Feature Store в ОК: Архитектура и эксплуатация высоконагруженном проекте

Андрей Кузнецов
Одноклассники, VK

OK в числах

- 28M MAU
- 100M+ событий фидбека в день
- 30+ TB новых данных в день
- 20+ ML сервисов в проектах

ML сервисы в ОК

Антиспам

Пуши/Оповещения

Группы

Ретеншен сервис

Лента

Видео

Музыка

Дружбы

Молл

Фото

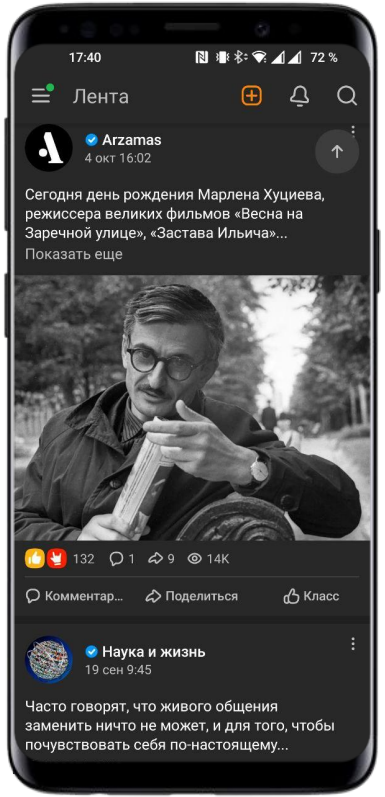
Поиск

Мини-приложения

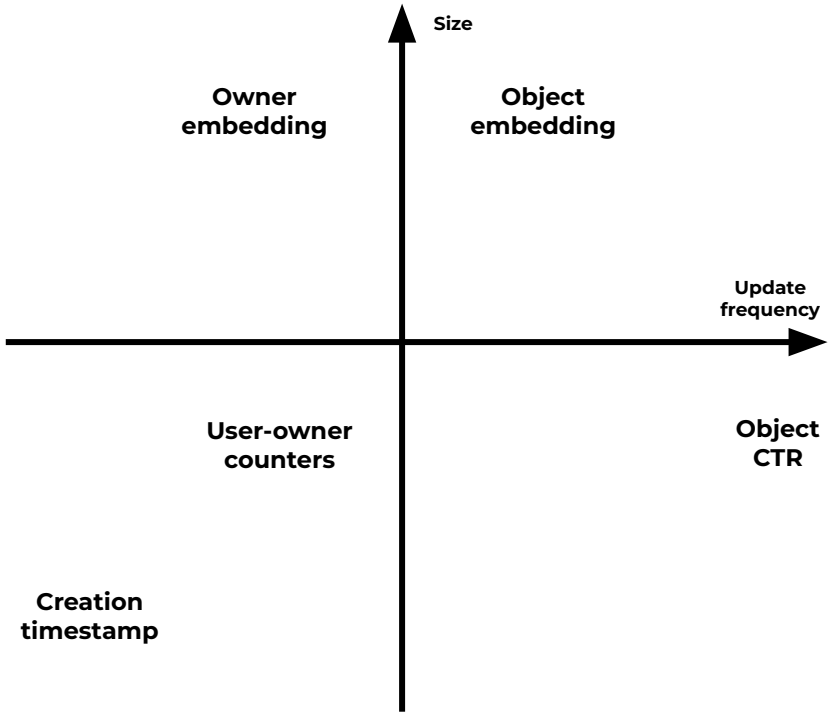
Поддержка пользователей

Самый нагруженный ML сервис в числах

- 500M запросов в день
- 10K RPS нагрузки в пике
- 2K+ объектов на взвешивание
- 100 ms avg клиентское время
- 100+ фичей
- ~15 одновременно запущенных моделей



ML в Ленте. Фичи



Могут принадлежать разным сущностям: юзеру, овнеру, объекту ранжирования

Некоторые могут изменяться очень быстро (показы), а другие остаются статичными

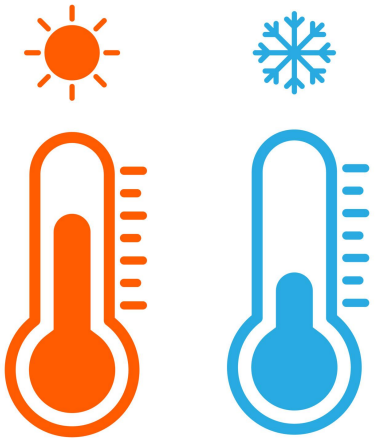
Сильно различаются по размеру

С разной скоростью становятся неактуальными

Сильно различаются по частоте запросов

Feature store в двух словах

- **Хранит и управляет** наборами фичей
- **Доставляет** фичи для тренировки и инференса ML моделей
- Можно разделить на два компонента: **холодный** и **горячий**



Feature store. Холодный компонент

- Нужен чтобы отобрать фичи и отправить их в модель для тренировки
- Распределенный сбор и обучение
- Обычно это обвязка поверх Спарка
- Отдельный метасторадж для хранения / Hive
- Низкие требования по скорости, высокие по объемам.
- Данные в HDFS или S3

Холодная часть - это Big Data сторона feature store

Feature store. Горячий компонент

- Хранит только фичи необходимые для онлайн применения
- Умеет быстро принимать фичи и управлять их состоянием
- Сервит фичи по запросу с высокими требованиями по скорости и доступности

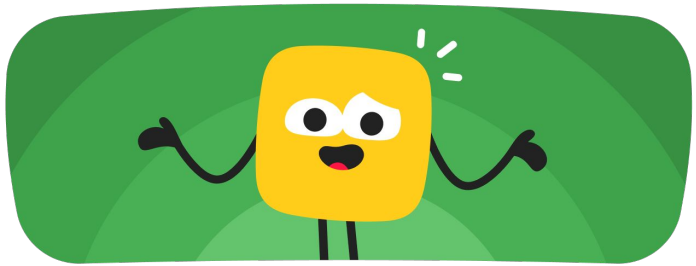
Это **Хайлоад** сторона feature store

Feature store. Горячий компонент. Что есть на рынке?

Feast - Redis (or pluggable modules)
Hopswork - RonDB (MySql cluster)
Featureform - Redis/Cassandra, Python API
Feathr - Redis, Cosmos DB
Tecton - DynamoDB, Snowflake

Butterfree - Framework for Feature Store dev

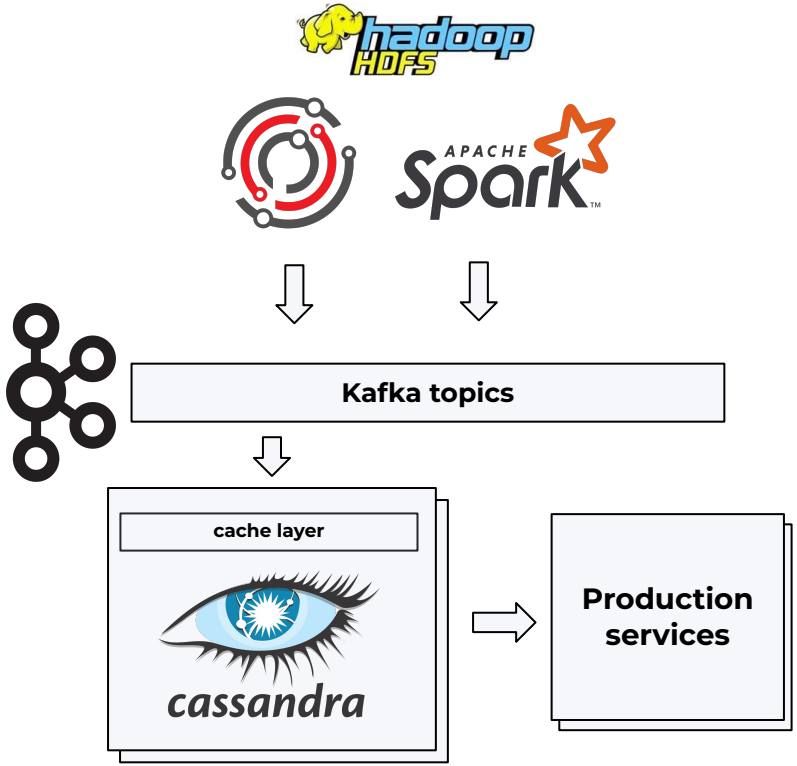
Redis хорош, но есть вопросы



Feature store. Горячий компонент. Требования

- доставка фичей и батч и стриминг
- быстрый сервинг фичей
- гибкие форматы хранения фичей
- большие объемы данных
- распределенность и отказоустойчивость
- эффективная утилизация железа

OK Feature Store



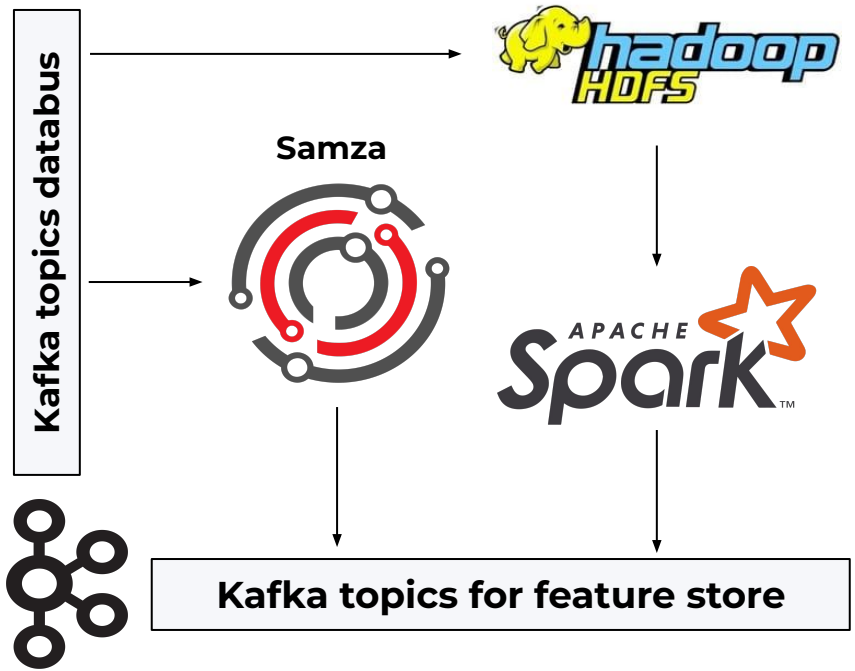
1. Java на сервере и на клиентах
2. Первая версия в проде с 2015
3. Полностью переехали в облако в 2022
4. 15 кластеров на данный момент
5. 500k RPS для 30kk объектов

   1.5-3 ms avg на запрос

   25 TB фичей

Дмитрий Бугайченко — Как посчитать CTR 100M объектов в реальном времени и не умереть

Доставка фичей. Продьюсеры



Стримовые джобы генерят фичи, используя события продакшен систем залогированных в Kafka

Батчевые джобы на Spark генерят тяжелые фичи собранные длинными окнами агрегаций, а также другими ML-моделями

Оркестрируем с помощью Airflow *[см. доклад Миши М]*

Формат фичей

В Kafka пишем сериализованные объекты

Декодеры на клиенте преобразуют байты в исходный тип. При создании топика ему назначается декодер.

В проекте 60+ декодеров:

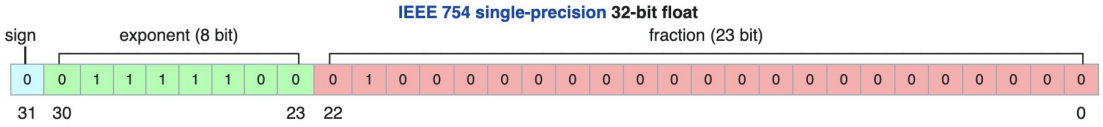
1) Простые Long, Double, DoubleVector
2) Декодер сложных объектов CtrDataDecoder
3) Декодеры компрессированных форматов

```
import java.nio.ByteBuffer;

public interface Decoder<T> {

    T toValue(ByteBuffer bytes);

}
```

Формат фичей. Сжатые форматы

Без потерь

DELTA компрессия [12,5%; 112,5%]
Сортирует набор неотрицательных целых чисел и записывает дельту между ними.

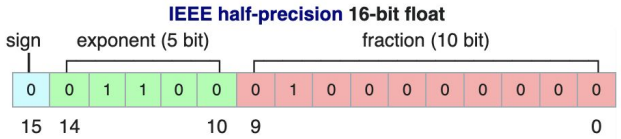


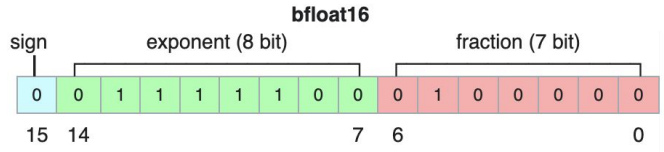
С потерями

DELTA_{THREE,FOUR}_DIGITS
Дельта-кодирование округлённого до третьего знака числа.

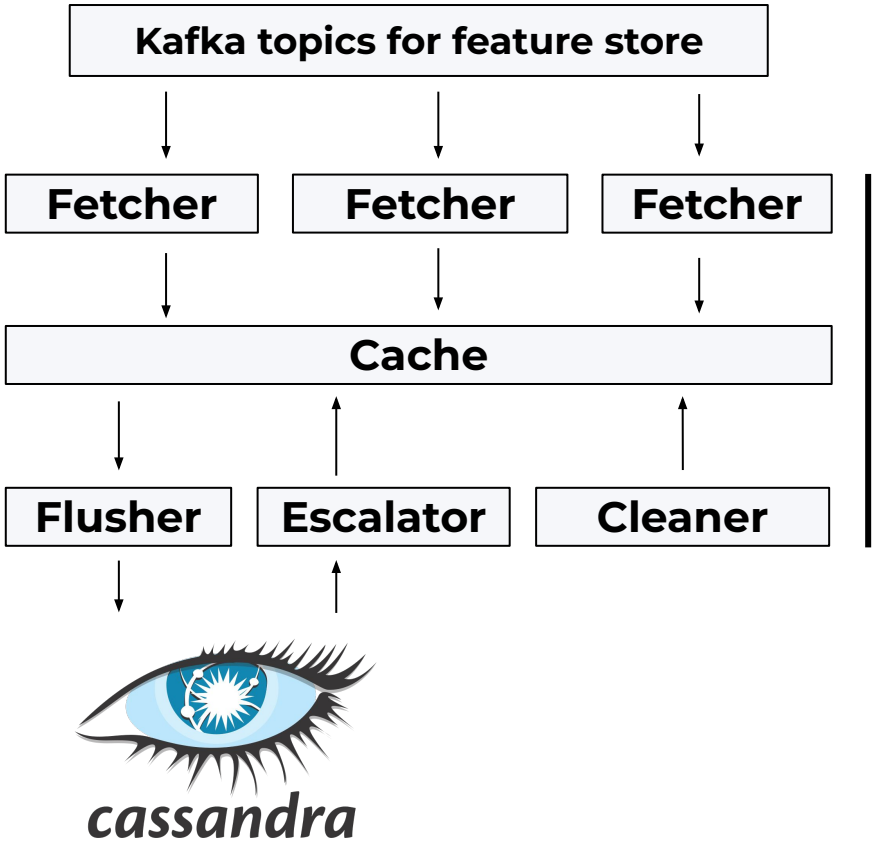
S10E5, BFLOAT16
режем диапазон и/или точность

{SINGLE,TWO}_BYTE_DISTRIBUTION
Делит диапазон значений на отрезки и присваивает числам в каждом отрезке индексы



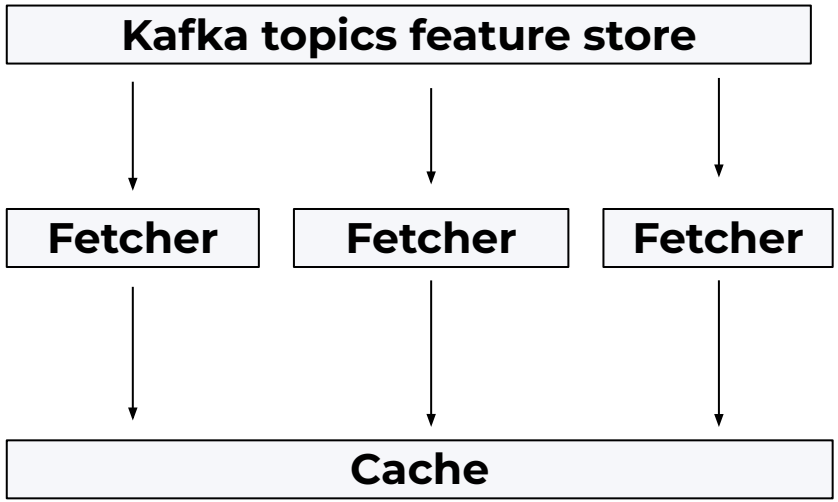


Хранение фичей. Архитектура



Управляемый кэширующий слой позволяет развязать нагрузку на запись и чтение

Cache layer



Fetcher - отдельный поток для выгрузки избранных партиций топиков инстанса

В кэш также сохраняем оффсеты зафетченных топиков по партициям

Можем ограничивать скорость поступления данных, что актуально для батчевых выгрузок

В одном фетчере группируем топики по смыслу (сервису), нагрузке

Cache layer. Cache data scheme

Version
Key length
Key
Column count
Column index 1
Dirty flag 1
Value 1

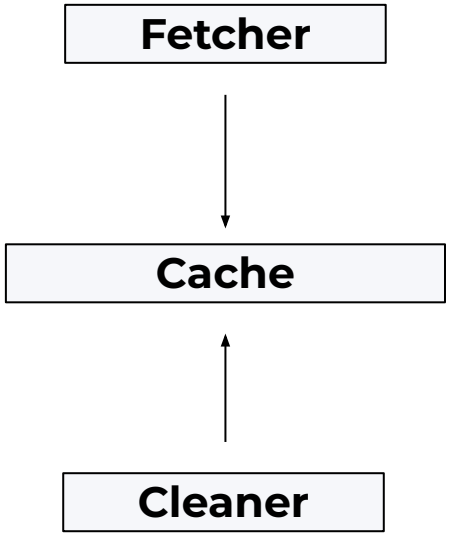
Column index N
Dirty flag N
Value N

В ключ записываем сведения о типе объекта

Индекс колонки (топика) имеет маппинг с его названием

Dirty flag используем для обозначения того, что топик еще не записан в персистентное хранилище

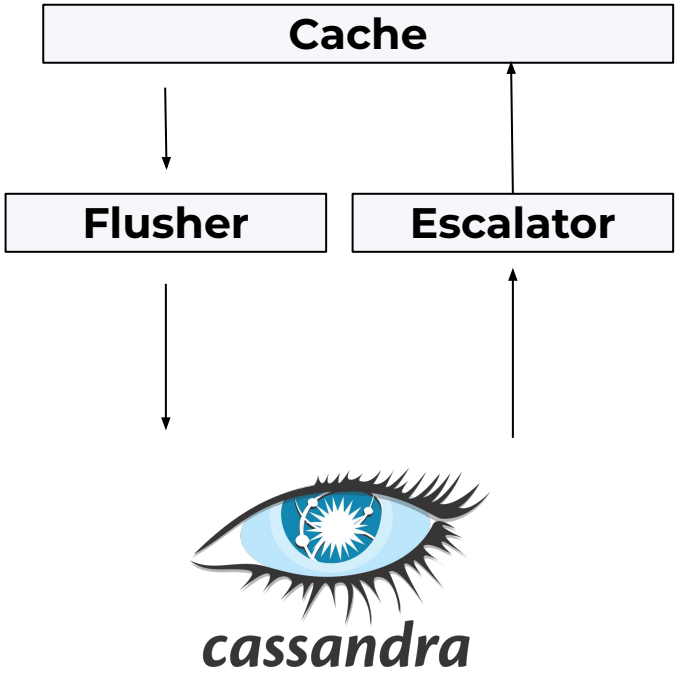
Cache layer. Cleaner



Отдельный поток клинера последовательно обходит все объекты в кэше и выкидывает давно неиспользуемые

- Чистим жестче если лимиты вместимости кончаются
- “Грязные” объекты трогаем только если других не осталось
- Удаляем топики специально отмеченные для чистки

Persistent layer



Синглнод Кассандра

Проваливаемся если не находим данные в кэше

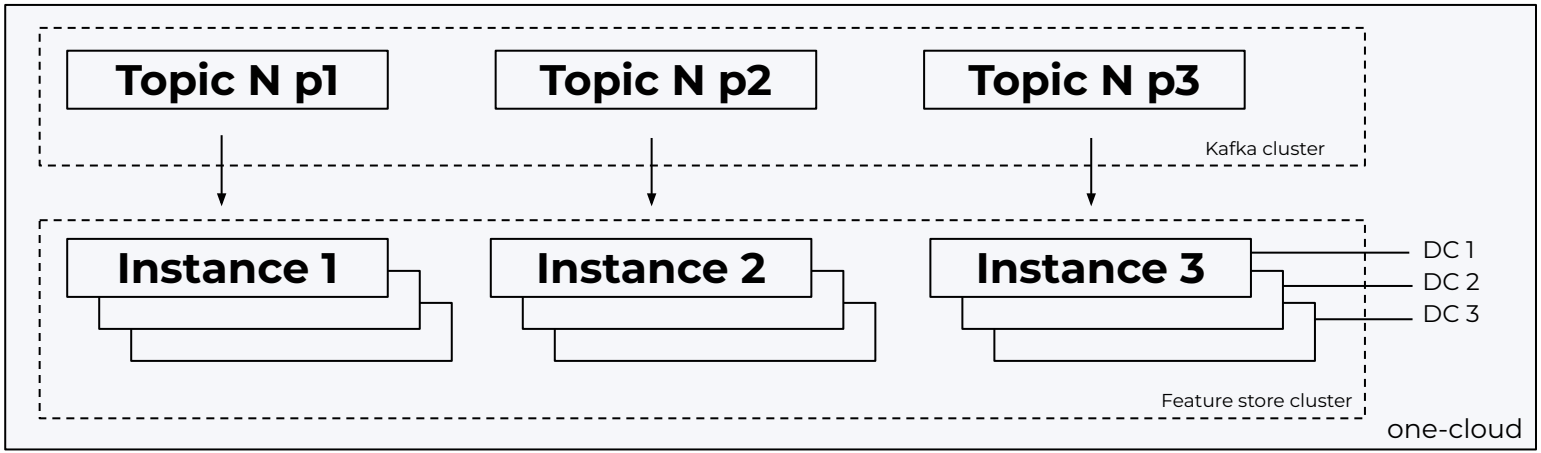
Эскалатор поднимает запрошенные данные в кэш

Флашер последовательно обходит кэш в поисках “грязных” объектов

Хранит фичи, хранит оффсеты прочитанных топиков

Чистит неактуальные фичи по TTL
Чистим топики принудительно

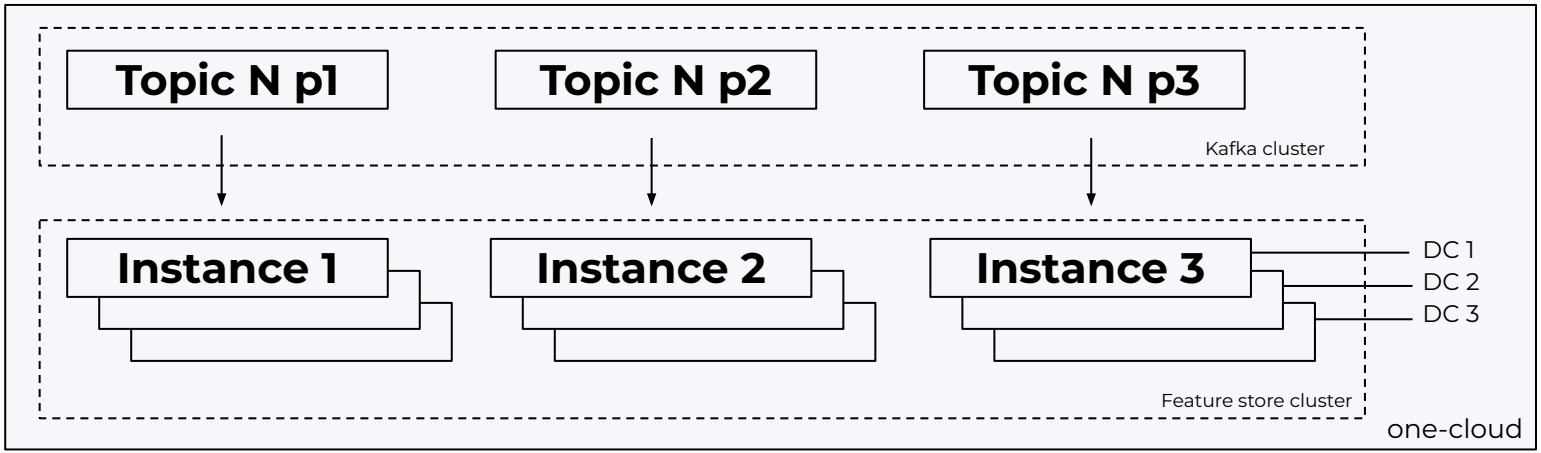
Кластера



Кластеры фичерсторов и управление кластерами в one-cloud:
- По какому id объекта складываем?
- Для какого сервиса поднимаем?
- Учитываем число и размер объектов

One-cloud — ОС уровня дата-центра в ОК

Кластера



Пример реального кластера:

- *counter* кластер 3x3
- 50k rps
- 30M объектов в кэше

vcores: '8'
mem: 12g
cache: 80G memdisk
persist: 160G

ctr *taste*

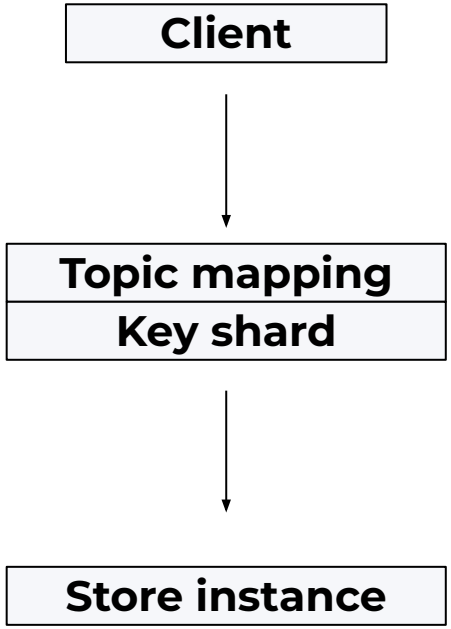
common *graph*

video *rating*

One-cloud — ОС уровня дата-центра в ОК

Клиенты

- Батчевые запросы фичей по ключам и топикам
- Декодирование в объекты
- Маппинг топиков в кластера
- Client-side caching
- Client-side partitioning
- one-nio RPC



Мониторинг

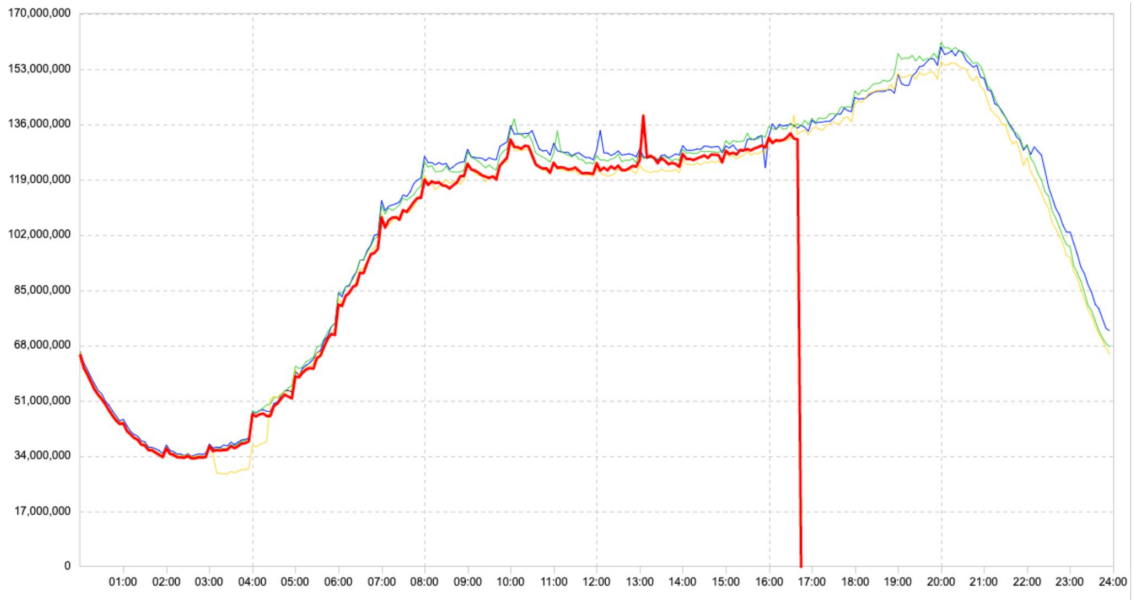
На доставку фичей:

- Offset lag
- Data availability

На сервинг фичей:

- Requests count
- Cache miss
- Storage escalations

\+ Full HW monitoring



Клиентские запросы к Feature Store

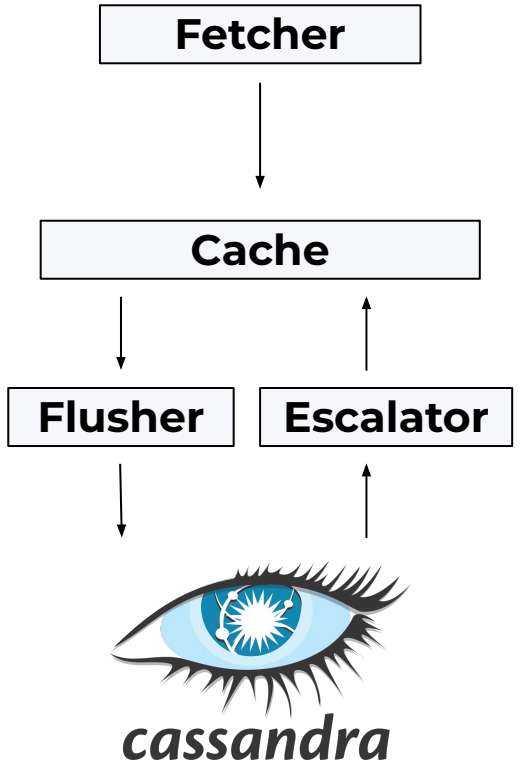
Backpressure

Лимиты серверные:

- Лимитер на фетчере
- Лимитер на флашер
- Лимитер на эскалатор
- Рандомайзер на запись в Кассандру

Лимиты клиентские:

- Лимитер на проваливание в персистент при превышении среднего времени чтения из Кассандры



Fault tolerance

Переживаем

- Потеря Cache
- Потеря Storage
- Consistency by Kafka

Рестарты инстансов могут быть даже без сброса кэша

Реплицируем потерянные стораджи с живых нод

Итоги

1. Архитектура горячего фичерстора может быть простой пока мало данных
2. Кэширующий слой - ключевой компонент
3. Кафка хороша как универсальная шина и решает проблему консистентности
4. Аккуратный выбор формата фичей может сэкономить ресурсы

Спасибо за внимание!

Мы ищем
новых коллег

hr@odnoklassniki.ru

@netcitizen

andrew.kuznetsov@corp.mail.ru