Case Study: Microservices Architecture for an E-Commerce Platform

Overview

This case study discusses the architecture of an **E-Commerce platform** leveraging a microservices approach. The diagram illustrates four services connected through an API Gateway and a service discovery mechanism (Eureka). The services include **Inventory Service**, **User Service**, **Cart Service**, and an **API Gateway**. Each service is associated with a MySQL database, emphasizing data independence.

---

1. Business Requirements

1. **User Features:**
    - Users can register, log in, and update their profiles.
    - Browse and add items to a shopping cart.
    - Place orders based on cart contents.
    - Manage inventory (admin-only functionality).

2. **System Goals:**
    - Modular architecture for independent scalability.
    - Efficient routing of user requests to respective services.
    - Centralized service discovery using Eureka.
    - High fault tolerance and reduced downtime.

---

2. Microservices Overview

1. Inventory Service

- **Responsibilities:**
    - Add, update, and remove inventory items (admin role).
    - Validate stock levels for items added to a cart.
- **Database:**
    - MySQL database to store product details, stock levels, and metadata.
- **Endpoints:**
    - **POST /inventory:** Add a new item.
    - **GET /inventory/{id}:** Fetch details of a specific product.
    - **PUT /inventory/{id}:** Update stock levels or item details.
    - **DELETE /inventory/{id}:** Remove a product from inventory.

---

2. User Service

- **Responsibilities:**
  - Manage user registration, authentication, and profile updates.
- **Database:**
  - MySQL database to store user details, hashed passwords, and roles (e.g., admin or customer).
- **Endpoints:**
  - **POST /users/register:** Register a new user.
  - **POST /users/login:** Authenticate user credentials.
  - **GET /users/{id}:** Fetch user profile details.
  - **PUT /users/{id}:** Update user profile.

---

3. Cart Service

- **Responsibilities:**
  - Manage cart items for each user, including adding, updating, and removing items.
  - Calculate the cart's total price and validate inventory stock before order placement.
- **Database:**
  - MySQL database to store cart sessions, including user ID, product IDs, quantities, and prices.
- **Endpoints:**
  - **POST /cart/{userId}:** Add an item to the cart.
  - **GET /cart/{userId}:** Fetch all items in the user’s cart.
  - **PUT /cart/{userId}/{productId}:** Update the quantity of a cart item.
  - **DELETE /cart/{userId}/{productId}:** Remove an item from the cart.

---

4. API Gateway

- **Responsibilities:**
  - Acts as a single entry point for client requests.
  - Routes requests to appropriate microservices.
  - Handles rate limiting, load balancing, and authentication.
- **Endpoints:**

    - **https://api.coforge.com/*:** All requests are routed through the gateway.
- **Technology:**
    - Implements Zuul or Spring Cloud Gateway.

---

5. Service Discovery (Eureka)

- **Responsibilities:**
    - Enables dynamic service discovery and load balancing.
    - Registers all services to ensure seamless communication between microservices.

---

3. High-Level Design

The attached architecture visualizes the following:

1. **Client Interaction:** Users interact with the system through the API Gateway.
2. **API Gateway:** Distributes requests to services (Inventory, User, and Cart Services).
3. **Service Discovery:** Eureka Server ensures services are discoverable and dynamically routable.
4. **Database:** Each microservice has its own dedicated MySQL database to maintain autonomy and prevent data coupling.
5. **OpenFeign** is used for inter-service communication, where services need real-time responses.
6. **Service Discovery** (via Eureka) ensures dynamic routing of requests.
7. **Fault Tolerance Mechanisms** like circuit breakers (e.g., Hystrix or Resilience4J) are added to OpenFeign clients for resilient communication.

---

4. Data Model

Inventory Service

Field	Type	Description
id	UUID	Unique identifier for the product.
name	String	Product name.
description	Text	Details about the product.

price	Decimal	Price per unit.
stockQuantity	Integer	Available stock for the product.

User Service

Field	Type	Description
id	UUID	Unique identifier for the user.
name	String	User's full name.
email	String	User's email address.
password	String	Securely password.
role	Enum	Role (admin or customer).

Cart Service

Field	Type	Description
id	UUID	Unique identifier for the cart.
userId	UUID	ID of the user who owns the cart.
items	JSON	List of product IDs with quantities.
totalPrice	Decimal	Total price of items in the cart.

---

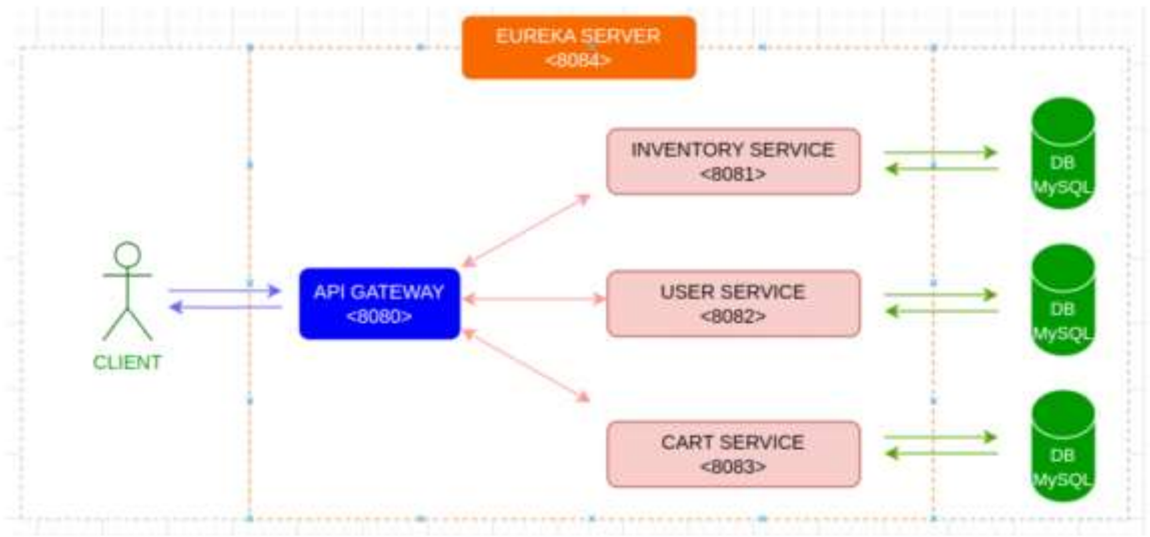
5. Advantages of the Design

1. **Scalability:** Each microservice can scale independently based on load.
2. **Fault Isolation:** Failure in one service does not cascade to others.
3. **Ease of Maintenance:** Microservices can be updated or replaced independently.
4. **Technology Flexibility:** Each service can use the most appropriate tech stack.

---

2. High-level design

At a high-level, we need some following services (or components) to handle above requirements:



- **Inventory Service**: Add item to the site, only admin role can manage the inventory.
- **User Service**: manage all users service and create new users
- **Cart Service**: manages customers shopping carts with CRUD operations.
- **API Gateway**: Route requests to multiple services using a single endpoint. This service allows us to expose multiple services on a single endpoint and route to the appropriate service based on the request.

6. Conclusion

The microservices architecture presented here ensures modularity, scalability, and fault tolerance. By leveraging an API Gateway and Eureka for service discovery, the system simplifies client interaction and maintains seamless inter-service communication. This design is ideal for e-commerce platforms aiming for rapid growth and high traffic resilience