Распил монолита в Леруа Мерлен

Юркин Павел

Java-разработчик в Леруа Мерлен

@pavel_yurkin

https://vk.com/night_wish

woodythegreat@yandex.ru

www.linkedin.com/in/pavel-yurkin

План доклада

История

API

Распил

Выбор БД

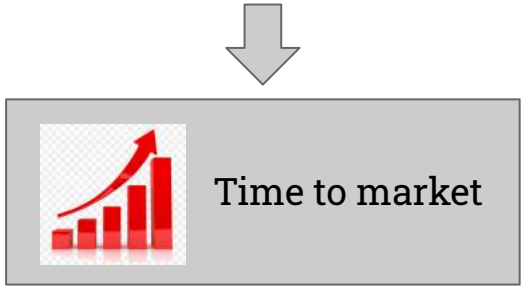
Сложные запросы

История IT в Леруа Мерлен

- Леруа Мерлен была создана во Франции в 1923 году
- В 1989 году был открыт первый магазин в Испании
- В 2004 - в России
- Магазины Леруа Мерлен находятся в 12 странах

IT в международной компании

- IT продукты для все бизнес-юнитов писала одна Франция
- Разработка не кастомная: куча ветвлений в бизнес-логике
- Монолитая архитектура

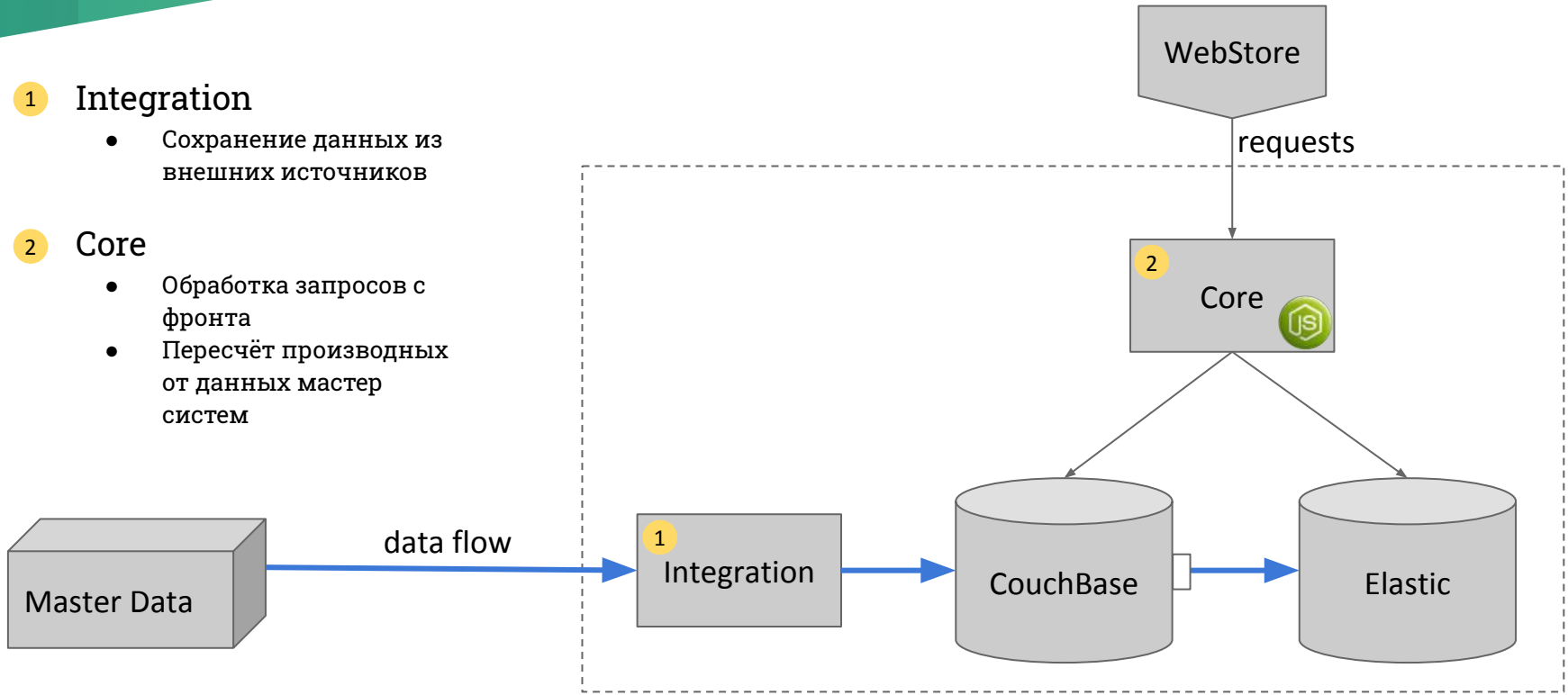


Хватит это терпеть!

Архитектура старой системы

Архитектура

1. Integration
   - Сохранение данных из внешних источников
2. Core
   - Обработка запросов с фронта
   - Пересчёт производных от данных мастер систем



Данные

```
“data”: [ {
        "id": "10035004",
        "type": "product",
        "publication": {
            "attributes": {
                "standardStoreSellingPrice@Price/contexts/store/73_dec": [
                  118
                ], ...
    }, {
        "id": "00000307",
        "type": "attribute",
        “name”: “standardStoreSellingPrice@Price”
        "displayName": “Цена продукта”,
         "translations": {
            "en": "Product price",
            "es": "Precio del producto",
    }, …. ]
```

Возможные типы:

"attribute",
"attributeType",
"content",
"contextAxis",
"group", "model",
"internal",
"modelMask",
"modelType",
"modelTypeMask",
"permission",
"repository", "source",
...

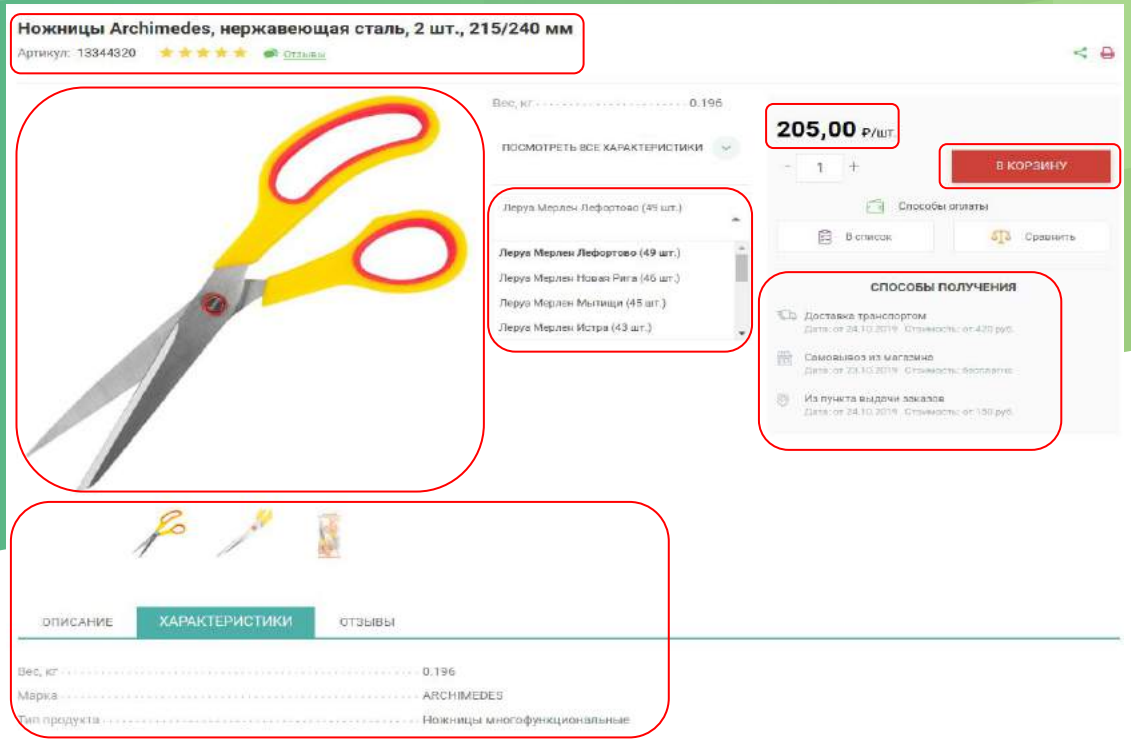
Итоги

- Быстрая разработка на старте
- Легко внедрять новые типы данных
- Подходит для небольших систем с небольшой вложенностью

- Долгое время ответа ответа при большой вложенности
- Проблемы с производительностью
- Проблемы с устойчивостью
- Маленькая скорость внедрения новых фич из-за большой связанности

Распил монолита

Publication platform



Проблема монолита

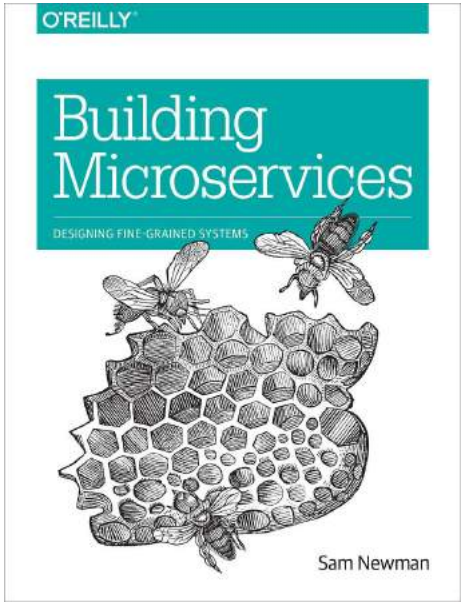
- Производительность
- Стабильность
- Time to market
- ...

Нужно что-то делать...

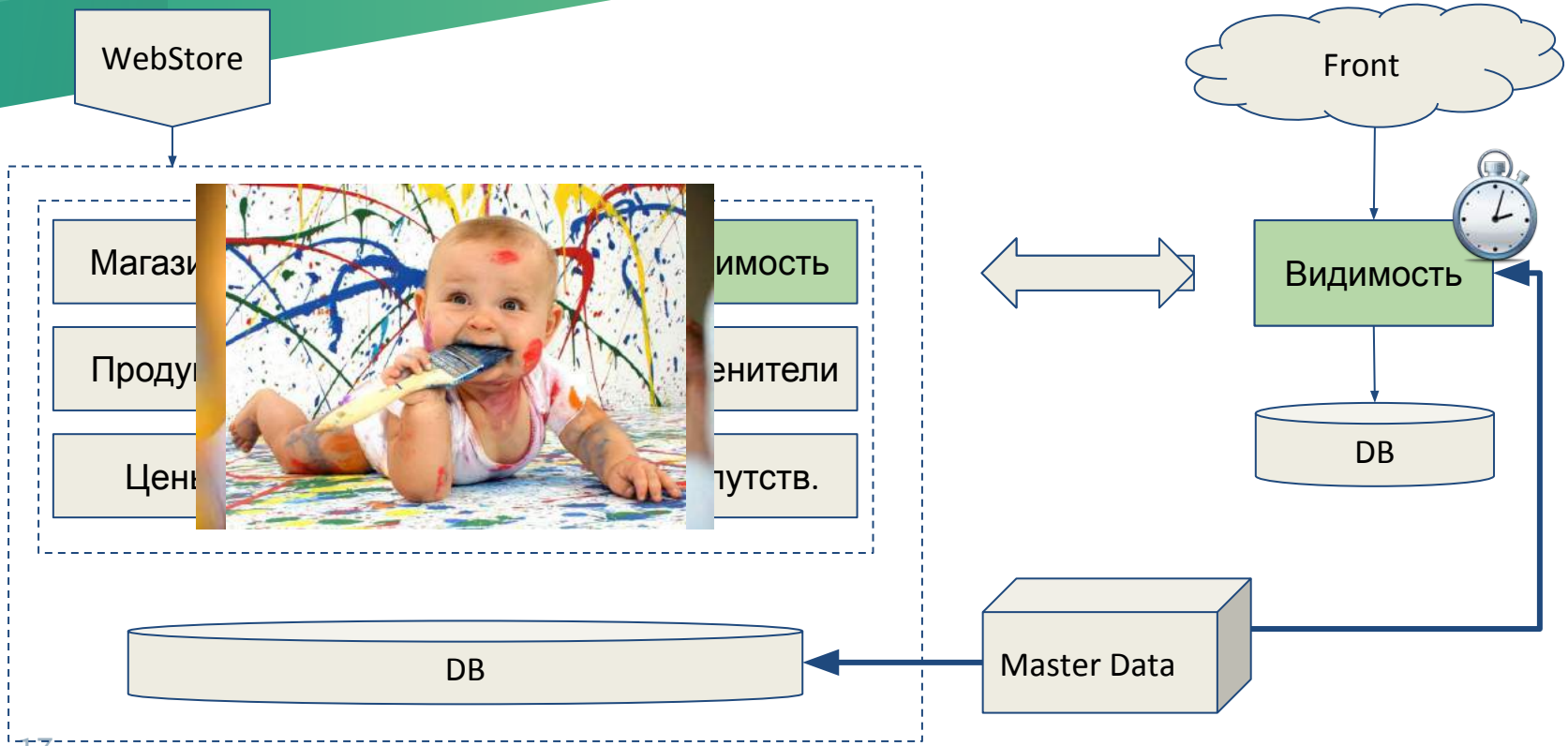
Выносим функционал

Как разбить на микросервисы?

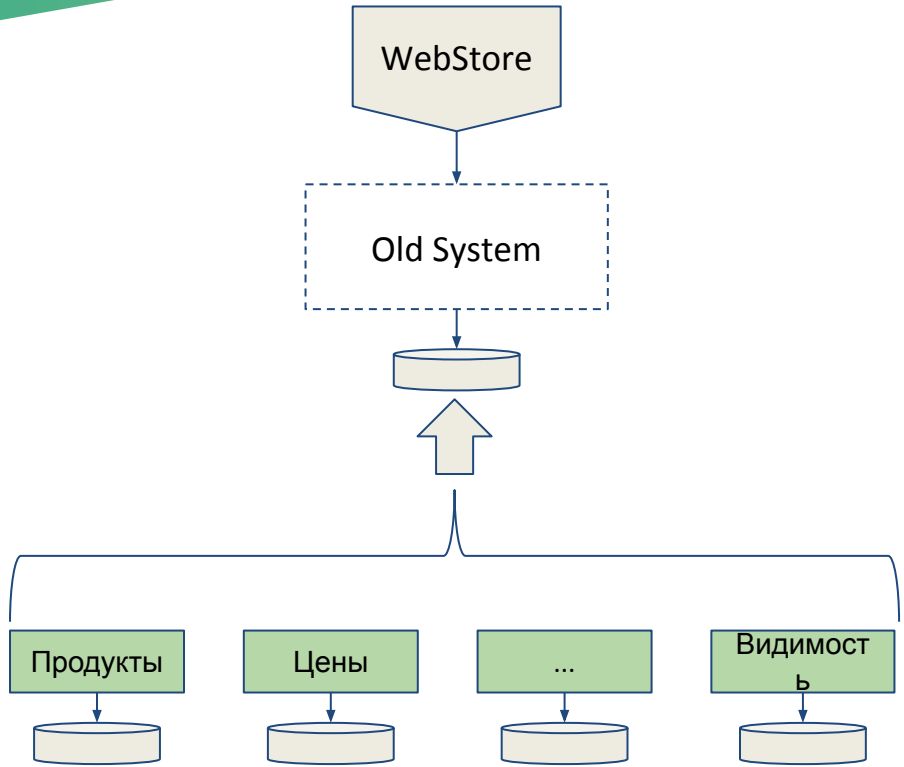
- Бизнес функционал
- Сложная бизнес-логика
- Есть контекст (различаются ли значения в зависимости от



Выносим функционал



Выносим функционал

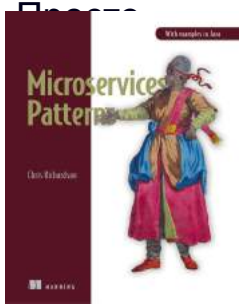


Каким образом выставить своё api?

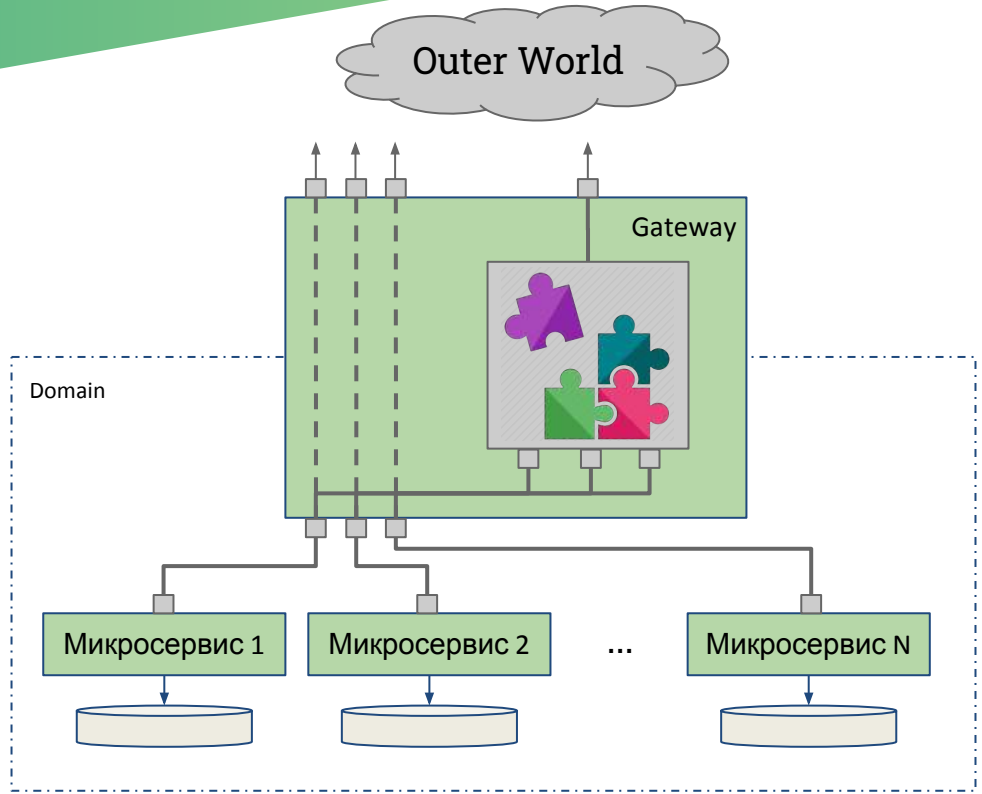
Вариант 1: Оркестрация на Gateway

- Единая точка входа
- Просто

- Чем больше клиентов и микросервисов - тем больше логики
- Связанность



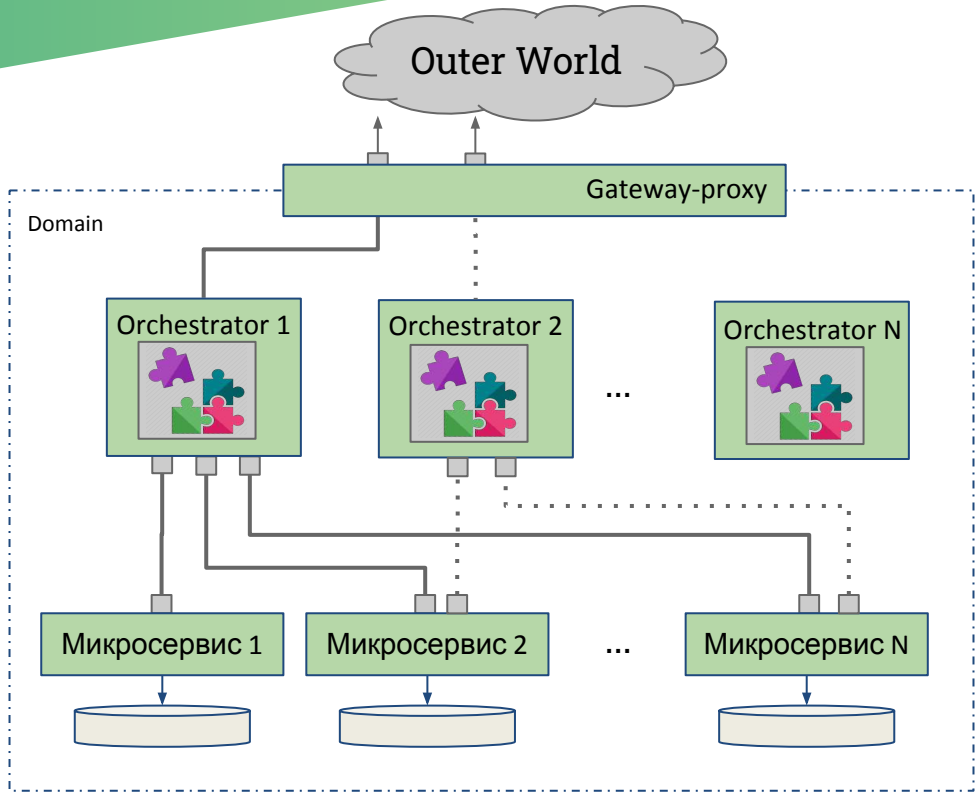
Chris Richardson



Вариант 2: Оркестратор + Gateway-proxy

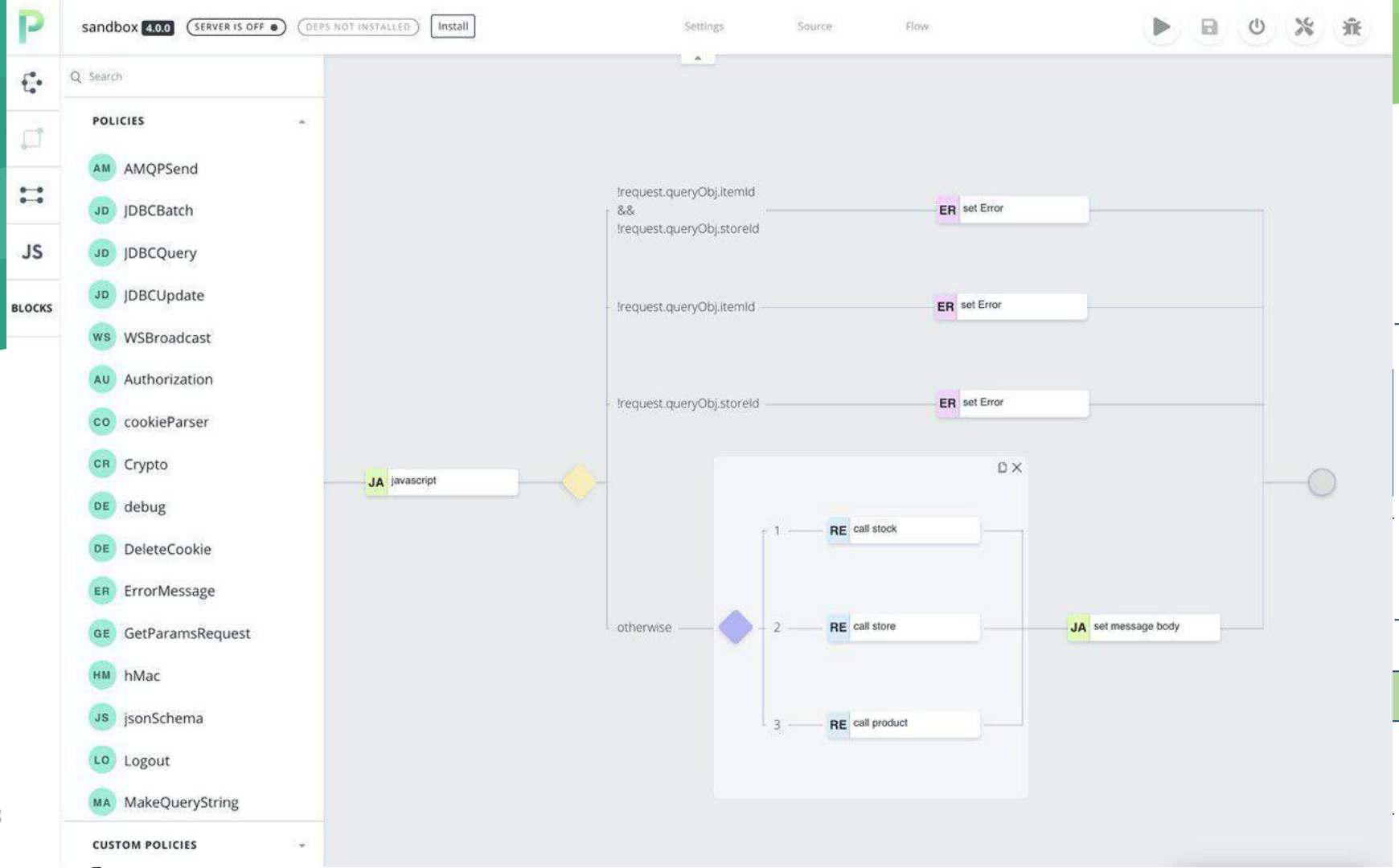
- Независимость
- В качестве Gateway можно использовать готовое решение

- Лишний вызов
- Больше количество сущностей



Проблемы 1-го и 2-го подходов

- Всеми доработками Gateway’ев / оркестраторов занимается доменная команда
- Снижается скорость появления новый фич
- Даже небольшие доработки фронта могут занять значительное время


sandbox 4.0.0
SERVER IS OFF
DEPS NOT INSTALLED
Install
Settings
Source
Flow
Search
POLICIES
AM AMQPSend
JD JDBCBatch
JD JDBCQuery
JD JDBCUpdate
WS WSBroadcast
AU Authorization
CO cookieParser
CR Crypto
DE debug
DE DeleteCookie
ER ErrorMessage
GE GetParamsRequest
HM hMac
JS jsonSchema
LO Logout
MA MakeQueryString
CUSTOM POLICIES
JS
BLOCKS
!request.queryObj.itemId && !request.queryObj.storeId
ER set Error
!request.queryObj.itemId
ER set Error
!request.queryObj.storeId
ER set Error
JA javascript
otherwise
1
RE call stock
2
RE call store
3
RE call product
JA set message body

Публикация api. Итоги.

- Оркестрация на gateway

- Единая точка входа
- Просто

- Чем больше клиентов и микросервисов - тем больше логики
- Связанность

- Оркестратор + gateway-proxy

- Независимость
- В качестве Gateway можно использовать готовое решение

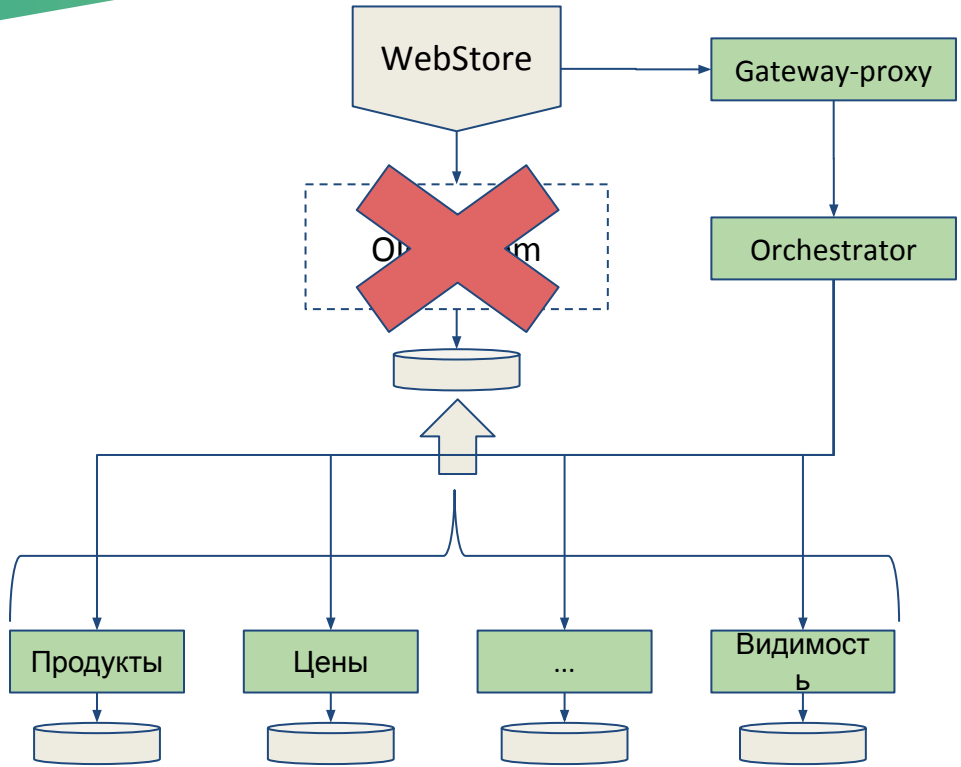
- Лишний вызов
- Возможна дубляция кода

- Api Manager

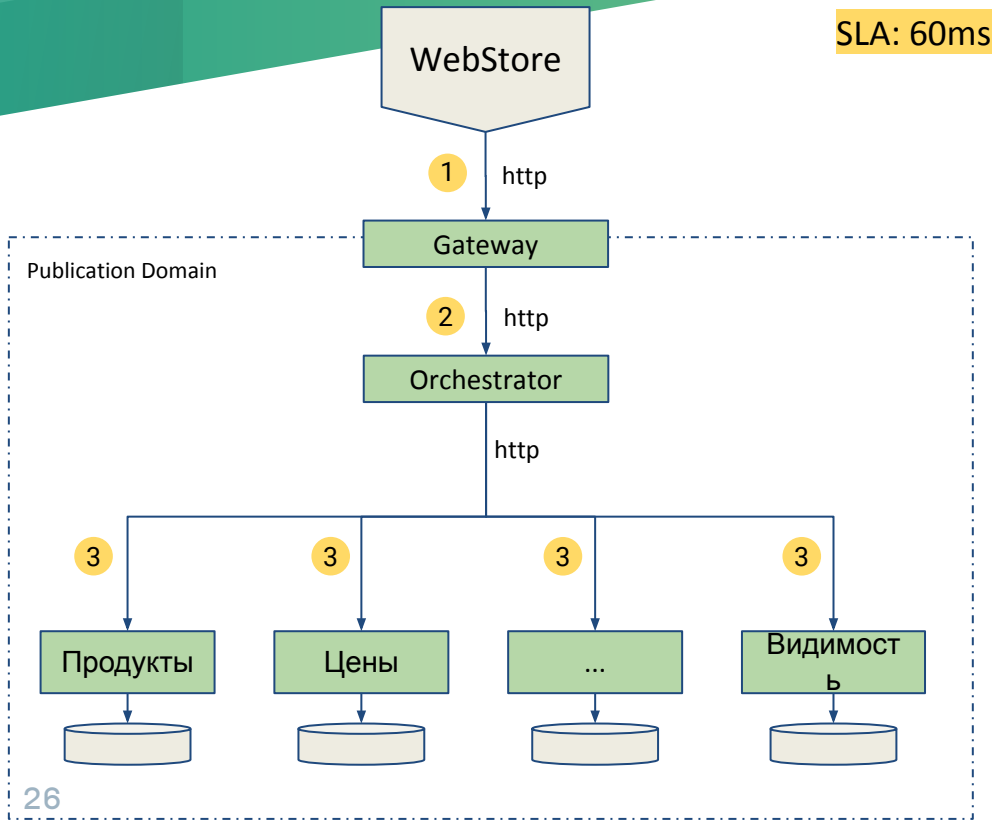
- Независимость от доменной команды
- Увеличение скорости разработки Gateway
- Доменная команда фокусируется на функционале

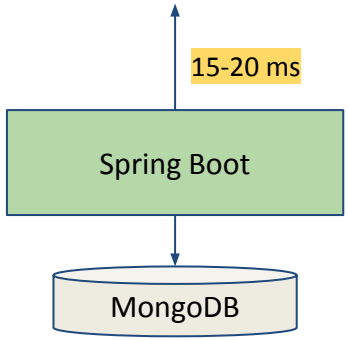
- Лишний вызов
- Сложная система

Выносим функционал



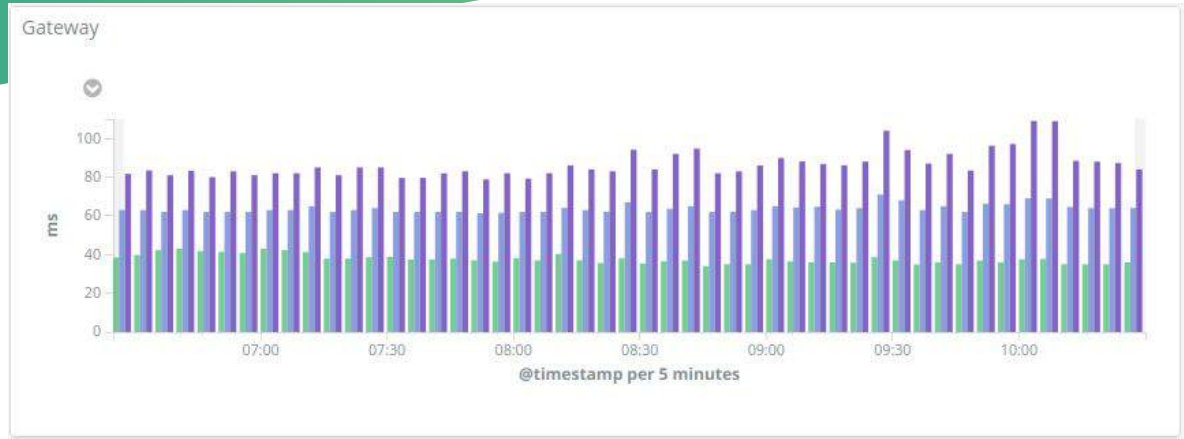
Производительность

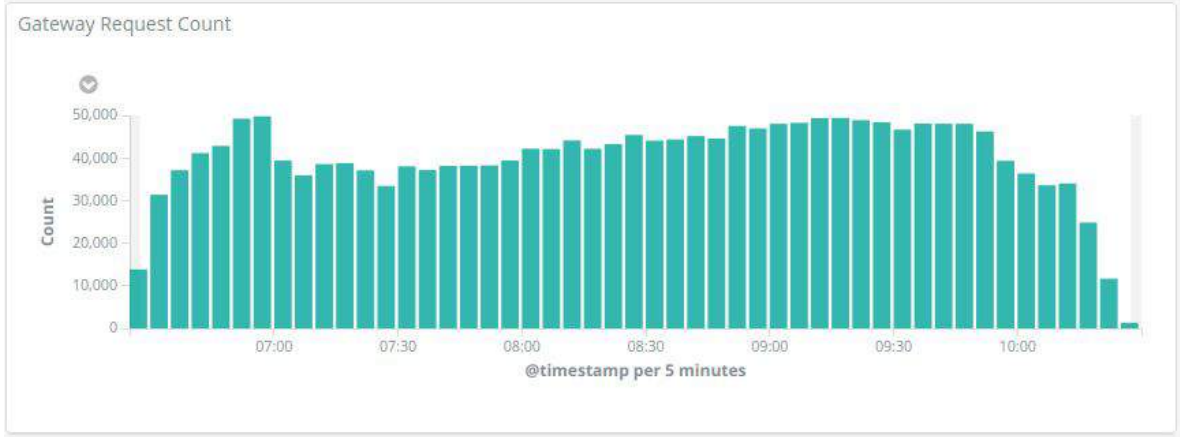


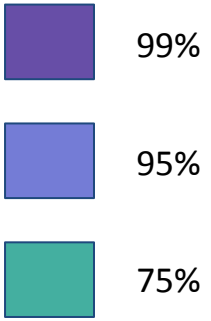



```
{
   "_id" : ObjectId("..."),
   "productId" : "10008671",
   "regions" : [
      {
         "regionId" : "1437",
         "visibility" : true
      }, ....
}
```

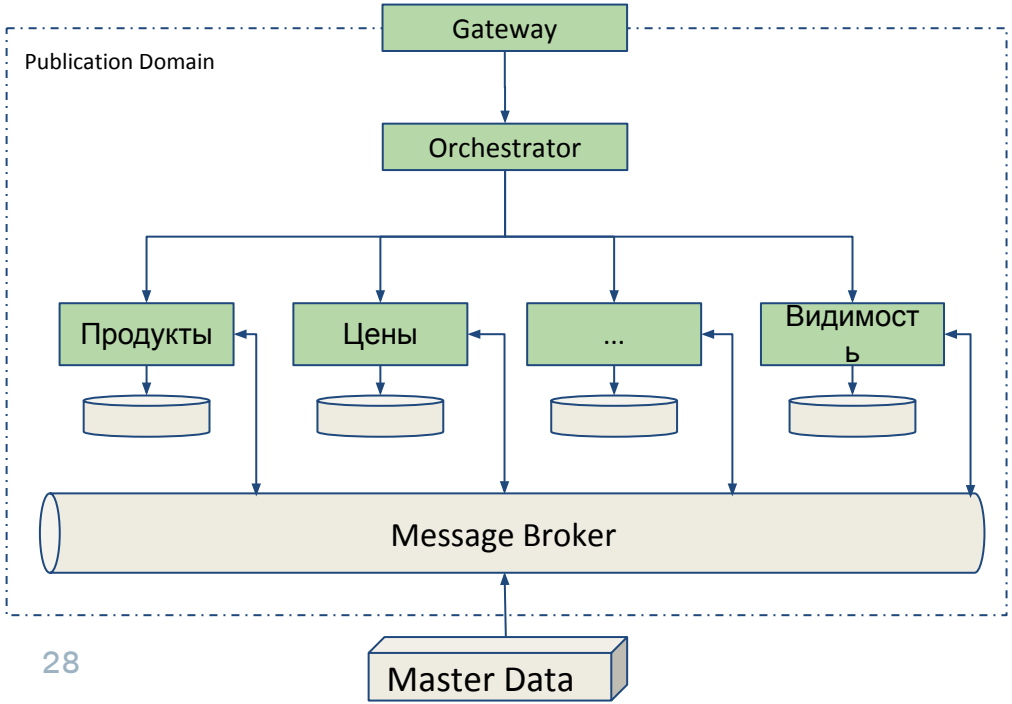
Производительность







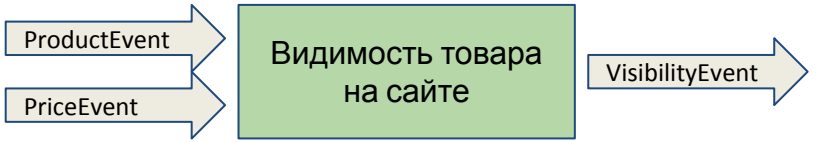
Потоки данных



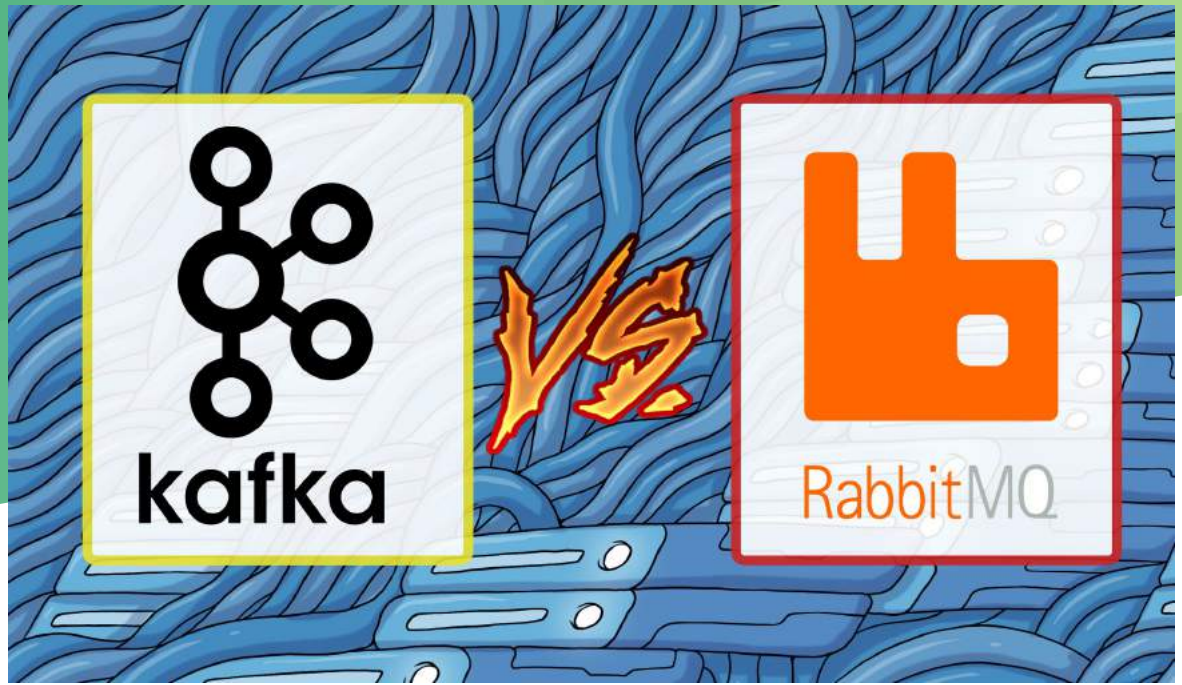
Бизнес-логика: Товар **видим** если у него есть:

- ➔ название (атрибут продукта)
- ➔ цена

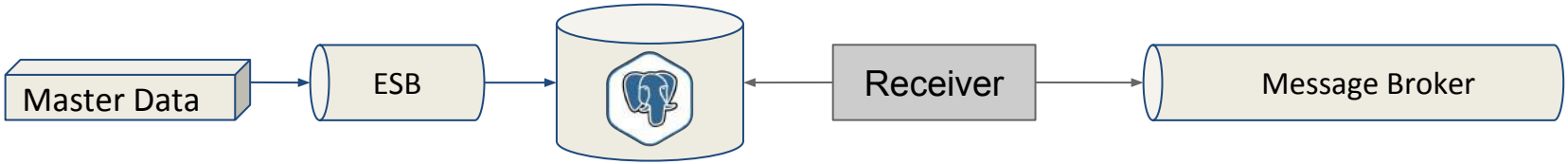
Используем Event Driven подход



Message broker



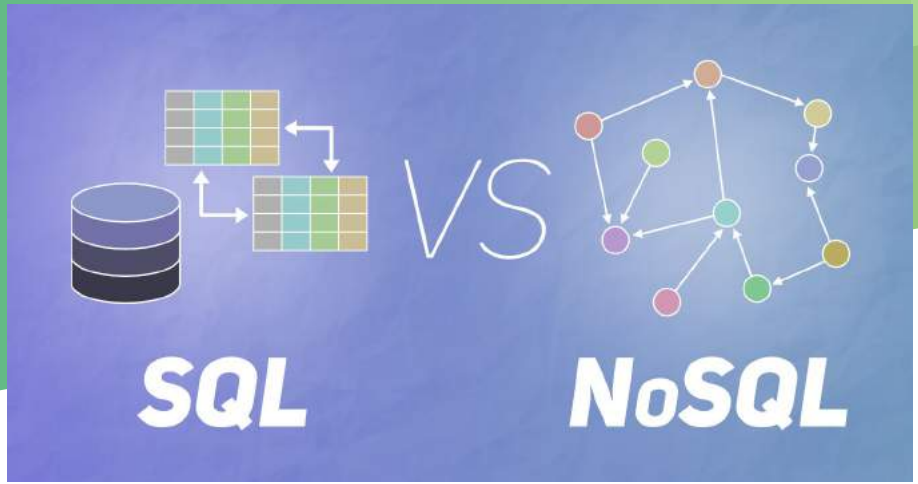
RabbitMQ vs Kafka



- Нет необходимости сохранять сообщения
- Есть необходимость в гибком роутинге

RabbitMQ

Выбор БД



Выбор БД

- getByProductId(...)
- getByFamilyId(...)

Horizontal scaling

Sharding

mongoDB®

https://www.integrant.com/when-to-use-sql-vs-nosql

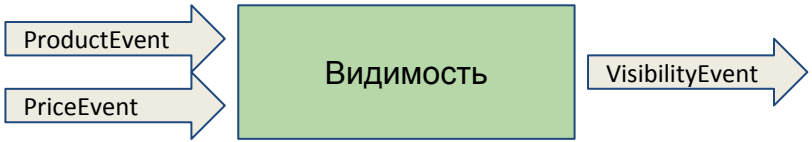
Workflow

1. Пришёл Product name
2. Считать Price
3. Рассчитать новый Visibility
4. Считать старый Visibility
5. Записать новый Visibility
6. Записать Product name

Бизнес-логика: Товар **видим** если у него есть:

➔ название (атрибут продукта)

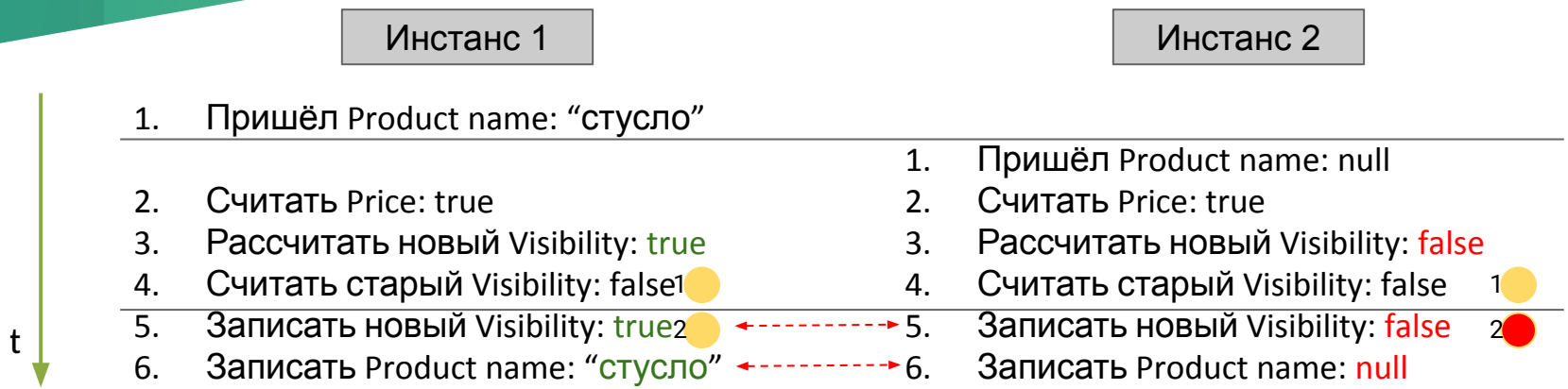
➔ цена



Создаём 3 коллекции:

- Product
- Price
- Visibility

Многопоточность



Optimistic locking:

```
@Version
private Long version;
```

Многопоточность

Инстанс 1	Инстанс 2
1. Пришёл Product name: “стусло”	
2. Считать Price: true	1. Пришёл Product name: null 2. Считать Price: true 3. Рассчитать новый Visibility: false 1 4. Считать старый Visibility: false 2 5. Записать новый Visibility: false
3. Рассчитать новый Visibility: true 4. Считать старый Visibility: false 5. Записать новый Visibility: true 3 6. Записать Product name: “стусло”	6. Записать Product name: null

t

No “select for update”

Что же делать?

- Реализовывать самому с mongo

см https://github.com/lukas-krecan/ShedLock

- Использовать Redis

см https://redis.io/topics/distlock

- Схлопнуть в одну коллекцию!

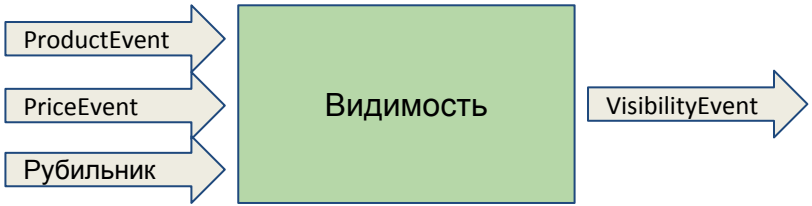
Почему вы никогда не должны использовать MongoDB
https://habr.com/ru/post/231213/

```
Visibility: [{
        productId: …
        visibility: …
        lastProductEvent: …
        lastPriceEvent: …
        ....
        Version: 1
}]
```

А если нельзя?

Бизнес-логика: Товар **видим** если у него есть:

- ➔ название (атрибут продукта)
- ➔ цена
- ➔ **Рубильник включен**



Согласованность данных

Дублирование данных не полное

Продукт

```
{
  “productId”: “1”,
  “attrs”: [
    { “attr”: “name”, “value”: “Стусло” },
    { “attr”: “rate”, “value”: 5 },
    ...
  ]
}
```

Видимость

```
{
  “productId”: “1”,
  “hasName”: true,
  ...
}
```

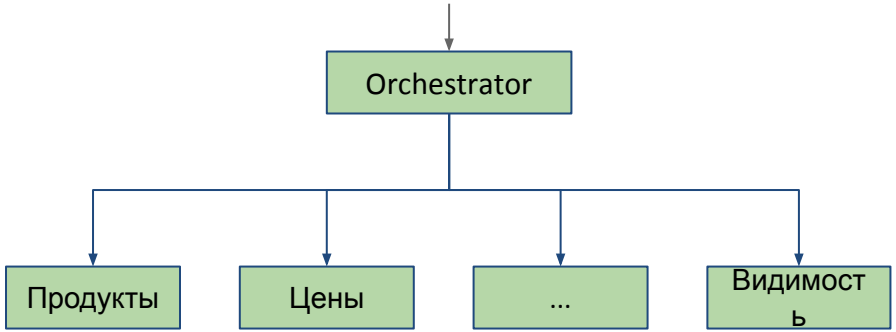
Реинит

У каждого микросервиса есть механизм реинита

Сложные запросы

Сложные запросы

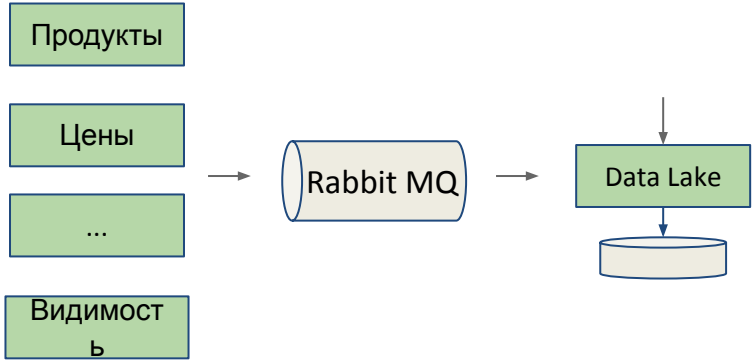
Агрегация через прокси



- Up to date info
- Универсальн

- Будет грузить систему
- Долго

Специфичный для определенного вида запросов сервис



- Быстро
- No impact

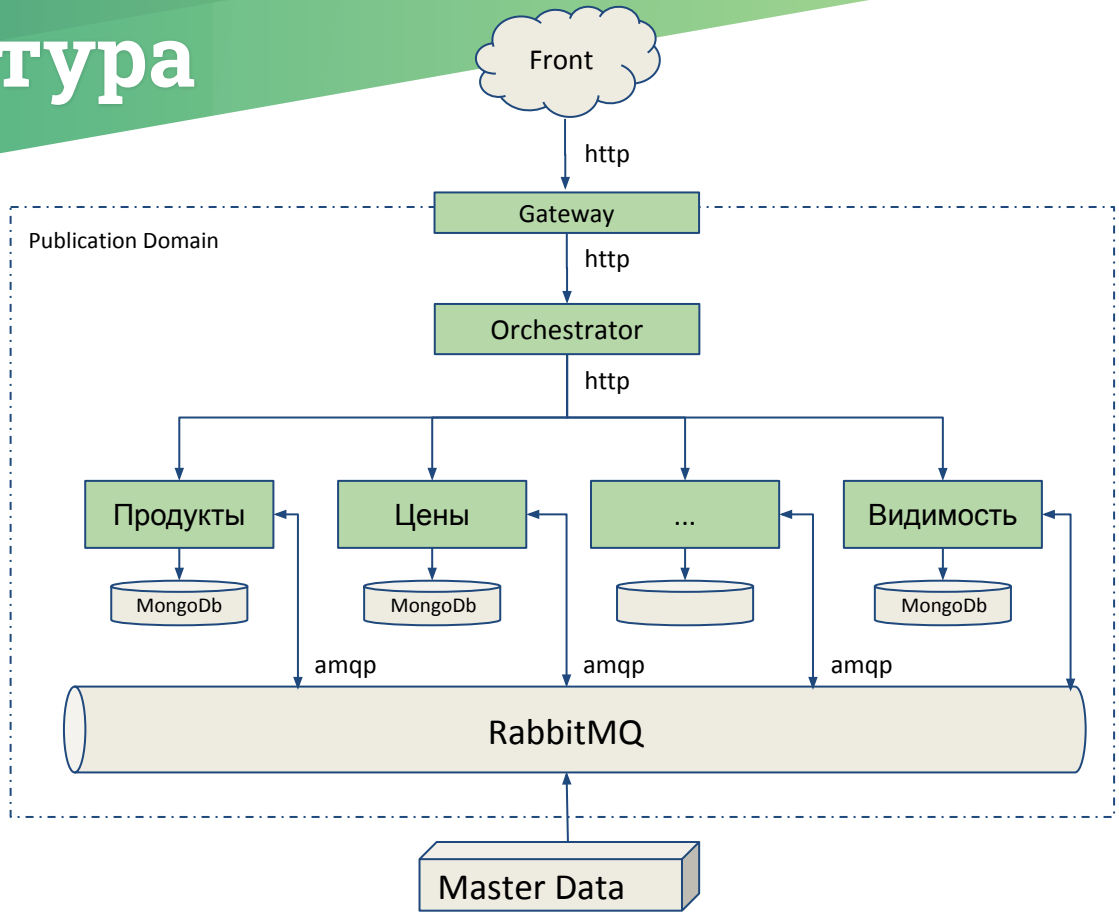
- Не универсально
- Консистентность

Итоги

Итоги: архитектура

Что у нас получилось?

- Распределённая система с производительностью 300 rps и временем ответа в 60 ms (95 персентиль), **без тюнинга**.
- Отказоустойчивость
- Гибкость
- ...



Итоги

1. Публикация api:
   - Оркестрация на gateway
   - Оркестратор + gateway-proxy
   - Api Manager
2. Аккуратнее с многопоточностью в MongoDb
3. Сложные запросы можно сделать с помощью:
   - Агрегации
   - Специального сервиса

Спасибо за внимание!

@pavel_yurkin

https://vk.com/night_wish

woodythegreat@yandex.ru

www.linkedin.com/in/pavel-yurkin