EVENT-DRIVEN ARCHITECTURE DEMO

Django + Kafka + Celery

Prerequisites

- Docker Desktop
- Python 3.8+
- PowerShell hoặc Terminal

1 Setup – First Time Only

1.1 Start Infrastructure (Kafka + Redis)

```
docker-compose up -d
docker ps
docker logs -f kafka
docker logs -f redis
```

1.2 Install Python Dependencies

```
pip install -r requirements.txt
```

1.3 Database Setup

```
python manage.py makemigrations
python manage.py migrate
```

1.4 Create Kafka Topics

```
docker exec -it kafka bash

/opt/kafka/bin/kafka-topics.sh --create \
--topic orders \
--bootstrap-server localhost:9092 \
--partitions 3 \
--replication-factor 1

/opt/kafka/bin/kafka-topics.sh --create \
--topic orders-dlq \
--bootstrap-server localhost:9092 \
--partitions 1 \
--replication-factor 1

/opt/kafka/bin/kafka-topics.sh --list \
--bootstrap-server localhost:9092

exit
```

1.5 Kafka UI

Mở trình duyệt: `http://localhost:8080`

2 Running the Demo

2.1 Django API Server

```
python manage.py runserver
```

2.2 Celery Worker

```
celery -A eda_demo worker --loglevel=info --pool=solo
```

2.3 Kafka Consumers

Inventory Consumer

```
cd orders/consumers
python inventory.py
```

Notification Consumer

```
python notification.py
```

Analytics Consumer

```
python analytics.py
```

2.4 Sending Test Data

```
python fakedatareal.py
```

Hoặc gửi thủ công:

```
curl -X POST http://127.0.0.1:8000/orders/ \
-H "Content-Type: application/json" \
-d "{\"product\":\"Laptop\",\"quantity\":2}"
```

Idempotency test:

```
curl -X POST http://127.0.0.1:8000/orders/ \
-H "Content-Type: application/json" \
-H "X-Idempotency-Key: test-key-123" \
-d "{\"product\":\"Mouse\",\"quantity\":5}"
```

3 Testing & Monitoring

3.1 Check Order Status

```
curl http://127.0.0.1:8000/orders/1/status/
```

3.2 Health Check

```
curl http://127.0.0.1:8000/health/
```

3.3 Celery Monitoring

```
celery -A eda_demo inspect active
celery -A eda_demo inspect stats
```

4 Demo Flow Explanation

Asynchronous Flow

1. Client gửi POST request tới Django API và nhận response ngay lập tức.
2. Celery worker publish event vào Kafka.
3. Kafka phân phối message tới các consumer độc lập.

4. Mỗi consumer xử lý nghiệp vụ riêng (Inventory, Notification, Analytics).
5. Khi hoàn tất, trạng thái đơn hàng được cập nhật thành `completed`.

Benefits

- Non-blocking
- Scalable
- Resilient
- Decoupled
- Idempotent

5 Error Handling & Retry

5.1 Dead Letter Queue

```
/opt/kafka/bin/kafka-console-consumer.sh \
--bootstrap-server localhost:9092 \
--topic orders-dlq \
--from-beginning
```

5.2 Reset Consumer Offset

```
/opt/kafka/bin/kafka-consumer-groups.sh \
--bootstrap-server localhost:9092 \
--group eda-demo-inventory \
--reset-offsets \
--to-earliest \
--topic orders \
--execute
```

6 Stop & Cleanup

6.1 Stop Containers

```
docker-compose down
docker-compose down -v
```

6.2 Clean Database

```
del db.sqlite3
del orders\migrations\0001_initial.py
```

7 Production Considerations

- Security (SECRET_KEY, DEBUG, ENV variables)
- Database: PostgreSQL/MySQL
- Kafka Cluster + Replication
- Celery + RabbitMQ
- Monitoring (Prometheus, Grafana, ELK)
- Load Balancing (Nginx, HAProxy)

End of Document