Implementation of Sockets in Microservices architecture

Introduction:

Microservices architecture is a design pattern that structures the application as a set of loosely coupled, collaborating services. The pattern ensures that each service is highly maintainable and testable, independently deployable, and capable of being worked on by a small team.

Monolithic Application

A conventional monolith is a single unit involving clients, a load balancer, a large application with components representing individual features, and a database to store and access information [1]. It is a single logical executable. The disadvantage in this architecture is that to reflect any changes made in a component, the entire application should be rebuilt and deployed.

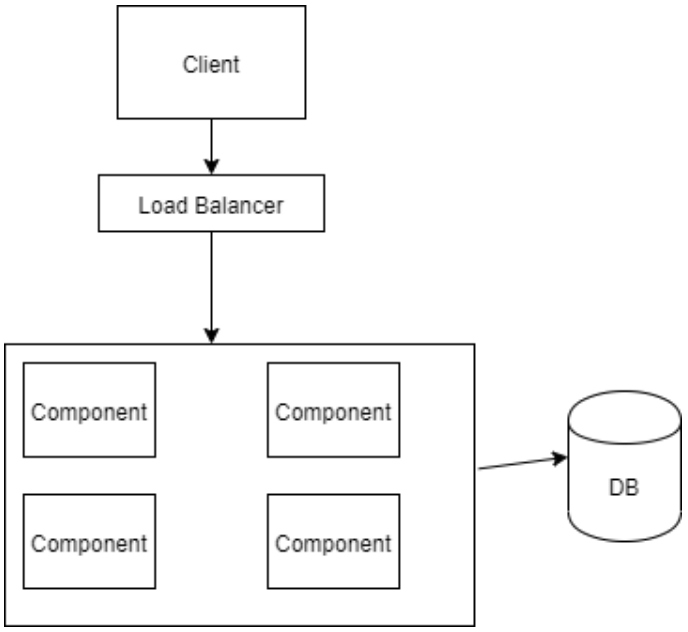


Fig: Monolithic Architecture [1]

Sockets in a Monolithic application

The implementation of sockets in a monolithic application is relatively simple. This is because the client and the server have direct communication with each other. The client can open a socket connection with the server requesting resources. A secure socket connection can be established between the client and the server using standard **OAuth 2.0 protocol** and **JWT (JSON Web Token)** tokens [2]

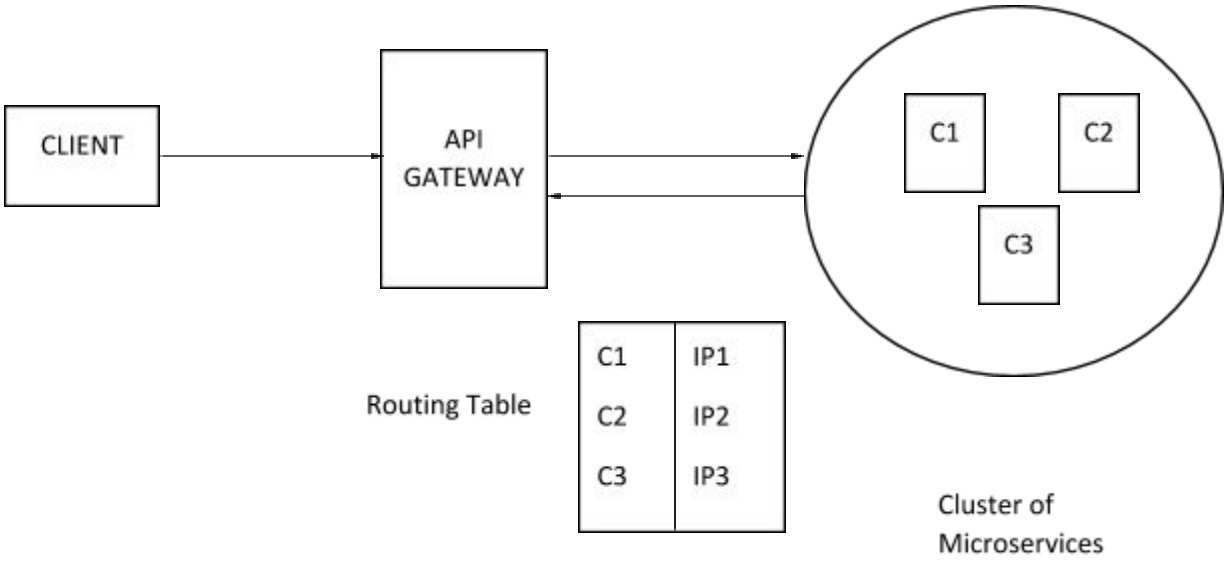
C1, C2, C3, C4 are the components representing feature services and their databases.

Fig: Sockets in Monolithic Architecture [2]

Sockets in Microservices Architecture

A microservices architecture consists of several small, scalable, and independent components. The clients access the services via API Gateway. API Gateway is a service that acts as a front-end to the backend microservices. In other words, API Gateway abstracts the infrastructure from being exposed to the clients. In contrast to the monolithic architecture, the client systems do not have any information about the backend services and their IP addresses. The API Gateway stores the required information in a routing table.

The client sends a request to the API Gateway requesting the resources. The requested data may be available at a single microservice or a combination of services. API Gateway takes care of the overhead to process and fetch the data from multiple servers in the specified sequence. The data is then transferred back to the client as a response [3].



C1, C2, C3 are the components representing microservices

. (Fig: Microservices Architecture [3])

Implementation of sockets in a microservices architecture is complex and challenging. The following are the two approaches to implement web sockets in a service-oriented architecture.

Approaches for implementing web sockets in Microservice architecture:

Approach1: Establish a socket connection between the client and the API Gateway

Procedure:

- Configure API Gateway to manage socket connections from the clients.
- To fetch the data, API Gateway may maintain open connections with the microservices or subscribe to the events published to a message bus.
- An API definition file specifies a sequence of steps for communicating with the services

Discussion Points:

- Firstly, the approach is in contrast with the principle of the API Gateway. The API Gateway is a lightweight service that routes the requests to the appropriate microservices. It should not perform any heavy lifting operations other than acting as an interface.

- Secondly, opening, closing, maintaining, and reconnecting web sockets increase hardware and software resource utilization. A heavy load on the server may cause unprecedented changes. There is also a possibility of **"Single Point Failure."** For instance, if the socket handlers throw an error, the entire API Gateway goes down. As a result, the application experiences downtime.

Approach 2: A WebSocket server parallel to the API Gateway

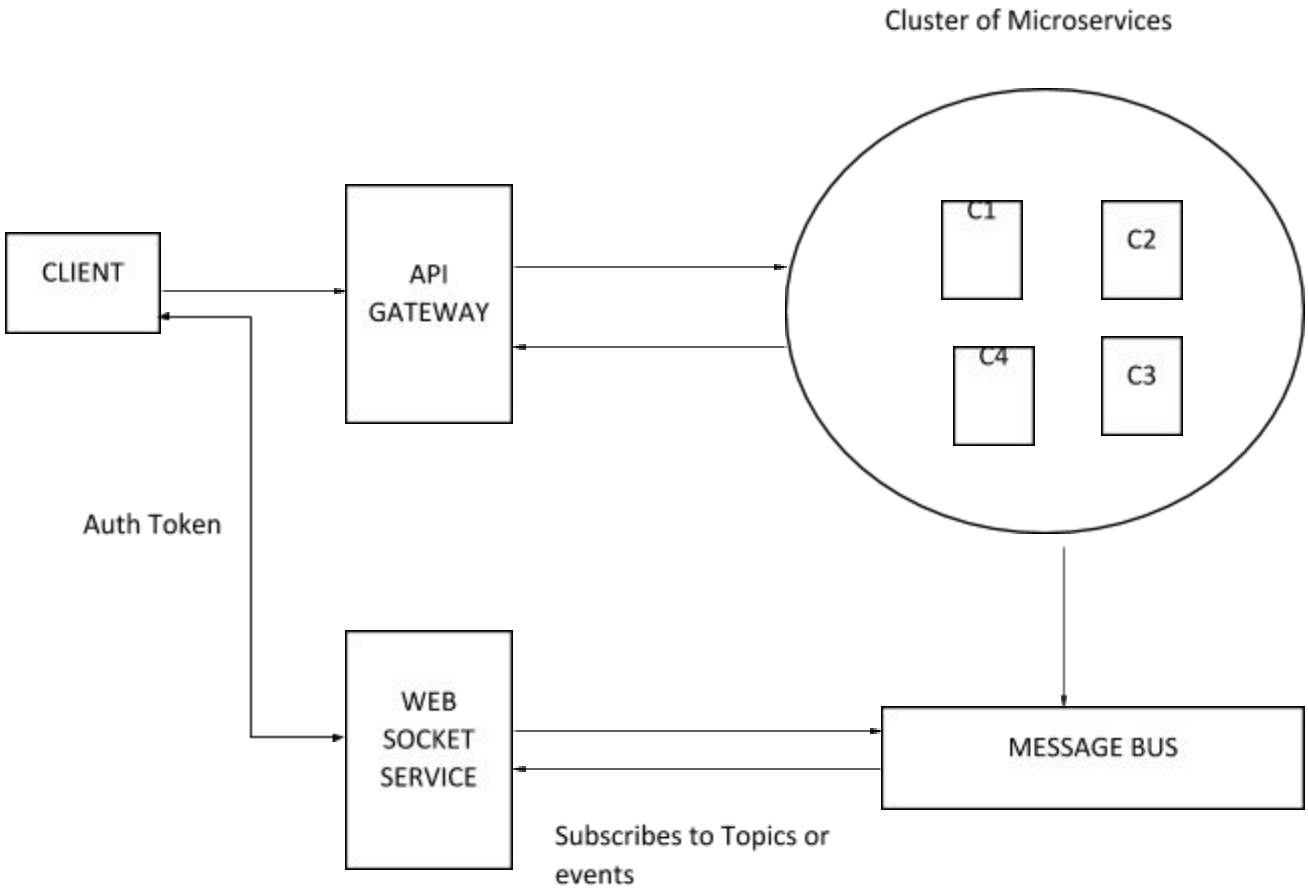
Procedure:

- The approach is to deploy a WebSocket server parallel to the API Gateway, on a separate domain and port.
- Initially, the client sends a request to the API Gateway requesting a security-access token (Bearer Token). The API Gateway interacts with the authentication microservice and responds with a token to the client. The token has an expiry time limit of 'N' seconds.
- Using the retrieved token, the client establishes a secure connection with the Web Socket Service.
- The Web Socket server communicates with the backend microservices via a message bus (KAFKA, Rabbit MQ) using the Producer-Consumer model.
- It subscribes to the topics published by the microservices and sends the response to the client.
- Whenever the token expires, the client requests the API Gateway to send a new authentication token from the Authentication service.

Discussion Points:

- This approach works effectively in an eventually consistent microservice architecture where the requests take time to complete.
- The approach solves the issue of a single point of failure.
- An independent socket service is equipped with the hardware and software resources to manage and scale relative to the traffic.
- The use of a message bus in the intercommunication makes the services loosely coupled.

Workflow – Approach 2:



C1, C2, C3 are the components representing microservices and their databases.

Fig: Microservices Architecture [4]

Further research:

- What is the ideal time limit for the expiry of the Authentication token?
- How many instances of a socket service should we maintain?
- Do we need to store the socket connection information in an in-memory database such as Redis?
- How to effectively implement handlers for managing subscriptions to the message bus.

Conclusion:

Web Sockets play an important role in providing a great user experience by dynamically updating the data on the screen. The Microservices architecture, although complex in its implementation, provides several characteristics such as consistency, reliability, and availability. The proposed approach of **"Parallel Web Socket Server"** provides an effective mechanism to implement the Web Socket functionality. The approach is extremely useful for enterprises that build and manage their infrastructure independently without relying on third party services.