📚 Event-Driven Architecture (EDA) – Lý thuyết và liên hệ với EDA-Demo

1) EDA là gì?

- **Event-Driven Architecture (EDA)**: Kiến trúc trong đó các thành phần (microservices) giao tiếp bằng cách phát và tiêu thụ các sự kiện (events) thông qua một message broker (Kafka).
- **Mục tiêu**: Loose coupling, khả năng mở rộng (scale độc lập), tăng khả năng quan sát, và xử lý bất đồng bộ.

Thành phần chính

- **Event**: Thông tin về điều gì đó đã xảy ra (ví dụ: user.created, registration.created).
- **Producer**: Dịch vụ phát sự kiện.
- **Consumer**: Dịch vụ lắng nghe và xử lý sự kiện.
- **Topic**: Dòng sự kiện theo chủ đề, lưu trong broker (Kafka).
- **Broker**: Hệ thống trung gian truyền sự kiện (Kafka + Zookeeper).

2) Đặc tính quan trọng

- **Pub/Sub, bất đồng bộ**: Producer không chờ Consumer, giúp tách biệt thời gian và không gian.
- **Loose coupling**: Thêm/bớt Consumer không ảnh hưởng Producer.
- **At-least-once delivery** (Kafka): Có thể trùng lặp message; Consumer cần idempotent.
- **Partition & ordering**: Kafka chia topic thành partitions; nên chọn key phù hợp (vd: userId, eventId) để giữ thứ tự cục bộ.

3) Các mẫu (patterns) phổ biến

- **Event Notification**: Sự kiện chỉ mang ID/ít dữ liệu; Consumer tự tra cứu dữ liệu. Dự án này dùng chủ đạo.
- **Event-Carried State Transfer**: Sự kiện mang toàn bộ state để Consumer không cần tra cứu.
- **Event Sourcing**: Lưu toàn bộ sự kiện làm nguồn sự thật, state được dựng lại từ events. Không dùng trong dự án này.
- **CQRS**: Tách kênh đọc/ghi; có thể kết hợp EDA để build read model. Dự án chưa áp dụng đầy đủ.
- **Outbox Pattern**: Tránh "dual write" giữa DB và Kafka; thường dùng với CDC (Debezium). Dự án này chưa dùng, có thể cân nhắc cho production.
- **Saga**: Điều phối giao dịch phân tán qua chuỗi sự kiện. Không dùng trong demo này.

4) Thiết kế sự kiện & tính ổn định

- **Đặt tên topic**: theo domain, dạng `domain.action` (vd: `user.created`, `registration.created`).
- **Schema & versioning**: JSON + version; cần backward/forward compatible (có thể dùng Schema Registry ở production).
- **Idempotency**: Consumer cần tránh ghi trùng (ví dụ kiểm tra tồn tại theo khóa tự nhiên/ID).

- **Retry & DLQ**: Thất bại nên retry; nếu vẫn lỗi, đẩy vào Dead Letter Queue (ví dụ: `notification.failed`, `audit.failed`).
- **Quan sát (observability)**: Dùng Kafka UI để xem topics, messages, consumer groups, lag.

5) Liên hệ với dự án EDA-Demo

- **Broker**: Kafka (kèm Kafka UI). Topics được tạo bởi script `shared/init-kafka-topics.sh` (partitions=1 trong demo).
- **Dịch vụ & vai trò**:
  - `gateway`: Nhận request từ frontend; route/emit sự kiện (producer `registration.created`).
  - `user-service` (PostgreSQL): Producer `user.created`, `user.logged_in`.
  - `registration-service` (PostgreSQL): Producer `registration.created`.
  - `event-service` (PostgreSQL): Consumer `registration.created` → cập nhật số lượng; có thể emit `event.updated`.
  - `notification-service` (SMTP): Consumer `registration.created` → tra email user → gửi mail → emit `notification.sent` (hoặc `notification.failed`).
  - `auditlog-service` (MongoDB): Consumer nhiều topic (`user.created`, `user.logged_in`, `registration.created`, `notification.sent`, …) → ghi audit; có guard tránh log lặp với `audit.logged`.
- **Pattern áp dụng**: Chủ đạo là **Event Notification** – sự kiện mang `userId`, `eventId`; `notification-service` tự tra cứu email từ `user-service` trước khi gửi.
- **Topics chính trong dự án**:
  - `user.created`, `user.logged_in`, `user.updated`
  - `event.created`, `event.updated`
  - `registration.created`, `registration.cancelled`
  - `notification.sent`, `notification.failed`
  - `audit.logged`, `audit.failed`
- **Lưu trữ**: Postgres cho nghiệp vụ (user, event, registration); MongoDB cho audit log.
- **Quan sát**: Kafka UI tại `http://localhost:8080`. Frontend tại `http://localhost:3000`. Gateway tại `http://localhost:3007`.
- **Email mặc định demo**: `truongkiet771@gmail.com` (cấu hình tại `docker-compose.yml` và `notification-service/.env`).

6) Mini flow minh họa

1. Frontend gọi `POST /users` → `user-service` tạo user → emit `user.created` → `auditlog-service` ghi log.
2. Frontend `POST /auth/login` → emit `user.logged_in` → `auditlog-service` ghi log.
3. Frontend `POST /registrations` (qua `gateway`) → emit `registration.created` →
   - `event-service` cập nhật số lượng tham gia (có thể emit `event.updated`).
   - `notification-service` tra cứu email, gửi thư → emit `notification.sent`.
   - `auditlog-service` ghi lại tất cả hành vi.

7) Khía cạnh sản xuất (nên cân nhắc nếu mở rộng)

- **Outbox + CDC (Debezium)** để đảm bảo đồng nhất DB-Kafka khi ghi sự kiện.
- **Idempotent consumer** (khóa duy nhất, upsert) và **exactly-once** (Kafka transactions) khi cần.
- **Retry/backoff & DLQ** rõ ràng cho các consumer quan trọng.

- **Schema Registry** để quản lý thay đổi schema.
- **Observability**: metrics (Prometheus/Grafana), tracing (OpenTelemetry), structured logs.
- **Bảo mật**: TLS/SASL cho Kafka, secrets management (ENV, Vault), giới hạn quyền.

8) Tài liệu nội bộ khác

- Quy trình demo chi tiết: xem `demo.md`.
- Kiến trúc & services: xem `README.md`, `docker-compose.yml`, `shared/event-types.js`.