E-commerce Microservices System

1. Overview

The goal of this project is to design and implement an **e-commerce system** with separate services for user management, product catalog, order processing, and real-time inventory tracking. The system will use **Kafka** for event streaming to ensure data consistency and real-time updates, and **Grafana** for monitoring service metrics. The application will leverage **Docker** for containerization and **Kubernetes** for orchestration, ensuring scalability and resilience.

2. Objectives

- **Scalability**: Ensure that each service is independently scalable to accommodate varying workloads, allowing the system to handle spikes in traffic without affecting performance.
- **Resilience**: Design a fault-tolerant system where each service operates independently, minimizing the impact of service failures.
- **Real-Time Data**: Use Kafka for real-time inventory tracking, ensuring product availability is always up-to-date.
- **Monitoring and Observability**: Implement Grafana dashboards for monitoring key metrics, such as response times, error rates, and resource usage.

3. Key Features

1. **User Management Service**:
    - Handles user registration, authentication, and profile management.
    - Supports login, logout, and password recovery.
    - User data is stored in a secure database, with data encryption for sensitive fields.
2. **Product Catalog Service**:
    - Manages product details including names, descriptions, prices, and inventory status.
    - Supports product categories and search functionalities.
    - Product updates trigger Kafka events to notify other services of changes.
3. **Order Processing Service**:
    - Manages customer orders, payment processing, and order status tracking.
    - Integrates with third-party payment gateways for secure payment processing.
    - Kafka is used to propagate order-related events (e.g., order placed, order shipped) to other services.
4. **Real-Time Inventory Tracking**:

- Uses Kafka to manage inventory updates in real time, triggered by product purchases or inventory adjustments.
- Ensures accurate inventory levels across the system and reduces the risk of over-selling.

5. **Monitoring with Grafana**:
   - Real-time dashboards for visualizing service health metrics such as CPU usage, memory consumption, request latency, and error rates.
   - Alerts to notify the development team of anomalies or issues in any of the services.

4. Technical Stack

- **Programming Language**: Python (FastAPI framework) for backend services.
- **Containerization**: Docker for containerizing each service.
- **Orchestration**: Kubernetes for managing and scaling containers.
- **Event Streaming**: Kafka for inter-service communication and real-time event propagation.
- **Database**: PostgreSQL for storing user, product, and order data.
- **Monitoring**: Grafana and Prometheus for monitoring and alerting.

5. Architecture

- **Microservices Architecture**: Each core function (user management, product catalog, order processing, inventory tracking) is a separate microservice.
- **Event-Driven Communication**: Kafka acts as the event bus, allowing services to publish and subscribe to relevant events.
- **Service Discovery**: Kubernetes is used to manage service discovery and load balancing.

6. User Stories

1. **User Registration**:
   - As a user, I want to be able to register an account so I can make purchases.
     - **Test Case**: Verify that a new user can register successfully with valid information.
     - **Test Case**: Verify that the system returns an error if mandatory fields are missing during registration.
   - As a user, I want to receive a confirmation email upon successful registration.
     - **Test Case**: Verify that a confirmation email is sent to the user after successful registration.
   - As a user, I want to be notified if my chosen username is already taken so I can pick a different one.
     - **Test Case**: Verify that the system notifies the user if the chosen username is already taken.
2. **Product Search**:
   - As a user, I want to search for products by name or category so I can quickly find what I need.

        - **Test Case**: Verify that the search returns relevant products when searching by name.
        - **Test Case**: Verify that products can be filtered by category.
    - As a user, I want to filter search results by price, rating, or availability to narrow down my choices.
        - **Test Case**: Verify that search results can be filtered by price range.
        - **Test Case**: Verify that search results can be filtered by product rating.
    - As a user, I want to see suggestions for similar products when I search to explore more options.
        - **Test Case**: Verify that similar product suggestions are displayed when viewing a product.
3. **Place Order**:
    - As a user, I want to place an order and receive updates on the order status.
        - **Test Case**: Verify that an order can be placed successfully with valid payment information.
        - **Test Case**: Verify that order status updates are sent to the user.
    - As a user, I want to receive an email or SMS confirmation once my order is placed.
        - **Test Case**: Verify that the user receives an email or SMS confirmation upon placing an order.
    - As a user, I want to be able to cancel my order within a certain time frame if I change my mind.
        - **Test Case**: Verify that an order can be canceled within the allowed time frame.
        - **Test Case**: Verify that the user receives a confirmation when an order is successfully canceled.
    - As a user, I want to be able to track the shipping status of my order in real time.
        - **Test Case**: Verify that the user can view the real-time shipping status of their order.
4. **Inventory Update**:
    - As an admin, I want inventory to update in real time whenever a product is purchased to prevent overselling.
        - **Test Case**: Verify that inventory levels are updated immediately after a product is purchased.
    - As an admin, I want to receive alerts if inventory for a product falls below a defined threshold so I can reorder stock.
        - **Test Case**: Verify that alerts are sent when inventory levels fall below the threshold.
    - As an admin, I want to be able to manually adjust inventory levels in case of discrepancies.
        - **Test Case**: Verify that an admin can manually update inventory levels.
5. **Service Monitoring**:
    - As a developer, I want to monitor system metrics to ensure all services are healthy and perform well.

        - **Test Case**: Verify that Grafana dashboards display real-time metrics for all services.
    - As a developer, I want to receive alerts if any service experiences high error rates or increased latency.
        - **Test Case**: Verify that alerts are triggered when a service's error rate exceeds a defined threshold.
    - As a developer, I want to access historical data on service performance to identify trends and optimize the system.
        - **Test Case**: Verify that historical performance data is available for analysis in Grafana.

7. Functional Requirements

- **Authentication and Authorization**: Users should be able to securely log in and out, and certain actions (e.g., adding products) should require admin privileges.
- **Product Availability Check**: The system should automatically check product availability before confirming an order.
- **Kafka Integration**: Kafka should handle event propagation for key actions, such as order placement and inventory updates.
- **Resilience and Failover**: Each service should handle failure gracefully and recover without impacting user experience.

8. Non-Functional Requirements

- **Scalability**: Each microservice should scale independently based on the traffic it receives.
- **Security**: Implement SSL/TLS for secure data transmission and use proper encryption for sensitive data (e.g., user passwords).
- **Performance**: Average response time for API calls should be under 200ms, and the system should support up to 10,000 concurrent users.
- **Monitoring and Alerts**: Real-time monitoring with alerts for anomalies or service failures using Grafana.

9. API Endpoints (Examples)

- **User Management**:
    - `POST /register`: Register a new user.
    - `POST /login`: User authentication.
    - `GET /profile`: Retrieve user profile information.
- **Product Catalog**:
    - `GET /products`: List all products.
    - `GET /products/{product_id}`: Retrieve product details.
    - `POST /products`: Add a new product (admin only).
- **Order Processing**:
    - `POST /orders`: Place a new order.
    - `GET /orders/{order_id}`: Retrieve order status.

10. Metrics to Monitor

- **User Management Service**: Login success/failure rate, response times, CPU/memory usage.
- **Product Catalog Service**: Product search latency, database query performance.
- **Order Processing Service**: Order creation rate, payment success rate, Kafka message processing times.
- **Inventory Tracking**: Accuracy of inventory data, event processing times in Kafka.

11. Risks and Mitigations

- **Service Downtime**: Use Kubernetes for auto-restarting failed containers to minimize downtime.
- **Data Consistency**: Utilize Kafka to ensure all services receive real-time updates, reducing the risk of data inconsistencies.
- **Scalability Bottlenecks**: Ensure each service can scale independently, and use load testing to identify potential bottlenecks before deployment.

12. Milestones

1. **Phase 1**: Set up Docker and Kubernetes environments, create user management and product catalog services.
2. **Phase 2**: Implement order processing service and integrate Kafka for event streaming.
3. **Phase 3**: Add real-time inventory tracking and Grafana for monitoring.
4. **Phase 4**: Final testing, load testing, and deployment.

13. Future Enhancements

- **Recommendation System**: Add a recommendation engine to suggest products based on user behavior.
- **Multi-Language Support**: Add support for multiple languages to enhance user experience.
- **Mobile App Integration**: Create a mobile app that integrates with the backend services for a better user experience.

14. Glossary

- **Kafka**: A distributed event streaming platform used for building real-time data pipelines.
- **Grafana**: An open-source platform for monitoring and observability, used for visualizing metrics.
- **Docker**: A tool for containerizing applications, making them easy to deploy and manage.
- **Kubernetes**: An open-source platform for automating deployment, scaling, and management of containerized applications.