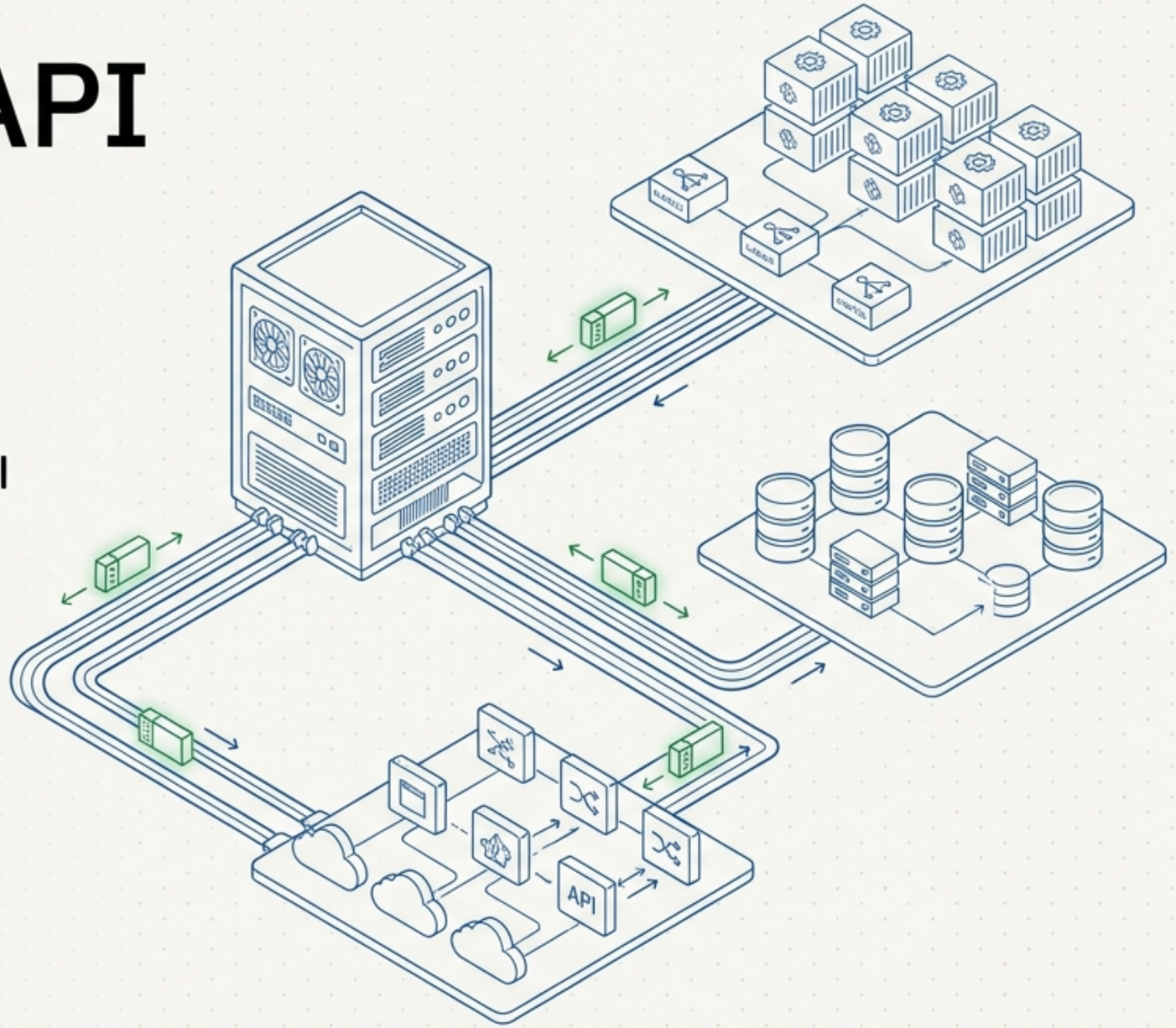
Архитектура API и интеграций

Главный чертёж аналитика: от контракта до экосистемы



ПРОЕКТИРОВАНИЕ ОТКАЗОУСТОЙЧИВЫХ РАСПРЕДЕЛЁННЫХ СИСТЕМ

Аналогия системы: API как официант

Клиент

Формирует заказ. Не имеет прямого доступа к базе данных (кухне).

HTTP Request

HTTP Response

API / Официант

Точка входа. Принимает заказ (Request), передаёт на сервер, возвращает результат (Response).

HTTP Request

HTTP Response

Сервер / Кухня

Обрабатывает бизнес-логику. Готовит данные.

⚠ Главный принцип REST (Stateless): Сервер не запоминает клиентов. Каждый запрос содержит всю информацию (токен), необходимую для выполнения. Сессии под запретом.

Анатомия HTTP-пакета и Content Negotiation

Запрос (Request)

```
POST /api/v1/tasks HTTP/1.1
```
(Метод + Эндпоинт)

```
Authorization: Bearer eyJhb...
```
(Токен)

```
Content-Type: application/json
```
(Формат тела запроса)

```
Accept: application/json; q=1.0
```
(Требуемый формат ответа.
q-factor = приоритет)

Ответ (Response)

```
HTTP/1.1 201 Created
```
(Статус-код)

```
Content-Type: application/json
```
(Реальный формат ответа)

```
{
  "id": 42,
  "title": "New Task"
}
```
(Тело / Body)

Content Negotiation — процесс согласования форматов. Если сервер не поддерживает запрошенный в `Accept` формат, он вернёт ошибку 406 Not Acceptable.

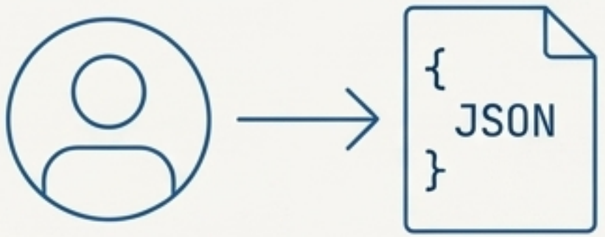
Матрица HTTP-методов и Идемпотентность

Метод	Действие	Аналог SQL	Безопасный (Safe)	Идемпотентный
GET	Прочитать	SELECT	Да ✔	Да ✔
POST	Создать	INSERT	Нет ✘	Нет (создаёт дубликаты)
PUT	Полностью заменить	REPLACE	Нет ✘	Да ✔
PATCH	Частично обновить	UPDATE	Нет ✘	Нет (зависит от реализации)
DELETE	Удалить	DELETE	Нет ✘	Да ✔

Идемпотентность: Свойство метода: вызов 1 раз или 10 раз подряд даёт одинаковый результат для состояния сервера.

Катастрофа №1: Исчезнувшее описание товара

1. Намерение



Задача: обновить только цену товара.

```
{
  "price": 9900
}
```

2. Исполнение (ОШИБКА)

Разработчик использует метод PUT вместо PATCH. Согласно RFC, PUT — это полная замена ресурса.

```
PUT /products/42
```

3. Последствия

id	title	description	price
1	~~NULL~~	~~NULL~~	9900
...			...

Сервер заменяет весь объект. Поля, которых не было в запросе, затираются.

```
title: NULL
description: NULL
price: 9900
```

Решение: Для частичного обновления используйте PATCH (JSON Merge Patch, RFC 7396).
Строго документируйте обязательные поля для PUT.

Язык машин: HTTP Статус-коды

Успех (2xx)

✅ `200 OK` — успешно
✅ `201 Created` — создано через POST
✅ `204 No Content` — успешно удалено через DELETE

Доступ (4xx)

⚠️ `401 Unauthorized` — нет токена или истёк
⚠️ `403 Forbidden` — токен есть, но нет прав на ресурс

Данные (4xx)

❗ `400 Bad Request` — невалидный JSON
`404 Not Found` — ресурс не найден
`409 Conflict` — дубликат email
`422 Unprocessable Entity` — ошибка бизнес-логики
`429 Too Many Requests` — превышен Rate Limit

Сервер (5xx)

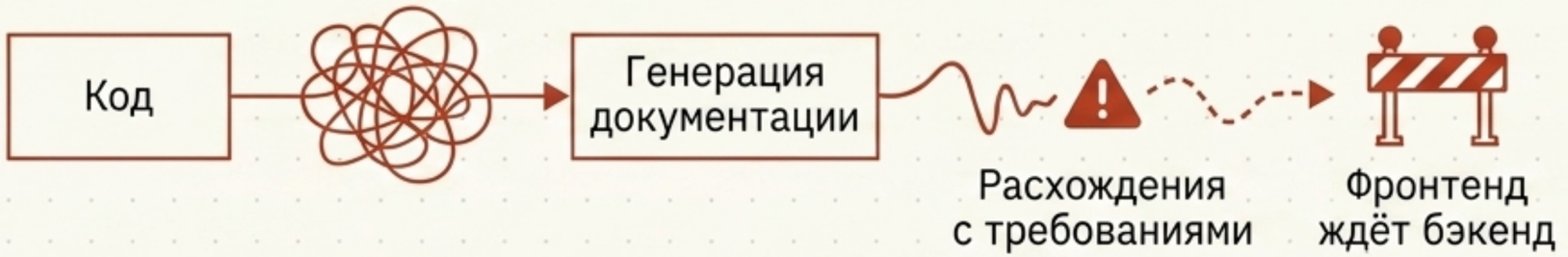
🕸️ `500 Internal Server Error` — падение кода/БД
🕸️ `502 Bad Gateway` — ошибка прокси

Важно: Если аналитик не опишет в OpenAPI код 500, клиентское приложение не сможет обработать падение сервера и покажет белый экран.

Контракт: OpenAPI и подход Design-First

OpenAPI Specification (OAS) — машиночитаемый стандарт описания REST API (текущая версия: 3.1).

Code-First (Устаревший подход)



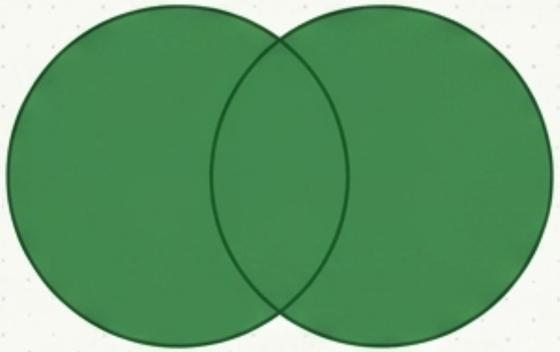
Design-First (Золотой стандарт)

Чек-лист Code Review

- ☑ - operationId уникален и понятен.
- ☑ - nullable явно указан для опциональных полей.
- ☑ - Ответы с ошибкой имеют единый формат ErrorResponse.
- ☑ - Пагинация вынесена в переиспользуемый компонент ($ref).

Полиморфизм схем JSON (OpenAPI 3.1)

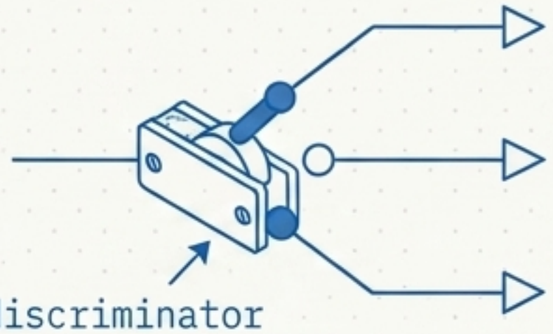
allOf (AND / Наследование)



Объединение. Схема должна соответствовать ВСЕМ перечисленным массивам.

Пример: Базовая модель Task + расширенные поля ExtendedFields для детального ответа.

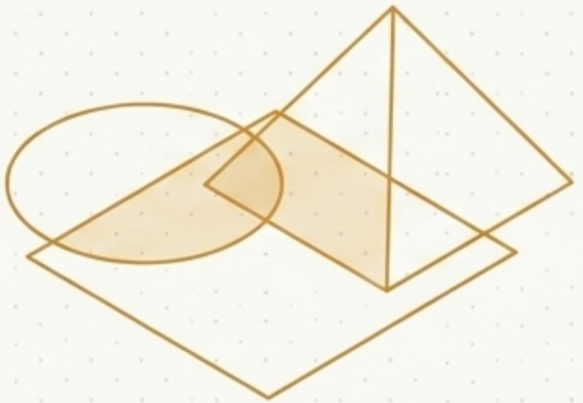
oneOf (XOR / Дискриминатор)



Ровно одна. Схема строго соответствует одной из моделей. Требует поле discriminator.

Пример: Уведомления (тип 'email' ИЛИ 'push' ИЛИ 'sms').

anyOf (OR / Нестрогое объединение)



Хотя бы одна. Мягкое условие для частичной валидации.

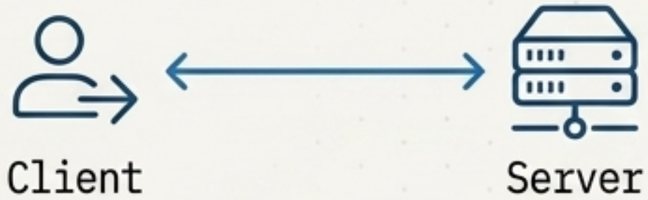
Пример: Поле может быть массивом ИЛИ единичной строкой.

Распутье интеграций: Синхронно или Асинхронно?

Нужна интеграция?

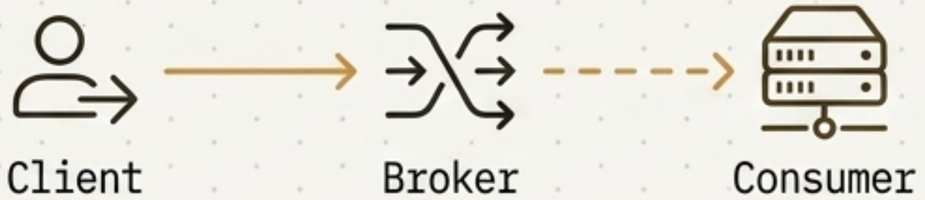
Синхронная (REST, gRPC)

- Поведение: Request -> Ждёт -> Response. Клиент блокируется.
- Риск: Каскадное ожидание. Если сервис 'C' завис, сервисы 'A' и 'B' исчерпают пул потоков.
- Когда выбирать: Нужен немедленный ответ (профиль пользователя).



Асинхронная (Kafka, RabbitMQ)

- Поведение: Event -> Message Broker -> Потребитель. Fire-and-forget.
- Риск: Сложность трассировки, дубликаты, задержки.
- Когда выбирать: Изоляция отказов, слабая связанность, фоновые процессы.

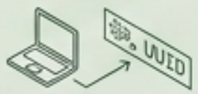


⚠ Миф о доставке и Идемпотентность

Модель доставки	Суть	Следствие	Применение
At-most-once (Максимум 1 раз)	Возможна потеря сообщений.	Нет дубликатов.	Метрики, логи.
At-least-once (Минимум 1 раз)	Потерь нет.	Будут дубликаты.	Самый частый стандарт индустрии.
Exactly-once (Ровно 1 раз)	Миф.	Технически невозможен в распределённых системах.	Теорема FLP.

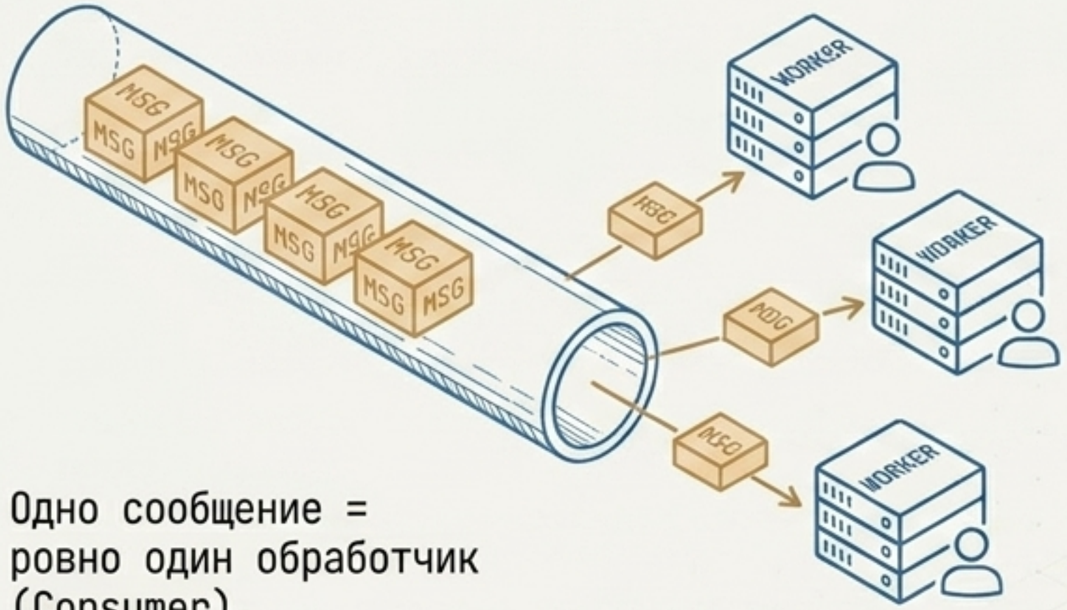
At-least-once + Idempotency = Effectively-once

Чтобы эмулировать Exactly-once, аналитик должен требовать идемпотентность потребителя. Клиент генерирует Idempotency-Key (UUID). Сервер проверяет ключ в БД. Если дубликат — возвращает сохранённый ответ без повторного выполнения бизнес-логики.



Маршрутизация: Queue vs Pub-Sub

Message Queue (Очередь / Точка-точка)

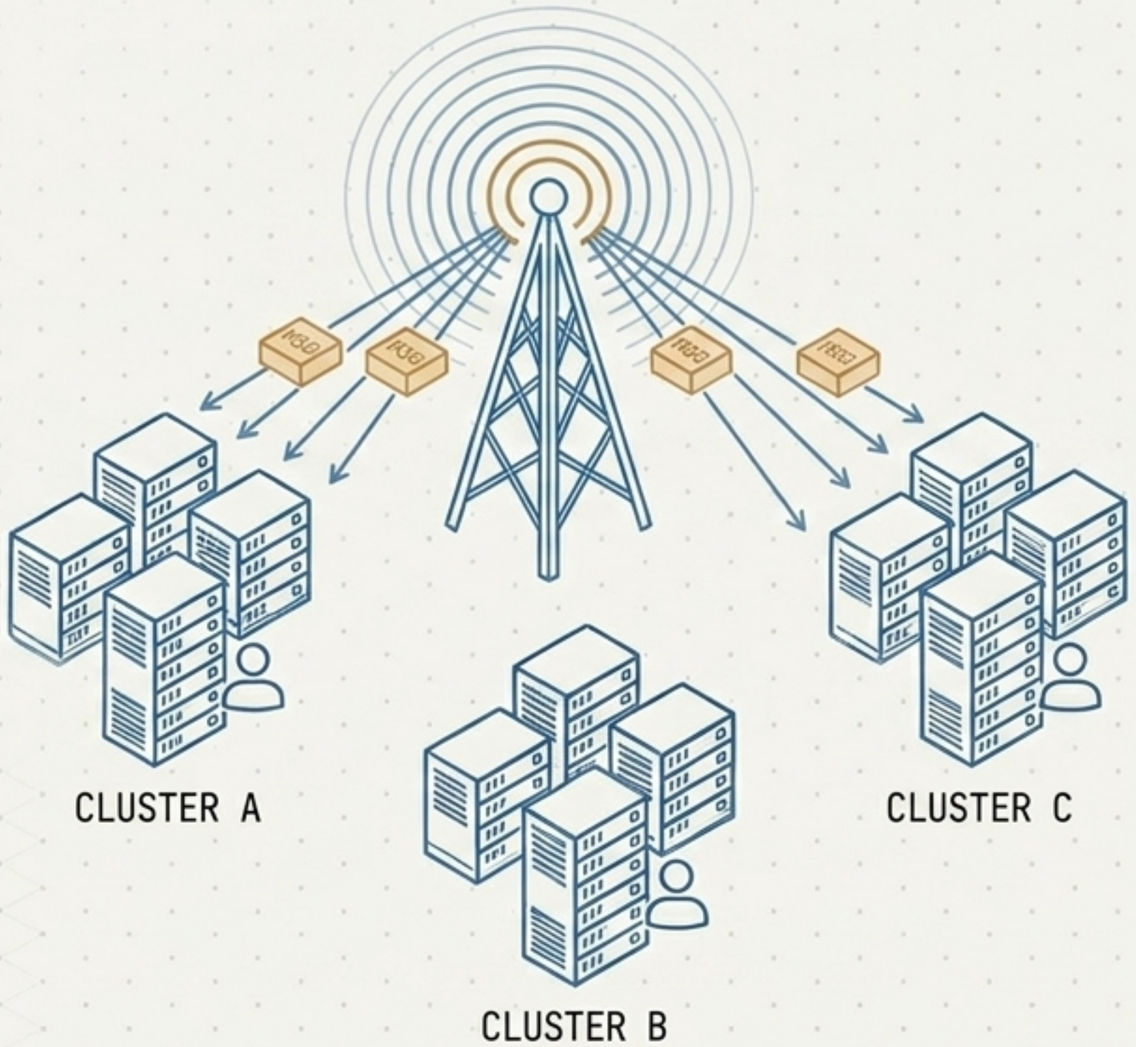


Одно сообщение =
ровно один обработчик
(Consumer).

Одно сообщение = N независимых подписчиков.
Издатель не знает, кто его слушает (слабая связанность).
Гарантируется порядок FIFO.

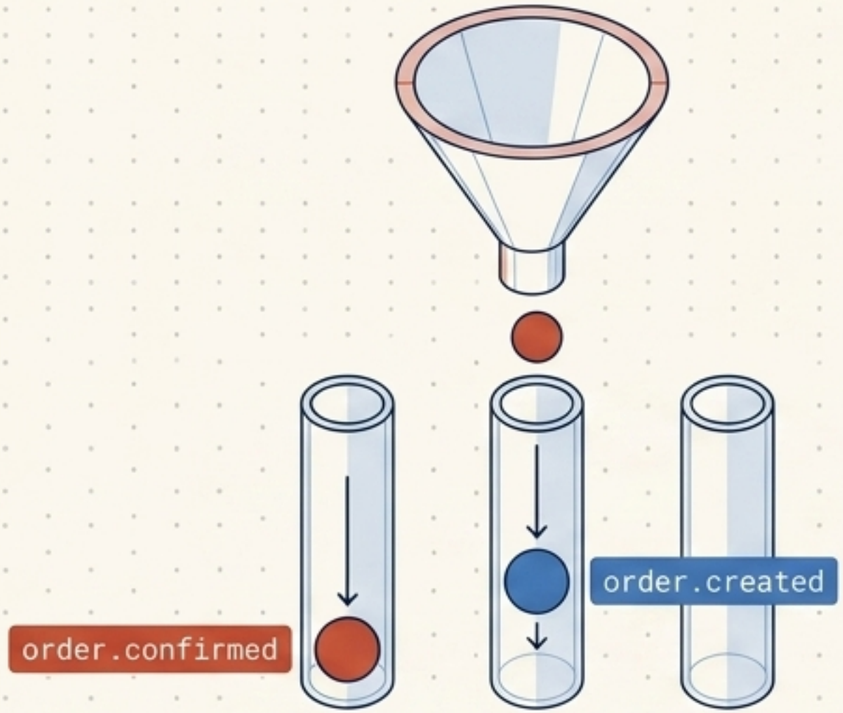
Пример: Apache Kafka.
Сценарий: Событие task.created. Уведомления, Аналитика и Поиск реагируют одновременно.

Event Bus / Pub-Sub (Издатель-Подписчик)



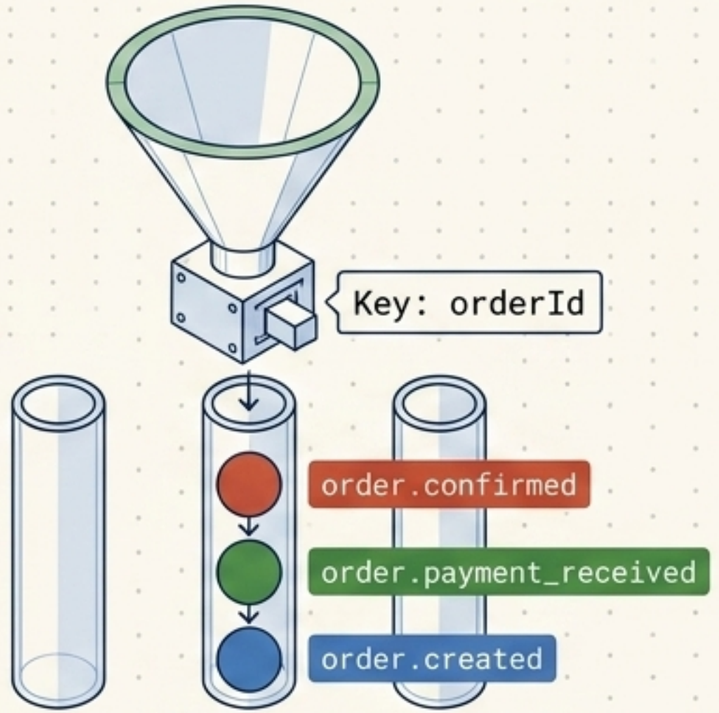
Катастрофа №2: Хаос в партициях Kafka

Случайный ключ = Потеря порядка



В Kafka порядок сообщений строго гарантирован только внутри одной партиции. Если ключ пустой или случайный, события одной сущности разлетаются по разным партициям. Склад получает confirmed раньше created.

Ключ партиционирования (orderId) = Строгий порядок

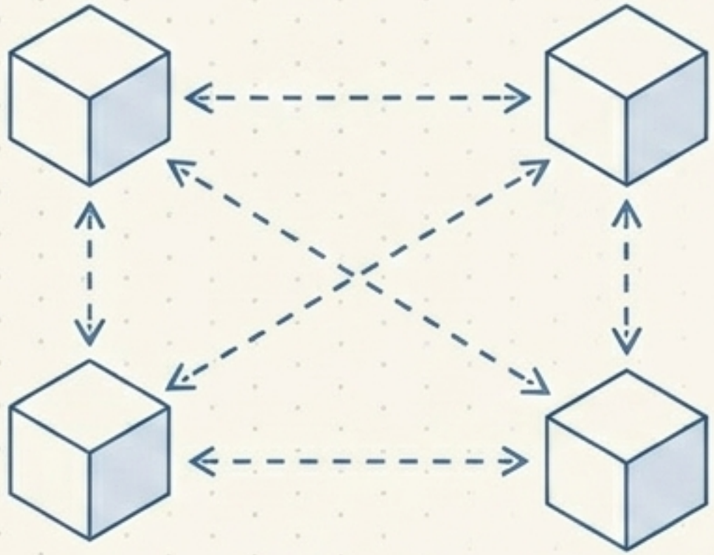


Решение: Все события одной бизнес-сущности должны иметь единый ключ. Укажите в NFR: "События публикуются с ключом orderId. Гарантируется строгий порядок в пределах одной сущности".

Распределённые транзакции: Паттерн Saga

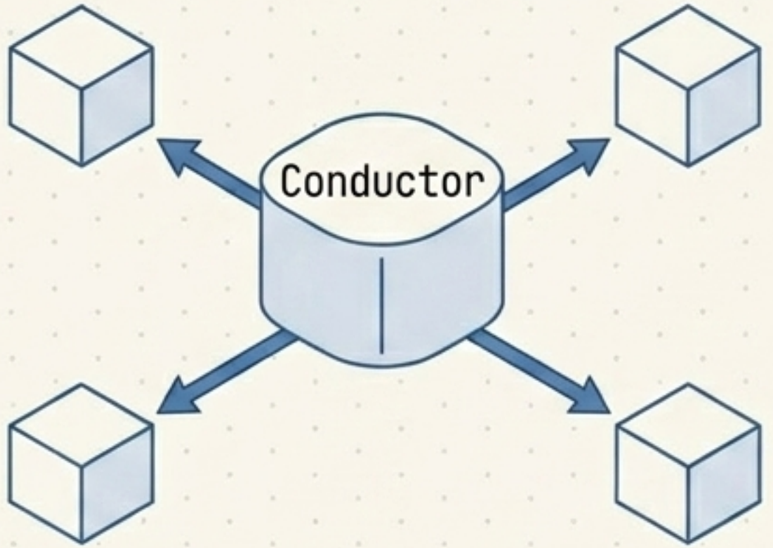
Суть: В микросервисах нет ACID-транзакций. Если финальный шаг падает, мы должны запустить компенсационные транзакции (откат) для всех предыдущих шагов.

Choreography (Хореография)



- Децентрализовано. Каждый сервис слушает других.
- Децентрализовано. Каждый сервис слушает других.
- Плюс: нет единой точки отказа.
- Минус: спагетти-события, сложно отслеживать.
- Идеально для простых потоков (2-3 шага).

Orchestration (Оркестрация)



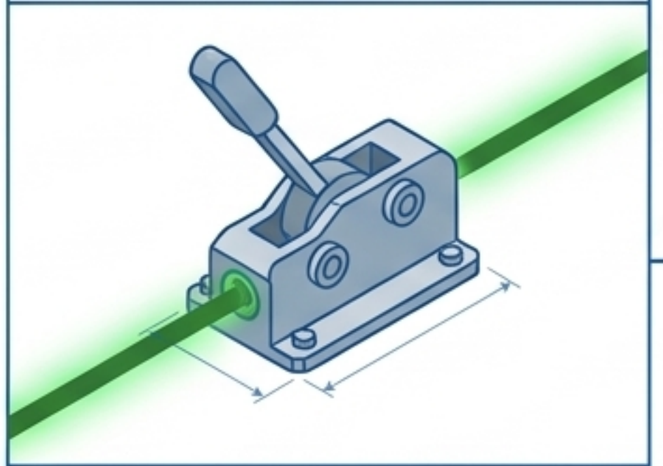
- Централизовано. Оркестратор управляет порядком.
- Плюс: весь процесс виден в одном месте.
- Минус: единая точка отказа, оркестратор знает слишком много.
- Идеально для сложных процессов (Camunda).

Предохранитель: Паттерн Circuit Breaker ⚠

Проблема (Каскадный отказ):
Если внешний сервис лёг, попытки достучаться (Retries) создадут эффект Thundering Herd, добивающее систему и сжигающее пулы потоков.

Успех

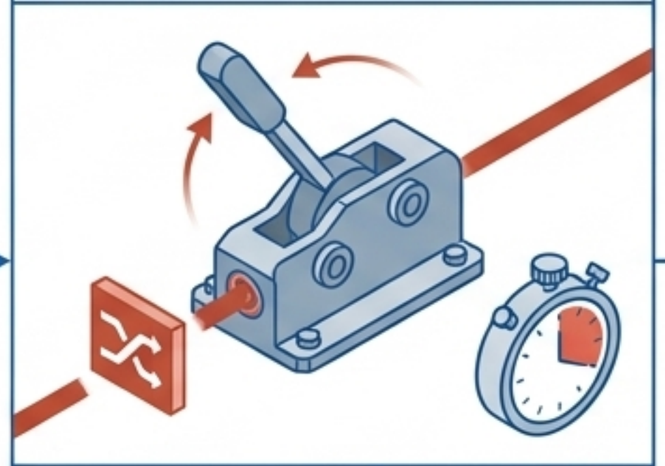
CLOSED (Закрыт)



Запросы идут.
Считается % ошибок.

Ошибка
> Порог

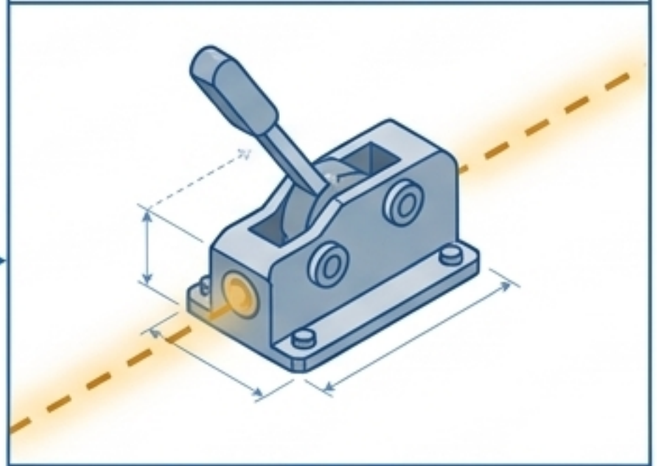
OPEN (Открыт)



Порог ошибок превышен.
Цепь размыкается. Запросы блокируются мгновенно, возвращается fallback.
Ждём таймаут.

Таймаут
истёк

HALF-OPEN (Полуоткрыт)



Пропускаем 1-2 пробных запроса. Если успех – закрываем цепь. Если ошибка – снова OPEN.

Ошибка

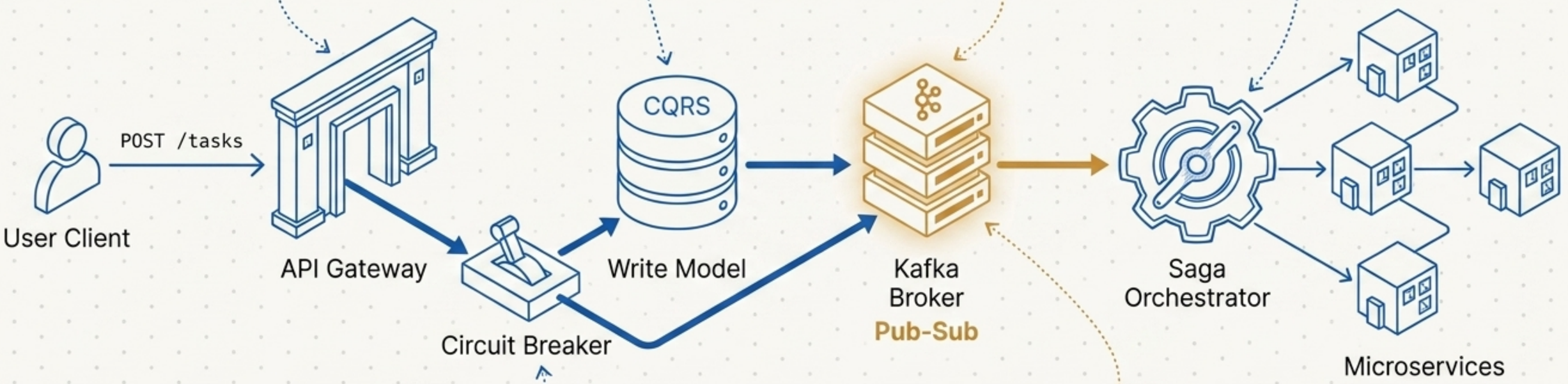
Синтез экосистемы: Жизненный цикл одного клика

1. Вход (API Gateway): Единая точка входа. Маршрутизация, проверка JWT-токена.

2. Запись (CQRS + Event Sourcing): Запрос попадает в оптимизированную Write Model. Состояние иммутабельно.

4. Дистрибуция (Message Broker): Событие попадает в Kafka. Строгий порядок обеспечен ключом taskId.

5. Оркестрация (Saga): Брокер запускает распределённую транзакцию.



3. Защита (Resilience): Circuit Breaker и Idempotency-ключи дежурят на переходе.

4. Дистрибуция (Message Broker): Событие попадает в Kafka. Строгий порядок обеспечен ключом `taskId`.