**E-Commerce Microservices Backend Documentation**

**Overview**

This project is a microservices-based backend for an e-commerce platform, developed using Spring Boot. It includes various services such as Product, Inventory, Order, and Notification, all designed to provide a robust and scalable solution for e-commerce operations.

**Key Features**

**Microservices Architecture**

* **Product Service**: Manages product information including adding, retrieving, and filtering products based on various criteria.
* **Inventory Service**: Keeps track of product stock levels and checks if products are in stock.
* **Order Service**: Handles customer orders including placing, updating, and deleting orders, as well as managing order items and statuses.
* **Notification Service**: Sends notifications related to orders and inventory updates.

**Service Discovery and API Gateway**

* **Netflix Eureka**:
  + Provides service discovery.
  + Allows efficient communication between microservices by registering and locating services.
* **Spring Cloud Gateway**:
  + Acts as an API Gateway for routing and load balancing.
  + Ensures efficient request handling and provides a single-entry point for all client requests.

**Security and Resilience**

* **Key cloak**:
  + Manages identity and access control.
  + Secures microservices by handling authentication and authorization.
* **Circuit Breaker**:
  + Implemented using Spring Cloud Circuit Breaker.
  + Provides resilience and prevents cascading failures by stopping the flow of requests to a failing service.

**Event-Driven Architecture**

* **Apache Kafka**:
  + Enables event-driven communication between microservices.
  + Facilitates asynchronous processing and improves system scalability by decoupling the producers and consumers of events.

**Containerization and Monitoring**

* **Docker**:
  + Dockerizes all microservices for consistent deployment across different environments.
  + Simplifies the deployment process and ensures that the application runs the same way everywhere.
* **Prometheus and Grafana**:
  + Prometheus collects metrics and provides monitoring capabilities.
  + Grafana offers visualization tools for monitoring application performance and health.

**Technologies Used**

* Spring Boot
* Spring Cloud Netflix (Eureka)
* Spring Cloud Gateway
* Spring Cloud Circuit Breaker
* Keycloak
* Apache Kafka
* Docker
* Prometheus
* Grafana

**Getting Started**

**Prerequisites**

Ensure you have the following installed:

* Java 11 or higher
* Docker
* Docker Compose
* Apache Kafka
* Prometheus
* Grafana

**Running the Application**

**Using Docker**

1. Build the applications and create the Docker images locally:

bash

mvn clean package -DskipTests

1. Start the applications:

bash

docker-compose up -d

**Without Docker**

1. Build the applications:

bash

mvn clean verify -DskipTests

1. Run the applications: Navigate to each service folder and run:

bash

mvn spring-boot:run

**Accessing Services**

* **Eureka Dashboard**: <http://localhost:8761>
* **API Gateway**: <http://localhost:8080>
* **Prometheus**: <http://localhost:9090>
* **Grafana**: <http://localhost:3000>

**API Endpoints**

**Product Service**

* **Create Product**
  + **Endpoint**: POST /api/product
  + **Description**: Adds a new product to the catalog.
  + **Request Body**: ProductRequest
* **Get Products by Category**
  + **Endpoint**: GET /api/product/{category}
  + **Description**: Retrieves products based on the specified category.
* **Filter Products**
  + **Endpoint**: GET /api/product/filter
  + **Description**: Retrieves products filtered by brand, name, and year.
  + **Query Parameters**: brand, name, year
* **Get All Products**
  + **Endpoint**: GET /api/product
  + **Description**: Retrieves all products in the catalog.

**Inventory Service**

* **Check Item Stock**
  + **Endpoint**: GET /api/inventory/{skuCode}
  + **Description**: Checks if a specific product (by SKU code) is in stock.
* **Check Items Stock**
  + **Endpoint**: GET /api/inventory
  + **Description**: Checks if multiple products (by SKU codes) are in stock.
  + **Query Parameters**: skuCodes (list of SKU codes)

**Order Service**

* **Place Order**
  + **Endpoint**: POST /api/order
  + **Description**: Places a new order.
  + **Request Body**: OrderRequest
* **Get All Orders**
  + **Endpoint**: GET /api/order
  + **Description**: Retrieves all customer orders.
* **Get Order by ID**
  + **Endpoint**: GET /api/order/{id}
  + **Description**: Retrieves a specific order by its ID.
* **Delete Order**
  + **Endpoint**: DELETE /api/order/{id}
  + **Description**: Deletes a specific order by its ID.
* **Update Order Status**
  + **Endpoint**: PATCH /api/order/{orderId}/status
  + **Description**: Updates the status of a specific order.
  + **Request Body**: Status
* **Get Orders by Status**
  + **Endpoint**: GET /api/order/orderStatus/{status}
  + **Description**: Retrieves orders based on their status.
* **Add Item to Order**
  + **Endpoint**: POST /api/order/{orderId}/items
  + **Description**: Adds an item to a specific order.
  + **Request Body**: OrderLineItemsDto
* **Remove Item from Order**
  + **Endpoint**: DELETE /api/order/{orderId}/items/{itemId}
  + **Description**: Removes an item from a specific order.

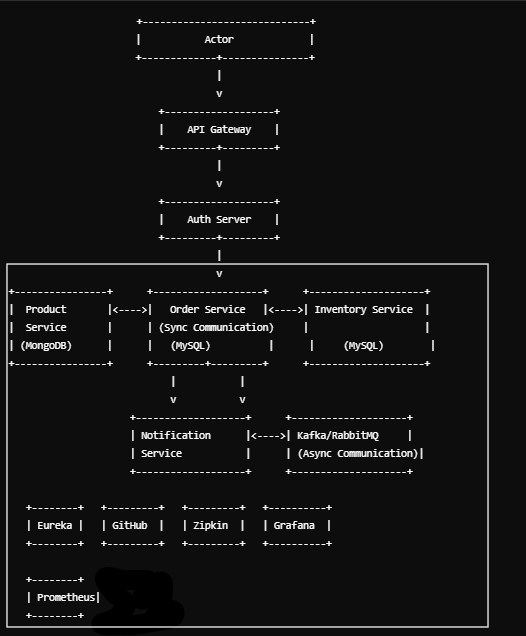
**Notification Service**

* This service listens to Kafka events and sends notifications accordingly.

**Communication Details**

* **Order Service and Inventory Service**: The Order Service communicates synchronously with the Inventory Service using WebClient to check if a product is available before placing an order.
* **Order Service and Notification Service**: Once an order is placed, the Order Service sends an event to Kafka, which is consumed by the Notification Service to publish email notifications or any further communication.

**High Level Architecture :**



**Docker**

Client

### Summary

The e-commerce microservices backend project is a robust and scalable solution built using modern technologies and microservices architecture. It comprises several key services including Product, Inventory, Order, and Notification, each fulfilling specific roles essential for e-commerce operations. The architecture leverages Docker for containerization, ensuring consistent deployment across environments, while Kubernetes orchestration simplifies management. Keycloak secures the APIs through centralized authentication and authorization, and Netflix Eureka facilitates service discovery for seamless communication between microservices. Resilience4J's Circuit Breaker enhances fault tolerance during interactions between services, and Kafka enables efficient event-driven communication for asynchronous processing. Prometheus and Grafana offer comprehensive monitoring and observability, ensuring optimal performance and timely issue resolution.

### Conclusion

In conclusion, the e-commerce microservices backend project demonstrates a modern approach to building scalable and resilient systems for e-commerce operations. By adopting microservices architecture and utilizing technologies like Docker, Kubernetes, Kafka, and Keycloak, the project ensures flexibility, scalability, and security. The implementation of Circuit Breaker and distributed tracing further enhances reliability and troubleshooting capabilities, while Prometheus and Grafana provide real-time monitoring and insights into system performance. Overall, the project not only meets current business needs but also lays a solid foundation for future enhancements and expansions in the e-commerce domain.