Software Engineer - Algotrading - Case Study

Objective

The goal of this challenge is to evaluate your ability to design, implement, and deploy a low-latency, scalable, and portable application capable of running algorithmic trading strategies in real time that aligns with the kind of work you'll do as a Software Engineer on our Algotrading team. This task will assess your skills in **Python or Golang**, **Docker**, **MongoDB**, **Redis** or **Kafka**, as well as your system design and problem-solving capabilities.

Scenario

You are tasked with creating a minimal **Event-Driven Application** that:

- Connects and ingests data of an orderbook on an exchange (eg. Binance, OKX, ByBit etc., one pair like BTC/USDT would suffice)
- Stores price data in the database
- Opens or closes positions depending on SMA crossover
- Stores orders in the database on creation and updates when a position/order gets updated on the exchange
- Could be monolith leveraging multi-threading or microservices satisfying all requirements
- Could be implemented in Python or Golang
- Could use Redis or Kafka, or both
- MongoDB chosen for simplicity yet PostgreSQL could be used

Task

1. **Data stream and processing**
    a. Consume real-time data from an exchange using websockets
    b. Make sure the app handles connection loss etc.

2. **Algorithm implementation**
    a. Calculate daily short-term (eg. 50) and long-term (eg. 200) moving averages using SMA. Generate **BUY** signals when the short-term average crosses above the long-term average, and **SELL** signals when it crosses below
    b. Log in JSON format and save in the database separately all trading signals

3. **Portability**
    a. Create a Dockerfile to containerize the application, including all services if implemented many

4. **Monitoring**
    a. Measure system performance using the following metrics:
        i. Latency per operation
        ii. CPU and memory usage
        iii. Data loss or error rate
    b. Optimize your code to reduce latency and resource consumption
    c. Provide an HTTP endpoint for monitoring system health (eg. Kubernetes readiness/liveness)

Bonus Points

- Metrics sidecar or endpoints that could be scraped by Prometheus will be a plus.
- Unit tests and integration tests (using pytest or similar frameworks).
- The system should handle multiple instances running simultaneously.
- Docker Compose config for development teams to run all on local environments.

Deliverables

1. A **GitHub repository** with:
    - All application source code.
    - Documentation on how to run and test the application in a local environment.
2. A short **document** that explains:
    - How you would ensure scalability, fault tolerance, and security.
    - Challenges faced and how you solved them.