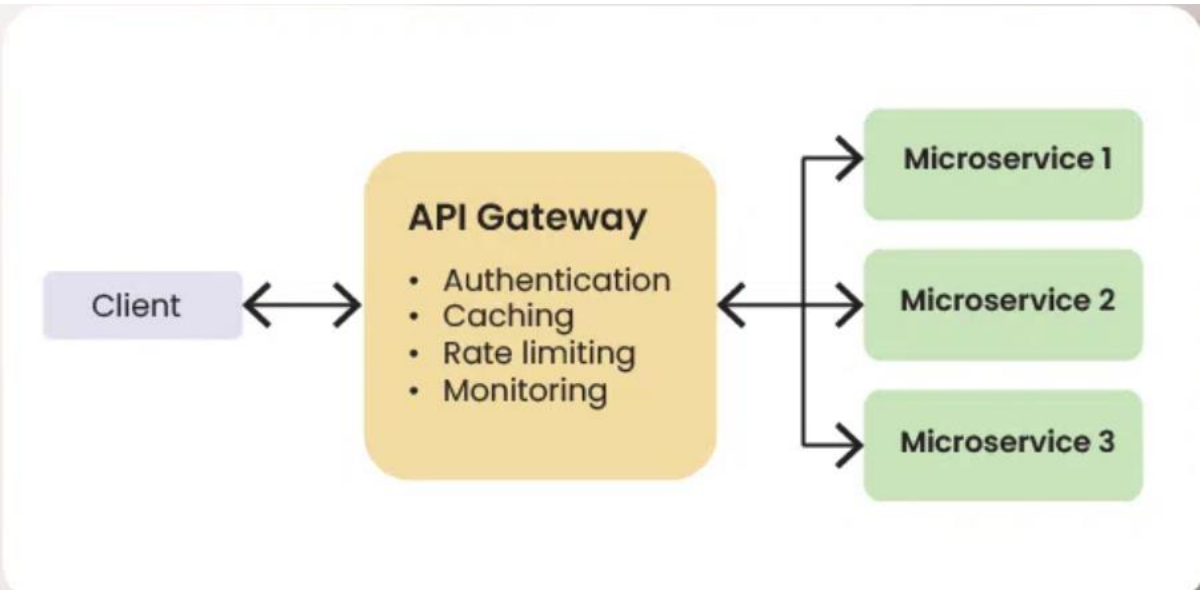
What is API Gateway | System Design ?

An API Gateway is a key component in system design, particularly in microservices architectures and modern web applications. It serves as a centralized entry point for managing and routing requests from clients to the appropriate microservices or backend services within a system.



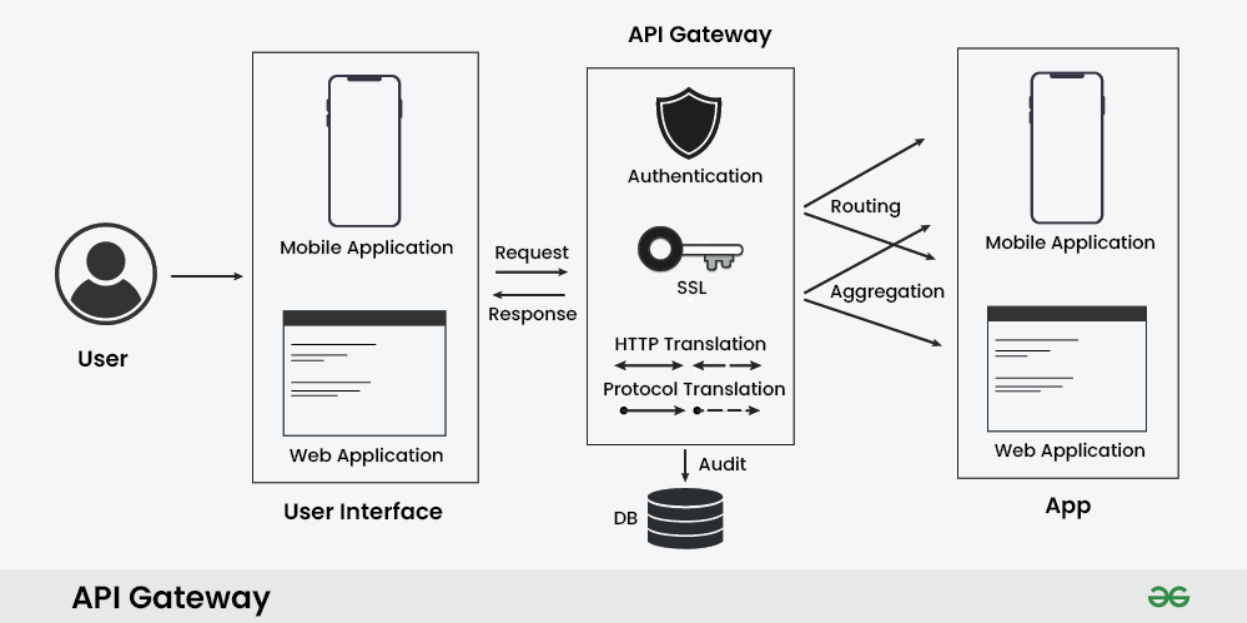
Important Topics for API Gateway in System Design

- What is an API Gateway?
- How does API Gateway work?
- How differently API Gateway work with Microservices and Monolith architecture?
- Best Practices for API Gateway
- Benefits of using an API Gateway:
- Challenges of using an API Gateway
- Popular API Gateway Solutions

What is an API Gateway?

API Gateway is a service that sits between clients and backend services, acting as a reverse proxy to accept incoming requests from clients, perform various operations such as routing, authentication, and rate limiting, and then forward those requests to the appropriate backend services.

It serves as a single entry point for clients to access multiple services, providing a unified interface and abstracting the complexities of the underlying architecture.



Working of API Gateway

In the above diagram:

- User will send the request from mobile or web application.
- API Gateway will determine which request is coming.
- Authentication means the user need to proof there identity to the server or client, by providing there User_Id and Password. Example :- Login or Signup page.
- SSL full form Secure Socket Layer, it is used to establish an encrypted link between a server and a client.
- It provides the ability to perform protocol translation, where incoming requests are translated from one channel to another. When requests are aggregated, a request received by an API gateway will trigger requests to different endpoints, and return response to the client.

The primary purpose of an API Gateway is to simplify the client's interaction with the underlying services, enhance security, and provide various features for managing and monitoring API traffic.

How does API Gateway work?

API Gateway acts as a single entry point for clients to access various microservices or backend services. It abstracts the underlying architecture and provides a unified interface for clients to interact with the system. Here's how it works:

- **Step 1 : Routing**
    - When a client sends a request to the API Gateway, it first examines the request to determine which service or

microservice should handle it. This routing can be based on various criteria such as the URL path, HTTP method, or headers.

- **Step 2 : Protocol translation**
    - The API Gateway can translate incoming requests from one protocol to another. For example, it can accept HTTP requests from clients and convert them to gRPC or WebSocket requests for backend services.
- **Step 3 : Request aggregation**
    - In some cases, a client may need to fetch data from multiple services to fulfill a single request. The API Gateway can aggregate these requests into a single call to improve efficiency and reduce the number of round trips.
- **Step 4 : Authentication and authorization**
    - The API Gateway can handle authentication and authorization for incoming requests. It can verify the identity of the client and check if the client has the necessary permissions to access the requested resources.
- **Step 5 : Rate limiting and throttling**
    - To prevent abuse and ensure fair usage of resources, the API Gateway can enforce rate limiting and throttling policies. It can limit the number of requests a client can make within a certain time period.
- **Step 6 : Load balancing**
    - The API Gateway can distribute incoming requests across multiple instances of a service to ensure high availability and scalability.
- **Step 7 : Caching**
    - To improve performance, the API Gateway can cache responses from backend services and serve them directly to clients for subsequent identical requests.
- **Step 8 : Monitoring and logging**
    - The API Gateway can collect metrics and logs for incoming requests, providing insights into the usage and performance of the system.

Overall, the API Gateway simplifies the client-server communication by providing a centralized entry point with various features to manage and secure the interactions between clients and backend services

How differently API Gateway works with Microservices and Monolith Architecture?

The way an API Gateway works with microservices differs from how it works with a monolithic architecture in several key aspects:

Aspect	Monolithic Architecture	Microservices Architecture
Request routing	In a monolithic architecture, the API Gateway typically routes requests to different parts of the monolith based on the request URL or other criteria	In a microservices architecture, the API Gateway routes requests to different microservices based on the request URL or other criteria, acting as a kind of "front door" to the microservices ecosystem.
Service discovery	In a monolithic architecture, service discovery is not typically a concern, as all parts of the application are contained within the same codebase.	In a microservices architecture, the API Gateway may need to use service discovery mechanisms to dynamically locate and route requests to the appropriate microservices.
Authentication and authorization	In both architectures, the API Gateway can handle authentication and authorization.	However, in a microservices architecture, there may be more complex authorization scenarios, as requests may need to be authorized by multiple microservices.
Load balancing	In both architectures, the API Gateway can perform load balancing.	However, in a microservices architecture, load balancing may be more complex, as requests may need to be load balanced across multiple instances of multiple microservices.
Fault tolerance	In both architectures, the API Gateway can provide fault tolerance by retrying failed requests and routing requests to healthy instances of services.	However, fault tolerance may be more critical in a microservices architecture, where the failure of a single microservice should not bring down the entire system.

Overall, the main difference in how an API Gateway works with microservices vs. monolith is in how it handles request routing, service discovery, and load balancing in a more distributed and decoupled microservices architecture

compared to the more centralized and integrated nature of a monolithic architecture

API Gateway with Microservices Example

Let's consider a hypothetical e-commerce system with microservices. The system has services for user management, product catalog, shopping cart, and order processing. Clients interact with the system through a web application.

Explanation of the below diagram

- The web application communicates with the API Gateway.
- The API Gateway routes requests to the appropriate microservices (e.g., user-related requests to the Users service).
- It handles authentication, rate limiting, caching, and other functions.
- Error responses are also standardized by the API Gateway.

API Gateway with Monolith Example

Consider a traditional e-commerce monolithic application. The API Gateway can still serve as a central entry point and manage authentication, request transformation, caching, and other features.

Explanation of below the diagram

- The web application communicates with the API Gateway.

- The API Gateway simplifies client interactions and provides security and caching and other features.
- It also manages API versioning and error handling.

Best Practices for API Gateway

Below are the best practices for API Gateway:

- **Security**: Implement strong authentication and authorization mechanisms, use SSL/TLS for encryption, and apply rate limiting and IP whitelisting to protect against abuse.
- **Performance Optimization**: Use caching, request/response compression, and efficient routing to reduce latency and improve response times.
- **Scalability**: Design for horizontal scalability, use load balancing, and monitor performance metrics to scale resources as needed.
- **Monitoring and Logging**: Implement comprehensive logging, use monitoring tools to track performance metrics, and integrate with logging and monitoring systems for centralized management.
- **Error Handling**: Implement robust error handling mechanisms and use standardized error codes and messages for consistency.
- **Versioning and Documentation**: Use versioning to manage changes and maintain backward compatibility, and keep up-to-date documentation for developers to understand how to use the API.

Benefits of using an API Gateway

- **Centralized Entry Point**
    - In complex systems with multiple microservices or backend services, clients (e.g., web or mobile applications) typically need to interact with various endpoints to access different functionalities.
    - An API Gateway acts as a single entry point, meaning that clients send their requests to the gateway, and the gateway takes responsibility for routing those requests to the appropriate services.
- **Routing and Load Balancing**
    - API Gateways analyze incoming requests and determine which backend service should handle them based on various factors such as the request's URL, headers, or even the content of the request.
    - Additionally, they can distribute incoming requests evenly across multiple instances of the same service to ensure load balancing.
- **Authentication and Authorization**
    - They can enforce authentication, ensuring that only authorized users or applications can access the services behind the gateway.
    - This is typically done using mechanisms like API keys, OAuth tokens, or JWTs. Furthermore, they handle authorization by checking if the authenticated user or application has the necessary permissions to access specific resources.
- **Request and Response Transformation**
    - API Gateways can transform requests and responses as they pass through.
    - For example, they can convert data formats (e.g., from JSON to XML or vice versa) to ensure compatibility between different parts of the system.
    - They can also aggregate data from multiple services into a single response, providing clients with a unified view.

Challenges of using an API Gateway

API Gateways can introduce several challenges, especially in complex environments or when not properly configured. Some common challenges include:

- **Performance bottlenecks**: API Gateways can become a single point of failure or a performance bottleneck, especially when handling a large

number of requests. Careful design and configuration are required to ensure they can handle the load.

- **Increased latency**: Introducing an API Gateway can add additional latency to requests, especially if it needs to perform complex routing, authentication, or other operations. Optimizing the Gateway's configuration and using caching can help mitigate this issue.
- **Complexity**: Managing and configuring an API Gateway can be complex, especially in environments with a large number of services and endpoints. Proper documentation and automation tools can help reduce this complexity.
- **Security risks**: Misconfigured API Gateways can introduce security vulnerabilities, such as improper authentication, authorization, or exposure of sensitive information. Regular security audits and updates are essential to mitigate these risks.
- **Scalability challenges**: Scaling an API Gateway can be challenging, especially in dynamic environments with fluctuating traffic. Load balancing and horizontal scaling strategies are essential to ensure scalability.
- **Monitoring and logging**: Monitoring and logging the API Gateway's performance and behavior can be complex, especially when dealing with a large number of requests and services. Proper monitoring and logging tools are crucial for maintaining visibility into the Gateway's operation.

Addressing these challenges requires careful planning, configuration, and monitoring to ensure that the API Gateway remains performant, secure, and scalable in a complex microservices environment

Popular API Gateway Solution

Below are some API Gateway Solution:

- **Amazon API Gateway**: It is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It supports RESTful APIs as well as WebSocket APIs for real-time communication. It integrates with other AWS services, such as AWS Lambda, to provide a serverless API development experience.
- **Apigee**: It now part of Google Cloud, is a platform that enables organizations to design, secure, deploy, monitor, and scale APIs. It offers features like API analytics, API monetization, and developer portal management. Apigee can be deployed on-premises, in the cloud, or in a hybrid environment.
- **Kong**: It is an open-source API Gateway and microservices management layer. It is built on top of Nginx and provides features like

request routing, authentication, rate limiting, and logging. Kong can be extended with plugins to add custom functionality and integrates with various authentication providers and monitoring tools.

- **Microsoft Azure API Management**: It is a fully managed service that helps organizations publish, secure, and manage APIs. It offers features like API gateway functionality, developer portal management, and API versioning. Azure API Management integrates with other Azure services and provides analytics and monitoring capabilities