, which contains the image used to boot the volume. In the end, you get a server you can connect to and install packages and any additional software. In this case, you install a web server along with the properly configured source code of your employee directory app.

One advantage of using AMIs is that they are reusable.

You might choose a Linux-based AMI and configure the HTTP server, application packages, and any additional software you may need to run your application.

If you wanted to create a second EC2 instance with the same configurations, how can you easily do that? One option is to go through the entire instance creation and configuration process and try to match your settings to the first instance. However, this is time consuming and leaves room for human error.

The second, better option, is to create an AMI from your running instance and use this AMI to start a new instance. This way, your new instance will have all the same configurations as your current instance, because the configurations set in the AMIs are the same.

Where Can You Find AMIs?

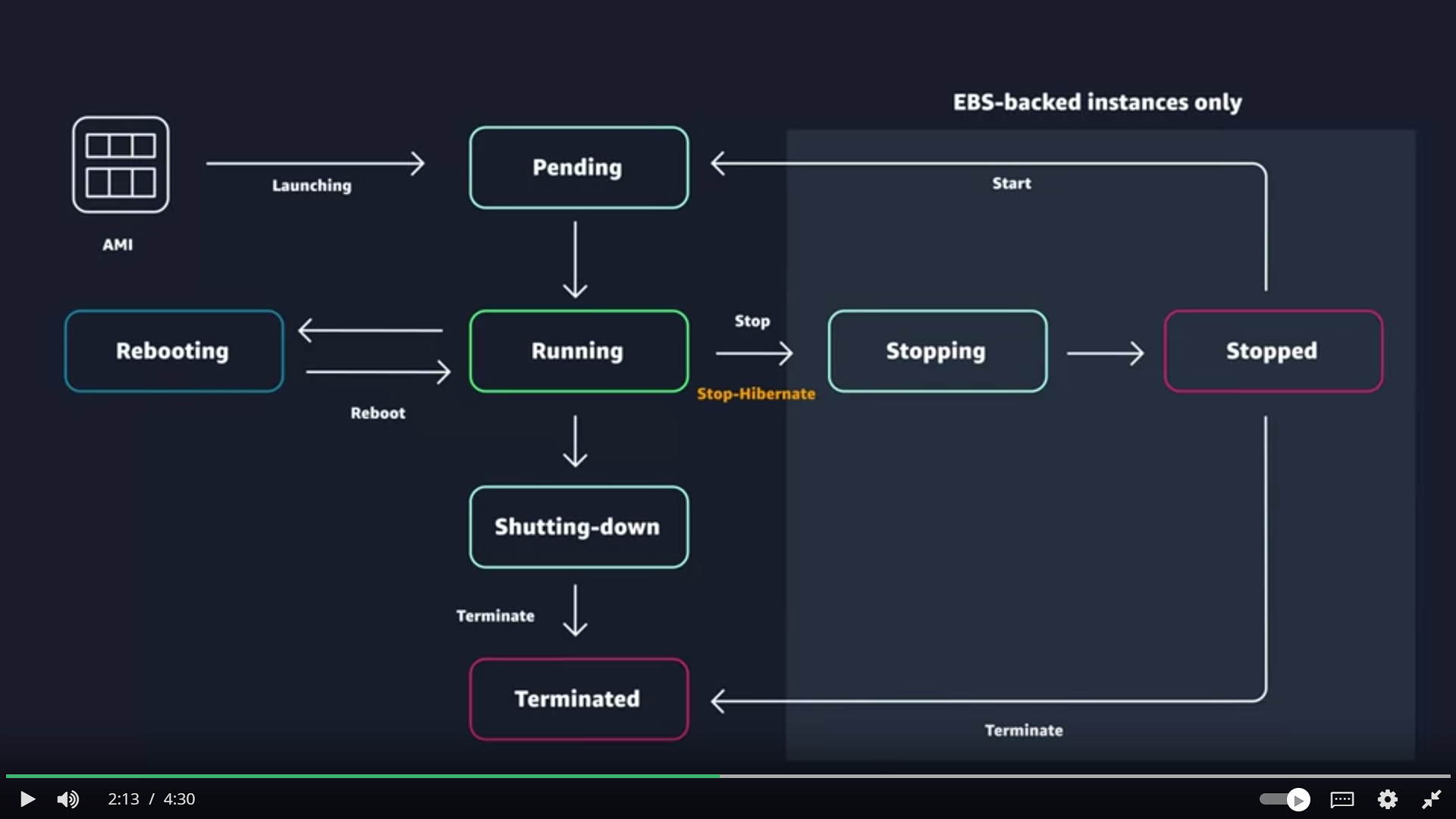
You can select an AMI from the following categories.

Quick Start AMIs that are premade by AWS and allow you to get started quickly.

AWS Marketplace AMIs that provide popular open source and commercial software from third-party vendors.My AMIs that are created from your EC2 instances Community AMIs that are provided by the AWS user community.

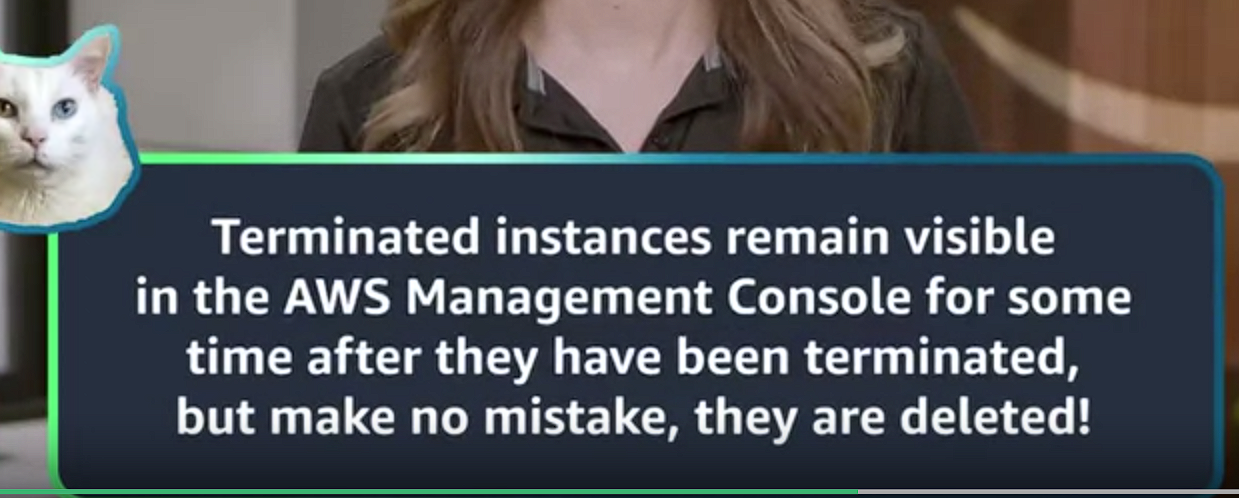
Build your own custom image with EC2 Image Builder.

Each AMI in the AWS Management Console has an AMI ID, which is prefixed by “ami-”, followed by a random hash of numbers and letters. These IDs are unique to each AWS region.



Sure, let's elaborate on each point:

1. **EC2 Provisioning and Disposal Value**:
   * EC2 (Elastic Compute Cloud) provides significant value because it allows users to provision new instances on-demand and dispose of them easily when they are no longer needed. This flexibility is crucial for businesses as it enables them to scale their resources according to their requirements, optimizing costs and performance.
2. **Charging Model Based on Instance State**:
   * Users are only charged for EC2 instances when they are in a running state. This means that if an instance is stopped or terminated, no charges are incurred, promoting cost-efficiency by ensuring that users only pay for the resources they actively use.
3. **Elastic Scaling**:
   * EC2's elasticity feature allows users to scale their fleet of instances in or out based on demand. This means that as demand for resources increases, more instances can be launched to handle the load, and as demand decreases, instances can be terminated to reduce costs.
4. **EC2 Instance Lifecycle**:
   * The lifecycle of an EC2 instance involves several states, including pending, running, stopping, stopped, and terminated. Understanding these states is essential for effectively managing EC2 instances and optimizing their usage.
5. **Options for Managing Instances**:
   * Users have various options for managing EC2 instances, including rebooting, stopping, stop-hibernating, and terminating them. Each option serves a specific purpose and allows users to control their instances according to their needs.
6. **Data Preservation Considerations**:
   * When terminating an EC2 instance, it's crucial to ensure that any important data or state stored on the instance is backed up. Once an instance is terminated, all data stored on it is lost unless it has been backed up elsewhere.
7. **Termination Protection**:
   * EC2 offers a termination protection feature that prevents instances from being terminated accidentally. This feature adds an extra layer of security for instances that need to be preserved.
8. **Persistent Storage Options**:
   * EC2 provides persistent storage options that allow users to store data independently of instances. This ensures that data remains accessible even if instances are terminated or replaced.
9. **Benefits of Terminating Instances**:
   * Terminating instances can be beneficial for troubleshooting or updating applications. Instead of fixing issues on existing instances, users can launch new instances with updates and terminate the old ones, ensuring a fresh and updated environment.
10. **Optimizing Costs**:
    * Launching and terminating instances based on demand helps optimize costs by ensuring that resources are only used when needed. This allows businesses to reduce expenses while maintaining performance.
11. **Costs Incurred Based on Instance State**:
    * Users are only charged for EC2 instances when they are in a running state or preparing to hibernate. Instances can be stopped when not in use to avoid unnecessary charges, making it cost-effective for intermittent workloads.



AWS offers a broad spectrum of compute offerings that give you the flexibility to choose the right tool for the right job. The three main categories of compute are *virtual machines*, *containers*, and *serverless*. There is no one-size-fits-all service because it depends on your needs.The key is to understand what each option has to offer in order to build a more appropriate cloud architecture for your use case. In this unit, you learn about containers and how to run them on AWS.Containers can host a variety of different workloads, including web applications, lift and shift migrations, distributed applications, and streamlining of development, test, and production environments.

**Reading 2.25: Amazon EC2 Instance Lifecycle**

Now that you know how to select an operating system for your EC2 instance, it’s time to choose other configurations to create your EC2 instance, such as the instance type, network, and storage. For an application like the employee directory application, you need instances with enough capacity to run web servers and process incoming customer requests. Your instance sizing will depend on both the demands of your application and the anticipated size of your user base. Forecasting server capacity for an on-premises application requires difficult decisions involving significant up-front capital spending, while changes to the allocation of your cloud-based services can be made with a simple API call. Because of AWS’s pay-as-you-go model, you can match your infrastructure capacity to your application’s demand, instead of the other way around.

**What Makes Up an EC2 Instance?**

**EC2 instances are a combination of virtual processors (vCPUs), memory, network, and in some cases, instance storage and graphics processing units (GPUs). When you create an EC2 instance, you need to choose how much you need of each of these components.**

AWS offers a variety of instances that differ based on performance. Some instances provide you with more capacity and others provide less. To get an overview of the capacity details for a particular instance, you should look at the instance type. Instance types consist of a **prefix identifying the type of workloads they’re optimized for, followed by a size. For example, the instance type c5.large can be broken down into the following elements.**

* **c5** determines the **instance family** and generation number. Here, the instance belongs to the fifth generation of instances in an instance family that’s optimized for generic computation.
* **large**, which determines the amount of instance capacity.

**What Are Instance Families?**

| **Instance Family** | **Description** | **Use Cases** |
| --- | --- | --- |
| General purpose | Provides a balance of compute, memory, and networking resources, and can be used for a variety of workloads. | Scale-out workloads such as web servers, containerized microservices, caching fleets, distributed data stores, and development environments. |
| Compute optimized | Ideal for compute-bound applications that benefit from high-performance processors. | High-performance web servers, scientific modeling, batch processing, distributed analytics, high-performance computing (HPC), machine/deep learning, ad serving, highly scalable multiplayer gaming. |
| Memory optimized | Designed to deliver fast performance for workloads that process large data sets in memory. | Memory-intensive applications such as high-performance databases, distributed web-scale in-memory caches, mid-size in-memory databases, real-time big-data analytics, and other enterprise applications. |
| Accelerated computing | Use hardware accelerators or co-processors to perform functions such as floating-point number calculations, graphics processing, or data pattern matching more efficiently than is possible with conventional CPUs. | 3D visualizations, graphics-intensive remote workstations, 3D rendering, application streaming, video encoding, and other server-side graphics workloads. |
| Storage optimized | Designed for workloads that require high, sequential read and write access to large data sets on local storage. They are optimized to deliver tens of thousands of low-latency random I/O operations per second (IOPS) to applications that replicate their data across different instances. | NoSQL databases, such as Cassandra, MongoDB, and Redis, in-memory databases, scale-out transactional databases, data warehousing, Elasticsearch, and analytics. |

**Where Does Your EC2 Instance Live?**

By default, your EC2 instances are placed in a network called the d**efault Amazon Virtual Private Cloud (VPC).** This network was created so that you can easily get started with Amazon EC2 without having to **learn how to create and configure a VPC.** Any resource you put inside the default VPC **will be public and accessible by the internet,** so you shouldn’t place any customer data or private information inside of it. Once you get more comfortable with networking on AWS, you should change this default setting to choose your own custom VPCs and restrict access with additional routing and connectivity mechanisms.

**Architect for High Availability**

Inside this network, your instance resides in an Availability Zone of your choice. AWS services that are scoped at the Availability Zone level must be architected with high availability in mind. While EC2 instances are typically reliable, two is better than one, and three is better than two. Specifying the instance size gives you an advantage when designing your architecture because you can use more smaller instances instead of a few larger ones. If your frontend only has a single instance and that instance fails, your application goes down. On the other hand, if your workload is distributed across 10 instances and one fails, you lose only 10 percent of your fleet and your application availability is hardly affected. When architecting any application for high availability, consider using at least two EC2 instances in two separate Availability Zones. ­­

**Explore the EC2 Instance Lifecycle**

An EC2 instance transitions between different states from the moment you create it all the way through to its termination.

When you launch an instance, it enters the **pending state** (1). When the instance is pending, **billing has not started**. At this stage, the instance is preparing to enter the running state. Pending is where AWS performs all actions needed to set up an instance, such as **copying the AMI content to the root device and allocating the necessary networking components**. When your instance is *running* (2), it's ready to use. This is also the stage **where billing begins**. As soon as an instance is running, you are then able to take other actions on the instance, such as ***reboot*, *terminate*, *stop*, and *stop-hibernate*.** When you reboot an instance (3), it’s different than performing a stop action and then a start action. **Rebooting an instance is equivalent to rebooting an operating system.** The instance remains on the same host computer and maintains its public and private IP address, and any data on its instance store. It typically takes a few minutes for the reboot to complete. When you stop and start an instance (4), your instance may be placed on a new underlying physical server. Therefore, you lose any data on the instance store that were on the previous host computer. When you stop an instance, the instance gets a new public IP address but maintains the same private IP address. When you *terminate* an instance (5), the instance store are erased, and you lose both the public IP address and private IP address of the machine. Termination of an instance means you can no longer access the machine.

**What Is the Difference Between Stop and Stop-Hibernate?**

When you stop your instance, it enters the *stopping* state, and then the *stopped* state. AWS does not charge usage or data transfer fees for your instance after you stop it, but storage for any Amazon EBS volumes is still charged. While your instance is in the stopped state, you can modify some attributes, like the instance type. When you stop your instance, the data stored in memory (RAM) is lost. When you *stop-hibernate* your instance, AWS signals the operating system to perform hibernation (suspend-to-disk), which saves the contents from the instance memory (RAM) to the Amazon EBS root volume. Consider a scenario where you build a standard three tier application, where you have web servers, application servers and database servers. Turns out, the application you built becomes extremely popular. To relieve some stress on the database that supports your application, you want to implement a custom backend layer that caches database information in memory (RAM). You decide to run this custom backend caching solution on Amazon EC2. In this scenario, the stop-hibernate feature would be instrumental in persisting storage. It would prevent you from having to manually create scripts to save this RAM data before shutting down the server.

**What Makes Up the Pricing?**

To understand EC2 pricing, let’s decouple the instance price from other services attached to it, such as storage and networking costs. In this unit we refer to the instance cost as the cost associated with the instance in terms of specifications and not the total blended cost of running an instance. Once an instance is launched in your AWS account, the billing usually accrues on a per-second basis. For simplicity of calculation, prices are stated per-hour. For example, if you have an instance running for 5 minutes and 38 seconds during a given month, you only pay for 338 seconds of utilization at the end of the month. One exception to this pricing convention may be third-party AMIs purchased from the AWS Marketplace, which may have a minimum billing of 1 hour. For more details, check out the resources section of this unit.

**What Are the EC2 Pricing Options?**

One of the ways to reduce costs with Amazon EC2 is to choose the right pricing option for the way your applications run. There are three main purchasing options for EC2 instances: on-demand, reserved, and spot instances.

**Pay As You Go with On-Demand Instances**

With *On-Demand* instances, you pay for compute capacity with no long-term commitments. Billing begins whenever the instance is running, and billing stops when the instance is in a stopped or terminated state. The price per second for a running On-Demand instance is fixed. For applications that require servers to be running all the time, you are less likely to benefit from the On-Demand pricing model, simply because there is no situation where you will need to turn servers off. For example, you might want the web server hosting the frontend of your corporate directory application to be running 24/7 so that users can access the website at any time. Even if there are no users connected to your website, you don’t want to shut down the servers supporting the site in case of potential user activity. In the case when servers cannot be stopped, consider using a Reserved Instance to save on costs.

**Reserve Capacity with Reserved Instances (RIs)**

RIs provide you with a significant discount compared to On-Demand instance pricing. RIs provide a discounted hourly rate and an optional capacity reservation for EC2 instances. You can choose between three payment options: *All Upfront*, *Partial Upfront*, or *No Upfront*. You can select either a 1-year or 3-year term for each of these options. Depending on which option you choose, you are discounted differently.

* All Upfront offers a higher discount than Partial Upfront instances.
* Partial Upfront instances offer a higher discount than No Upfront.
* No Upfront offers a higher discount than On-Demand.

*On-Demand* and *No Upfront* are similar since both do not require any upfront payment. However, there is a major difference. When you choose an On-Demand instance, you stop paying for the instance when you stop or terminate the instance. When you stop an RI, you still pay for it because you committed to a 1-year or 3-year term. Reserved Instances are associated with an instance type and an Availability Zone depending on how you reserve it. The discount applied by a Reserved Instance purchase is not directly associated with a specific instance ID, but with an instance type.

**Save on Costs with Spot Instances**

Another way of paying for EC2 instances is by using *Spot Instances*. Amazon EC2 Spot Instances allow you to take advantage of unused EC2 capacity in the AWS Cloud. They are available at up to a 90% discount compared to *On-Demand* prices. With Spot Instances, you set a limit on how much you would like to pay for the instance hour. This is compared against the current Spot price that AWS determines. If the amount you pay is more than the current Spot price and there is capacity, then you will receive an instance. While they are very promising from the billing perspective, there are some architectural considerations you will need to consider in order to use them effectively. One consideration is that your spot instance may be interrupted. For example, if AWS determines that capacity is no longer available for a particular spot instance or if the Spot price exceeds how much you are willing to pay, AWS will give you a 2-minute warning before it interrupts your instance. That means any application or workload that runs on a Spot instance must be able to be interrupted. Because of this unique consideration, inherently fault-tolerant workloads are typically good candidates to use with Spot instances. These include big data, containerized workloads, continuous integration/continuous delivery (CI/CD), web servers, high-performance computing (HPC), image and media rendering, or other test and development workloads.

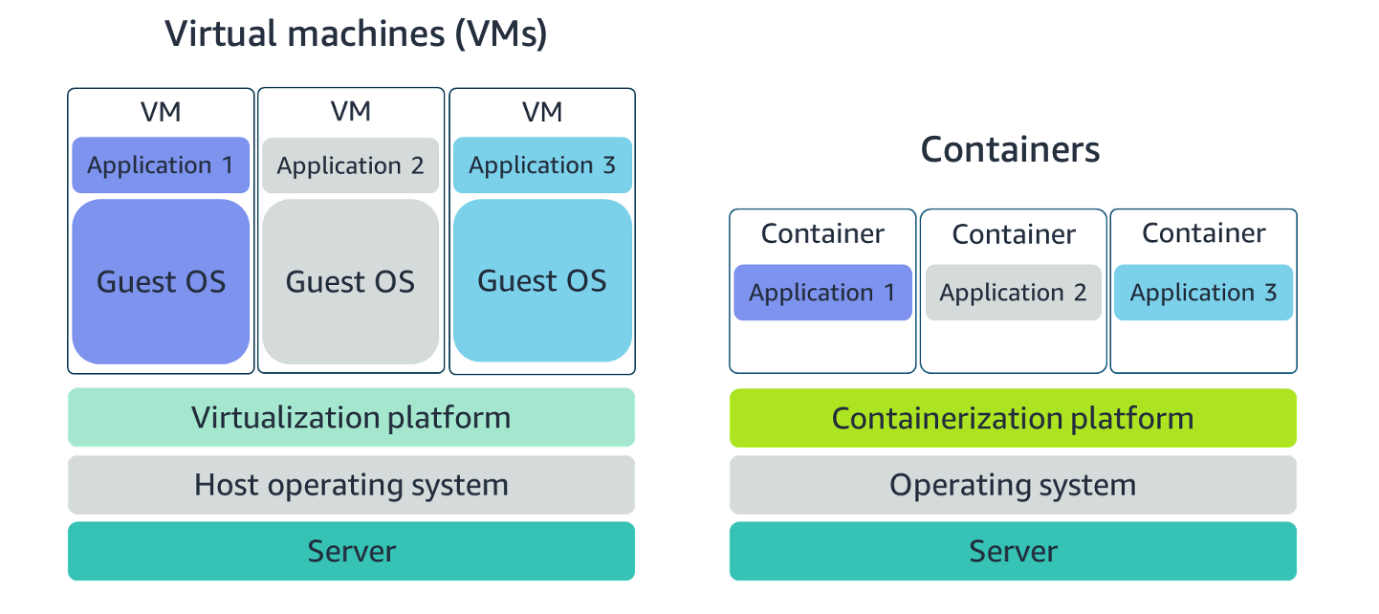
**WHAT IS A CONTAINER?**

While containers are often referred to as a new technology, the idea started in the 1970s with certain Linux kernels having the ability to separate their processes through isolation. At the time, this was configured manually, making operations complex.With the evolution of the open source software community, containers evolved. Today, containers are used as a solution to problems of traditional compute, including the issue of getting software to run reliably when it moves from one compute environment to another.A container is a standardized unit that packages up your code and all of its dependencies. This package is designed to run reliably on any platform, because the container creates its own independent environment. This makes it easy to carry workloads from one place to another, such as from development to production or from on-premises to the cloud.

**WHAT IS DOCKER?**

When you hear the word *container*, you may associate it with Docker. Docker is a popular container runtime that simplifies the management of the entire operating system stack needed for container isolation, including networking and storage. Docker makes it easy to create, package, deploy, and run containers.

**WHAT IS THE DIFFERENCE BETWEEN CONTAINERS AND VMS?**



Containers share the **same operating system and kernel** as the host they exist on, whereas virtual machines contain **their operating system**. Since each virtual machine has to maintain a copy of an operating system, there’s a degree of wasted space.A container is more lightweight. They spin up quicker, almost instantly. This difference in startup time becomes instrumental when designing applications that need to scale quickly during input/output (I/O) bursts.While containers can provide speed, virtual machines offer you the full strength of an operating system and offer more resources, like package installation, a dedicated kernel, and more.

**ORCHESTRATE CONTAINERS**

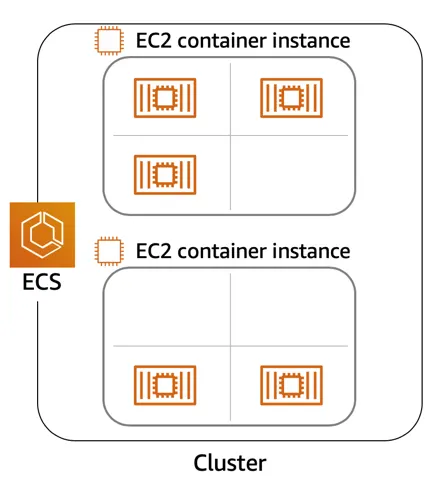
In AWS, containers run on EC2 instances. For example, you may have a large instance and run a few containers on that instance.While running one instance is easy to manage, it lacks high availability and scalability. Most companies and organizations run many containers on many EC2 instances across several Availability Zones.If you’re trying to manage your compute at a large scale, you need to know:

* How to place your containers on your instances.
* What happens if your container fails.
* What happens if your instance fails.
* How to monitor deployments of your containers.

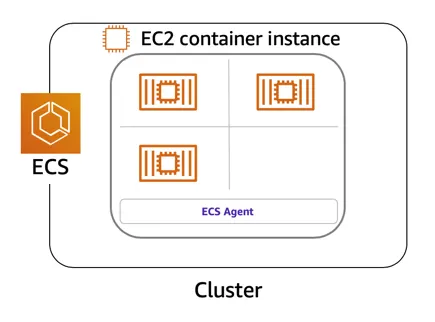
This coordination is handled by **a container orchestration service. AWS offers two container orchestration services: Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS).**

**MANAGE CONTAINERS WITH AMAZON ELASTIC CONTAINER SERVICE (AMAZON ECS)**

Amazon ECS is an end-to-end container orchestration service that allows you to quickly spin up new containers and manage them across a cluster of EC2 instances.



To run and manage your containers, you need to install the Amazon ECS Container Agent on your EC2 instances. This agent is open source and responsible for communicating back to the Amazon ECS service about cluster management details. You can run this agent on both Linux and Windows AMIs. An **instance with the container agent** installed is often called a *container instance*.



Once the Amazon ECS container instances are up and running, you can perform actions that include, but are not limited to, **launching and stopping containers, getting cluster state, scaling in and out, scheduling the placement of containers across your cluster, assigning permissions, and meeting availability requirements.**

To prepare your application to run on Amazon ECS, you create a task definition**. The task definition is a text file, in JSON format, that describes one or more containers. A task definition is similar to a blueprint that describes the resources you need to run that container, such as CPU, memory, ports, images, storage, and networking information.**

Here is a simple task definition that you can use for your corporate director application. In this example, the runs on the Nginx web server.

**{**

**"family": "webserver",**

**"containerDefinitions": [ {**

**"name": "web",**

**"image": "nginx",**

**"memory": "100",**

**"cpu": "99"**

**} ],**

**"requiresCompatibilities": [ "FARGATE" ],**

**"networkMode": "awsvpc",**

**"memory": "512",**

**"cpu": "256"**

**}**

**USE KUBERNETES WITH AMAZON ELASTIC KUBERNETES SERVICE (AMAZON EKS)**

Kubernetes **is a portable, extensible, open source platform for managing containerized workloads and services.** By bringing software development and operations together by design, Kubernetes created a rapidly growing ecosystem that is very popular and well established in the market.If you already use Kubernetes, you can use Amazon EKS to orchestrate these workloads in the AWS Cloud.Amazon EKS is conceptually similar to Amazon ECS, but there are some differences.

* An EC2 instance with the ECS Agent installed and configured is called a container instance. In Amazon EKS, it is called a **worker node.**
* An ECS Container is called a task. In the Amazon EKS ecosystem, it is called a pod.
* While Amazon ECS runs on AWS native technology, Amazon EKS runs on top of Kubernetes.

If you have containers running on Kubernetes and want an advanced orchestration solution that can provide simplicity, high availability, and fine-grained control over your infrastructure, Amazon EKS is the tool for you.

* **Responsibilities in EC2 Setup**:
  + Users are responsible for setting up and managing their fleet of EC2 instances.
  + Tasks include patching instances, scaling setup, and ensuring high availability (deploying instances across multiple Availability Zones).
  + Although less management overhead compared to on-premises solutions, users still need to establish management processes tailored to their use case.
* **Control and Flexibility in EC2**:
  + Users have significant control over their solutions on EC2, allowing for customization and scalability according to specifications.
  + This level of control is beneficial for certain use cases but may not be necessary for every solution.
* **Introduction to Serverless Computing**:
  + Serverless computing offers a different approach where users focus solely on their application without managing the underlying infrastructure.
  + AWS services, like AWS Lambda, exemplify serverless computing by abstracting away provisioning, scaling, fault tolerance, and maintenance tasks.
* **Benefits of Serverless Computing**:
  + Serverless offerings are convenient to use, allowing users to concentrate on the value of their application rather than implementation details.
  + Reduces the need for operational support processes as many management tasks are handled by the service provider.
* **Impact on Shared Responsibility Model**:
  + Serverless computing shifts responsibilities, with the service provider assuming more of the management tasks.
  + Users are still responsible for tasks like data encryption and access management, but tasks such as patching the operating system are handled by the service provider.
* **AWS Services Spectrum**:
  + AWS services can be viewed on a spectrum ranging from control to convenience.
  + Services like Amazon EC2 offer control, while serverless compute services like AWS Lambda prioritize convenience.
  + Users can choose services based on their specific needs and preferences.

**REMOVE THE UNDIFFERENTIATED HEAVY LIFTING**

If you run your code on Amazon EC2, AWS is responsible for the physical hardware and you are responsible for the logical controls, such as guest operating system, security and patching, networking, security, and scaling.If you run your code in containers on Amazon ECS and Amazon EKS, AWS is responsible for more of the container management, such as deploying containers across EC2 instances and managing the container cluster. However, when running ECS and EKS on EC2, you are still responsible for maintaining the underlying EC2 instances.If you want to deploy your workloads and applications without having to manage any EC2 instances, you can do that on AWS with *serverless* compute.

**GO SERVERLESS**

Every definition of serverless mentions four aspects.

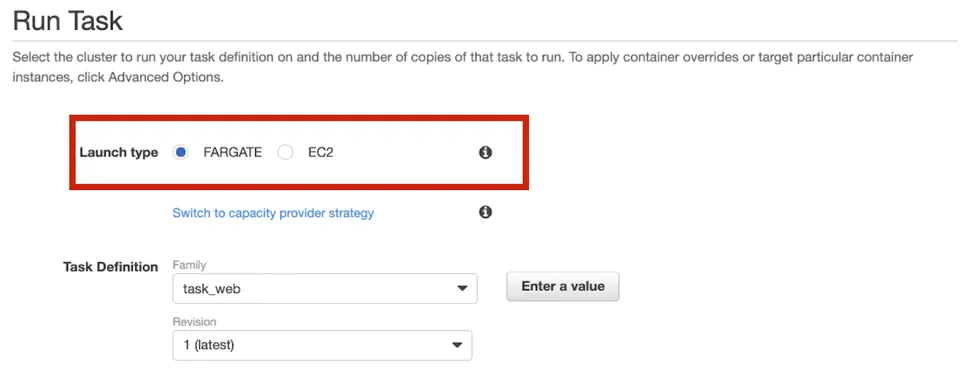
* No servers to provision or manage.
* Scales with usage.
* You never pay for idle resources.
* Availability and fault tolerance are built-in.

With serverless, spend time on the things that differentiate your application, rather than spending time on ensuring availability, scaling, and managing servers.AWS has several serverless compute options, including AWS Fargate and AWS Lambda.

**EXPLORE SERVERLESS CONTAINERS WITH AWS FARGATE**

Amazon ECS and Amazon EKS enable you to run your containers in two modes.

* Amazon EC2 mode
* AWS Fargate mode



AWS Fargate is a purpose-built serverless compute engine for containers. Fargate scales and manages the infrastructure, allowing developers to work on what they do best: application development.It achieves this by allocating the right amount of compute, eliminating the need to choose and handle EC2 Instances and cluster capacity and scaling. Fargate supports both Amazon ECS and Amazon EKS architecture and provides workload isolation and improved security by design.

AWS Fargate abstracts the EC2 instance so you’re not required to manage it. However, with AWS Fargate, you can use all the same ECS primitives, APIs, and AWS integrations. It natively integrates with AWS Identity and Access Management (IAM) and Amazon Virtual Private Cloud (VPC). Having native integration with Amazon VPC allows you to launch Fargate containers inside your network and control connectivity to your applications.

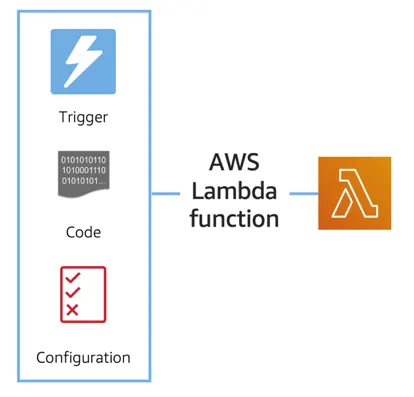
**RUN YOUR CODE ON AWS LAMBDA**

If you want to deploy your workloads and applications without having to manage any EC2 instances or containers, you can use AWS Lambda.AWS Lambda lets you run code without provisioning or managing servers or containers. You can run code for virtually any type of application or backend service, including data processing, real-time stream processing, machine learning, WebSockets, IoT backends, mobile backends, and web apps, like your corporate directory app!

AWS Lambda requires zero administration from the user. You upload your source code and Lambda takes care of everything required to run and scale your code with high availability. There are no servers to manage, bringing you continuous scaling with subsecond metering and consistent performance.

**HOW LAMBDA WORKS**

There are three primary components of a Lambda function: the trigger, code, and configuration.The code is source code, that describes what the Lambda function should run. This code can be authored in three ways.



* You create the code from scratch.
* You use a blueprint that AWS provides.
* You use same code from the AWS Serverless Application Repository, a resource that contains sample applications, such as “hello world” code, Amazon Alexa Skill sample code, image resizing code, video encoding, and more.

When you create your Lambda function, you specify the runtime you want your code to run in. There are built-in runtimes such as Python, Node.js, Ruby, Go, Java, .NET Core, or you can implement your Lambda functions to run on a custom runtime.The configuration of a Lambda function consists of information that describes how the function should run. In the configuration, you specify network placement, environment variables, memory, invocation type, permission sets, and other configurations. To dive deeper into these configurations, check out the resources section of this unit.Triggers describe when the Lambda function should run.

A trigger integrates your Lambda function with other AWS services, enabling you to run your Lambda function in response to certain API calls that occur in your AWS account. This makes you quicker to respond to events in your console without having to perform manual actions.All you need is the what, how, and when of a Lambda function to have functional compute capacity that runs only when you need it to.Amazon’s CTO, Werner Vogels, says, “No server is easier to manage than no server.” This quote summarizes the convenience you can have when running serverless solutions, like AWS Fargate and AWS Lambda.

In the next unit, you apply all the information you’ve learned about Amazon EC2, Amazon ECS and Amazon EKS, and AWS Fargate and learn the use cases for each service.

**AWS Lambda function handler**

The AWS Lambda function handler is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method. When the handler exits or returns a response, it becomes available to handle another event.You can use the following general syntax when creating a function handler in Python:

def handler\_name(event, context): ... return some\_value

**NAMING**

The Lambda function *handler* name specified at the time you create a Lambda function is derived from the following:the name of the file in which the Lambda handler function is locatedthe name of the Python handler functionA function handler can be any name; however, the default on the Lambda console is lambda\_function.lambda\_handler. This name reflects the function name as lambda\_handler, and the file where the handler code is stored in lambda\_function.py.If you choose a different name for your function handler on the Lambda console, you must update the name on the **Runtime settings** pane.

**BILLING GRANULARITY**

AWS Lambda lets you run code without provisioning or managing servers, and you pay only for what you use. You are charged for the number of times your code is triggered (requests) and for the time your code executes, rounded up to the nearest 1ms (duration). AWS rounds up duration to the nearest millisecond with no minimum execution time. With this pricing, it can be very cost effective to run functions whose execution time is very low, such as functions with durations under 100ms or low latency APIs. For more information, see AWS News Blog.

**SOURCE CODE**

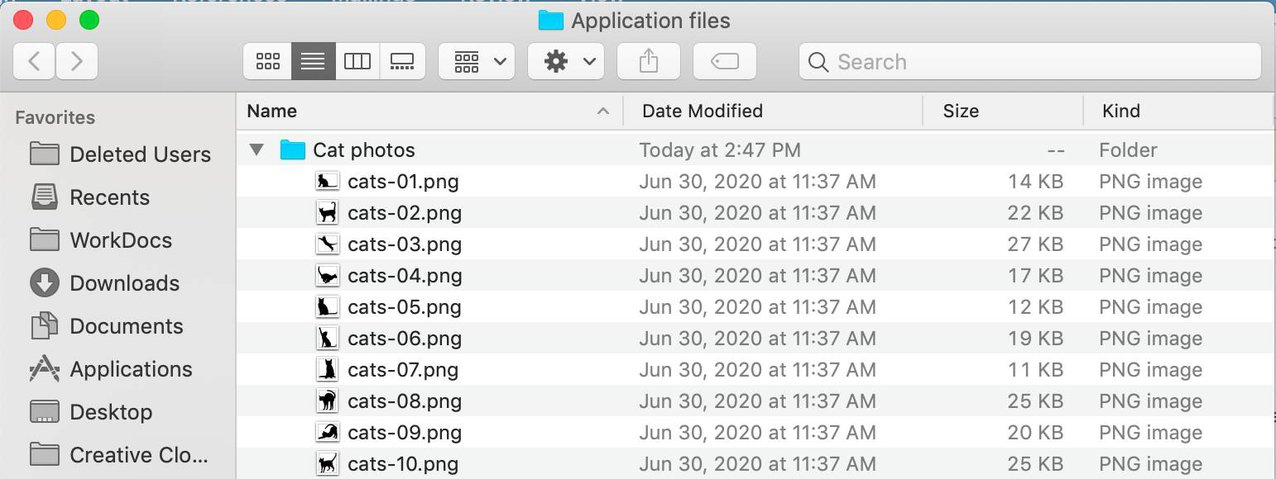
This video used a small amount of sample code illustrating a pattern for lazily generating assets using AWS Lambda and Amazon S3. If you’re looking to deploy a service to resize images to production, consider using the new release  [Serverless Image Handler](https://aws.amazon.com/answers/web-applications/serverless-image-handler/)which is a robust solution to handle image manipulation and can be deployed via an AWS CloudFormation template.

You can find a tutorial on creating the AWS Lambda function as well as the code used in the AWS Lambda demo here: see [AWS News Blog](https://aws.amazon.com/blogs/compute/resize-images-on-the-fly-with-amazon-s3-aws-lambda-and-amazon-api-gateway/).

* Configure storage for the employee directory app.
* Application requires storage for:
  + Operating system, software, and system files.
  + Static assets (e.g., employee headshot photos).
  + Structured data (e.g., employee names, titles, locations).
* Focus on storing application files and static content for now.
* Two main storage types: block and object.
* Block storage:
  + Splits files into fixed size chunks.
  + Easy to update specific parts of a file.
* Object storage:
  + Treats each file as a single unit.
  + Requires updating the entire file for changes.
* Static data (e.g., photos):
  + Accessed often, modified rarely.
  + Suitable for object storage.
* Frequently updated data (e.g., application/system files):
  + High transaction rates.
  + Performs better with block storage.

**File Storage**

You may be familiar with file storage if you’ve interacted with file storage systems like Windows File Explorer or Finder on MacOS. You place your files in a tree-like hierarchy that consists of folders and subfolders. For example, if you have hundreds of cat photos on your laptop, you may want to create a folder called Cat photos, and place those images inside that folder to organize them. Since you know these images will be used in an application, you may want to place the cat photos folder inside another folder called Application files.



 Each file has metadata such as file name, file size, and the date the file was created. The file also has a path, for example, computer/Application\_files/Cat\_photos/cats-03.png. When you need to retrieve a file, your system can use the path to find it in the file hierarchy.

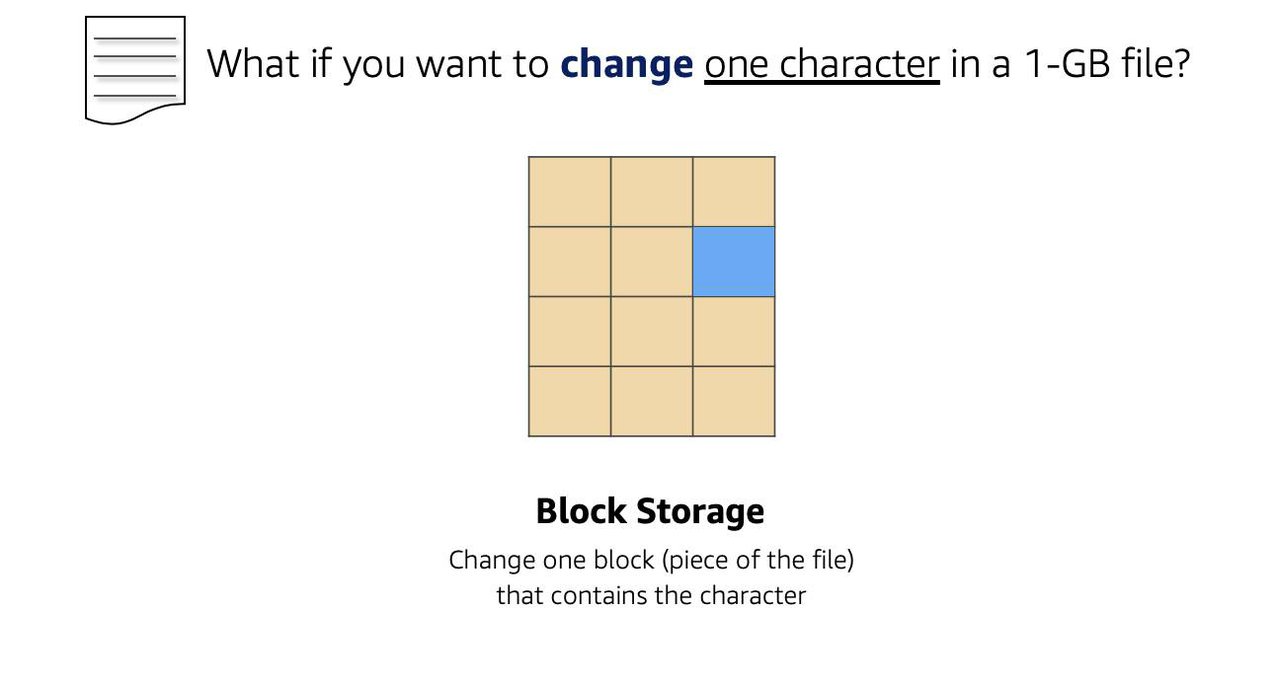
File storage is ideal when you require centralized access to files that need to be easily shared and managed by multiple host computers. Typically, this storage is mounted onto multiple hosts and requires file locking and integration with existing file system communication protocols.Common use cases for file storage include:

* Large content repositories
* Development environments
* User home directories

**Block Storage**

While file storage treats files as a singular unit, block storage splits files into fixed-size chunks of data called **blocks** that have their own addresses. Since each block is addressable, blocks can be retrieved efficiently.

When data is requested, these addresses are used by the storage system to organize the blocks in the correct order to form a complete file to present back to the requestor. Outside of the address, there is no additional metadata associated with each block. So, when you want to change a character in a file, you just change the block, or the piece of the file, that contains the character. This ease of access is why block storage solutions are fast and use less bandwidth.

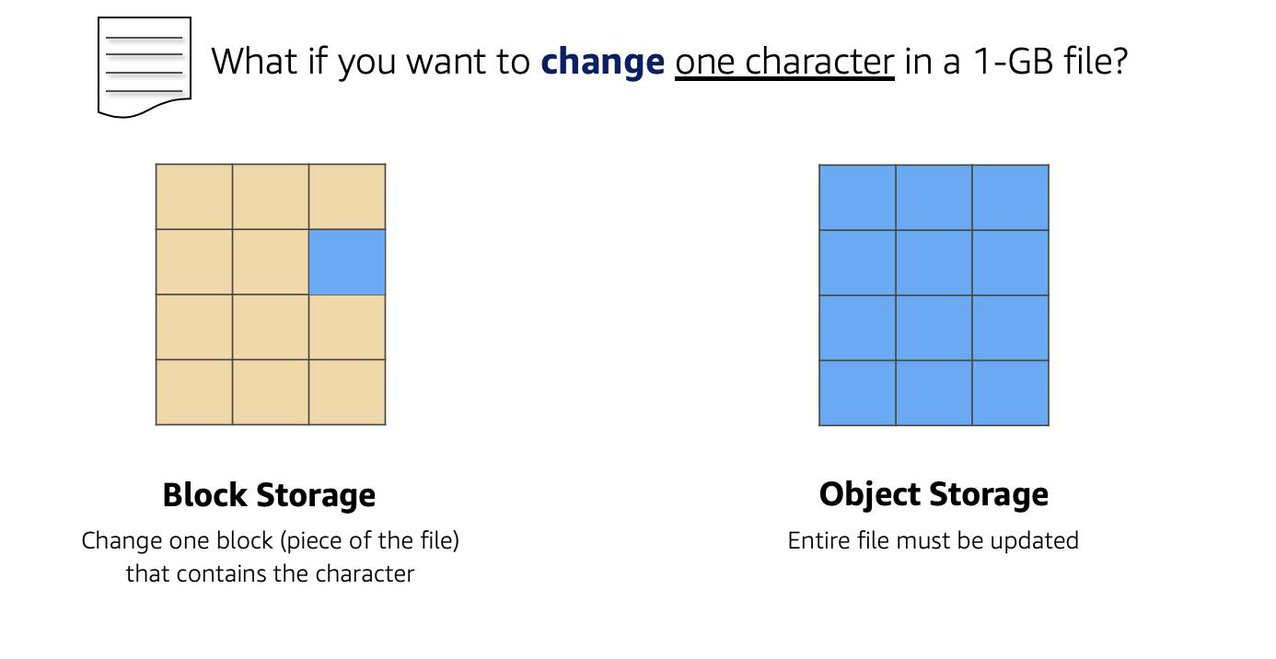


Since block storage is optimized for low-latency operations, it is a typical storage choice for high-performance enterprise workloads, such as databases or enterprise resource planning (ERP) systems, that require low-latency storage.

**Object Storage**

Objects, much like files, are also treated as a single unit of data when stored. However, unlike file storage, these objects are stored in a flat structure instead of a hierarchy. Each object is a file with a unique identifier. This identifier, along with any additional metadata, is bundled with the data and stored.

Changing just one character in an object is more difficult than with block storage. When you want to change one character in a file, the entire file must be updated.



 With object storage, you can store almost any type of data, and there is no limit to the number of objects stored, making it easy to scale. Object storage is generally useful when storing large data sets, unstructured files like media assets, and static assets, such as photos.

**Reading 3.2: Amazon EC2 Instance Storage and Amazon Elastic Block Store**

**Amazon EC2 Instance Store**

Amazon EC2 Instance Store provides temporary block-level storage for your instance. This storage is located on disks that are physically attached to the host computer. This ties the lifecycle of your data to the lifecycle of your EC2 instance. If you delete your instance, the instance store is deleted as well. Due to this, instance store is considered ephemeral storage. Read more about it in the [AWS documentation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html) .

Instance store is ideal if you are hosting applications that replicate data to other EC2 instances, such as Hadoop clusters. For these cluster-based workloads, having the speed of locally attached volumes and the resiliency of replicated data helps you achieve data distribution at high performance. It’s also ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content.

**Amazon Elastic Block Storage (Amazon EBS)** As the name implies, Amazon EBS is a block-level storage device that you can attach to an Amazon EC2 instance. These storage devices are called Amazon EBS volumes. EBS volumes are essentially drives of a user-configured size attached to an EC2 instance, similar to how you might attach an external drive to your laptop.

EBS volumes act similarly to external drives in more than one way.

* Most Amazon EBS volumes can only be connected with one computer at a time. Most EBS volumes have a one-to-one relationship with EC2 instances, so they cannot be shared by or attached to multiple instances at one time. *Note: Recently, AWS announced the Amazon EBS multi-attach feature that enables volumes to be attached to multiple EC2 instances at one time. This feature is not available for all instance types and all instances must be in the same Availability Zone. Read more about this scenario in the* [*EBS documentation*](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html) *.*
* You can detach an EBS volume from one EC2 instance and attach it to another EC2 instance in the same Availability Zone, to access the data on it.
* The external drive is separate from the computer. That means, if an accident happens and the computer goes down, you still have your data on your external drive. The same is true for EBS volumes.
* You’re limited to the size of the external drive, since it has a fixed limit to how scalable it can be. For example, you may have a 2 TB external drive and that means you can only have 2 TB of content on there. This relates to EBS as well, since volumes also have a max limitation of how much content you can store on the volume.

**Scale Amazon EBS Volumes**

You can scale Amazon EBS volumes in two ways.

1. Increase the volume size, as long as it doesn’t increase above the maximum size limit. For EBS volumes, the maximum amount of storage you can have is 16 TB. That means if you provision a 5 TB EBS volume, you can choose to increase the size of your volume until you get to 16 TB.
2. Attach multiple volumes to a single Amazon EC2 instance. EC2 has a one-to-many relationship with EBS volumes. You can add these additional volumes during or after EC2 instance creation to provide more storage capacity for your hosts.

**Amazon EBS Use Cases**

Amazon EBS is useful when you need to retrieve data quickly and have data persist long-term. Volumes are commonly used in the following scenarios.

* **Operating systems:** Boot/root volumes to store an operating system. The root device for an instance launched from an Amazon Machine Image (AMI) is typically an Amazon EBS volume. These are commonly referred to as EBS-backed AMIs.
* **Databases:** A storage layer for databases running on Amazon EC2 that rely on transactional reads and writes.
* **Enterprise applications:** Amazon EBS provides reliable block storage to run business-critical applications.
* **Throughput-intensive applications:** Applications that perform long, continuous reads and writes.

**Amazon EBS Volume Types**

|  | **EBS Provisioned IOPS SSD** | **EBS General Purpose SSD** | **Throughput Optimized HDD** | **Cold HDD** |
| --- | --- | --- | --- | --- |
| **Description** | Highest performance SSD designed for latency-sensitive transactional workloads | General purpose SSD that balances price and performance for a wide variety of transactional workloads | Low-cost HDD designed for frequently accessed, throughput intensive workloads | Lowest cost HDD designed for less frequently accessed workloads |
| **Use Cases** | I/O-intensive NoSQL and relational databases | Boot volumes, low-latency interactive apps, development, and test | Big data, data warehouses, log processing | Colder data requiring fewer scans per day |
| **Volume Size** | 4 GB-16 TB | 1 GB-16 TB | 500 GB-16 TB | 500 GB-16 TB |
| **Max IOPS/Volume** | 64,000 | 16,000 | 500 | 250 |
| **Max Throughput/Volume** | 1,000 MB/s | 250 MB/s | 500 MB/s | 250 MB/s |

There are two main categories of Amazon EBS volumes: solid-state drives (SSDs) and hard-disk drives (HDDs). SSDs provide strong performance for random input/output (I/O), while HDDs provide strong performance for sequential I/O. AWS offers two types of each. The following chart can help you decide which EBS volume is the right option for your workload.

**Benefits of Using Amazon EBS**

Here are the following benefits of using Amazon EBS (in case you need a quick cheat sheet).

* High availability: When you create an EBS volume, it is automatically replicated within its Availability Zone to prevent data loss from single points of failure.
* Data persistence: The storage persists even when your instance doesn’t.
* Data encryption: All EBS volumes support encryption.
* Flexibility: EBS volumes support on-the-fly changes. You can modify volume type, volume size, and input/output operations per second (IOPS) capacity without stopping your instance.
* Backups: Amazon EBS provides you the ability to create backups of any EBS volume.

**EBS Snapshots**

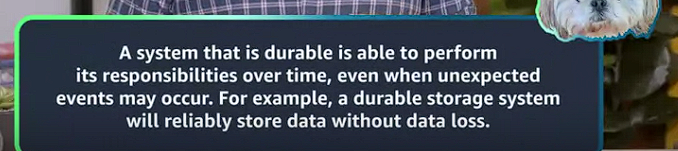
Errors happen. One of those errors is not backing up data, and then, inevitably losing that data. To prevent this from happening to you, you should back up your data—even in AWS. Since your EBS volumes consist of the data from your Amazon EC2 instance, you’ll want to take backups of these volumes, called snapshots.

EBS snapshots are incremental backups that only save the blocks on the volume that have changed after your most recent snapshot. For example, if you have 10 GB of data on a volume, and only 2 GB of data have been modified since your last snapshot, only the 2 GB that have been changed are written to Amazon Simple Storage Service (Amazon S3).

When you take a snapshot of any of your EBS volumes, these backups are stored redundantly in multiple Availability Zones using Amazon S3. This aspect of storing the backup in Amazon S3 will be handled by AWS, so you won’t need to interact with Amazon S3 to work with your EBS snapshots. You simply manage them in the EBS console (which is part of the EC2 console).

EBS snapshots can be used to create multiple new volumes, whether they’re in the same Availability Zone or a different one. When you create a new volume from a snapshot, it’s an exact copy of the original volume at the time the snapshot was taken.

* **Storage for Employee Photos**:
  + **Inadequacy of Amazon EBS**:
    - **Single Instance Limitation**: Most EBS volumes can only be connected to one EC2 instance at a time. Multi-attach is not universally supported.
    - **Size Limitations**: EBS volumes have size constraints, which is not ideal for storing a large number of high-definition photos.
* **Amazon S3 as a Solution**:
  + **Standalone Storage**: Not tied to compute; accessed via URLs, making it "storage for the internet."
  + **Scalability**: No limit to the number of objects, with individual object size limit of 5 TB.
  + **Object Storage**: S3 uses a **flat structure with unique identifiers for objects**, distributed across multiple facilities within one AWS region.
  + **High Availability and Durability**: Designed for 99.99% availability and 11 nines (99.999999999%) of durability.



* **S3 Storage Concepts**:
  + **Buckets**: Containers for storing objects.
    - **Creating Buckets**: Must be globally unique and DNS compliant. Region-specific placement (e.g., Oregon).( You can't upload any object, not even a single photo to S3 without creating a bucket first.)
    - **Organizing Objects**: Use folders within buckets for organization.
    - **Uploading Objects**: Objects are uploaded into buckets and can be accessed via URLs.
    - our bucket name has to be globally unique across all AWS accounts and must be DNS compliant. Once you create your bucket, AWS will construct a URL using this name, so it has to be something that is reachable over HTTP or HTTPS, meaning there can be no special characters, no spaces, et cetera. So for this bucket's name, let's choose employee-photo-bucket-sr-001,
* **Accessing Data in S3**:
  + **Default Privacy**: All S3 resources (buckets, folders, objects) are private by default.
  + **Public Access**: Explicitly set via bucket permissions and object actions. Requires several steps to ensure intentional public access to prevent accidental data exposure.
  + **Granular Access Control**: Achieved through IAM policies and S
  + buckets are region specific, so we can choose where we want to place our bucket

3 bucket policies.

* **IAM Policies**: Attached to users, groups, and roles to manage access to S3 content.
* **S3 Bucket Policies**:
  + Attached directly to buckets.
  + Written in JSON format.
  + Define allowed or denied actions on the bucket and its contents.
  + Apply policies for specific scenarios, such as allowing another AWS account to upload objects or enabling read-only access to anonymous users.
  + Policies are applied at the bucket level but affect all objects within the bucket.
* **Backup and Restore with S3**:
  + **Durability and Redundancy**: S3 data is stored across multiple facilities, ensuring high durability.
  + **Access Methods**: Objects can be retrieved via HTTP/HTTPS URLs, ensuring easy access from anywhere.
* **Key Takeaways**:
  + **S3 Suitability**: Ideal for scalable, durable storage of employee photos.
  + **Privacy and Security**: S3 ensures default privacy but allows fine-grained control over access through policies.
  + **Ease of Access**: URLs provide simple, direct access to stored objects.

By choosing Amazon S3 for storing employee photos, you leverage its scalability, durability, and ease of access, making it a more suitable solution compared to Amazon EBS for this specific use case.

WS uses this name as part of the object identifier. In S3, each object is identified using a URL, which looks like this:



After the http://, you see the bucket name. In this example, the bucket is named *doc*. Then, the identifier uses the service name, *s3* and specifies the service provider *amazonaws*. After that, you have an implied folder inside the bucket called *2006-03-01* and the object inside the folder that is named *AmazonS3.html*. The object name is often referred to as *the key name*.

Note, you can have folders inside of buckets to help you organize objects. However, remember that there’s no actual file hierarchy that supports this on the back end. It is instead a flat structure where all files and folders live at the same level. Using buckets and folders implies a hierarchy, which makes it easy to understand for the human eye.

**S3 USE CASES**

Amazon S3 is one of the most widely used storage services, with far more use cases than could fit on one screen. The following list summarizes some of the most common ways you can use Amazon S3.

* **Backup and storage:** S3 is a natural place to back up files because it is highly redundant. As mentioned in the last unit, AWS stores your EBS snapshots in S3 to take advantage of its high availability.
* **Media hosting:** Because you can store unlimited objects, and each individual object can be up to 5 TBs, S3 is an ideal location to host video, photo, or music uploads.
* **Software delivery:** You can use S3 to host your software applications that customers can download.
* **Data lakes:** S3 is an optimal foundation for a data lake because of its virtually unlimited scalability. You can increase storage from gigabytes to petabytes of content, paying only for what you use.
* **Static websites:** You can configure your bucket to host a static website of HTML, CSS, and client-side scripts.
* **Static content:** Because of the limitless scaling, the support for large files, and the fact that you access any object over the web at any time, S3 is the perfect place to store static content.

**CHOOSE THE RIGHT CONNECTIVITY OPTION FOR YOUR RESOURCES**

Everything in Amazon S3 is private by default. This means that all S3 resources, such as buckets, folders, and objects can only be viewed by the user or AWS account that created that resource. Amazon S3 resources are all private and protected to begin with.

If you decide that you want everyone on the internet to see your photos, you can choose to make your buckets, folders, and objects public. Keep in mind that a public resource means that everyone on the internet can see it. Most of the time, you don’t want your permissions to be all or nothing. Typically, you want to be more granular about the way you provide access to your resources.

To be more specific about who can do what with your S3 resources, Amazon S3 provides two main access management features: IAM policies and S3 bucket policies.

**UNDERSTAND IAM POLICIES**

Previously, you learned about creating and using IAM policies, and now you get to apply this to Amazon S3. When IAM policies are attached to IAM users, groups, and roles, the policies define which actions they can perform. IAM policies are not tied to any one AWS service and can be used to define access to nearly any AWS action. You should use IAM policies for private buckets when:

* You have many buckets with different permission requirements. Instead of defining many different S3 bucket policies, you can use IAM policies instead.
* You want all policies to be in a centralized location. Using IAM policies allows you to manage all policy information in one location.

**UNDERSTAND S3 BUCKET POLICIES**

S3 bucket policies are similar to IAM policies, in that they are both defined using the same policy language in a JSON format. The difference is IAM policies are attached to users, groups, and roles, whereas S3 bucket policies are only attached to buckets. S3 bucket policies specify what actions are allowed or denied on the bucket.

For example, if you have a bucket called employeebucket, you can attach an S3 bucket policy to it that allows another AWS account to put objects in that bucket.

Or if you wanted to allow anonymous viewers to read the objects in employeebucket, then you can apply a policy to that bucket that allows anyone to read objects in the bucket using *"Effect":Allow* on the *"Action:["s3:GetObject"]"*.

Here’s an example of what that S3 bucket policy might look like.

*{*

*"Version":"2012-10-17",*

*"Statement":[*

*{*

*"Sid":"PublicRead",*

*"Effect":"Allow",*

*"Principal": "\*",*

*"Action":["s3:GetObject"],*

*"Resource":["arn:aws:s3:::employeebucket/\*"]*

*}*

*]*

*}*

S3 Bucket policies can only be placed on buckets, and cannot be used for folders or objects. However, the policy that is placed on the bucket applies to every object in that bucket. You should use S3 bucket policies when:

* You need a simple way to do cross-account access to S3, without using IAM roles.
* Your IAM policies bump up against the defined size limit. S3 bucket policies have a larger size limit.

**ENCRYPT S3**

Amazon S3 reinforces encryption in transit (as it travels to and from Amazon S3) and at rest. To protect data at rest, you can use:

* **Server-side encryption:** This allows Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects.
* **Client-side encryption:** Encrypt your data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and all related tools.

To encrypt in transit, you can use client-side encryption or Secure Sockets Layer (SSL).

**USE VERSIONING TO PRESERVE OBJECTS**

As you know, Amazon S3 identifies objects in part by using the object name. For example, when you upload an employee photo to S3, you may name the object employee.jpg and store it in a folder called employees. If you don’t use Amazon S3 versioning, anytime you upload an object called employee.jpg to the employees folder, it overwrites the original file. This can be an issue for several reasons.

* employee.jpg is a common name for an employee photo object. You or someone else who has access to that bucket might not have intended to overwrite it, and now that you have, you no longer have access to the original file.
* You may want to preserve different versions of employee.jpg. Without versioning, if you wanted to create a new version of employee.jpg, you would need to upload the object and choose a different name for it. Having several objects all with slight differences in naming variations may cause confusion and clutter in your bucket.

So, what do you do? You use S3 versioning! Versioning enables you to keep multiple versions of a single object in the same bucket. This allows you to preserve old versions of an object without having to use different naming constructs, in case you need to recover from accidental deletions, accidental overwrites, or even application failures. Let’s see how this works.

If you enable versioning for a bucket, Amazon S3 automatically generates a unique version ID for the object being stored. In one bucket, for example, you can have two objects with the same key, but different version IDs, such as employeephoto.gif (version 111111) and employeephoto.gif (version 121212). Versioning-enabled buckets let you recover objects from accidental deletion or overwrite.

* Deleting an object does not remove the object permanently. Instead, Amazon S3 puts a marker on the object that shows you tried to delete it. If you want to restore the object, you can remove this marker and it reinstates the object.
* If you overwrite an object, it results in a new object version in the bucket. You still have access to previous versions of the object.

**UNDERSTAND VERSIONING STATES**

Buckets can be in one of three states.

* Unversioned (the default): No new or existing objects in the bucket have a version.
* Versioning-enabled: This enables versioning for all objects in the bucket.
* Versioning-suspended: This suspends versioning for new objects. All new objects in the bucket will not have a version. However, all existing objects keep their object versions.

The versioning state applies to all of the objects in that bucket. Keep in mind that storage costs are incurred for all objects in your bucket and all versions of those objects. To reduce your S3 bill, you may want to delete previous versions of your objects that are no longer in use.

**WHAT ARE AMAZON S3 STORAGE CLASSES?**

When you upload an object to Amazon S3 and you don’t specify the storage class, you’re uploading it to the default storage class—often referred to as standard storage. When you learned about Amazon S3 in previous units, you were learning about the standard storage class without even knowing it! S3 storage classes let you change your storage tier as your data characteristics change. For example, if you are now accessing your old photos infrequently, you may want to change the storage class those photos are stored in to save on costs. There are six S3 storage classes.

1. **Amazon S3 Standard:** This is considered general purpose storage for cloud applications, dynamic websites, content distribution, mobile and gaming applications, and big data analytics.
2. **Amazon S3 Intelligent-Tiering:** This tier is useful if your data has unknown or changing access patterns. S3 Intelligent-Tiering stores objects in two tiers, a frequent access tier and an infrequent access tier. Amazon S3 monitors access patterns of your data, and automatically moves your data to the most cost-effective storage tier based on frequency of access.
3. **Amazon S3 Standard-Infrequent Access (S3 Standard-IA):** S3 Standard-IA is for data that is accessed less frequently, but requires rapid access when needed. S3 Standard-IA offers the high durability, high throughput, and low latency of S3 Standard, with a low per-GB storage price and per-GB retrieval fee. This storage tier is ideal if you want to store long-term backups, disaster recovery files, and so on.
4. **Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA):** Unlike other S3 storage classes which store data in a minimum of three Availability Zones (AZs), S3 One Zone-IA stores data in a single AZ and costs 20% less than S3 Standard-IA. S3 One Zone-IA is ideal for customers who want a lower-cost option for infrequently accessed data but do not require the availability and resilience of S3 Standard or S3 Standard-IA. It’s a good choice for storing secondary backup copies of on-premises data or easily re-creatable data.
5. **Amazon S3 Glacier Instant Retrieval:**Amazon S3 Glacier Instant Retrieval is an archive storage class that delivers the lowest-cost storage for long-lived data that is rarely accessed and requires retrieval in milliseconds.
6. **Amazon S3 Glacier Flexible Retrieval:**S3 Glacier Flexible Retrieval delivers low-cost storage, up to 10% lower cost (than S3 Glacier Instant Retrieval), for archive data that is accessed 1—2 times per year and is retrieved asynchronously.
7. **Amazon S3 Glacier Deep Archive:** S3 Glacier Deep Archive is Amazon S3’s lowest-cost storage class and supports long-term retention and digital preservation for data that may be accessed once or twice in a year. It is designed for customers—particularly those in highly regulated industries, such as the Financial Services, Healthcare, and Public Sectors—that retain data sets for 7 to 10 years or longer to meet regulatory compliance requirements.
8. **Amazon S3 Outposts:**Amazon S3 on Outposts delivers object storage to your on-premises AWS Outposts environment.

**AUTOMATE TIER TRANSITIONS WITH OBJECT LIFECYCLE MANAGEMENT**

If you keep manually changing your objects, such as your employee photos, from storage tier to storage tier, you may want to look into automating this process using a lifecycle policy. When you define a lifecycle policy configuration for an object or group of objects, you can choose to automate two actions: transition and expiration actions.

* **Transition actions** are used to define when you should transition your objects to another storage class.
* **Expiration actions** define when objects expire and should be permanently deleted.

For example, you might choose to transition objects to S3 Standard-IA storage class 30 days after you created them, or archive objects to the S3 Glacier storage class one year after creating them.

The following use cases are good candidates for lifecycle management.

* **Periodic logs:** If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you might want to delete them.
* **Data that changes in access frequency:** Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them, but your organization or regulations might require you to archive them for a specific period. After that, you can delete them.

**Reading 3.4: Choose the Right Storage Service**

Here’s a recap of all the storage services mentioned so far. By the end of this reading, you should be able to better answer the question “Which storage service should I use?” for some of the more common scenarios.

**Amazon EC2 Instance Store**

Instance store is ephemeral block storage. This is preconfigured storage that exists on the same physical server that hosts the EC2 instance and cannot be detached from Amazon EC2. You can think of it as a built-in drive for your EC2 instance. Instance store is generally well-suited for temporary storage of information that is constantly changing, such as buffers, caches, and scratch data. It is not meant for data that is persistent or long-lasting. If you need persistent long-term block storage that can be detached from Amazon EC2 and provide you more management flexibility, such as increasing volume size or creating snapshots, then you should use Amazon EBS.

**Amazon EBS**

Amazon EBS is meant for data that changes frequently and needs to persist through instance stops, terminations, or hardware failures. Amazon EBS has two different types of volumes: SSD-backed volumes and HDD-backed volumes.SSD-backed volumes have the following characteristics.

* Performance depends on IOPS (input/output operations per second).
* Ideal for transactional workloads such as databases and boot volumes.

HDD-backed volumes have the following characteristics:

* Performance depends on MB/s.
* Ideal for throughput-intensive workloads, such as big data, data warehouses, log processing, and sequential data I/O.

Here are a few important features of Amazon EBS that you need to know when comparing it to other services.

* It is block storage.
* You pay for what you provision (you have to provision storage in advance).
* EBS volumes are replicated across multiple servers in a single Availability Zone.
* Most EBS volumes can only be attached to a single EC2 instance at a time.

**Amazon S3**

If your data doesn’t change that often, Amazon S3 might be a more cost-effective and scalable storage solution. S3 is ideal for storing static web content and media, backups and archiving, data for analytics, and can even be used to host entire static websites with custom domain names.Here are a few important features of Amazon S3 to know about when comparing it to other services.

* It is object storage.
* You pay for what you use (you don’t have to provision storage in advance).
* Amazon S3 replicates your objects across multiple Availability Zones in a Region.
* Amazon S3 is not storage attached to compute.

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

In this module, you’ve already learned about Amazon S3 and Amazon EBS. You learned that S3 uses a flat namespace and isn’t meant to serve as a standalone file system. You also learned most EBS volumes can only be attached to one EC2 instance at a time. So, if you need file storage on AWS, which service should you use?For file storage that can mount on to multiple EC2 instances, you can use Amazon Elastic File System (Amazon EFS) or Amazon FSx. Use the following table for more information about each of these services.

|  |  |  |
| --- | --- | --- |
| Service | Characteristic | More Information |
| Amazon Elastic File System (EFS) | Fully managed NFS file system. | [EFS FAQs](https://aws.amazon.com/efs/faq/) |
| Amazon FSx for Windows File Server | Fully managed file server built on Windows Server that supports the SMB protocol. | [FSx for Windows File Server FAQs](https://aws.amazon.com/fsx/windows/faqs/?nc=sn&loc=8) |
| Amazon FSx for Lustre | Fully managed Lustre file system that integrates with S3. | [FSx for Lustre FAQs](https://aws.amazon.com/fsx/lustre/faqs/?nc=sn&loc=5) |

Here are a few important features of Amazon EFS and FSx to know about when comparing them to other services.

* It is file storage.
* You pay for what you use (you don’t have to provision storage in advance).
* Amazon EFS and Amazon FSx can be mounted onto multiple EC2 instances.

**Resources**: