Avito

Инструкция по отпилу, или как мы сервис сессий из монолита выносили

Павел Лакосников

Для чего Авито микросервисы

Монолит

- Долго и трудно тестировать
- Сложный и большой поток изменений
- Долго деплоить и откатывать
- Высокая связность кода и большие издержки времени при разработке

Микросервис

+ Легче протестировать
+ Малое количество единовременных изменений
+ Быстро деплоить и откатывать
+ Меньшая связность

Сервис сессий

- Генерация sessionID - уникального ID посетителя
- Привязка sessionID к UserId
- Хранение данных, связанных непосредственно с сессией

- 528 млн ключей в Redis кластере
- 200 гигабайт данных
- 400к RPS операций с данными

Исследование бизнес-области

- Кто потребитель?
- Какие задачи и сценарии решает?
- Какие внешние зависимости?
- Какие NFR\SLA

Кто потребитель?

Кто использует сервис сессий

- Мессенджер
- Сервис гейтвея
- Сервис объявлений
- Сервис профилей

Задачи сервиса

Сервис сессий до исследования

```
type SessionOld interface {
    ReadOrCreate(cookie string)
    Prolongate(cookie string)
    SetData(key string, val interface{}, ttl int64)
    GetData(key string) interface{}
}
```

Сервис сессий после исследования

```
type SessionApi interface {
    Create() Session
    Delete(sessionID string) bool
    Get(sessionID string) Session
    Prolongate(sessionID string) Session

    SetData(sessionID string, key string, val string) bool
    GetData(sessionID string, key string) string
    DelData(sessionID string, key string) bool

    SessionLogin(sessionID string, userId int64) string
    SessionLogout(sessionID string) bool
    SessionRefresh(sessionID string, refreshToken string)
}
```

Сервис сессий после исследования

```
type SessionApi interface {
    Create() Session
    Delete(sessionID string) bool
    Get(sessionID string) Session
    Prolongate(sessionID string) Session

    SetData(sessionID string, key string, val string) bool
    GetData(sessionID string, key string) string
    DelData(sessionID string, key string) bool

    SessionLogin(sessionID string, userId int64) string
    SessionLogout(sessionID string) bool
    SessionRefresh(sessionID string, refreshToken string)
}
```

Сервис сессий после исследования

```
type SessionApi interface {
    Create() Session
    Delete(sessionID string) bool
    Get(sessionID string) Session
    Prolongate(sessionID string) Session

    SetData(sessionID string, key string, val string) bool
    GetData(sessionID string, key string) string
    DelData(sessionID string, key string) bool

    SessionLogin(sessionID string, userId int64) string
    SessionLogout(sessionID string) bool
    SessionRefresh(sessionID string, refreshToken string)
}
```

Внешние зависимости

- Базы данных, хранящие наши сущности
- Список других сервисов, в которые делаются запросы

Зависимости сервиса сессий

1. **Redis**
   - Хранение основных данных
2. **Postgress**
   - Хранение refresh-tokens
   - Хранение истории
   - Несколько тригеров, хранимых процедур, вложенных подзапросов, вызываемых из других процедур базы
3. **Другие сервисы**
   - Сервис пользователей
   - Шина данных

NFR\SLA

Response time			Error rate
P75	P99	P99.9	
1мс	10мс	25мс	99.99

Вендоризация

- go dep
- composer
- pip
- luarocks
- npm

Зачем?

- Контроль за изменениями
- Изоляции всех внешних зависимостей
- Проверка удобства API
- Первичное исследование предполагаемой нагрузки

Зачем?

- Контроль за изменениями
- Изоляции всех внешних зависимостей
- Проверка удобства API
- Первичное исследование предполагаемой нагрузки

Зачем?

- Контроль за изменениями
- Изоляции всех внешних зависимостей
- Проверка удобства API
- Первичное исследование предполагаемой нагрузки

Зачем?

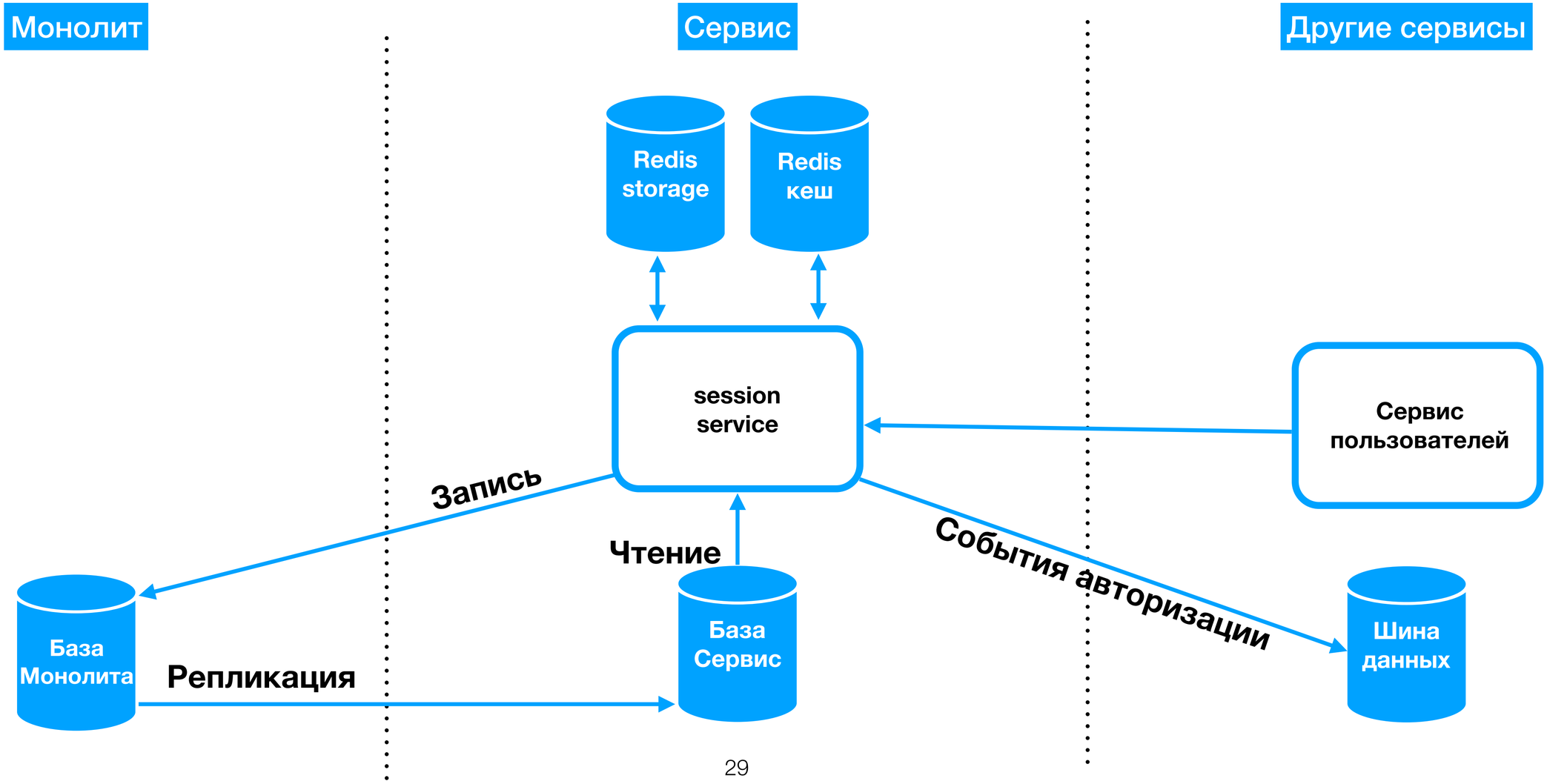
- Контроль за изменениями
- Изоляции всех внешних зависимостей
- Проверка удобства API
- Первичное исследование предполагаемой нагрузки

Вынос функциональности кода в микросервис

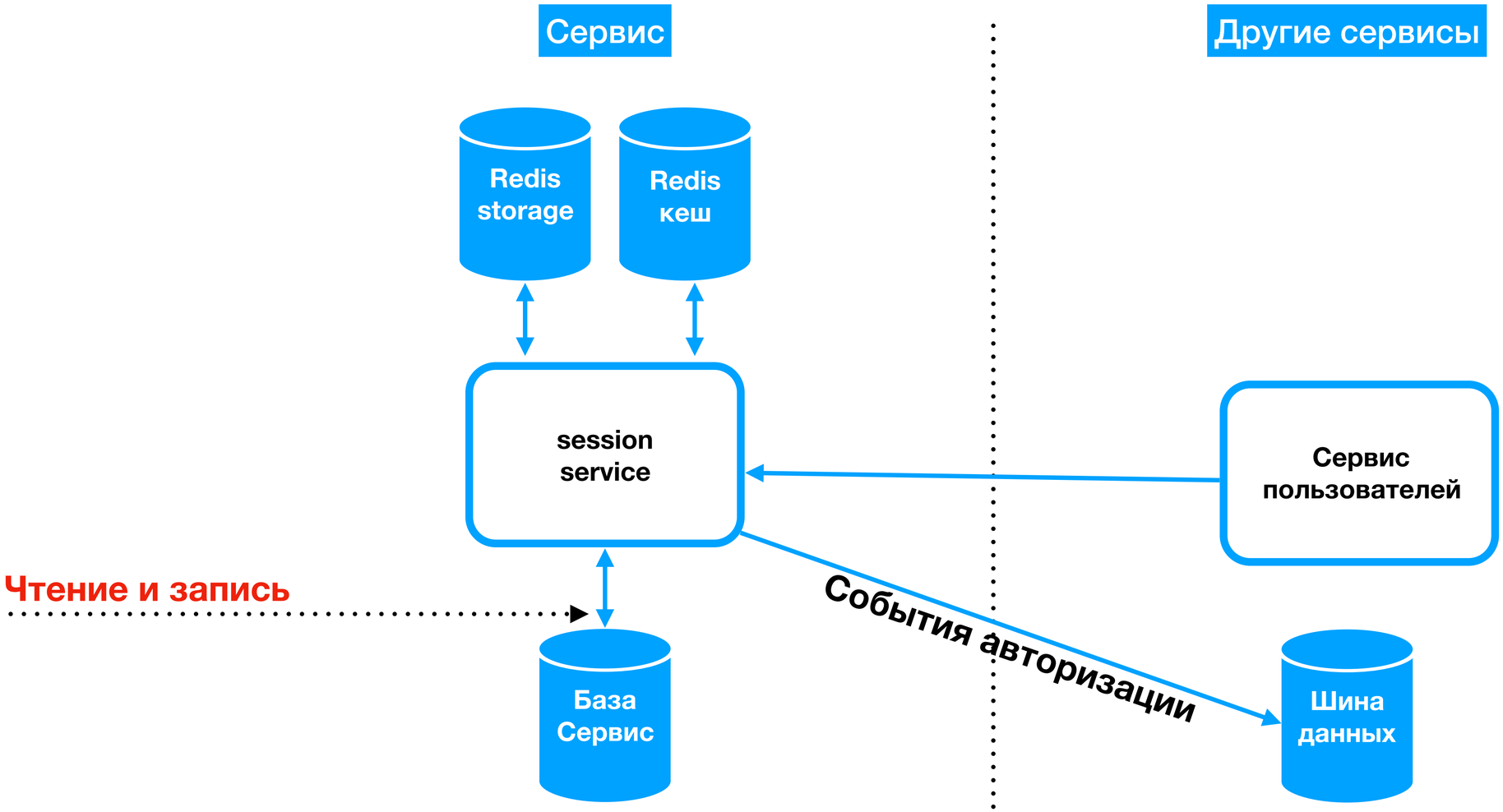
- Создание отдельного хранилища для сервиса
- Перенос данных из монолита в микросервис
- Промежуточный этап с работой на 2 базы
- Окончательный вынос функционала в микросервис

Промежуточный этап

Промежуточная схема сервиса сессий



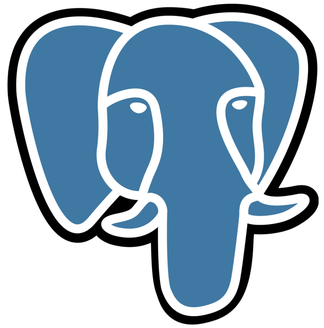
Итоговая схема сервиса сессий



Работа с базой данных
Как ее отпилить?

Трудности

- Структура хранилища не готова к выносу
- Тигеры, хранимые процедуры базы, view, подзапросы и join’ы
- Сильно денормализованный способ хранения данных сервиса
- Постоянная REALTIME нагрузка
- Сложность в переключении



Трюки

- Работа над инкапсуляцией данных сервиса в одном месте
- Отдельный пользователь с доступом только в нужные таблицы
- Отдельная база данных для сервиса
- Репликация из **Main** базы в Базу **Сервиса**

Метрики

- Memory
- CPU
- Тротлинг
- Рестарты реплик

Метрики

- RPS в разрезе методов и кодов
- Server ResponseTime
- Client ResponceTime
- Состояние пулов коннектов

Даём нагрузку на сервис

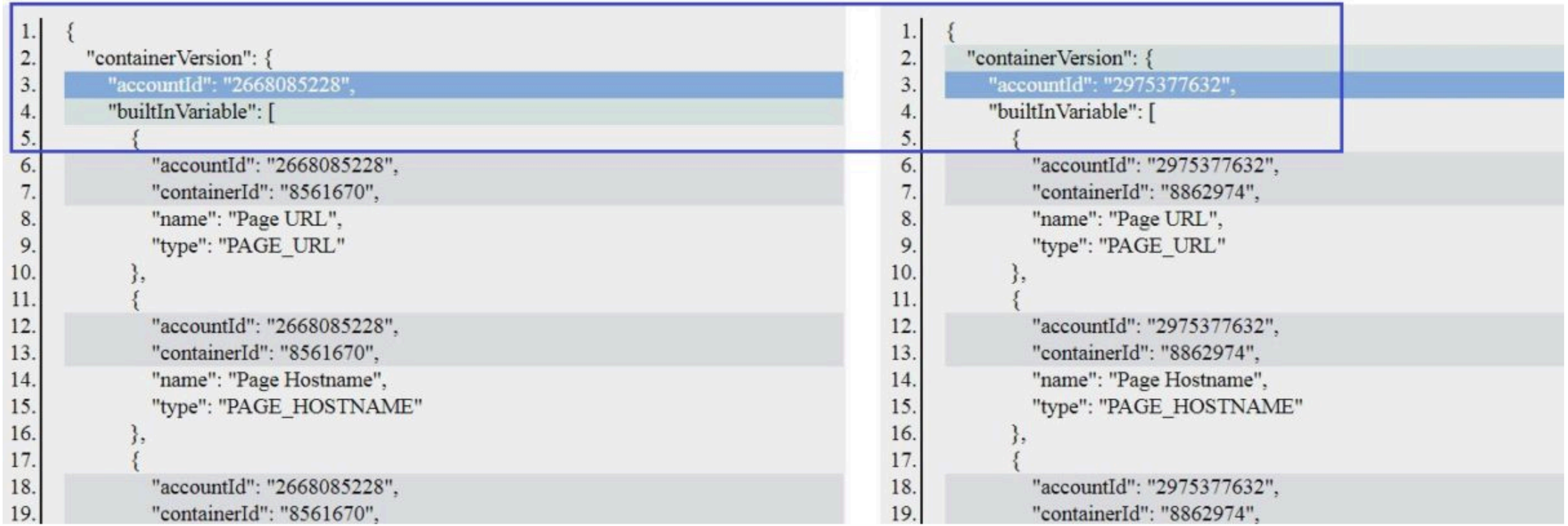
Переключение трафика

```
func isServiceEnabled(toggleState int) bool {
    if toggleState == 0 {
        return false
    }
    if toggleState ⩾ 100 {
        return true
    }

    r1 := rand.New(rand.NewSource(time.Now().UnixNano()))
    return r1.Intn( n: 100) ⩾ (100 - toggleState)
}
```


RHEOSTAT
AMPS.
3·3
OHMS.
24

Сравнение ответов



```
{
  "containerVersion": {
    "accountId": "2668085228",
    "builtInVariable": [
      {
        "accountId": "2668085228",
        "containerId": "8561670",
        "name": "Page URL",
        "type": "PAGE_URL"
      },
      {
        "accountId": "2668085228",
        "containerId": "8561670",
        "name": "Page Hostname",
        "type": "PAGE_HOSTNAME"
      },
      {
        "accountId": "2668085228",
        "containerId": "8561670",
```

```
{
  "containerVersion": {
    "accountId": "2975377632",
    "builtInVariable": [
      {
        "accountId": "2975377632",
        "containerId": "8862974",
        "name": "Page URL",
        "type": "PAGE_URL"
      },
      {
        "accountId": "2975377632",
        "containerId": "8862974",
        "name": "Page Hostname",
        "type": "PAGE_HOSTNAME"
      },
      {
        "accountId": "2975377632",
        "containerId": "8862974",
```

Что в итоге получилось

Что хорошо

- Покрытие тестами подняли до 90%
- Оптимизировали сериализацию данных
- Ускорили всех потребителей
- Получили возможность масштабироваться независимо от монолита

Что плохо

- Сервис-мейкер теперь немножко девопс
- Неоднократно в процессе выноса сервис падал
- Бесшовная выкатка это трудно

Tips & Tricks

Не забывать про то, что сервис на GO

- Memory-кеш
- Удобные Goroutines
- Асинхронность
- Пулы коннектов

Сайд-эффекты кеша

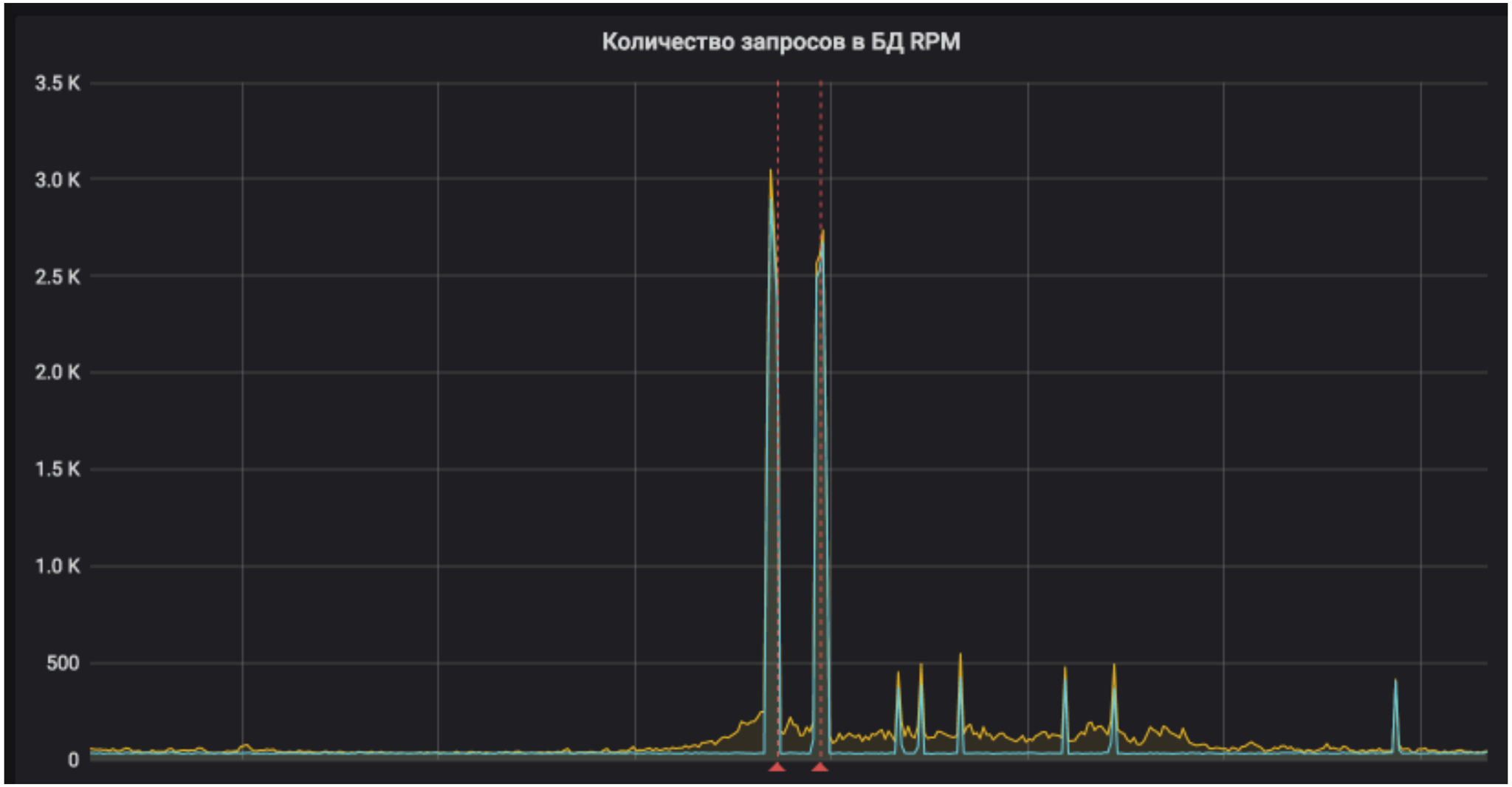
Вот у нас он появился

- Вы добавили в сервис кеш
- Ускоряется ответ
- Разгружается база
- Рестартуем сервис, штук на 10 реплик

Вот у нас он появился

- Вы добавили в сервис кеш
- Ускоряется ответ
- Разгружается база
- Рестартуем сервис реплик

База легла под нагрузкой

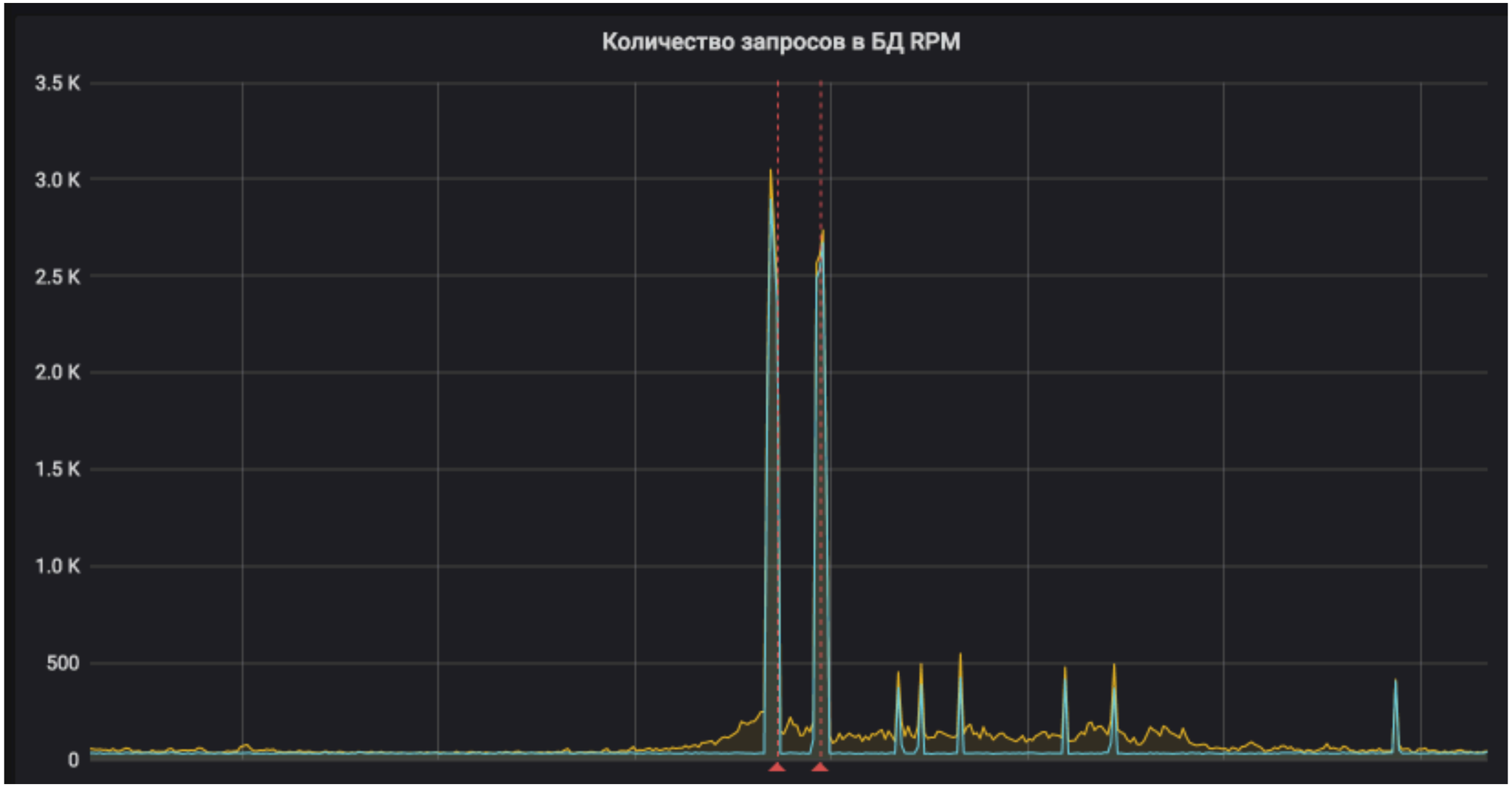


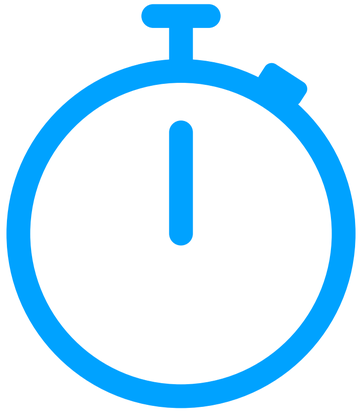
Добавляем прогрев кеша на старте

Добавляем прогрев кеша на старте

Некоторые догадываются, что будет дальше

База легла под нагрузкой





Добавляем рандомную задержку для каждой реплики перед началом прогрева кеша

Вот теперь получилось!

Вот теперь получилось!

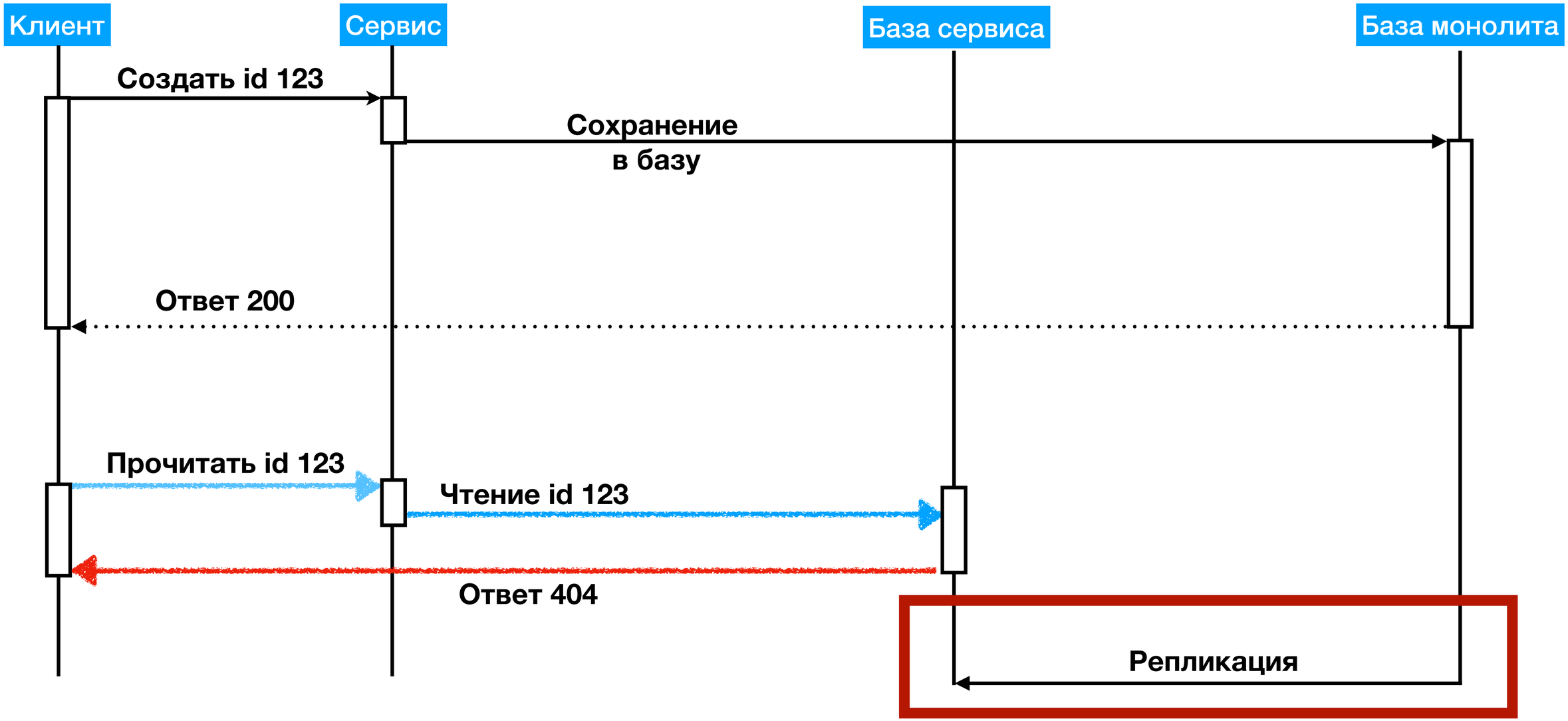
А что ещё может кеш?

Кеш на промежуточном этапе распила

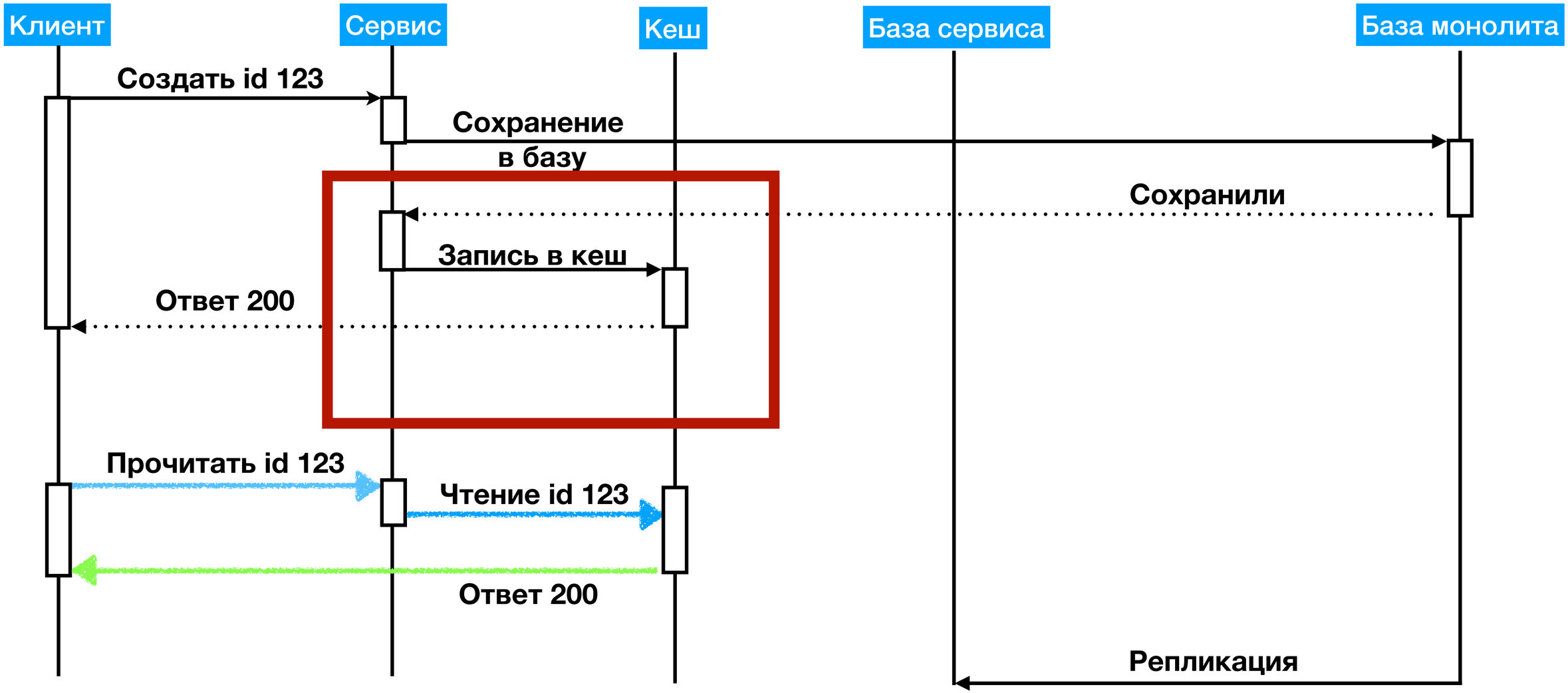
- Ускорение ответа
- Снижение нагрузки на базу
- …

Консистентность!

Запрос на запись



Запрос на запись



Хранение индексов в памяти

Схема распила сервиса

- Исследование бизнес-области
- Вендоризация
- Написание сервиса
- Развязка внешних зависимостей
- Переключение нагрузки