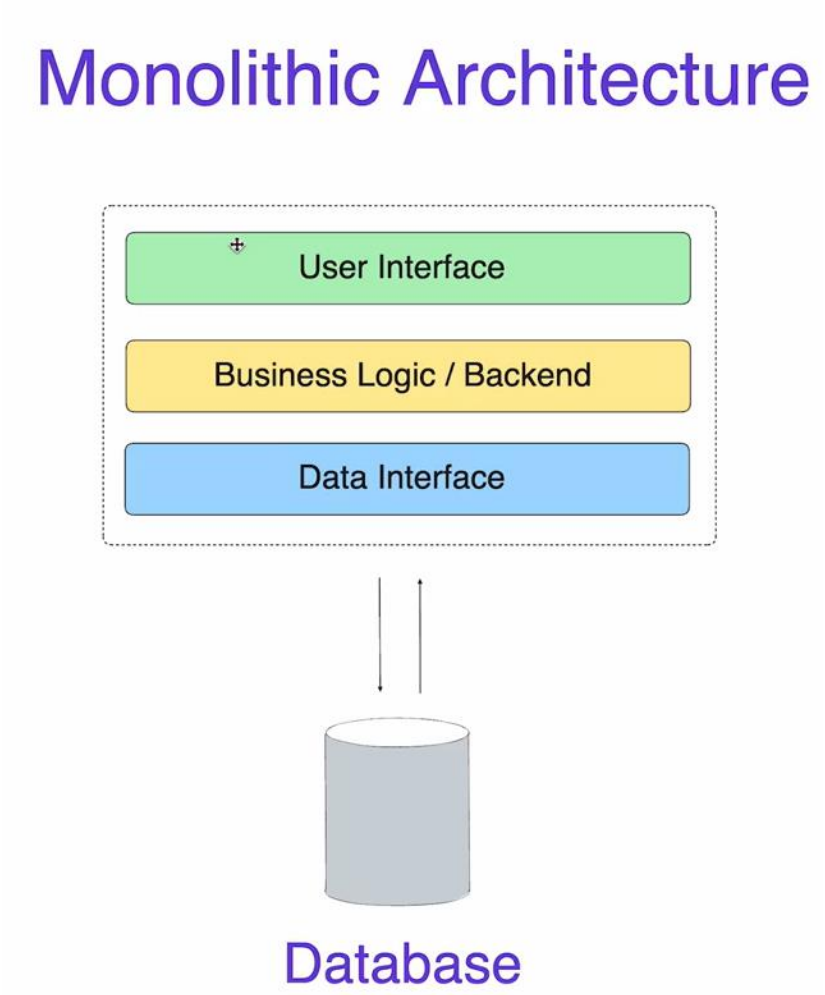
**Week 3 – System Design Applications**

1. Monolithic Architecture

- Monolithic Architecture is a single-tiered software application built as a single and indivisible unit.




- This includes typically 3 components such as user interface, business logic/backend and data interface. User interface is the frontend part of the application where users interact with software, collecting users' inputs and display data retrieved from backend servers. The business logic contains the core computational logic of the software application, processing users' inputs and applying relevant business rules and making decisions based on the data that provided by the users. The data interface is the data access layer that communicates with the database, handling data retrieval, updating data and storing data based on the operations performed by business logics.

- The components are tightly integrated and run as a single service.

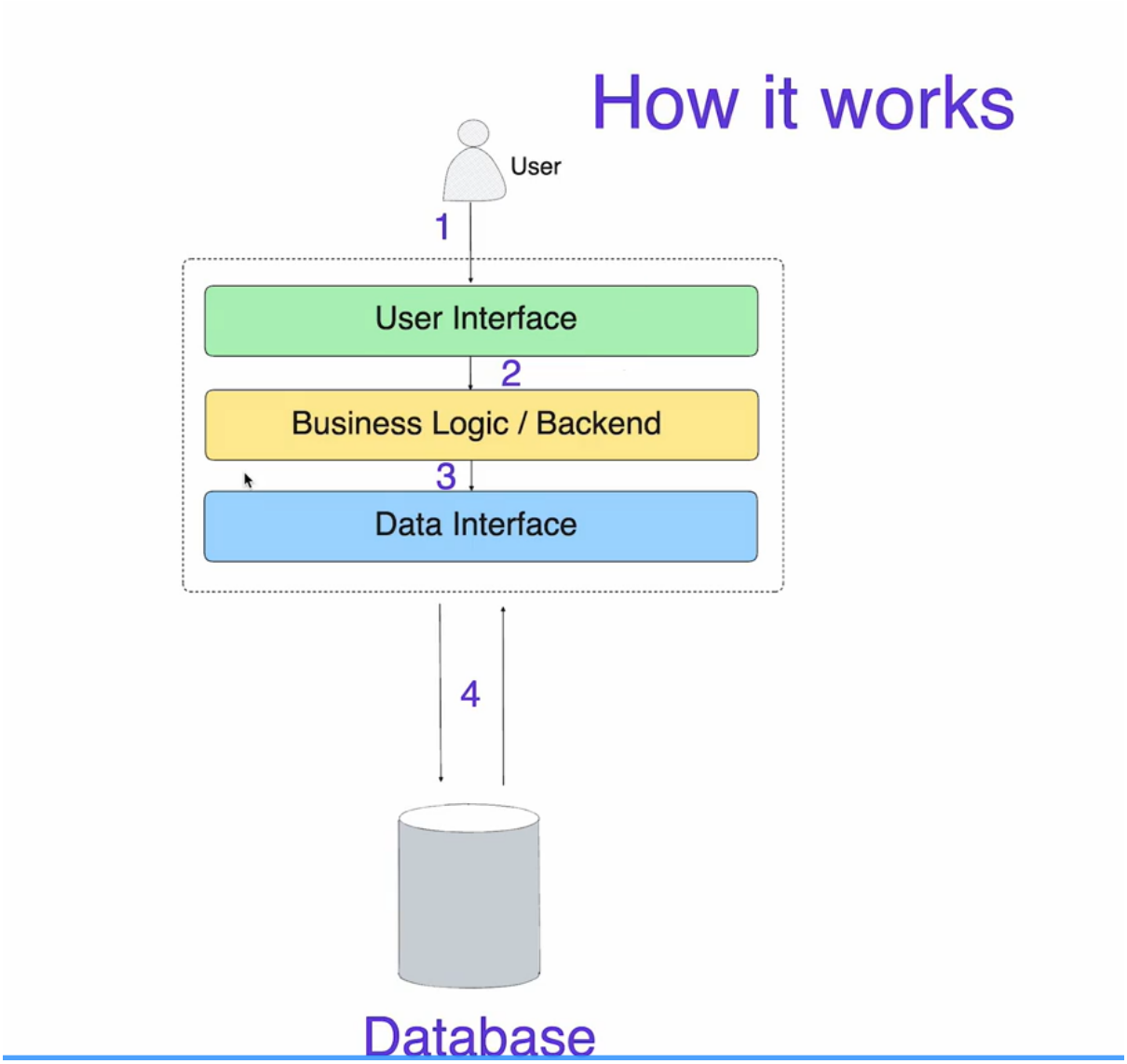
- If you want to update or scale a single monolithic software application, you need to deploy the entire stack, even if this update or scaling only affects one of the components.

(2) How does a monolithic application work?

- When a user takes an action to the user interface, their request is sent to the backend, where the

request is processed. Depending on the type of requests, the backend might interact with the data interface to fetch stored data.

- After the backed processes all the requests, the results are sent back to the user interface, which provides the responses to the user.




(3) What are the pros and cons?

1. Pros: Simplicity and Uniformity.

- For a small application and a small team, using a monolithic architecture makes it simple to deploy all the components on a single code base. This architecture design is most suitable for small developer teams.

- Developers can work in a more unified environment for developing, testing and deploying applications, which simplifies debugging and developing processes.

2. Cons: Scalability, Flexibility and Deployment

- Scaling monolithic applications can be challenging as it often requires scaling the entire

application rather than specific parts or resources due to the tightly integrated nature of the architecture.

- Making changes and updating a monolithic application can be complex and risky because changes to one part of the application can affect other parts.

- Deploying a new version of a monolithic application can be time-consuming and resource-intensive as it requires the deployment of the entire stack of the application.

(3) Why do we care about the monolithic architecture?

1. Legacy systems.

- Applications that are planning to move to the cloud are often built using monolithic architectures, known as legacy systems. Cloud experts must know how these applications are structured to effectively plan and execute their migration to a cloud environment.

2. Integration

- Once monolithic applications are migrated to the cloud, they need to be integrated with cloud - native services. Understanding monolithic application systems helps in identifying the best approach for integration without disrupting the entire system.

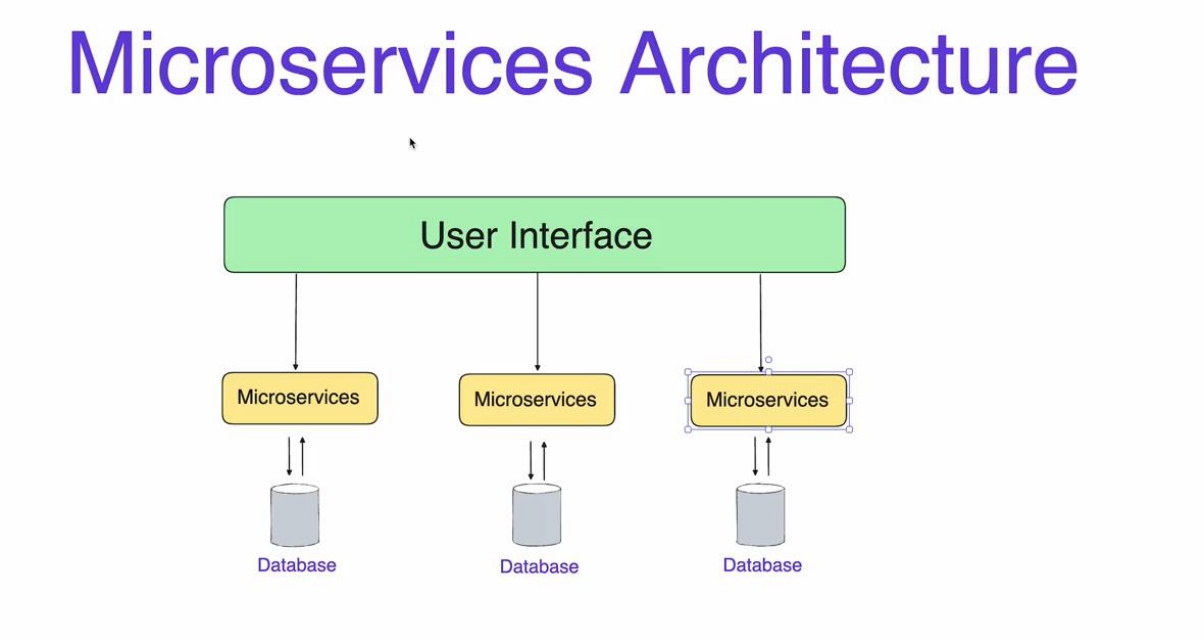
3. In the cloud, scalability is a priority, but monolithic application systems cannot scale as easily as microservices. Cloud experts must know how to handle this scalability challenges of monolithic systems, including vertical scaling (adding more power to the existing system) or horizontal scaling (increasing the number of monolithic systems, managed by a load balancer).

4. Monolithic applications consume resources differently compared to microservice applications. It is important to understand this to efficiently allocate computing resources, such as CPU, memory and storage to meet the performance requirements of monolithic applications.

5. Cloud experts must adopt a cost-effective strategy because monolithic applications are resource-intensive. This involves selecting the right types and sizes of cloud resources that align with performance requirements of monolithic applications to minimize unnecessary costs.

2. Microservices applications

- Microservices architecture is a software application development method that involves a collection of small and independent service units.



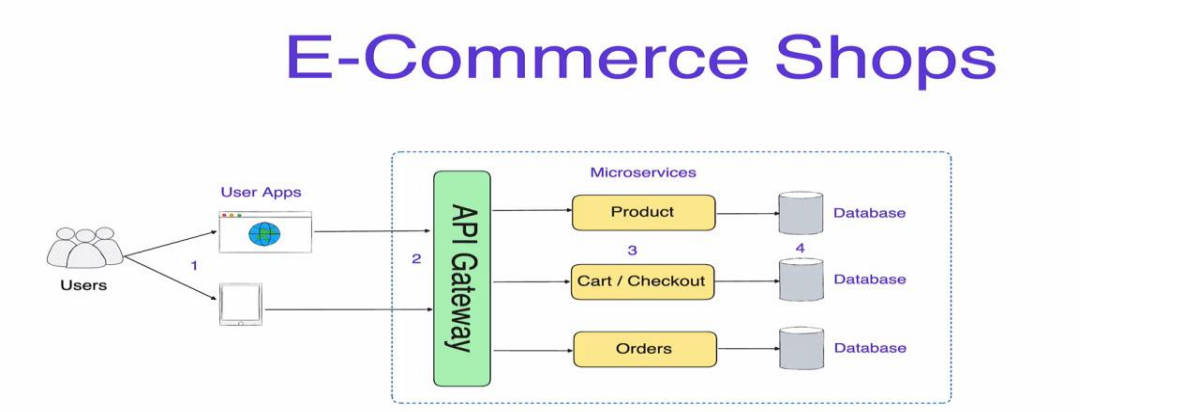

- Each service in a microservice architecture is a self-contained piece of business functionality and operates autonomously.

- The user interface interacts with various independent services, each of which handles a single business capability, such as user authentication, product catalog management, and other processes.

- These services can be developed, scaled and deployed independently of each other and each service has its own dedicated database that ensures loose-coupling and independent scalability. This means that changes to one microservice's database do not impact any other microservice.

- In large companies, each team often works on an independent microservice, with its own business logic, database and codebase, enabling each of them to focus on a specific business functionality that will not affect the rest of the functionalities.

## 2. How does microservices work?




- Users interact with e-commerce platforms through user applications, which could be a web browser or a user application.

- An API gateway is a crucial component in a microservices architecture, acting as a single point

of entry, receiving all clients' requests and routes them to the appropriate microservices. In the context of e-commerce shop, there are multiple microservices handling different business functionalities, such as product list management, managing the cart and checkout, and processing orders, each communicating with its own database.

- Each microservice has its own dedicated database that stores relevant data. For example, the database connected to the product management microservice contains data related to the products listed in the e-commerce application, whereas the database connected to the orders management microservice contains order history and details.

3. What are the importances of microservices?

(1) Scalability

- Microservice architecture allows individual services to scale independently to meet the increasing demands for specific functionalities without scaling the entire stack.

(2) Flexibility

- It enables different development teams to use various programming languages and technologies best suited for their specific service functionality. Each microservice can operate autonomously as part of the overall business platform.

(3) Resilience

- A Failure in a single microservice does not cause the entire application to go down, enhancing the overall system's reliability.

(4) Iteration

- Microservices support agile development practices, enabling rapid deployment and integration of new features with existing system. They also allow for quick bug fixes and updates because developers can focus on a single microservice without needing to update the entire backend system, integrating changes seamlessly with the original components.

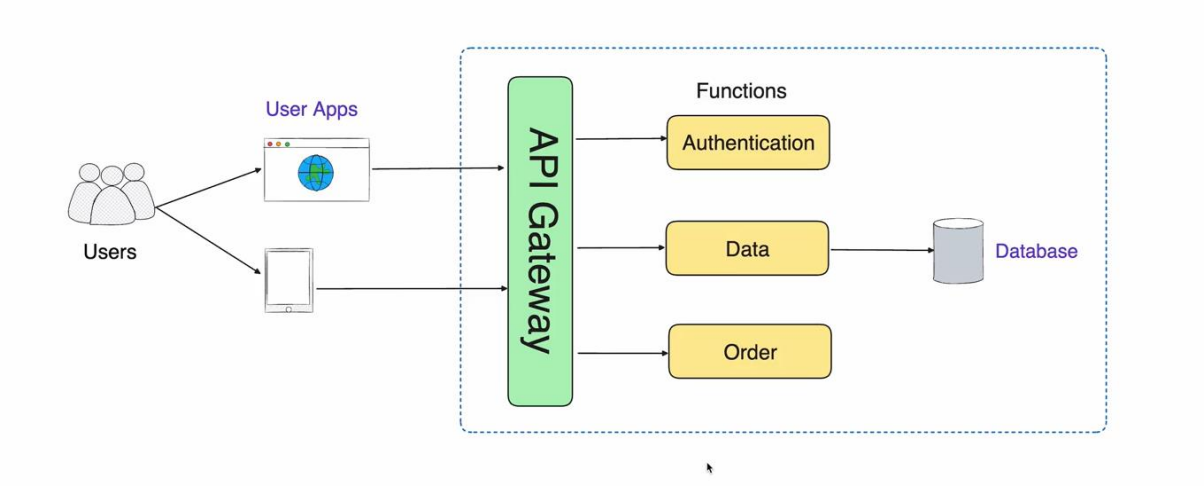
## 3. Serverless Architecture

- Serverless architecture is a cloud application deployment model where cloud service providers dynamically allocate and provision servers as needed. Cloud applications refer to software applications running in the cloud, interacting with end-users. However, cloud services are technologies and tools provided by cloud service providers to support the operations and deployment of cloud applications, such as cloud computing resources and various cloud services like EC2, ELB, ASG and CDN.

- A serverless application runs in a stateless compute container, which are event triggered and fully managed by cloud providers. This includes managing the serverless architecture and handling operational tasks, such as scaling and server provisioning. Serverless applications are software applications that operate within a serverless architecture, allowing developers to solely focus on writing code and deploying functions without the need to manage the underlying infrastructure.

- Stateless means that applications do not retain any previous data or state between executions. When stateless applications execute a specific user request, they reset everything afterward and prepare to execute another request, that enables them to require minimum computing resources and scale efficiently.

- In serverless architecture, developers do not need to own, rent or manage servers. Instead, they can focus on writing code and deploying functions without the burden of managing the entire underlying infrastructure.




- Users interact with software applications, such as web applications, mobile applications or any other interface through various devices. These interactions often involve making HTTP request to an API.

- The API gateway receives HTTP requests from the user interface and routes them to the appropriate backend functions based on the API endpoints specified in each HTTP request. The API endpoints are associated with specific functions and represent the desired operations for each HTTP request, which are processed by the relevant functions.

- Functions handle different tasks. For example, user authentication functions perform user authentication requests by verifying users' credentials and providing tokens for secure access. Data functions create, read, update and delete relevant users' or applications' data. An order function can proceed orders, checkouts and payment procedures.

- The database is connected to serverless data functions and can also be connected to other functions, providing persistent data necessary for each function to operate.

- The database can be a managed service like Amazon dynamo DB, providing seamless scaling and integration with AWS Lambda functions.

- In serverless architecture, cloud service providers invoke and execute functions for HTTP requests. This means that you only pay for the execution time that is running on an event-triggered basis. They also provide seamless scaling to automatically handle increased traffic without the need for manual interventions and disrupting application performance.

4. What is the difference between serverless and microservices

- They are different in how they manage services and resources. In serverless architecture, developers do not need to manage the underlying infrastructure, as the server layer is completely abstracted away. Each component of the software application is hosted by cloud service providers and runs only as needed that makes it cost-effective. It can also scale automatically.

- Microservices break down the entire application into small and independent services. Each service has its own computing resources and communicates with each other through lightweight mechanisms. However, it requires managing the underlying infrastructure, including servers and networking. They always need to be active and running at all times, which differs from the on-demand nature of serverless functions.

- The lightweight mechanism means that microservices communicate with each other through lightweight protocols like HTTP/REST, gRPC or messaging queues that are simpler and less resource-intensive compared to the traditional inter-process communication in monolithic systems.

## 4. API's Explained

- An Application Programming Interface (API) is a means of communication that allows different software systems to interact with servers.

### (1) What is an API?

- For example, when a customer orders something in a restaurant, an API is like a menu listing all the operations that can be performed, which helps developers interact with a service or application. APIs are more related to servers or databases, facilitating interactions between a client (such as a user or an application) and a server or database, enabling data exchange or processing requests.

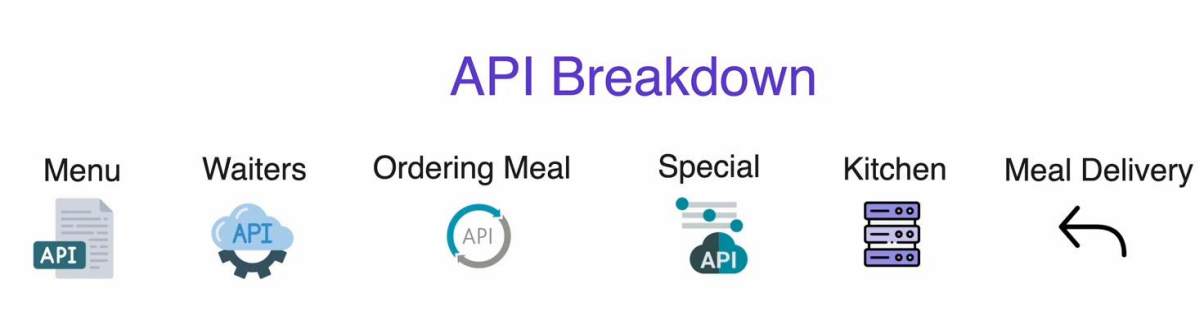
- API documentation provides all the requests you can make, the required parameters and the expected responses. In the restaurant analogy, the waiters act as APIs, handling orders from a customer (requests from users) and bringing them to the kitchen (server).

- APIs take a request from a client, and send it to the server, and return the response back to the client. Ordering a meal is like making an API request, where you specify what you want and how you want it prepared. When you make an API request, you can choose what data you want and how it should be processed.

- You can ask additional request to a waiter to make your dish spicier. Likewise, within an API, special API requests are available by querying parameters that modify how the API functions and customize the response.

- In the analogy, the kitchen represents the server that processes requests and makes appropriate responses.

- A meal delivery is related to the API response. When API requests are processed, the server sends them back to the applications.




(2) What is API important?

1. Efficiency

- An API allow users to communicate efficiently with different software components (Frontend components, backend components, APIs and Microservices) without needing to understand how the servers process their requests.

2. Flexibility

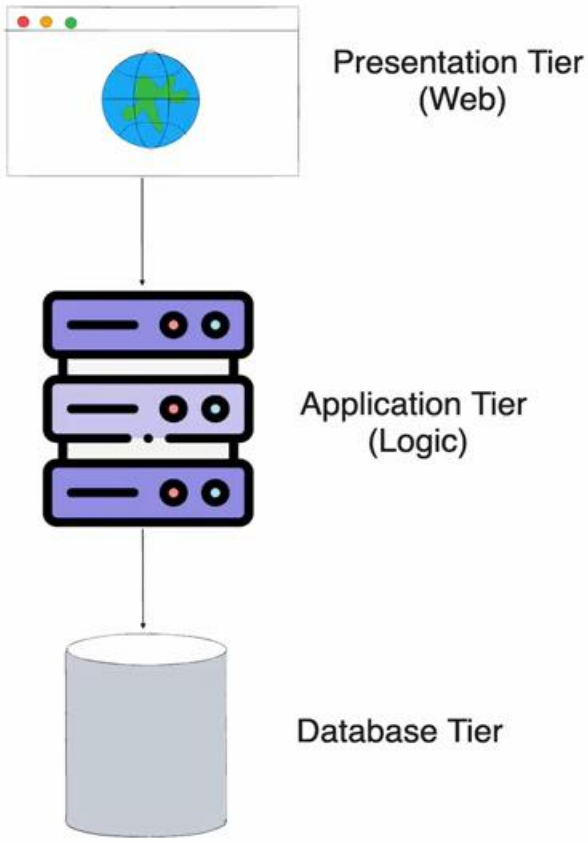
- An API enables customization of data and service requests, similar to how a meal can be tailed to an individual's taste.

3. Integration

- APIs are essential for integrating different services (EC2, Lambda, DynamoDB, S3 and so on) to make complex software applications. They are crucial for integrating various cloud services and automating tasks, which are fundamental aspects of modern cloud application development. APIs serve as the medium, helping developers integrate different services to create unique features of an application.

## 5. Three tier architecture

- Both monolithic architecture and microservices architecture include three common tiers, including user interface, business logic and database, which are the foundations of modern application architecture. These three tiers are developed separately on different platforms.




**(1) Presentation tier (User interface/Client tier)**

- This is the User Interface of applications, providing components such as web pages and web applications needed for interaction between software applications and users.

**(2) Application Tier**

- It contains the business logic or the rules of the applications, where core functions are executed, including processing data, performing calculations, and connecting to the database.

**(3) Database Tier (Data storage tier)**

- This is the database and database management layer, responsible for storing, securing and retrieving the applications' data.

2. What is the importance of three tier architecture

(1) Scalability

- Each tier can scale independently, allowing cost-effective handling of growing demands without

affecting other tiers.

(2) Maintainability

- The separation of concerns makes it easier to manage and update each tier independently. Each tier has a specific and distinct role, for example, the presentation tier handles the user interface, the application tier manages business logic and the database tier stores and retrieve data. Each tier can be modified independently without disrupting the others.

(3) Reusability

- Business rules and logic within the application can be reused by multiple front-end applications, improving efficiency and consistency in how your applications work, regardless of how users interact with it. For example, both web and mobile versions of your application can use the same business logic, reducing redundancy without needing to recreate the logic repeatedly.

(4) Flexibility

- Changes and upgrades in technologies can be implemented only one tier that will not affect the others, providing greater flexibility in development and maintenance.

**3. Monolithic architecture and Microservices architecture**

- Both architectures incorporate the same tiers, including presentation tier, application tier and database tier.

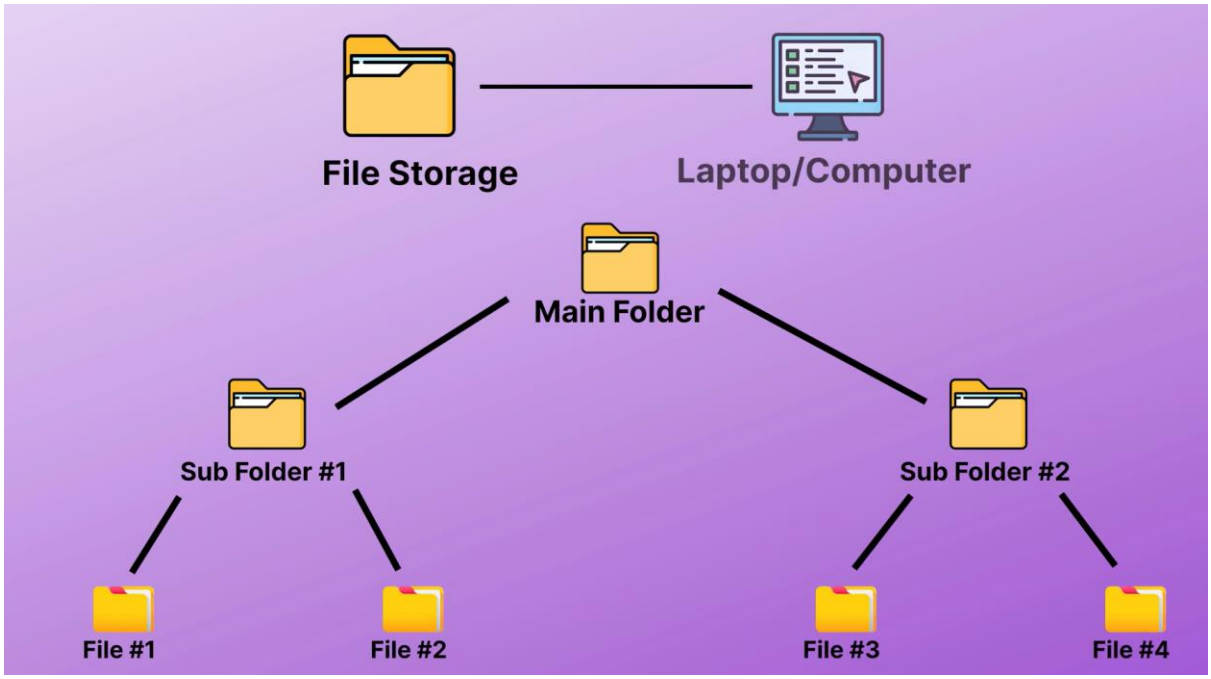
- However, microservices build these tiers separately, allowing each tier to scale, update and change independently. This leads to greater cost-effective, flexibility and resource efficiency. In contrast, monolithic services encapsulate all three tiers within a single and unified service and each service cannot scale or update independently, leading to less cost-effective and higher resource consumptions.

## 제6강. Cloud storage options (Object, Block, File)

- There are 3 primary storage options, including file storage, block storage and object storage, each with unique characteristics and use cases.

(1) File storage

- File storage organizes data in a hierarchical structure of files and folders. If managed by the cloud, file storage allows employees to access shared files such as documents, spreadsheets and presentations.




(2) When to use File storage:

- When applications require a shared file system.

- For user directories or home directories, which means that there are multiple employees in a company using individual directories so file storage can provide an individual directory within a shared file system and the company can maintain the centralized control over these multiple individual directories.

- For storing files and folders in a structured hierarchy.

- For legacy applications that were designed to interact with a file system.

(3) Advantages of File storage:

- File storage has familiar structure that most people have seen before.

- File storage supports file-level operations such as open, close, read, write and file directory navigation.

(4) Disadvantages of file storage:

- It is not as scalable as object storage.

- The performance of file storage degrades with high concurrency when a large number of users access the file storage simultaneously. This can lead to the increased latency and slower responsive times for operations such as read and write operations.

- In AWS, file storage is provided Amazon EFS.

2. Block storage

- Block storage stores data in uniformly sized blocks, each with a unique identifier and managed independently. Blocks are stored in Storage Area Network (SAN) environments and can be combined as needed to make a larger storage structure.

- A Storage Area Network (SAN) is a high-speed network providing high I/O capabilities for databases, allowing them to quickly read and write data to block storage. It consolidates data stored in uniformly sized blocks into a comprehensive storage system, offering improved performance, scalability, redundancy and high availability for data recovery.

- Database often needs block storage's high I/O capabilities to quickly access and write data.

(2) When to use block storage?

- For databases and transactional data requiring high I/O rates.

- When running virtual machines or containers, where each needs a file system, a block storage can provide reliable, high-performance storage solution, acting as an independent disk for each VM or container.

- For applications that need raw, unformatted storage that can be managed by operating systems. The block storage can provide raw and unformatted data, without a predefined structure (such as a file system). The operating systems or applications can manage this unstructured data based on their needs, giving full control over how data is organized, managed and utilized.

(3) Advantages

1. High performance, especially for high read and write speeds, suitable for high-demand applications like databases and virtual machines.

2. Low latency

3. Fine-grained control of storage: data is stored as uniformly-sized block and each block can be managed independently as an individual hard drive, providing more flexibility in data management.

4. Each block can be controlled as an individual hard drive

(4) Disadvantages of Block Storage:

1. More expensive than file or object storage due to high I/O capabilities and fine-grained control.

2. Less scalable in capacity and management compared to object storage.

3. Requires more manual intervention and overhead for scaling, compared with object storage that can handle more massive unstructured data.

4. In AWS, block storage is provided by Amazon EBS.

5. Blocks contain portions of data and are stored on physical hardware drives separately and independently. A Storage Area Network (SAN) can control over these blocks, providing centralized control and management over these blocks with high I/O capabilities necessary for databases or other high-performance applications.

3. Object storage

- Object storage manages and stores data as objects, each including data itself, metadata (information about the data) and a globally unique identifier (like a URL of web contents or static assets).

- In a valet parking system analogy, when you hand over your car which is data, you get a ticket which is a unique identifier using for retrieval. Valets store cars as objects efficiently, regardless

of their size or type.

(2) When to use

- For large sets of unstructured data, such as photos, videos and logs. Object storage is suitable for storing unstructured data which does not fit neatly into rows and columns, such as photos, videos, audio files, and logs.

- For data that needs to be accessed via HTTP/HTTPS such as web contents.

- For archiving and backup systems due to its scalability. It can store a huge amount of data cost-effectively, suitable for long-term storage and disaster recovery.

(3) Advantages

- Highly scalable with unlimited capacity and lower cost than block or file storage.

- Object storage can be accessed from anywhere via standard web protocols like HTTP/HTTPS. This storage is fully compatible with HTTP/HTTPS, ideal for storing web contents, including video, images and audio files and web assets (static assets) like HTML, CSS, JavaScript and so on that are essential components for hosting static websites.

- Every object stored in the object storage has a unique URL. Users or applications can retrieve these objects by sending HTTP/HTTPS requests to these URLs. For example, if you have an image stored in the object storage, you can access it using its URL like https://bucket-name.s3.amazonaws.com/image.jpg.

- Object storage can be integrated with CDNs, caching web contents in the object storage and moving them closer to users, reducing latency and enhancing users' experiences.

- Object storage provides rich built-in metadata capabilities, powerful for search, categorization and analytics. Objects can be stored with metadata, including attributes such as object size, date created and last modified that allows users to search and manage a large volume of data efficiently.

**(4) Disadvantages**

- Not suitable for traditional databases or applications, needing a file system with frequent small update with low-latency random access.

- It typically has higher latencies than block storage, which is not suitable for high-performance applications requiring rapid data access.

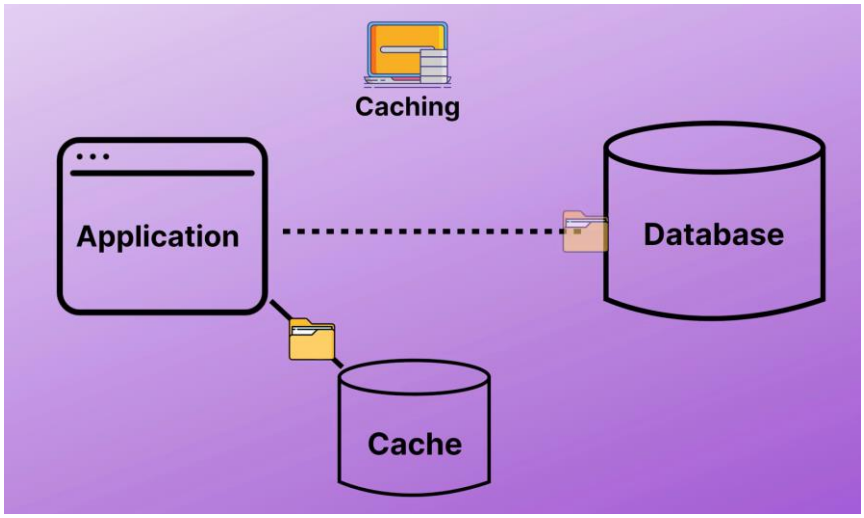
- It is not ideal for data that requires frequent and complex updates because objects need to be rewritten entirely rather than modified in specific points, making it ideal for 'write once and read many' scenarios.

- In AWS, object storage is provided by Amazon S3.

Each storage option should be chosen based on application requirements, performance needs, scalability and cost considerations.

## 제7강. Caching

- Caching is a technique temporarily storing data in a cache, which is a temporary storage location.




- This technique enables faster data retrieval by fetching data from the cache rather than from the primary storage location.

- Caching helps applications speed up access to frequently requested data, enhancing overall system efficiency and performance. Caching saves time and resources by avoiding unnecessary trips to the main storage every time data is requested.

(2) There are 3 key reasons why caching is important

1. Performance

- One of the primary advantages of caching is the reduction of response times and lower latency for applications. Applications can retrieve frequently accessed data from the cache much quicker than fetching the data from the main database every time it is needed.

- This speed enhancement improves user experiences, especially for data-intensive applications where even milliseconds of delays can affect user satisfaction.

2. Scalability

- As user requests increase, the response time of back-end databases can decrease due to larger workloads, potentially leading to system failures. Caching mitigates this issue by serving a large portion of requests directly from the cache, thereby reducing the workload on the back-end database.

- It allows applications to maintain performance standards even as they scale to support more users or transactions.

3. Cost

- In cloud environments, resource usage is directly related to increase in operational costs. Caching offers significant cost savings due to the fact that accessing data from the cache requires less computational power than querying the original database.

- Reducing the number of reads and writes to the primary database lower data transfer and storage I/O costs, leading to substantial financial benefits.

(3) Different types of caching

1. Browser caching

- Browser caching involves storing files such as HTML pages, images and stylesheets (CSS files) locally in the users' browser.

- When users visit a website and navigate it, the browser loads these files from the browser cache rather than retrieving them from the server again, that reduces server load and network latency, leading to faster page loads.

## 2. Content Delivery Networks caching

- A CDN is a distributed network of servers that work together to deliver the Internet content quickly. CDN servers can cache contents at various locations geographically closer to the users, reducing data travel time and resulting in reduced website load times and bandwidth costs.

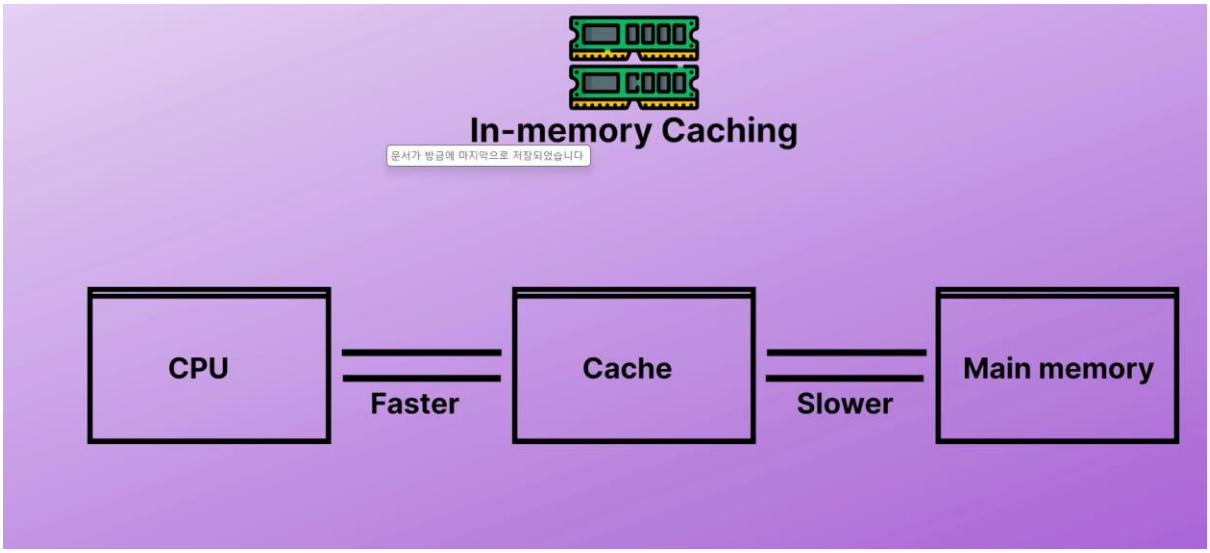
- A CDN uses a network of distributed servers to serve contents to users via the closest servers, reducing latency and improving load times. Bandwidth is a maximum rate at which data can be transmitted over a network connection, meaning that how much data can be transferred in a given amount of time.

- The most popular CDN is Amazon CloudFront.

## 3. In-memory caching

- In-memory caching stores data in the RAM of servers using technologies like Redis and Memcached. It is much faster to read data from memory rather than fetching it from the main storage.

- In-memory caching is used for data that requires frequent access but not permanent storage, such as session information, temporary calculations and frequently read database queries.




## 4. Database caching

- Database caching temporarily stores the results of expensive database queries. When queries are made by users, the systems first check if the results of those queries are stored in the cache. If they are, the cached results are returned quickly, avoiding the need to re-execute the queries against the primary database.

- This speeds up read operations for complex queries and heavily accessed data.

## 5. Application caching

- It stores data used by applications to prevent repeated operations, including user session data and user preferences, making applications more responsive. Application caching can directly implement in-memory caching (RAM) or use external caching systems like Redish and Memcached (both of them have their own RAM) for scalability.

Using the proper caching systems results in optimizing resources usage, reducing costs and improving overall performance of systems and applications.

## 제 8강 Load Balancer vs API gateway

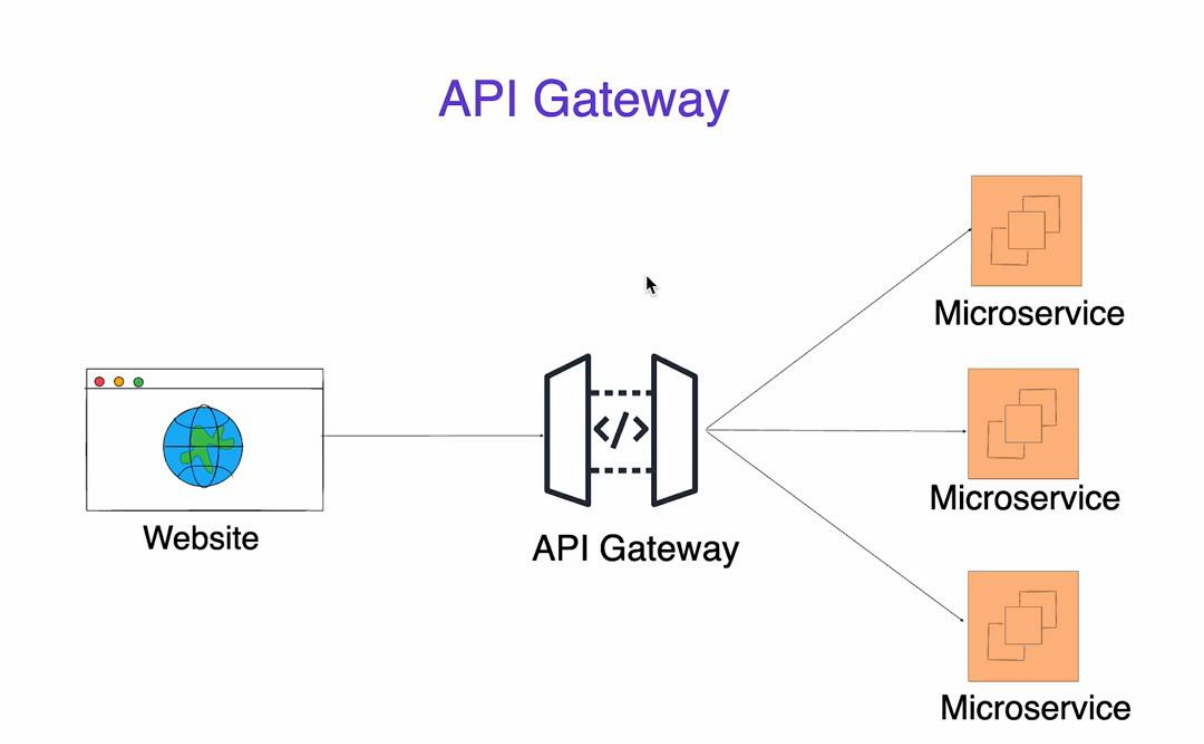
(1) A Load Balancer distributes incoming traffic across multiple servers, ensuring that no single server becomes overwhelmed, thereby it increases reliability, availability and performance of applications.

(2) Load balancers can be installed as hardware devices, such as traditional and dedicated appliances in on-premise data centers or as software solutions, such as virtual machines running in the cloud.

(3) Load balancers distribute incoming traffic to the least busy or closest servers based on algorithms like Round Robin, Least Connections and IP Hash, enhancing applications responsiveness and availability. In conjunction with an Auto Scaling Group, Load Balancers can also scale server resources in response to traffic spikes or reduce servers during low demand, ensuring efficient resource utilization.

(4) In addition, Load balancers incorporate functionalities of an Application Delivery Controller also known as ADC, including SSL offloading, caching, WAF integration. The ADC offloads the burden of encryption and decryption workloads from servers, allowing them to focus on delivering contents rather than encryption tasks. It also caches frequently accessed web contents such as images, scripts and HTML pages to improve delivery speed. Finally, it integrates a Web Application Firewall (WAF) to protect servers from malicious requests coming from the frontend systems.

(4) Load balancers are critical components in minimizing performance issues, resulting in high availability of applications.




- An API gateway acts as a single entry point that manages all API calls/requests from clients to back-end services. It decouples the user interface from the back-end microservices, allowing consistent communication between them. This decoupling means that the user interface does not need to know about the changes or updates on the back-end services, allowing the back-end services to be updated or scaled independently.

- The API gateway handles and coordinates API calls/requests efficiently, rerouting them to the back-end services, performing load balancing, request transformations, or even authentication that enforces security polices and rate limiting. It also compiles responses and send them back to the clients.

- The API gateway simplifies communication between the user interface and microservices by abstracting the complexity of the back-end infrastructure. It handles security concerns, SSL termination (encryption and decryption process), authorization and authentication, allowing only legitimate requests. This reduces the overall overhead on back-end services and keeps internal services hidden and protected from direct external access.

3. What are the differences between Load balancer and API gateway?

1. A Load balancer purely focuses on traffic distribution, while an API gateway typically manages API requests, including protocol translation, response aggregation from multiple microservices and API management features.

2. A Load balancer directs traffic to the optimized servers to prevent a single server from being overwhelmed for high availability and high performance of applications, whereas API gateway ensures the requests are routed to the appropriate services, handling additional functionalities such as protocol translation and aggregating responses.

3. A Load balancer uses algorithms for distributing traffic, whereas API gateway offers advanced functionalities such as API security, authentication and authorization, rate limiting and monitoring, logging and API transformation that facilitates communication between different

protocols or technologies stacks.

(1) Protocol translation

- API gateway converts one protocol to another to facilitate communication between user interface and back-end servers. The user interface and backend servers often use different and incompatible protocols. An API gateway can translate these different protocols, enabling efficient communication between the systems. By acting as a translator, the API gateway ensures that data and requests can flow seamlessly between the user interface and backend servers, despite their different protocols.

(2) Response aggregation

- If a client requests several API calls, the API gateway distributes each API call to different servers and aggregate responses to compile them into a single response and send it back to the client.

(4) Rate limiting

- The API gateway restricts the number of API calls that a user can make to prevent resource abuse, server overload and fair usage among users.

(5) Authentication and Authorization

- User authentication is the process of verifying the identity of a user or client, addressing the question, “Who are you?”.

- If the user is authenticated, the API gateway checks whether the user has the right permissions to access a specific resource, addressing the question, “What can you do?”.

(6) Monitoring

- Monitoring involves tracking and analyzing the performance and health of API calls, including response times, error rates and request volumes.

(7) Logging

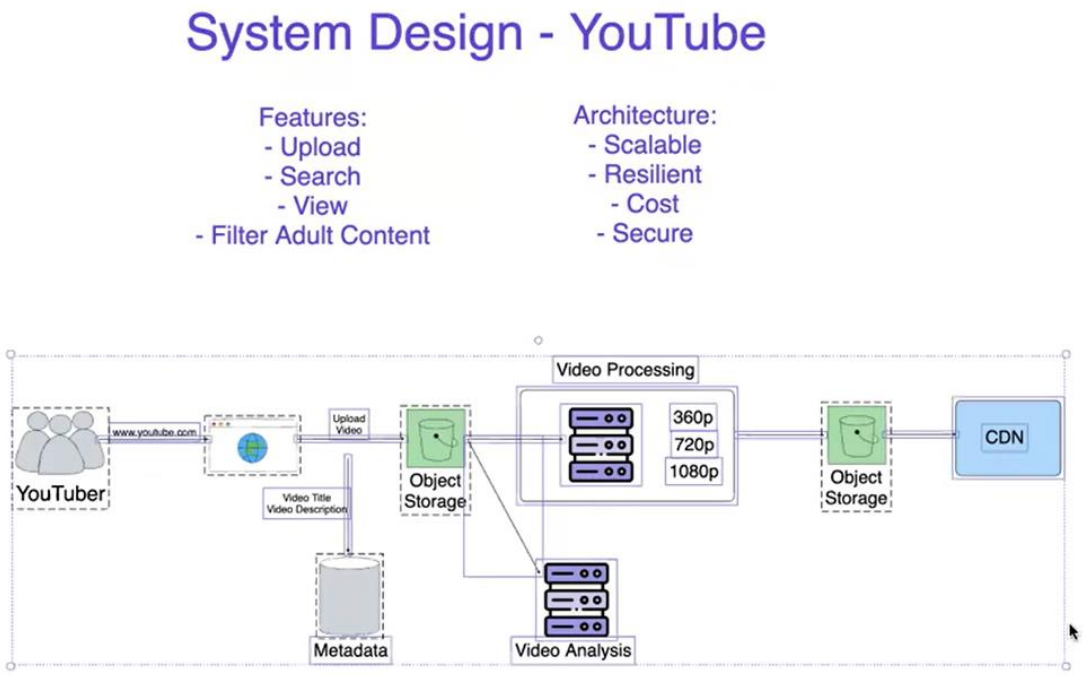
- Logging records all incoming requests, outgoing responses and details of each interaction, useful for debugging, auditing and analyzing.

(8) API transformation

- API transformation involves modifying or transforming incoming requests or outgoing responses to meet specific requirements, including changes in data formats, such as converting from JSON to XML format.

## 제 9강. System Design – Youtube

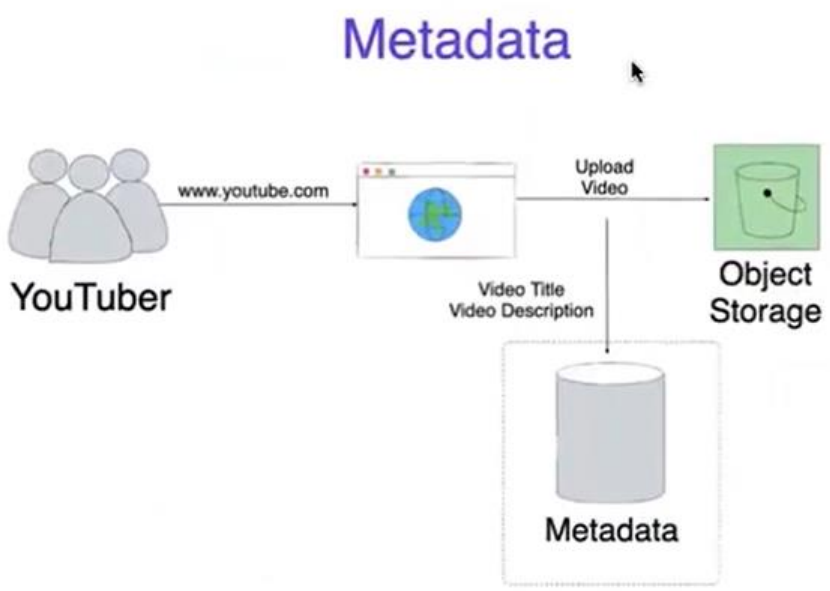
- In this lesson, we learn how YouTube – one of the largest video streaming platforms - handles millions of user requests while maintaining high availability and security.




- Understanding the features, architectural components, and functionalities required to design a video streaming platform is essential.


(1) Accessing YouTube and Uploading features

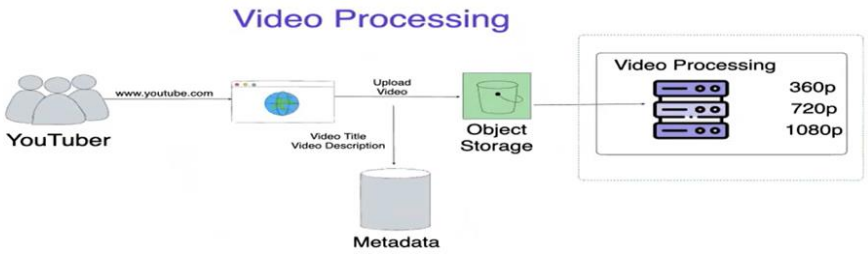
- When users access YouTube via 'www.youtube.com' and upload videos in formats like 4K or MP4, these files need to be stored in a robust storage solution. Among the three different kinds of cloud storage options, object storage is the most appropriate option to store media files due to its scalability and durability. For example, in AWS, Amazon S3 is used, ensuring that video files remain securely stored and accessible even if a data center fails.



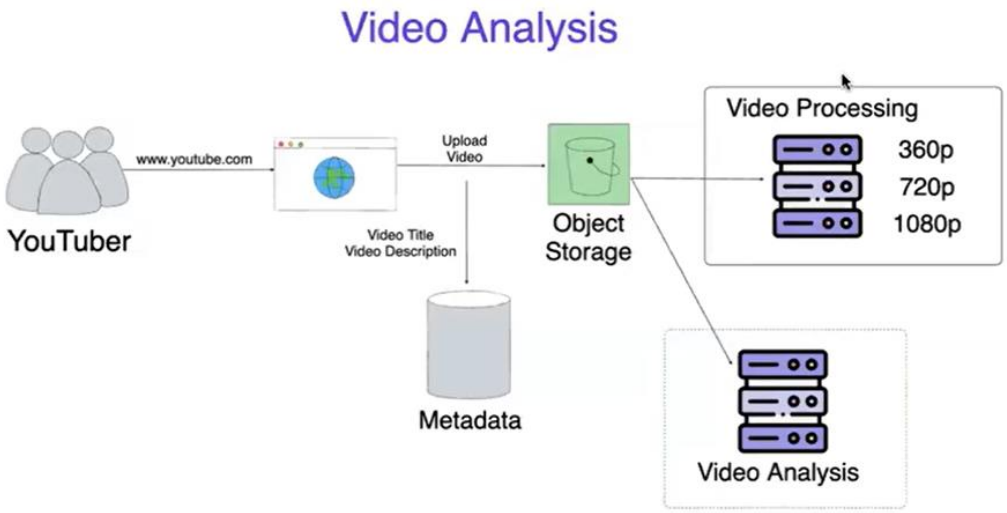

- Video files are stored alongside metadata, including detailed information such as video titles, video descriptions and video tags, which are text-based information. This free-form text-based metadata can be stored in object storage or a NoSQL database like DynamoDB or OpenSearch, which are more suited for handling free-form data and scaling horizontally compared to traditional relational databases.

- This free-from and text-based metadata can be stored in a relational database like RDS but the relational database struggles to scale horizontally due to its inherent complex relationships and strict constraints on structured data so we need to use a NoSQL database like DynamoDB or OpenSearch, avoiding these complexities and strict constraints on relational integrity. NoSQL database can scale horizontally, especially ideal for free-form data.

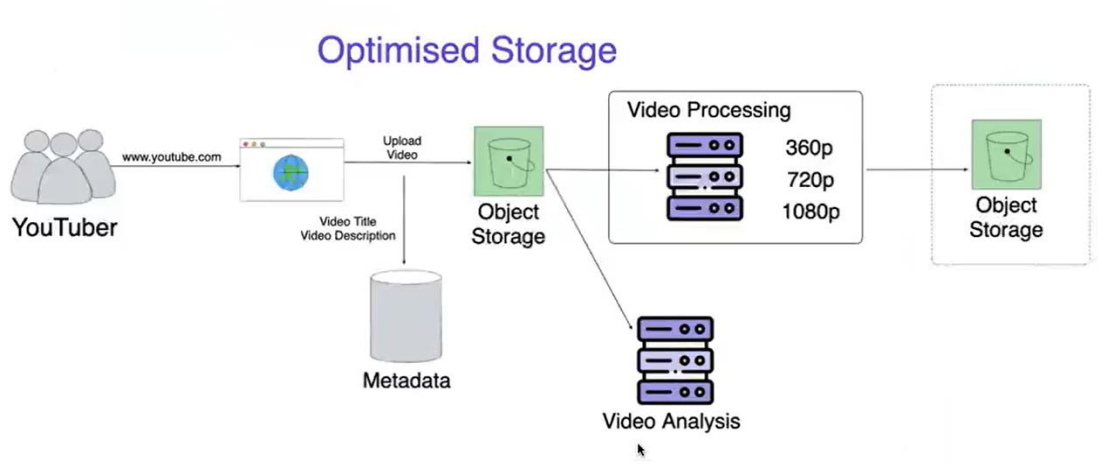
- After video files being stored in NoSQL database and the object storage, videos processing and analysis will be started.




- Once uploaded, videos undergo processing, including security checks and conversion into multiple resolutions (e.g., 360p, 720p and 1080p). Any microservice or serverless function can convert the original video files into multiple resolutions to serve them to users with different devices and internet speed, increasing accessibility. For example, not all users can handle 4K videos, so converting to lower resolutions enables users to access them, making the contents more widely available.



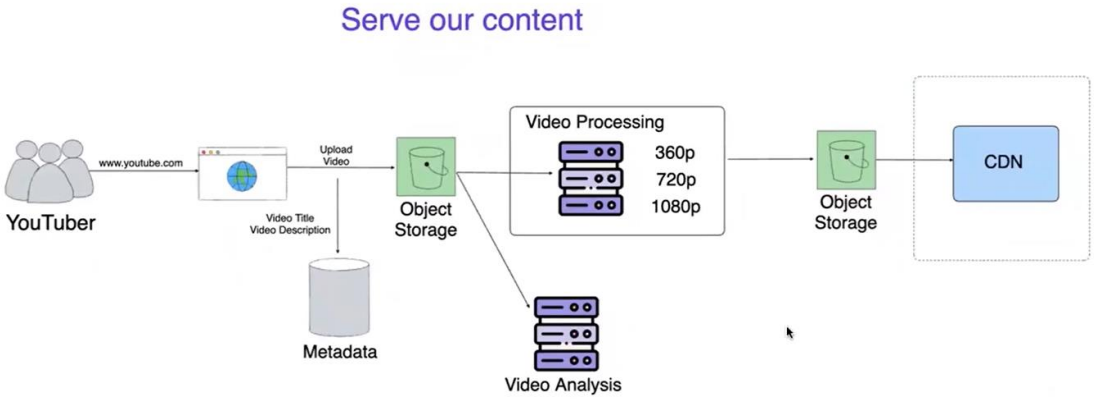

- Simultaneously, video analysis is performed on different servers or business logic, using AWS Rekognition, which utilizes machine learning algorithms to automatically analyze content and scan for inappropriate material, ensuring a safe and secure environment on the platform. If any inappropriate content is detected, it is removed before being made available on the platform. These processes are triggered as soon as contents is uploaded to the object storage.

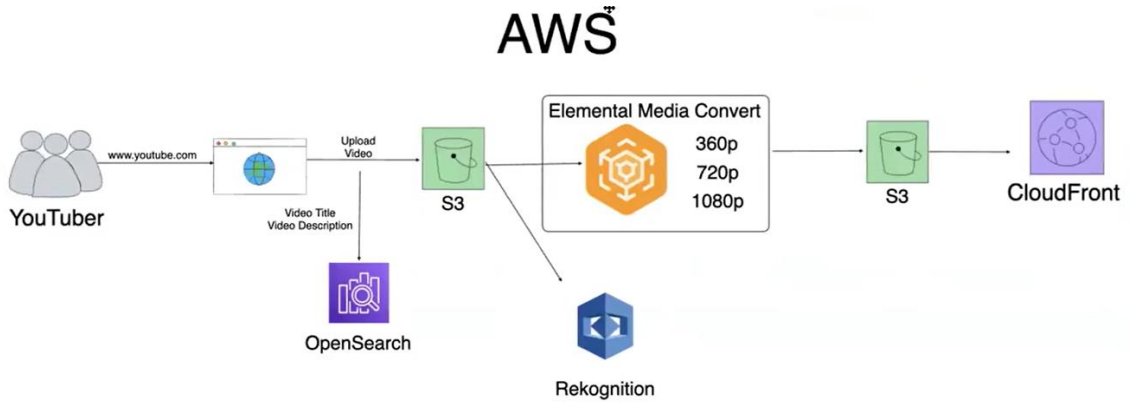



- After video processing and analysis, videos are stored in another object storage, organized into different resolution folders for streaming. AWS Lambda functions and AWS Step Functions are often used to automate tasks like organizing video files into specific folders based on resolution. For example, if a video file is converted into 360p, AWS step functions make a folder for 360p resolution and place the file into that folder.

- AWS lambda functions are triggered in response to specific events, and are stateless, meaning that each lambda function execution is independent and does not retain the state of previous executions. A specific event does not affect another execution, ideal for isolated and event-driven tasks. However, AWS step functions manage and orchestrate entire workflows composed of multiple Lambda functions and other AWS services. They are stateful, meaning they remember the execution state and track each step and retry any failed process to orchestrate the successful completion of the entire workflows.

- And finally, videos are streamed to users with low latency by using a CDN. The CDN caches the video content at any edge location nearest to viewers, reducing latencies and enhancing the user experience. When a viewer in the US watches a video for the first time, it is served from the object storage. For subsequent viewers in the same location, the CDN provides the cached video, providing high-speed content delivery.

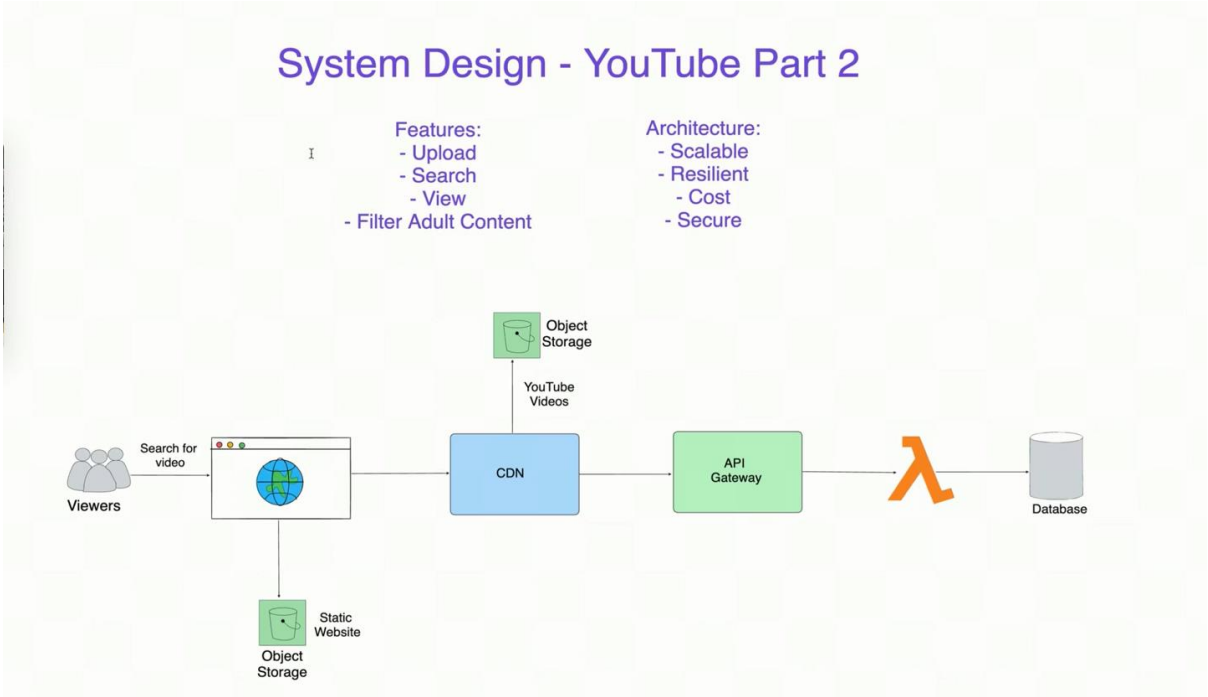




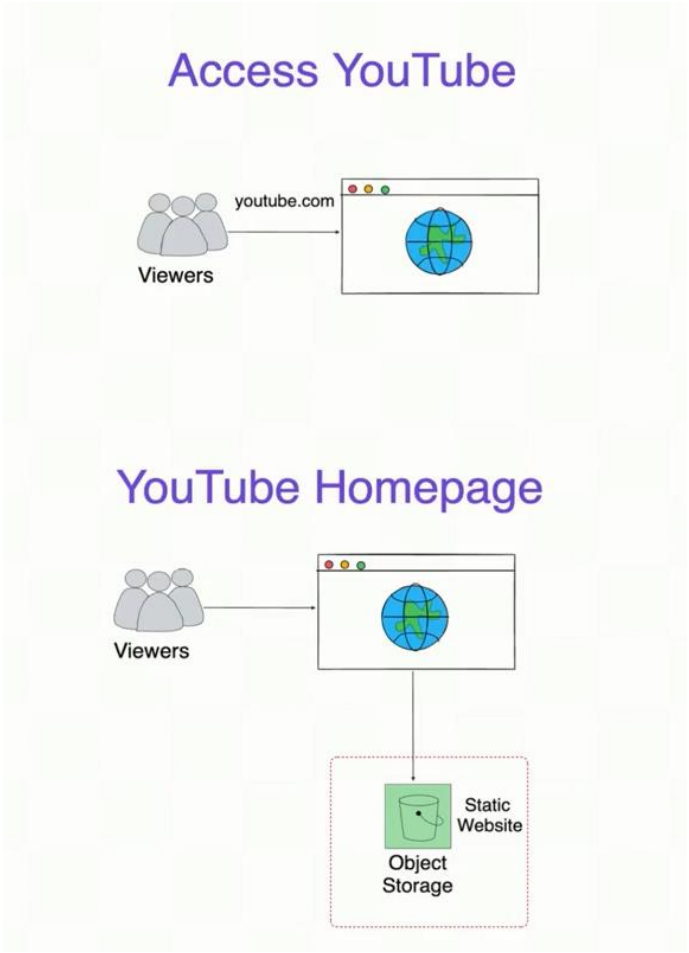

- OpenSearch is suitable for storing and indexing YouTube videos, offering efficient search and retrieval capabilities for users. It allows for quick, full-text search capabilities and can handle large volumes of data.

## 제 10강. System Design – YouTube 2

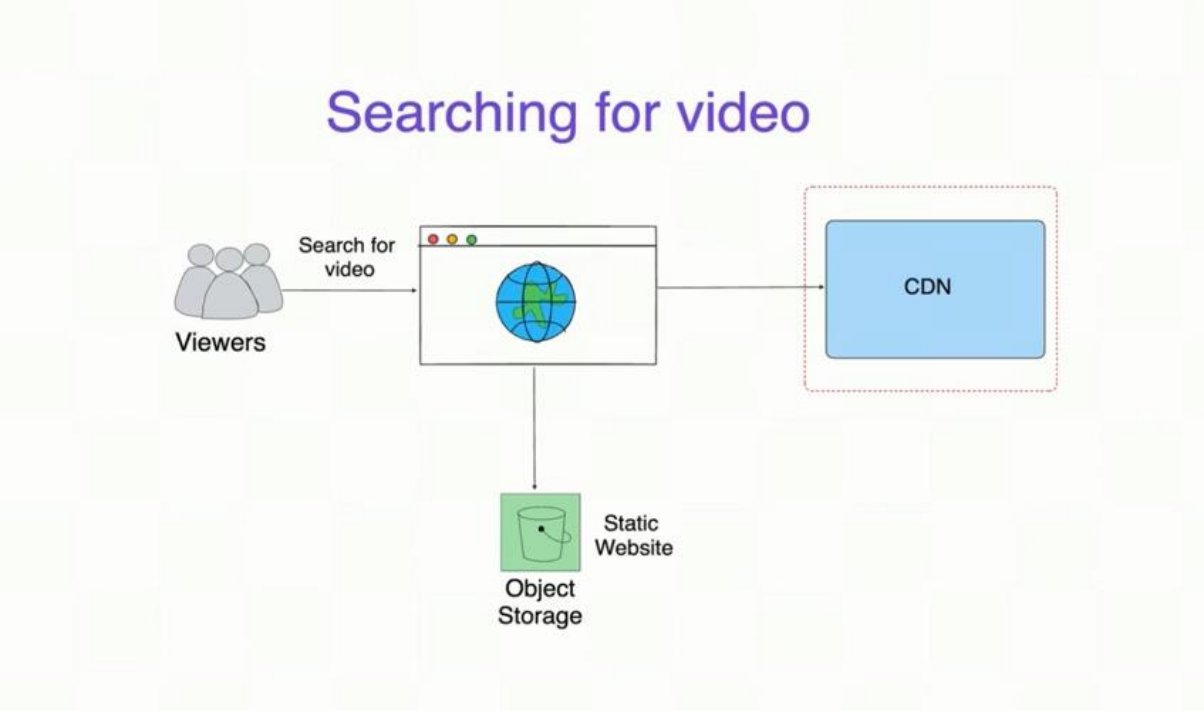
- In this lesson, we will learn about the process of watching a video on YouTube and searching for a video and how these processes are handled by the system.




(1) Searching for a YouTube video journey

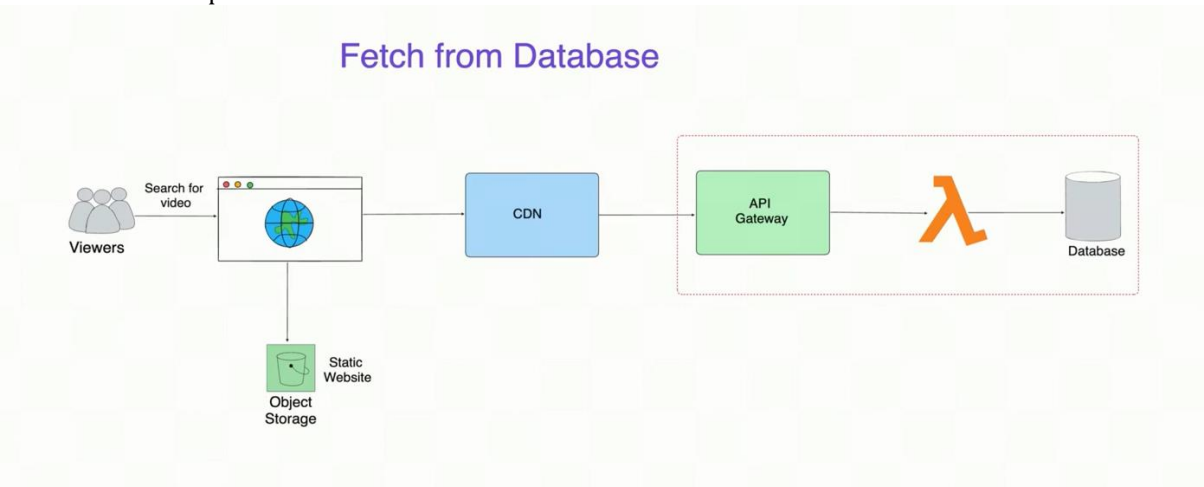



- Users access YouTube via www.youtube.com where they are presented with the main page of YouTube. This home page is a static website composed of HTML, CSS or other front-end frameworks. The static website consists of static contents such as text and images, which are stored in Amazon S3, an object storage service that is ideal for storing these static contents.

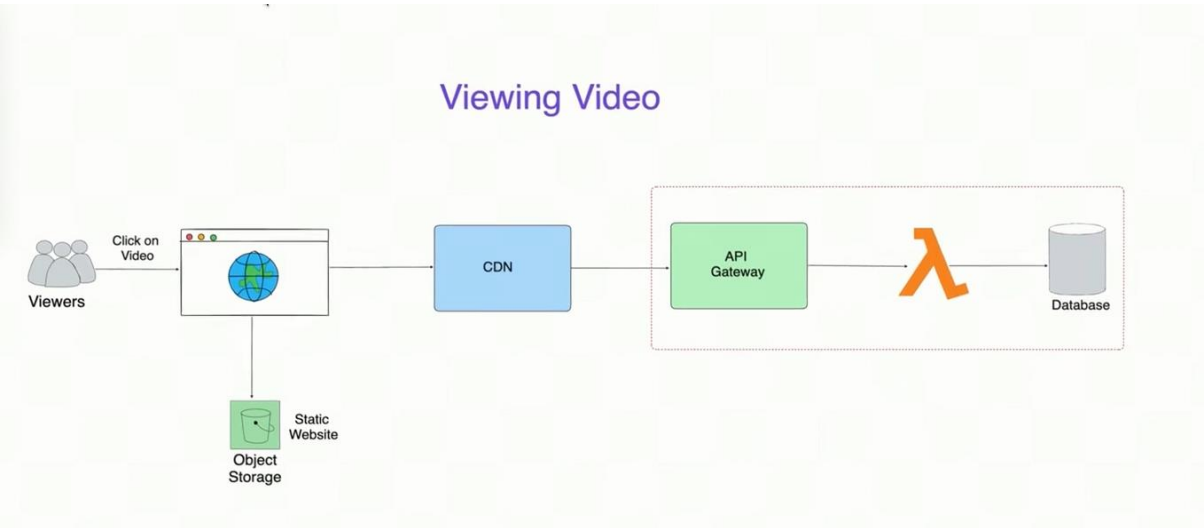



- When a user searches for a specific channel, our request is routed to a CDN and the result is initially served by the object storage. For subsequent cases, the same video will be served directly from the CDN, which instantly delivers the video to users.

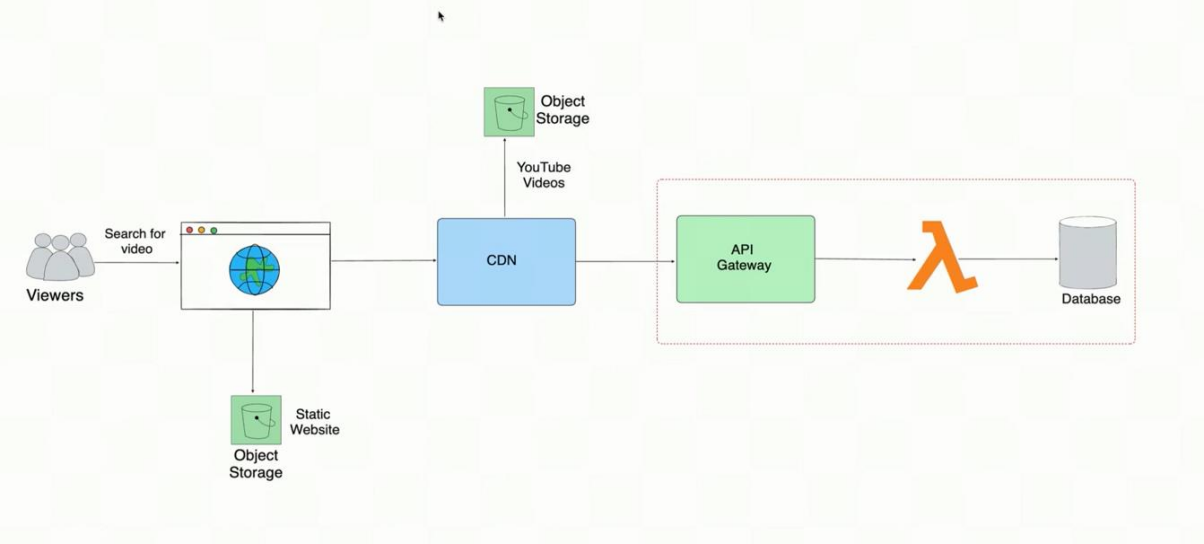
- However, if the search query is not stored in the CDN, what happens then?
- When a user searches for a video using a keyword in the search bar on YouTube, this request is transformed into the search query and is sent to the API gateway. There are 2 different responsibilities handled in this step.




- First of all, the search query is sent to an API gateway. The API gateway routes this request to a Lambda function, which is triggered to search a database containing relevant metadata that matches the video search query. This process handles the search functionality, displaying a list of video content for the user. After that, the result of the query (a list of video file information (such as titles, descriptions and other metadata stored in the database) is sent back to the user. This allows the user to see a list of videos that matches their search keyword.
- When the user searches for a video using a free-form, text-based keyword, this query triggers the Lambda function, which retrieves metadata from the database and displays the results on the YouTube website. At this point, the website transitions from static to dynamic, showing content processed by backend systems.




- The second responsibility is serving the actual video to the user. If a user clicks on a specific video from the list and the video is cached by the CDN, the CDN servers the video immediately.
- However, if the video is not cached in the CDN, the CDN retrieves the video content from the object storage. After that, this video is cached at the nearest edge locations by the CDN, and for future requests from other users in the same geographic location, the CDN servers the video directly from the nearest edge location, providing high-speed content delivery.




How is scalability managed in YouTube.
- YouTube's scalability relies on using CDNs and object storage, both of which are ideal for handling the large-scale storage and efficient delivery of media files like videos. The CDN reduces latency by caching content close to users, while object storage provides scalable, durable storage for vast amounts of video data.

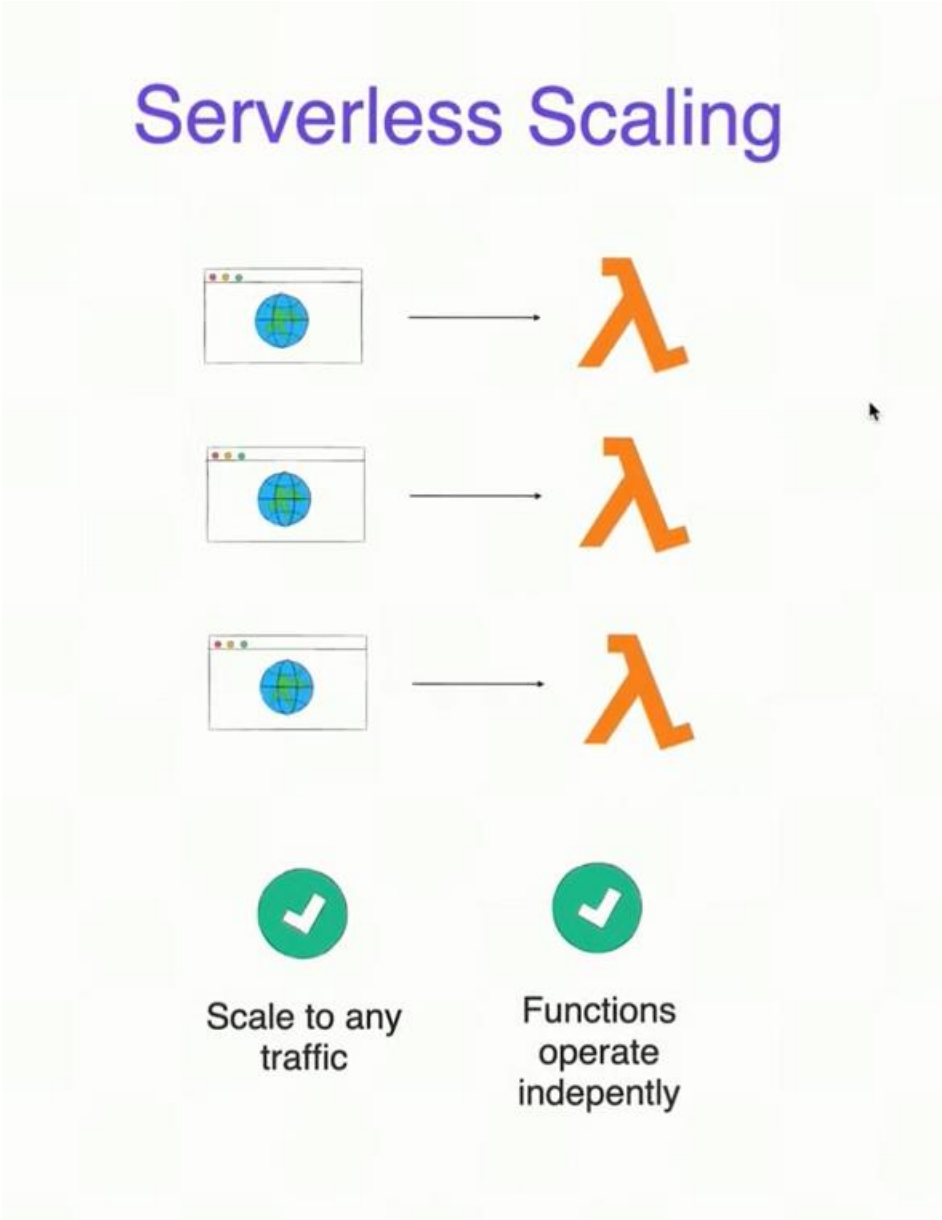
## 제11강. Server vs Serverless scaling

### 1. Horizontal scaling vs Vertical scaling

- Server is a virtual machine and vertical scaling involves increasing the capacity of an existing server by upgrading its resources, such as moving from a smaller instance type (t2.micro) to a larger one (m4.2xlarge). This process typically requires stopping the old instance, transitioning to more powerful instance and rerouting incoming traffic to the new instance.
- The main advantage of the vertical scaling is simplicity, just adding more CPU, memory or storage to a single server but the disadvantage is the inherent capacity limit to how much a single instance can be scaled. There is a maximum capacity for upgrading a single server, beyond which further scaling is not possible.
- However, horizontal scaling involves adding more servers or virtual machines to handle increasing loads

by using Auto Scaling Group (ASG) to manage the scaling process, adding or removing instances based on demand, enhancing high availability and fault tolerance of applications. This approach distributes the load across multiple instances and can scale indefinitely, allowing applications to handle large and variable workloads more effectively.

2. Serverless Scaling




- AWS Lambda functions automatically manage the scaling of applications without the need for manual intervention. When incoming traffic increases, AWS Lambda triggers additional instances of the function as needed, enabling applications to adapt to any amount of traffic. And Lambda function can operate independently and automatically scale the number of virtual machines without manual intervention.
- Developers can focus on code and AWS Lambda functions automatically scale the number of instances and provision them.

Week 3 – System Design Applications Project

Project 1 – Create your Architecture
Designing and Comparing Architectural styles
Objective:

You will design a simple e-commerce application using monolithic architecture, then refactor it into microservices, and finally, propose a serverless approach for certain functionalities. This exercise aims to provide practical insights into the advantages, challenges, and use cases of each architectural style.

## Part 1. Monolithic Architecture Design

### 1. Design a Simple E-commerce Application:

(1) Define the core functionalities: user registration, product catalog, shopping cart, and order processing.

- In the monolithic diagram for a simple E-commerce application, there are different business functionalities that we should define: user registration, product catalog, shopping cart and order processing.

(1) User registration

- User registration is quite related to user interface and business logic. A user logs into the website via the Internet or mobile device and the user interface displays the web page and handles registration form for the user. The business logic processes the user data, checks for existing user accounts or store new user information through the data interface. The data interface fetches user information and store new user information into the database.

(2) Product catalog

- Product lists are displayed to users, that is fetched by the business logic through the data interface. The business logic fetches product information from the database and the data interface fetches product details from the database and update new product information in the database.

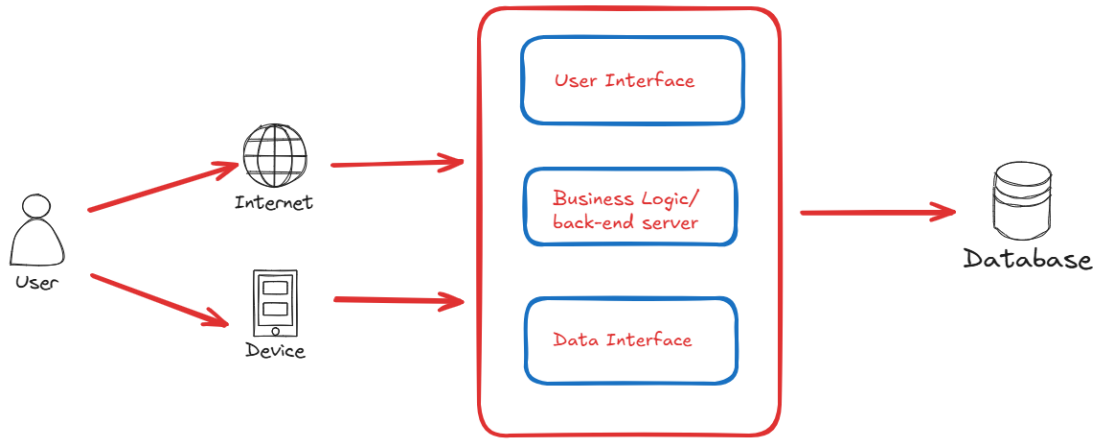
(3) Shopping cart

- A User updates its shopping cart by adding or removing items from the cart through the user interface. The business logic updates and manages the cart operations, such as calculating totals and applying discounts and ensure all items are available. The database interface saves the cart status and retrieve the cart-related data from the database.

(4) Order processing

- A user confirms and submits its order through the user interface and the business logic processes order validation and payments, checks inventory and updates order status for the user. The database interface tracks order statuses and update inventory levels from the database.

(2) Create a high-level architecture diagram showing the UI, business logic/backend, and data interface layers.



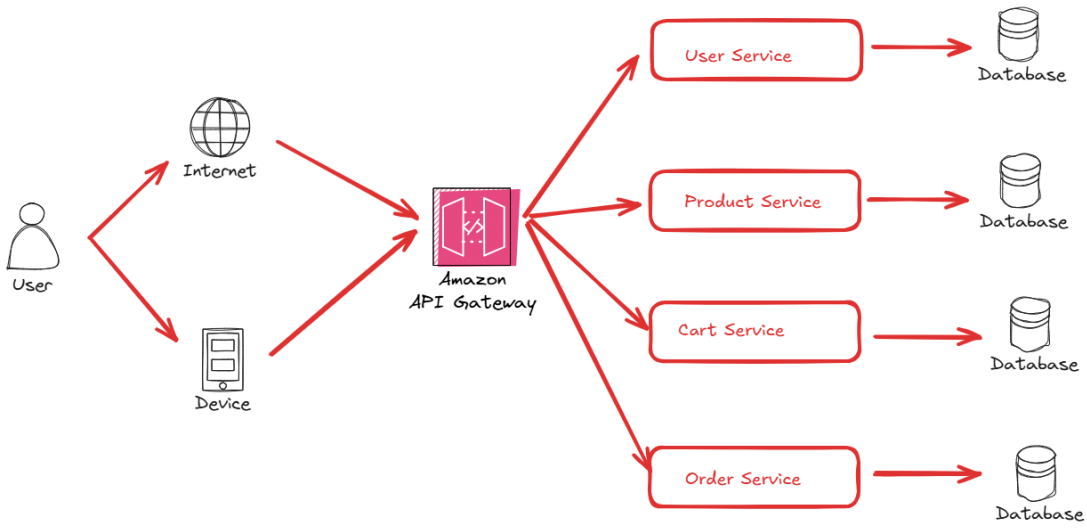

(3) Discuss the simplicity and uniformity advantages and address the scalability, flexibility, and deployment disadvantages within the context of your design.

- All components (User interface, business logic and data interface) are tightly integrated and suitable for small-scale applications with limited functionalities. Deployment is straightforward, with only one package to deploy and manage.
- However, when demand increases, the entire application needs to be scaled, which is more complex and resource-intensive than scaling a single component independently. They are tightly coupled with one another and less flexibility because any change in a specific part affects other functionalities, making the

system less adaptable to change.
- When we deploy a new service or functionality, even if it is minor and small updates, we should redeploy the entire application, that is time-consuming and riskier in terms of potential downtime.

## Part2. Refactoring into Microservices

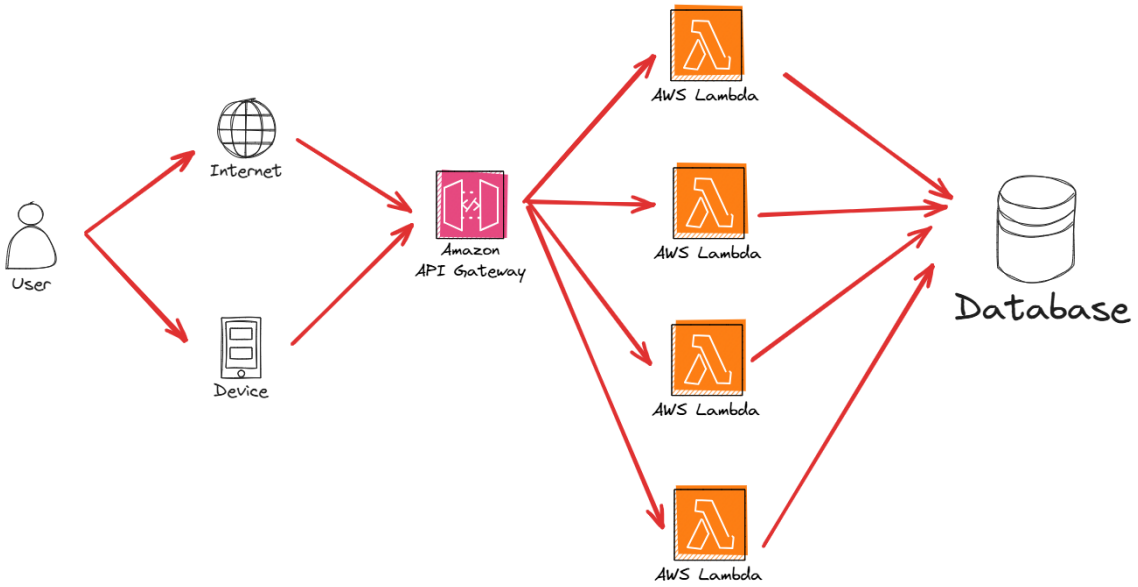
1. Identify and Design Microservices:
(1) Break down the monolithic design into microservices (e.g., User Service, Product Services, Cart Services, Order Service)
(2) Draw an updated architecture diagram showing the microservices, their databases, and how they interact with the UI and each other.




(3) Explain how this refactoring addresses the scalability and flexibility issues. Discuss the challenges in deployment, development, and maintaining consistency.
- Microservices architecture consists of isolated and independent components that can scale independently based on the demand for specific functionalities. This addresses scalability issues because only the needed services are scaled rather than the entire application.
- Any change or update in a specific component does not affect other functionalities that provides flexibility. This independence allows for more efficient development and deployment of new features because it does not require redeploying the entire application.
- However, microservices can introduce challenges in maintaining the entire application because each microservice is deployed separately and independently that needs to be integrated and synchronized with other services to maintain the application's integrity and seamless functionality. Any isolated update and change do not disrupt the entire system, requiring robust integration and testing strategies.

## Part3. Incorporating Serverless Architecture

1. Serverless Functions for E-commerce:
(1) Identify components of the microservices design that could be effectively handled by serverless functions (e.g., user authentication, payment processing).
- Components of the microservices design that could be effectively handled by serverless functions include user authentication, payment processing, order processing, and other event-driven tasks like sending notifications or handling file uploads.
(2) Update the architecture diagram to include serverless functions and their triggers.




(3) Discuss the benefits of serverless architecture in terms of scaling, cost, and operational management. Highlight the differences and potential challenges compared to microservices.
- AWS Lambda functions abstract the complexity of the management of underlying infrastructure. Lamba functions automatically scale in response to the volume of incoming requests without manual interventions. Developers do not need to concern about the entire underlying architecture but solely focus on the code.
- AWS Lambda functions are cost-effective due to their event-driven and stateless nature. They do not remember the previous state between executions that leads to save a significant amount of computational resources. They run only when triggered by events that can optimize resource consumption compared to microservices running continuously.
- Developers still manage the underlying servers and networks for running microservices that is abstracted by serverless architecture, reducing operational burden.
- Microservices have dedicated databases that can help them scale and operate independently but AWS Lambda functions share a single database that simplifies maintaining the overall consistency and integrity across functions. However, this serverless architecture occur a bottleneck under heavy load due to the increase in incoming traffic if the shared database cannot handle the increase load, leading to downtime or performance issues.

Part4. Comparison Report:
(1) Write a report comparing the three architectural styles, focusing on scalability, development complexity, deployment, maintenance, and cost implications.

(1) Scalability
- Serverless architecture > Microservices > Monolithic services
- Serverless architecture abstracts the complexity of underlying infrastructure, enabling automatic scalability. Lambda functions scale automatically based on the increase in incoming traffic. In microservices, each microservice can scale independently but developers still consider the complexity of scaling the number of servers and the network interactions between them. Monolithic services require re-deploying the entire stack in order to scale a specific component, making this approach resource-intensive and time-consuming.
(2) Development complexity
- Microservices > Serverless architecture > Monolithic services
- Monolithic services are tightly coupled, allowing all components to be developed and deployed simultaneously, suitable for small businesses. Microservices, on the other hand, can be developed independently without affecting other functionalities and often use dedicated databases, which can improve efficiency. However, developers need to consider the underlying infrastructure, such as provisioning new servers when launching a new microservice. Lambda functions can also be developed and launched independently when a new functionality is needed for an application but developers do not need

to consider the underlying infrastructure, as this is handled by cloud service providers.
(3) Deployment
- Microservices > Serverless architecture > Monolithic services
- Monolithic services deploy the entire stack, even when only minor changes are made to a single component. Both microservices and serverless architectures allow for the deployment of new functionalities without affecting other backend servers. However, microservices use dedicated databases so any update must be integrated with other services to maintain the integrity and consistency of the application. In addition, microservices require consideration of backend servers and other underlying infrastructure to deploy new functionalities, unlike serverless architecture, abstracting these layers entirely.

(4) Maintenance
- Monolithic services > Microservices > Serverless architecture
- Monolithic services are tightly coupled so any update or change in a single component requires redeployment of the entire stack, making the process time-consuming and inefficient. Both microservices and serverless architecture allow specific components to be updated easily, but microservices require changes to be integrated with other functionalities, often involving different dedicated databases for service integrity and consistency. This integration can be more complex than in serverless architecture, typically using a single database and the entire underlying maintenance process can be taken by cloud service providers.
(5) Cost implications
- Monolithic services > Microservices > Serverless architecture
- Due to its tightly coupled nature, monolithic service is time-consuming and intensive-resource. Lambda functions are stateless and event-driven, that significantly reduces resource consumption because each function is only triggered when incoming traffic is received via the API gateway. This contrasts with microservices, where all components must run continuously, leading to higher resources use.

(2) Include a section on the suitability of each architecture style for different types of projects (e.g., small vs. large scale, consistent load vs. variable load).

(1) Small vs Large scale
- For small businesses, monolithic services are appropriate because the overall architecture is simple to develop and deploy. As businesses grow larger, teams may use microservices with dedicated database for different business functionalities, which is more efficient in terms of resource consumption. However, microservices still involve complexity, as developers must manage the underlying infrastructure and updates to each microservice need to be integrated seamlessly into the overall application. Lambda functions are suitable for larger companies that need to focus solely on code without handling the maintenance process and they can easily maintain the consistency using a single, automatically scaled database managed by cloud service providers, which enhances availability and consistency.

(2) Consistent load vs. Variable load
- For consistent loads, microservices are more suitable than serverless architecture because each component can always run, ensuring consistent application performance. For fluctuating demand, serverless architecture is more appropriate as Lambda functions are event-triggered, allowing users to pay based on the actual runtime of the functions, which can be cost-effective. While microservices can provide consistent performance, serverless functions may introduce latency due to cold starts but offer more cost-savings for variable loads.

Week 3 – System Design Applications

Project 2 – Design Scalable Architecture

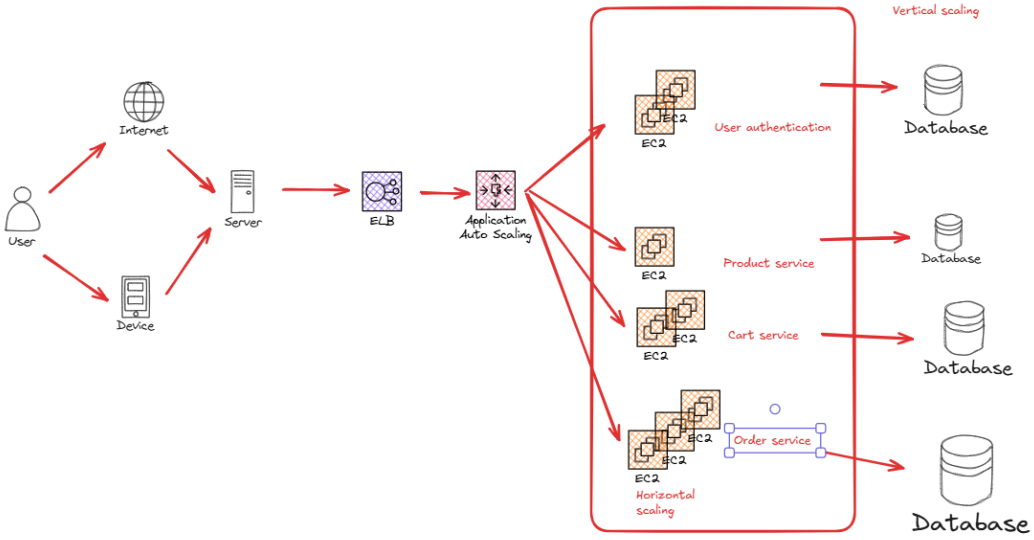
For this project, you will design an architecture for a scalable web application. You will create two versions of the design: one using traditional server scaling (both vertical and horizontal) and another leveraging serverless technologies. This exercise aims to illustrate the practical applications of load balancing, API gateways, and different scaling strategies in real-world scenarios.

## Part 1: Traditional Server Scaling Design

1. Design Overview:
(1) Propose a web application idea (e.g., e-commerce platform, social media site) and outline its core functionalities.

- A user accesses an e-commerce platform via the public internet or mobile device which retrieve static contents from a web server. A user's request is routed to a load balancer which distribute incoming traffic to backend application servers, serving different application servers, such as user registration and authentication, product list-up, shopping cart update and order processing. Each microservice retrieve necessary data from dedicated database. When user traffic changes, Auto scaling Group (ASG) adjust the number of backend servers based on needs.

(2) Create a high-level architecture diagram incorporating load balancers, web servers, application servers, and databases.




## 2. Vertical and Horizontal Scaling:

(1) Explain how you would implement vertical scaling for your database layer.
(2) Design a horizontal scaling strategy using an Auto Scaling Group for the web and application server layers. Include a load balancer in your architecture to distribute incoming traffic.

- For implementing vertical scaling for databases, computing power such as CPU, Memory or Storage must be larger to adapt to increased incoming traffic from application servers. A web server serves static contents such as images, HTML, CSS and so on to a user and send HTTP requests to backend servers via a load balancer to retrieve dynamic contents such as video files from the backend database.
- If incoming traffic from web servers increases, a load balancer distributes traffic across multiple servers to prevent them from being overwhelmed and also health-checks for high availability of application services. An auto-scaling group automatically increases the number of backend servers when the load balancer receives a significant amount of traffic.

3. Discuss the benefits and limitations of each scaling strategy in the context of your application.
- The vertical scaling enhances the computing power of databases by enlarging the size of CPU, Memory and Storage. This improves the I/O capabilities of each database to speed up the response for each user request. However, there are limitations to increase the size of the computing power, meaning that it is definite scalability.
- For horizontal scaling, it adjusts the number of instances based on demand. If incoming HTTP requests from users increase, this scaling method increases the number of backend servers. However, if incoming HTTP requests from users decrease, this scaling method decreases the number of them via auto-scaling group.

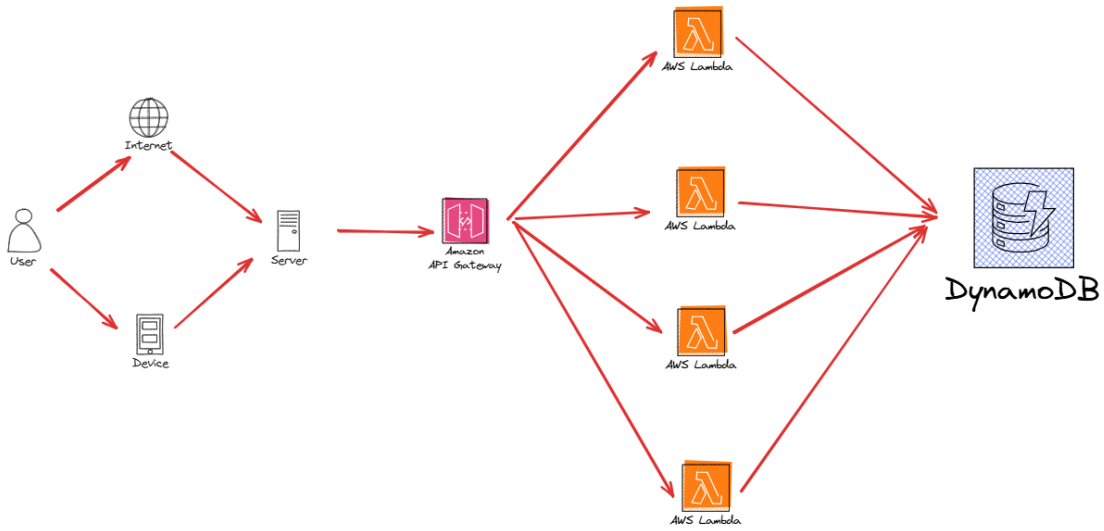
## Part 2: Serverless Architecture Design

### 1. Serverless Components:

(1) Redesign the architecture of the same web applications using AWS Lambda functions for backend processing.

- If chosen serverless architecture, AWS Lambda functions abstract the complexity of underlying infrastructure and process all incoming traffic via Amazon API gateway. Users access web pages via the public internet or mobile devices and web servers serve static contents such as images to the users. If users request dynamic contents, web servers forward that requests to the backend functions to retrieve data from the Dynamo DB or other databases.

(2) Incorporate other serverless services (e.g., Amazon DynamoDB, API Gateway) to support your application's functionalities.




(3) Include a diagram showing how these serverless components interact with each other and how they are exposed to the front end.

- AWS Lambda functions automatically scale the instances of functions when incoming traffic via Amazon API gateway increases. In the serverless architecture, developers do not need to consider the underlying infrastructure such as provision instances. If user requests forwarded by AWS Lamba functions increase, serverless NoSQL database service, DynamoDB automatically increases its computing power such as CPU, Memory or Storage to respond to the increased requests.

### 2. Scaling and Management:

- Discuss how serverless architecture simplifies scaling and management aspects of the application.
- Highlight the differences in scaling behavior between serverless and traditional server-based scaling, emphasizing aspects like cost, operational overhead, and scalability limits.

For microservices, it needs auto-scaling groups to adjust the number of backend servers. Developers need to handle the underlying infrastructure for scaling and managing them. In contrast, in serverless architecture, this manual process is completely abstracted and handled by cloud service providers. Serverless Lambda functions automatically adjust the number of instances running functions.

Traditional server-based scaling requires higher operational overhead costs because developers must provision instances based on demands. Backend servers operated as microservices typically run all the time that is inefficient for fluctuated loads. In terms of scalability limits, especially for vertical scaling, there are limitations on enlarging the size of instances and for microservices, if there are too much backend servers, the maintenance charge for integrating all servers is resource-intensive and cost-ineffective.

However, serverless architecture saves operational overhead due to its abstraction of the underlying infrastructure maintenance. In addition, serverless functions are event-driven that trigger functions only when incoming traffic is forwarded, which is cost-savings. There are no limits to run functions and automatically increase the instances of functions as needed.

## Part 3: Comparison and Analysis

1. Performance and Cost:

(1) Compare the two architectures in terms of performance under varying load conditions and cost-efficiency at scale.

(2) Consider the implications of each architecture on development, deployment, and maintenance.

2. Use Cases:

(1) Identify scenarios where one architecture might be preferred over the other, considering factors like application complexity, traffic patterns, and operational capacity.

## Deliverables:

1. Architecture Diagrams: Detailed diagrams for both traditional server scaling and serverless architecture designs.
2. Design Document:

(1) A comprehensive document explaining each design's components, scaling strategies, and rationale behind architectural decision.

(2) A comparative analysis of both architectures, focusing on scalability, cost, operational complexity, and suitability for various application types.