SolarLab>_

Взаимодействие сервисов: независимые данные и event-driven архитектура

Алексей Андреев https://t.me/AlekseyAndreev1984

Содержание

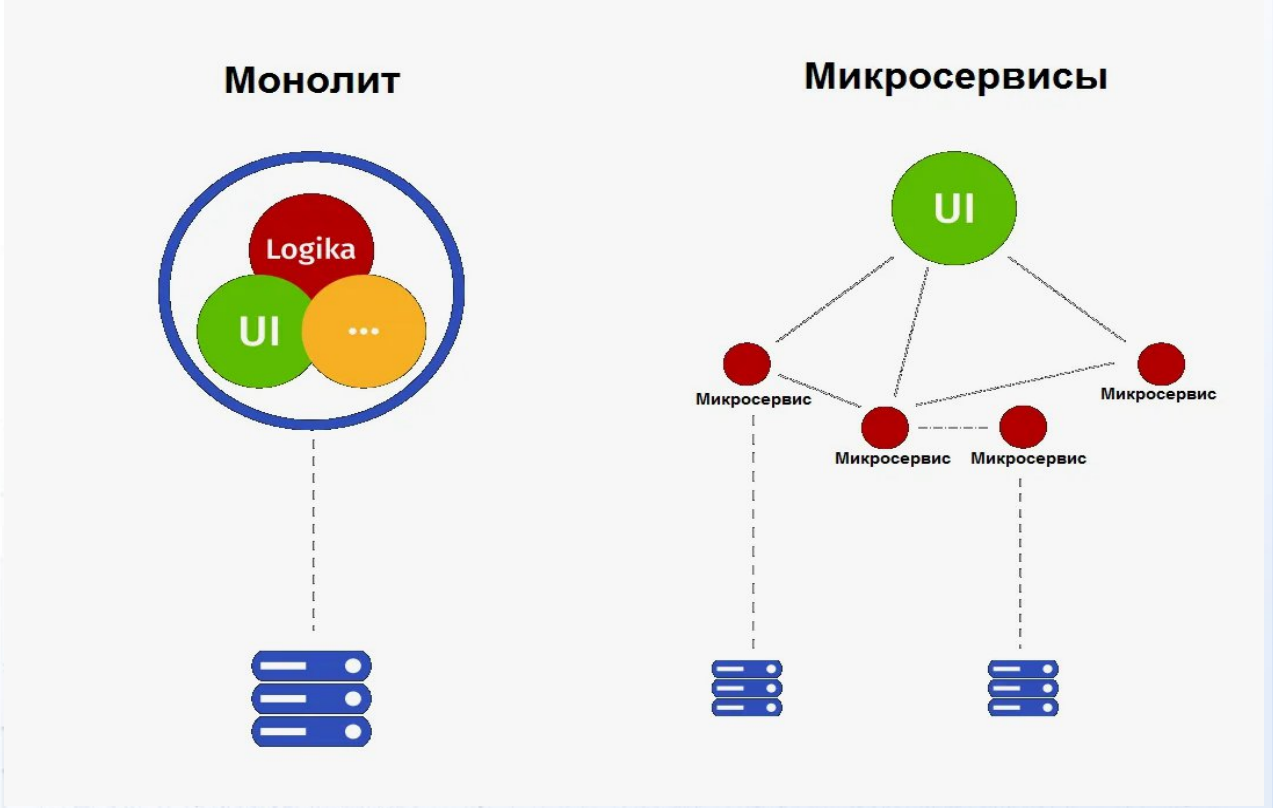
- История из жизни - как возник доклад
- Основная проблема, Цель доклада
- Почему данные сервиса должны быть в его собственной БД?
- Схема взаимодействия: событийная модель, плюсы и минусы подхода
- Советы
- Выводы
- Вопросы

SolarLab>_

История из жизни - как возник доклад

История из жизни - как возник доклад

У нас был монолит. С одной большой базой данных. Монолит стал перероастать в микросервисы. У каждого микросервиса своя база данных. Если нужны общие данные, то мы можем ходить к одному источнику данных, либо иметь все данные, необходимые для работы внутри собственной БД.



Как выбирали – CAP теорема

Если перенести CAP теорему с БД на взаимодействие сервисов, то мы выбрали AP(доступность и устойчивость к фрагментации)

SolarLab>_

Основная проблема, Цель доклада

Основная проблема: как обеспечить согласованность данных между сервисами?

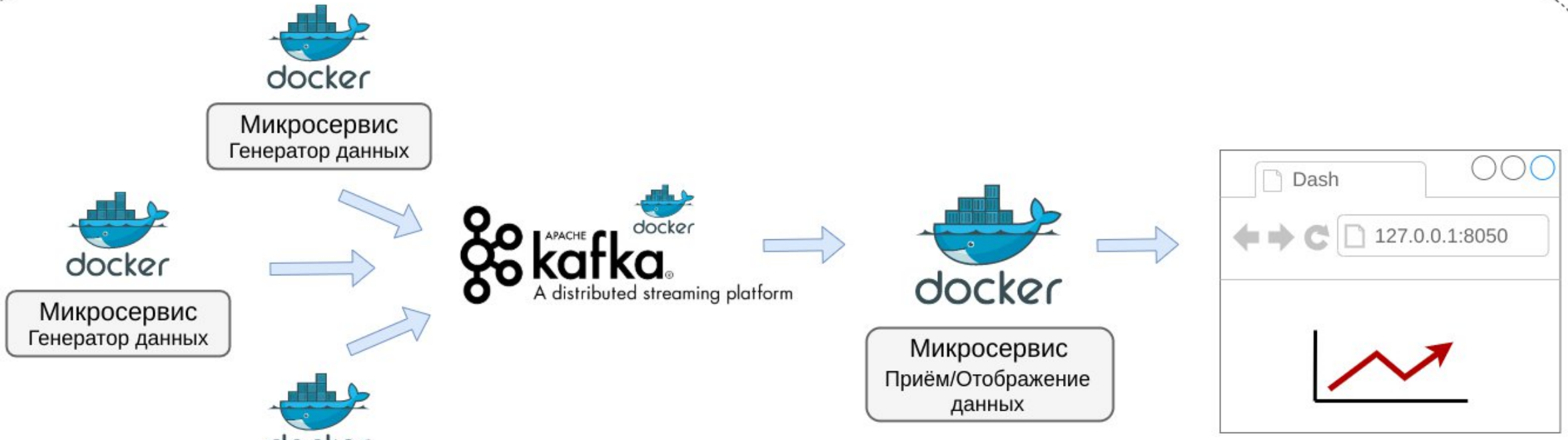
Каждый сервис работает автономно, управляя своим собственным набором данных. Как гарантировать, что данные, распределенные по разным сервисам, будут консистентными? Как избежать потери данных и обеспечить их актуальность в реальном времени?

Основная проблема: как обеспечить согласованность данных между сервисами?

Для обеспечения согласованности данных между сервисами с разными БД можно использовать:

- Событийно-ориентированную архитектуру: сервисы публикуют события при изменении данных.
- API-интерфейсы: сервисы обмениваются данными через API(HTTP REST API, gRPS, thrift, ODATA, что-то ещё)
- Использование единой базы данных: Если все сервисы используют одну и ту же базу данных.
- Использование распределенных транзакций: Распределенные транзакции могут быть использованы для обеспечения согласованности данных между сервисами. Они позволяют выполнять несколько операций как единое целое
- Использование паттерн "Saga": Паттерн Saga представляет собой методологию разбиения сложных и длинных транзакций на более мелкие, управляемые шаги. Каждый микросервис в цепочке выполняет свою локальную транзакцию, которая может быть атомарной и возвращать систему в предыдущее согласованное состояние при ошибке.
- Использование сервиса обнаружения изменений: Сервис обнаружения изменений может быть использован для обнаружения изменений данных в одном сервисе и уведомления других сервисов об этих изменениях.
- Другие способы

Цель доклада: разобрать схему, где каждый сервис владеет своими данными и распространяет их через события

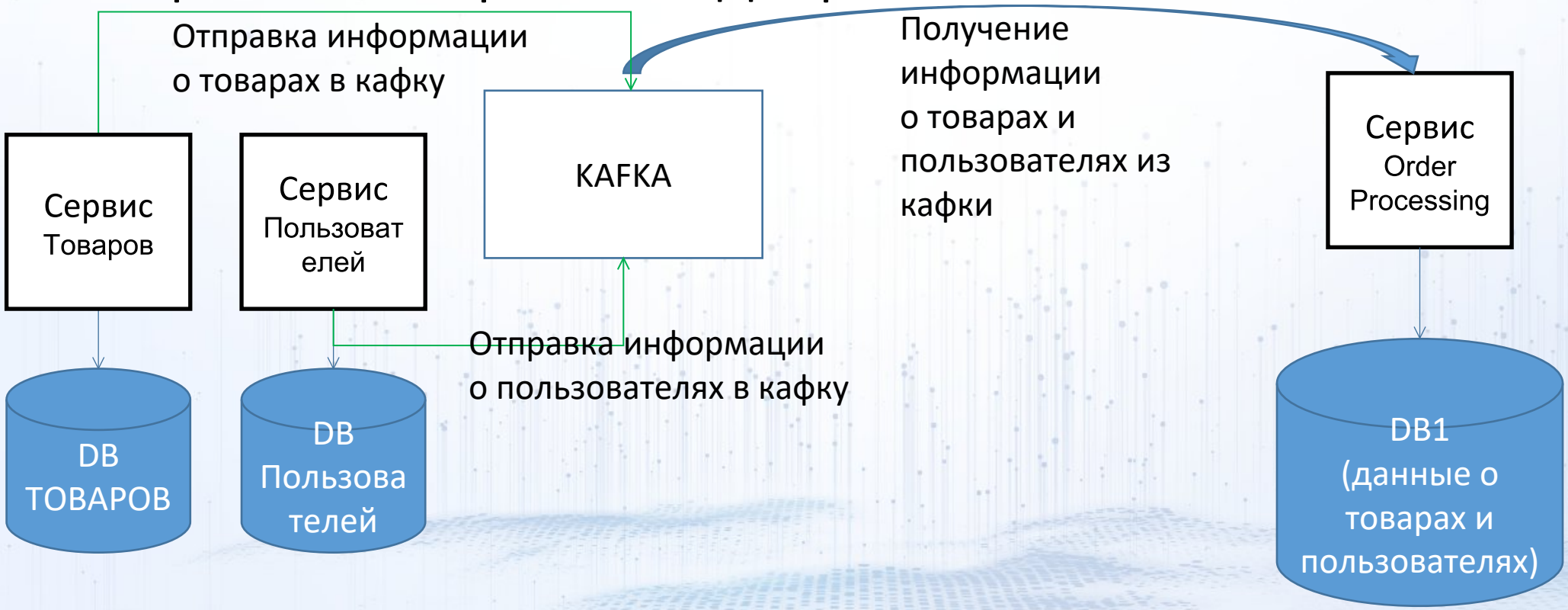


SolarLab>_

Почему данные сервиса должны быть в его собственной БД?

Почему данные сервиса должны быть в его собственной БД?

- Все данные, которые есть в нашей системе, и которые необходимы для работы сервиса – мы храним в БД сервиса.



Почему данные сервиса должны быть в его собственной БД?

В предметно-ориентированном проектировании (Domain-Driven Design, DDD) есть bounded context: каждый сервис — владелец своих данных.

Что такое Bounded Context?

Граница контекста (Bounded Context) — это четко определённая область внутри системы, в которой применяется конкретная модель предметной области. В DDD каждая модель имеет своё предназначение и ограничения, поэтому важно не смешивать различные модели в одном контексте.

Пример: В интернет-магазине есть два разных контекста:

Контекст управления товарами (Product Management) — здесь модель описывает характеристики товаров, их категории, цены и наличие на складе.

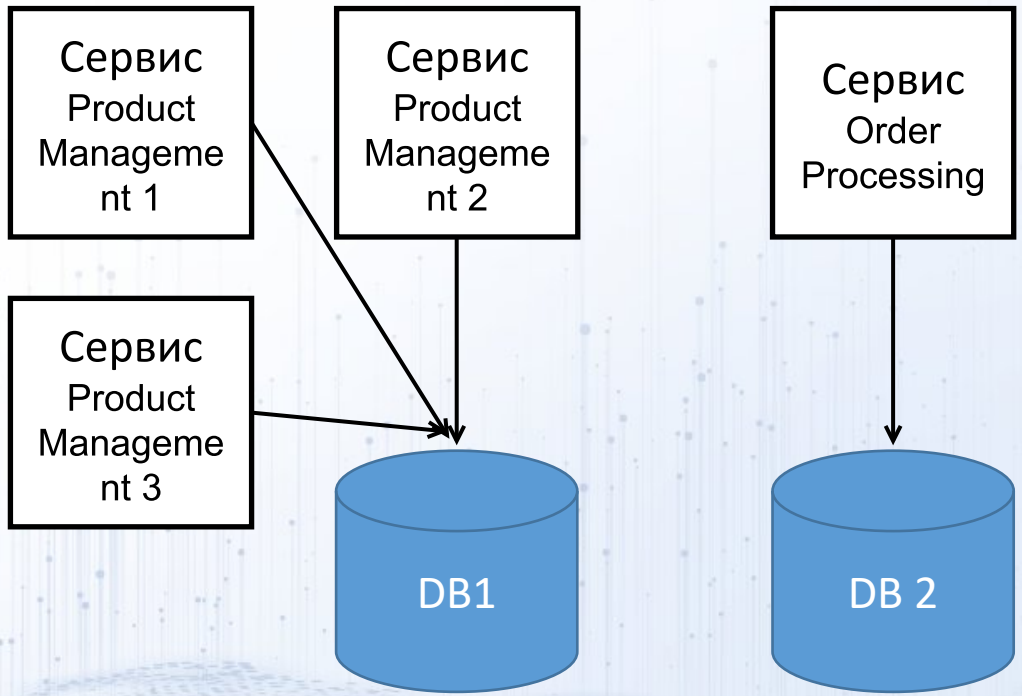
Контекст оформления заказов (Order Processing) — здесь модель фокусируется на заказах, клиентах и оплатах.

Хотя в обоих контекстах фигурирует понятие "товар", его модель в каждом случае будет разной. В контексте управления товарами важны атрибуты (название, описание, цена), а в контексте заказов — количество, статус и стоимость.

Почему данные сервиса должны быть в его собственной БД?

Независимость сервисов:

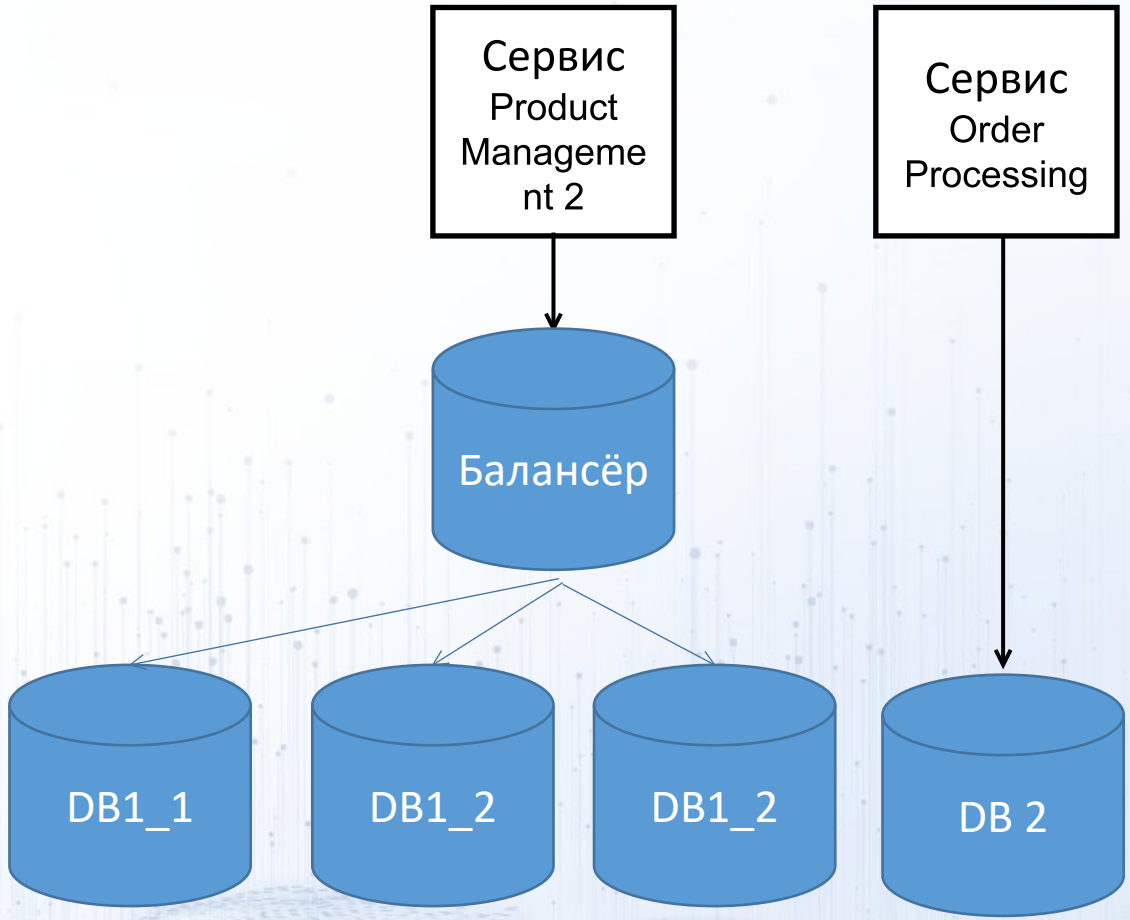
- **Масштабируемость**
- **Можно менять схему БД без влияния на другие сервисы**
- **Отказоустойчивость (падение одного сервиса не ломает всю систему).**
- **Гибкость в выборе технологий (разные БД для разных задач).**



Почему данные сервиса должны быть в его собственной БД?

Независимость сервисов:

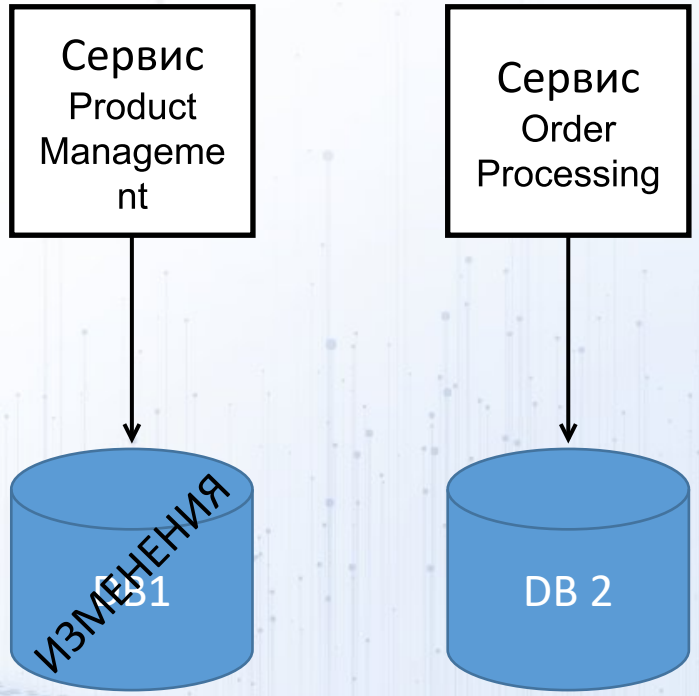
- **Масштабируемость**
- **Можно менять схему БД без влияния на другие сервисы**
- **Отказоустойчивость (падение одного сервиса не ломает всю систему).**
- **Гибкость в выборе технологий (разные БД для разных задач).**



Почему данные сервиса должны быть в его собственной БД?

Независимость сервисов:

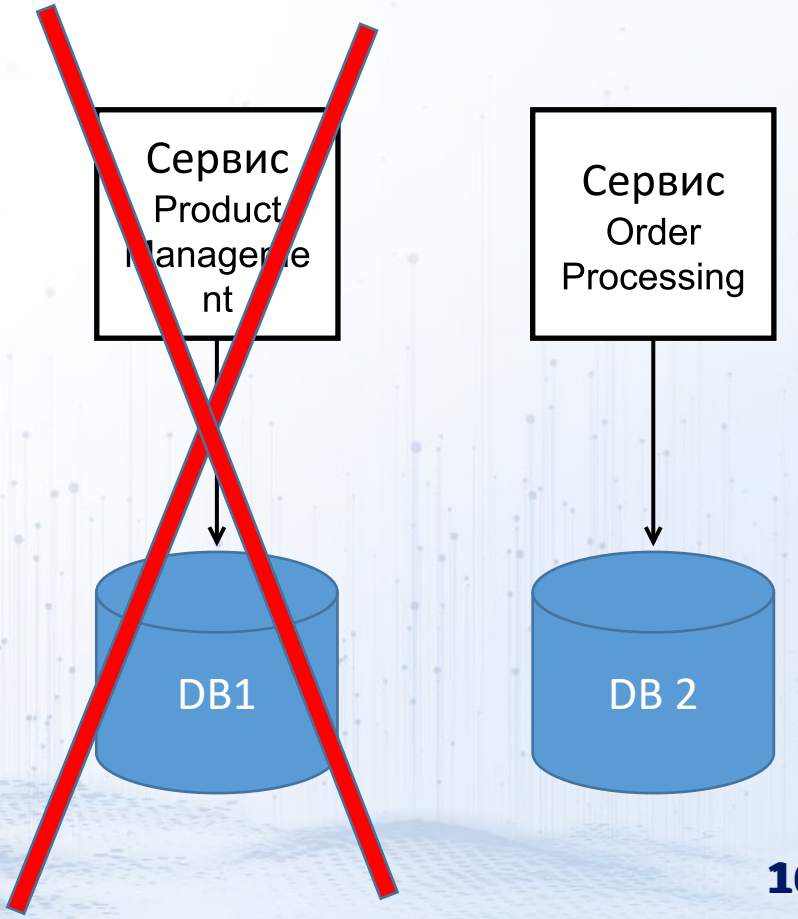
- **Масштабируемость**
- **Можно менять схему БД без влияния на другие сервисы**
- **Отказоустойчивость (падение одного сервиса не ломает всю систему).**
- **Гибкость в выборе технологий (разные БД для разных задач).**



Почему данные сервиса должны быть в его собственной БД?

Независимость сервисов:

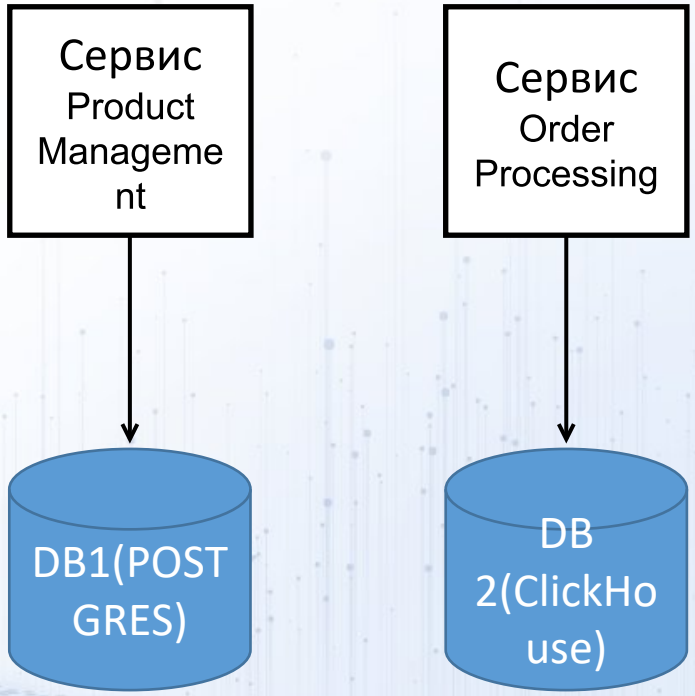
- **Масштабируемость**
- **Можно менять схему БД без влияния на другие сервисы**
- **Отказоустойчивость (падение одного сервиса или выход из строя БД не ломает всю систему).**
- **Гибкость в выборе технологий (разные БД для разных задач).**



Почему данные сервиса должны быть в его собственной БД?

Независимость сервисов:

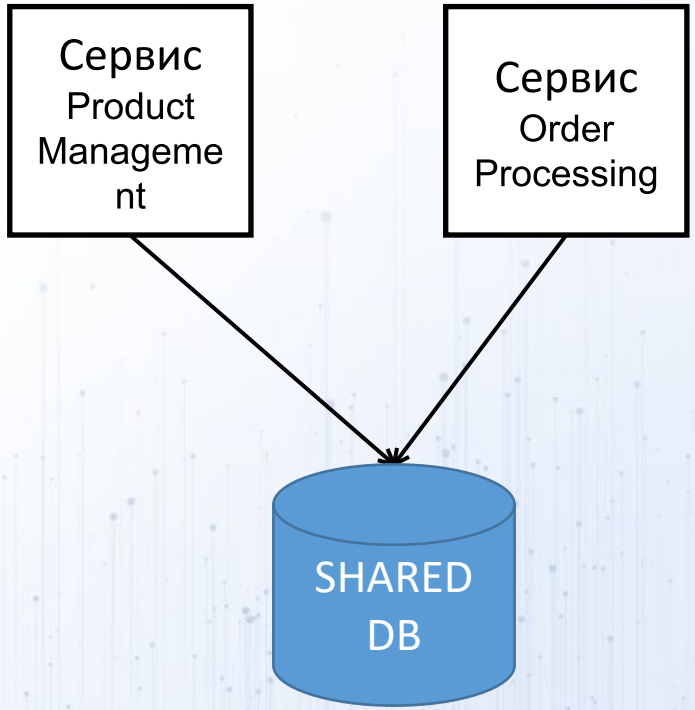
- **Масштабируемость**
- **Можно менять схему БД без влияния на другие сервисы**
- **Отказоустойчивость (падение одного сервиса или выход из строя БД не ломает всю систему).**
- **Гибкость в выборе технологий (разные БД для разных задач).**



Почему данные сервиса должны быть в его собственной БД?

Проблемы shared database:

- Жёсткая связность
- Блокировки
- Сложность изменений.



SolarLab>_

Схема взаимодействия: событийная модель

Схема взаимодействия: событийная модель

Основной flow:

1. Сервис-источник получает данные → сохраняет в свою БД.
2. Данные публикуются в очередь (Kafka/RabbitMQ/etc.).
3. Подписчики получают событие → обновляют свои БД.

Схема взаимодействия: событийная модель

Сервис-источник получает данные → сохраняет в свою БД.

Схема взаимодействия: событийная модель

Данные публикуются в очередь (Kafka/RabbitMQ/etc.).

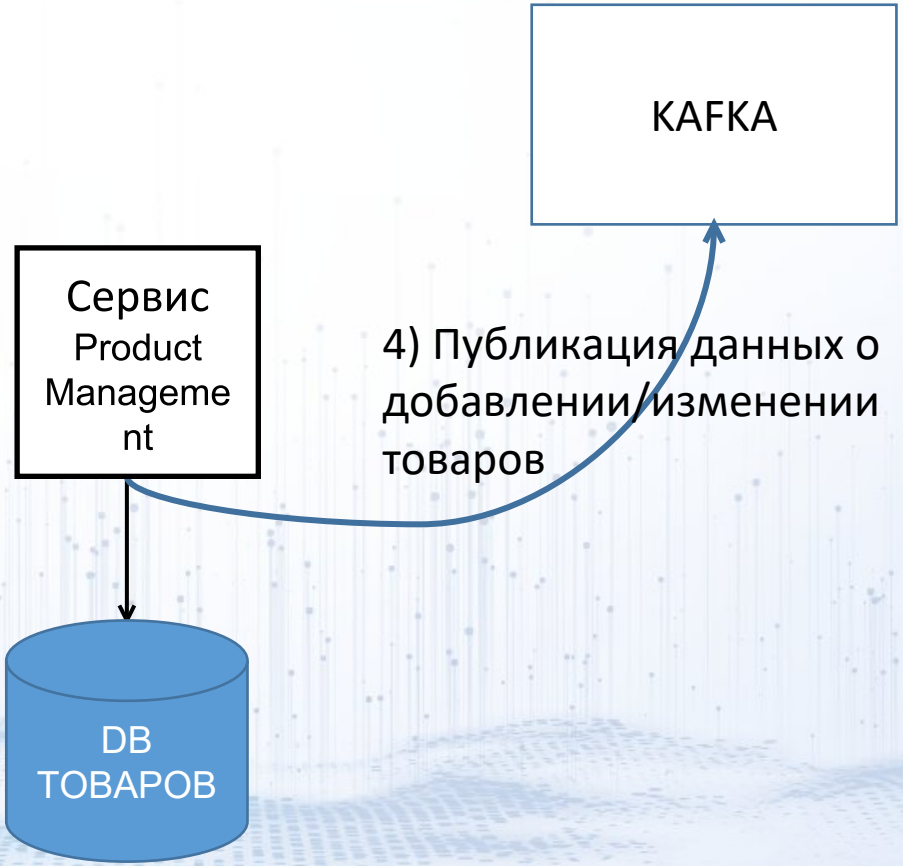


Схема взаимодействия: событийная модель

Подписчики получают событие → обновляют свои БД.

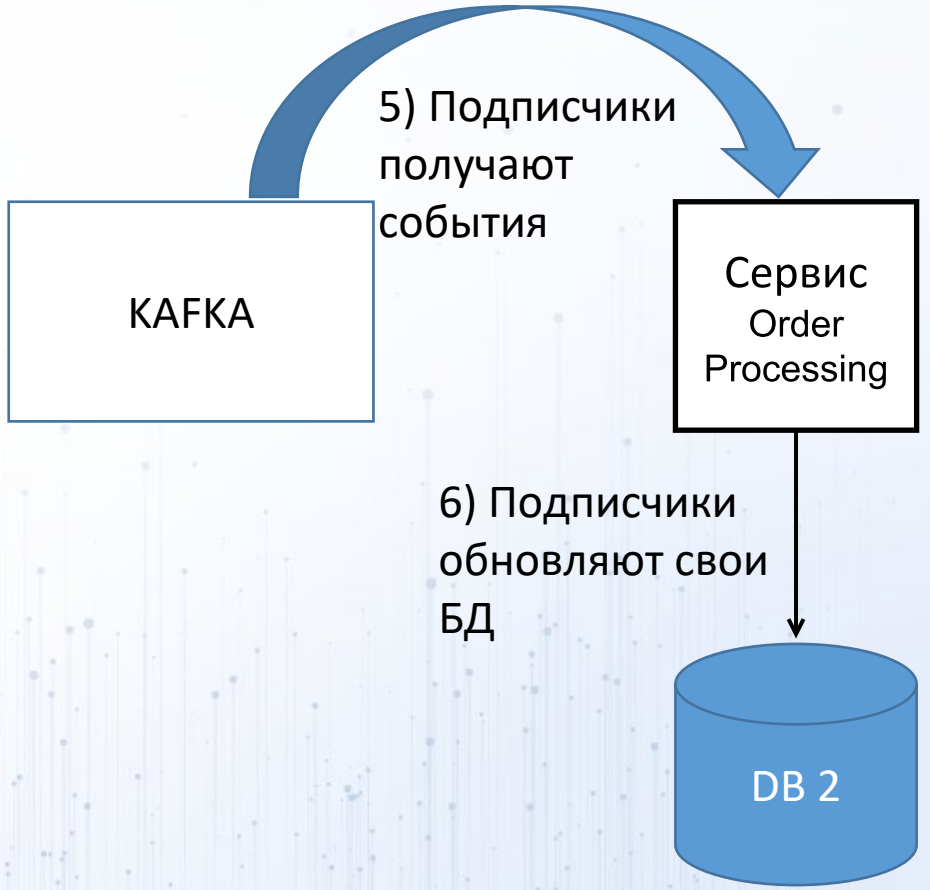


Схема взаимодействия: событийная модель

Основной flow:

1. Сервис-источник получает данные → сохраняет в свою БД.
2. Данные публикуются в очередь (Kafka/RabbitMQ/etc.).
3. Подписчики получают событие → обновляют свои БД.

Схема взаимодействия: событийная модель. Плюсы подхода

- **Асинхронность** (не нужно ждать ответа).
- Отказоустойчивость. Сервис работает только на своих данных, и при выходе из строя источника данных – работа продолжается(но новых данных не появляется)
- Отложенная обработка (подписчики могут работать в своем темпе).
- Масштабируемость (можно добавлять новых подписчиков без изменений источника).

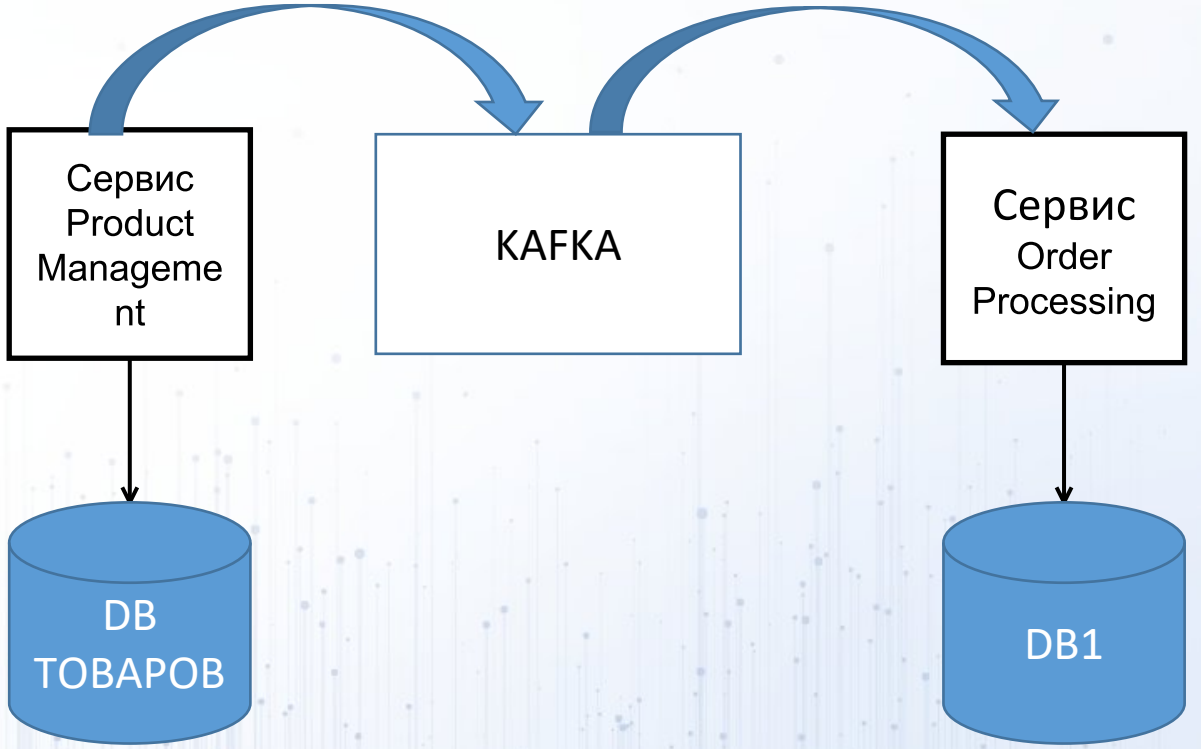


Схема взаимодействия: событийная модель. Плюсы подхода

- Асинхронность (не нужно ждать ответа).
- **Отказоустойчивость**. Сервис работает только на своих данных, и при выходе из строя источника данных – работа продолжается(но новых данных не появляется)
- Отложенная обработка (подписчики могут работать в своем темпе).
- Масштабируемость (можно добавлять новых подписчиков без изменений источника).

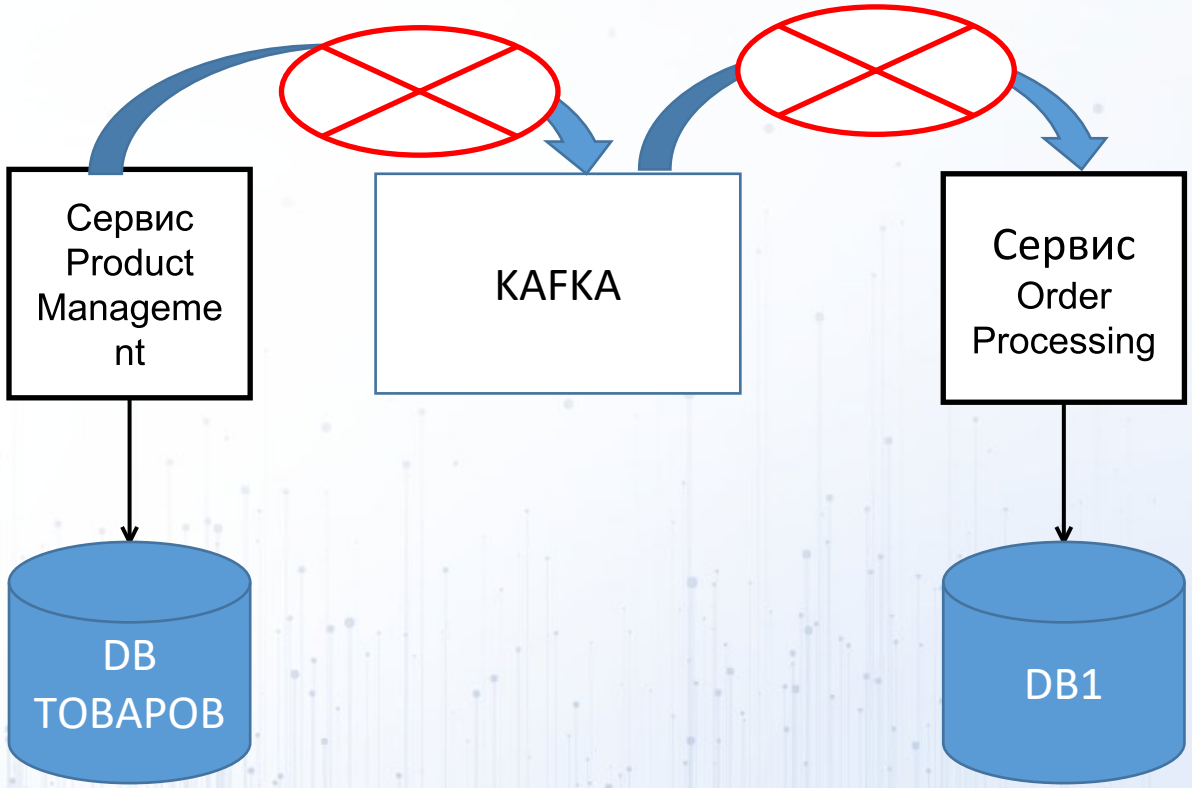


Схема взаимодействия: событийная модель. Плюсы подхода

- Асинхронность (не нужно ждать ответа).
- Отказоустойчивость. Сервис работает только на своих данных, и при выходе из строя источника данных – работа продолжается(но новых данных не появляется)
- **Отложенная обработка** (подписчики могут работать в своем темпе).
- Масштабируемость (можно добавлять новых подписчиков без изменений источника).

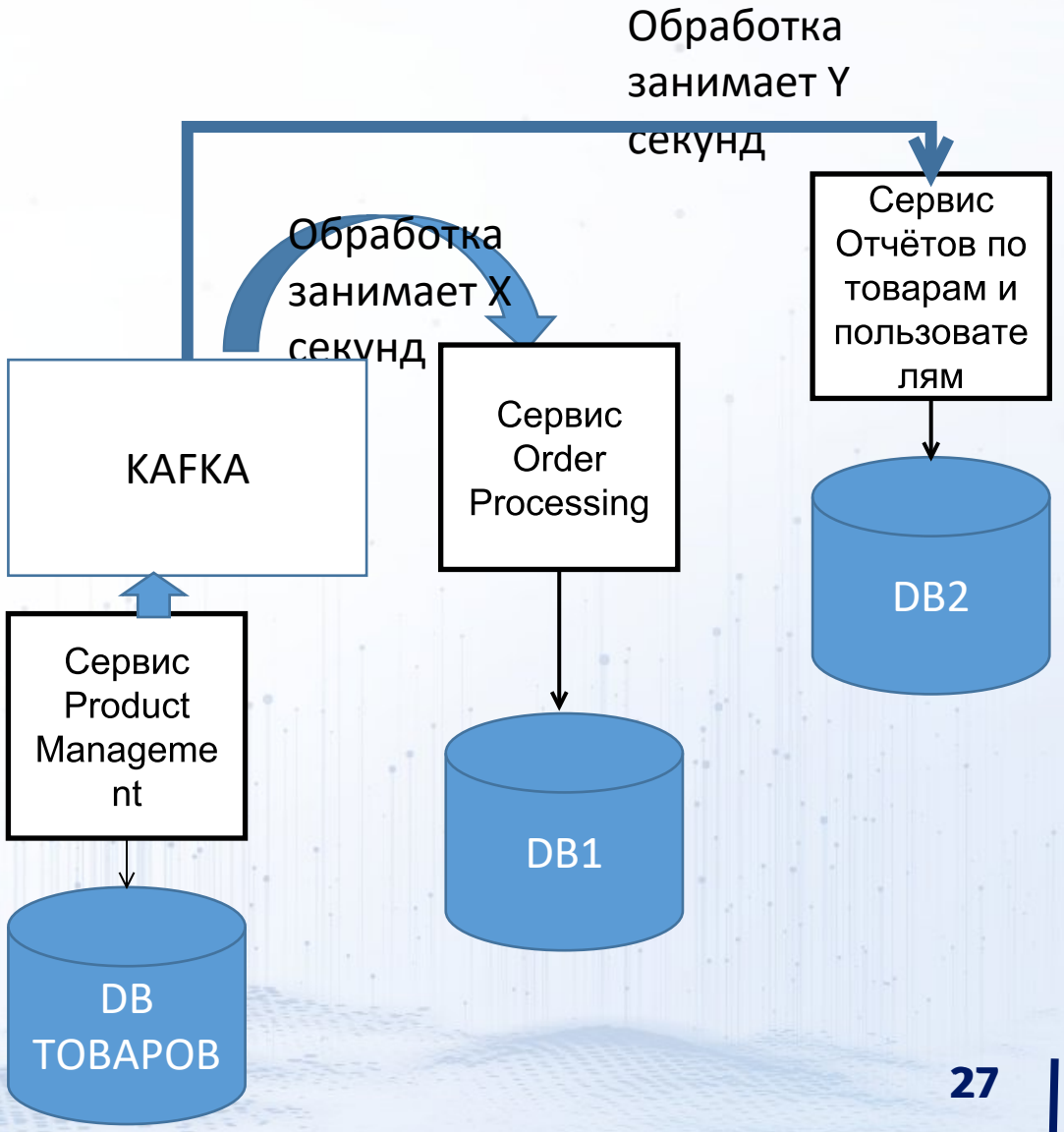


Схема взаимодействия: событийная модель. Плюсы подхода

- Асинхронность (не нужно ждать ответа).
- Отказоустойчивость. Сервис работает только на своих данных, и при выходе из строя источника данных – работа продолжается(но новых данных не появляется)
- Отложенная обработка (подписчики могут работать в своем темпе).
- **Масштабируемость** (можно добавлять новых подписчиков без изменений источника).

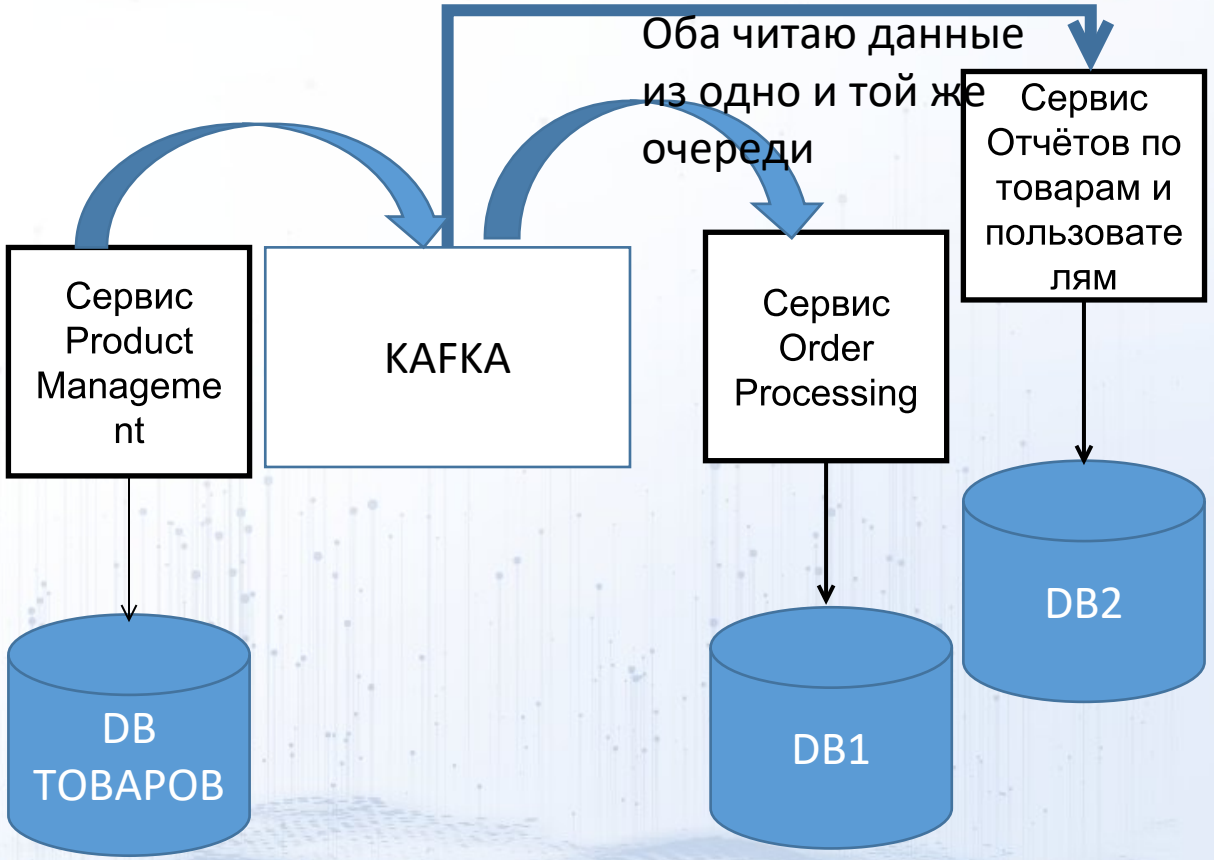


Схема взаимодействия: событийная модель. Минусы и подводные камни

- Обеспечение согласованности и целостности данных.
- Идемпотентность обработки событий (как избежать дублей). Обработка порядка событий.
- Отладка.
- Повышенная сложность.
- Решение проблем с задержкой и производительностью.
- Управление и мониторинг потоков событий. Распределённая трассировка и метрики
- Изменения контрактов шины данных ведёт к изменению всех потребителей
- Большая проблема роста объема сообщений и размера контрактов

SolarLab>_

Советы

Советы. Шина передачи данных

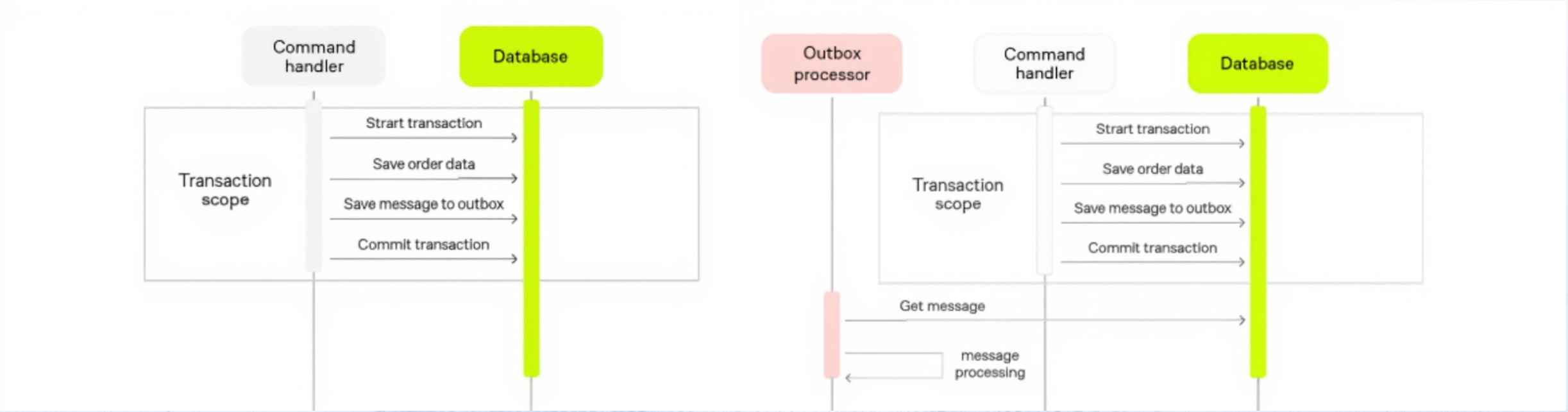
Используйте кафку, как шину данных, для взаимодействия.

- Кафка позволяет легко подключить новых подписчиков
- Кафка имеет возможность работать пачками
- Кафка имеет различные политики сжатия – есть delete, compact, delete + compact
- И много ещё чего

Советы. Kafka – политики сжатия

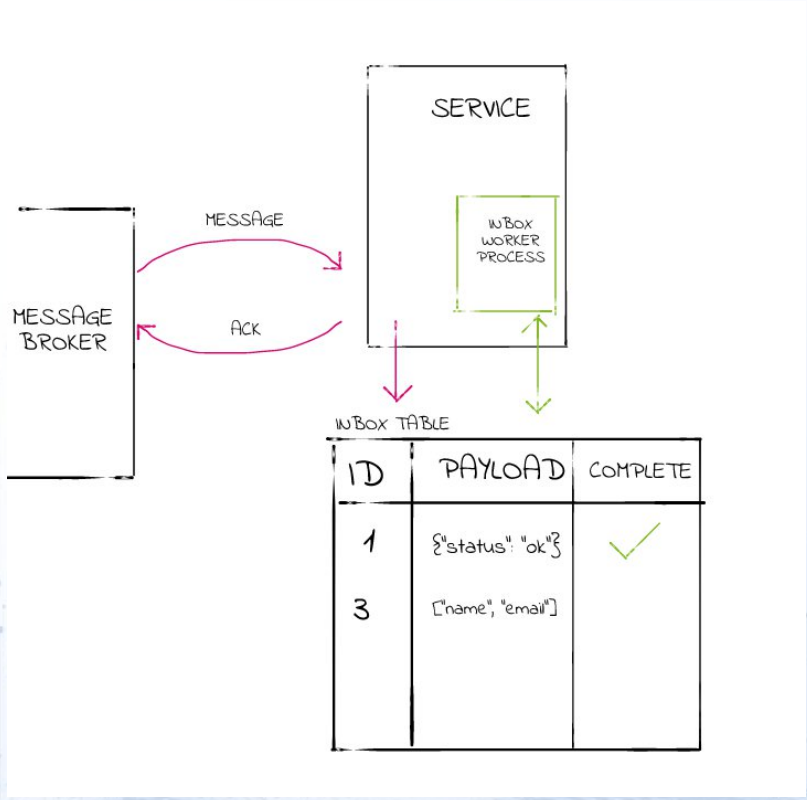
Советы. Работа с шиной данных. Outbox

- При работе с шиной данных используйте паттерн Outbox для отправки. Это помогает решать проблему гарантии доставки(At-most-once, At-least-once, Exactly once)



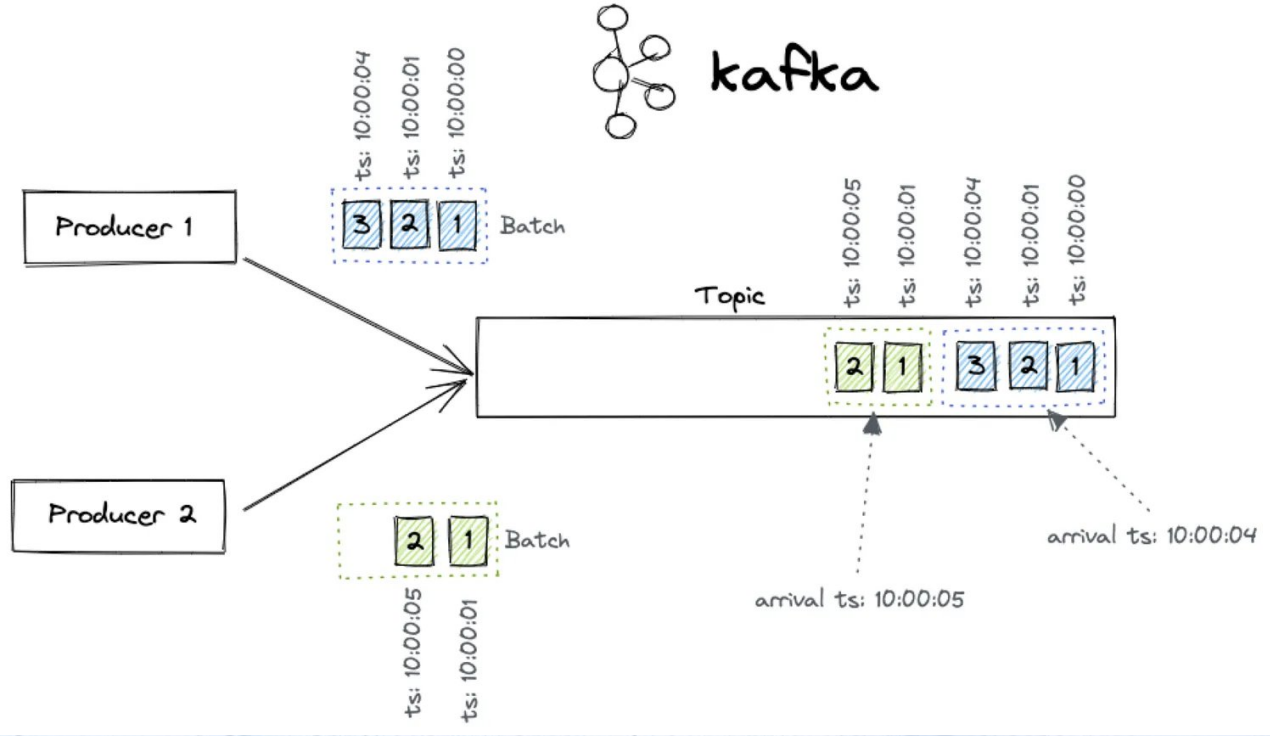
Советы. Работа с шиной данных. Inbox

При работе с шиной данных используйте паттерн Inbox для получения сообщения. Это поможет так же в случае, когда есть проблема зависимых сообщений



Советы. Порядок событий

При работе с шиной данных не забывайте про последовательность событий. В кафке это можно реализовать различными способами



Советы. Дубликаты

При работе с шиной данных не забывайте про дубликаты. Их не должно быть. Или заложить логику на случай дубликатов

Советы. Прямые вызовы своих же сервисов

Старайтесь избавляться от прямых вызовы своих же сервисов. В этом случае логика размазывается. И при выходе из строя одного сервиса – другие перестают работать

Советы. Документация

Тщательно документируйте все взаимодействия. Все топики(какие данные, кто пишет, кто читает). Как документировать потоки данных (Event Storming, диаграммы последовательностей)

Советы. Данные

Храните все необходимые вам данные для работы внутри вашего сервиса

Советы. Тестировщикам

Для тестировщиков: делайте моки очередей. Посмотрите, что такое Skaffold

Советы. Полезные книги

Прочитайте книгу Вульф Бобби, Хоп Грегор «Шаблоны интеграции корпоративных приложений» https://www.ozon.ru/product/shablony-integratsii-korporativnyh-prilozheniy-vulf-bobbi-hop-gregor-elektronnaya-kniga-946613685/

SolarLab>_

Выводы

Выводы

- Масштабируемость при создании изолированных сервисов - это парадигма, к достижению которой постоянно стремятся многие предприятия, а событийно-ориентированная архитектура построена с оглядкой на это. Благодаря инструментам, таким как Kafka или RabbitMQ, событийно-ориентированные архитектуры становятся более гибкими, универсальными и надежными, что удовлетворяет множество актуальных бизнес-запросов.
- Но этот мощный инструмент имеет и свою цену: высокую кривую обучения и достаточно сложную первоначальную настройку. Однако после создания такой структуры преимущества очевидны, поскольку данные передаются с меньшей задержкой и большей пропускной способностью.
- Тщательная документация позволит внедрять новые сервисы быстрее и качественнее

SolarLab>_

Вопросы?

Вопросы?

SolarLab>_

Ссылки

Ссылки

1. What are microservices? https://microservices.io/
2. Книга Фаулера Microservices: https://martinfowler.com/articles/microservices.html
3. Жизненный опыт https://t.me/AlekseyAndreev1984
4. Kafka compact https://docs.arenadata.io/ru/ADStreaming/current/concept/architecture/kafka/log_compaction.html?ysclid=mcynd4bfc033426985
5. Книга Вульф Бобби, Хоп Грегор «Шаблоны интеграции корпоративных приложений» https://www.ozon.ru/product/shablony-integratsii-korporativnyh-prilozheniy-vulf-bobbi-hop-gregor-elektronnaya-kniga-946613685/
6. Статьи на хабре про Inbox + Outbox https://habr.com/ru/articles/862066/ https://habr.com/ru/companies/lamoda/articles/678932/

SolarLab>_

Спасибо за внимание