Сегментация: Единое окно для знаний о пользователе

Зачем мы тут?

- Расскажем:
  - как на java стеке собрали систему сегментации пользователей
  - как мы сами себе выстрелили в ногу Clickhouse-ом, а потом сделали его супероружием
  - как такая система помогает легко проверять сложные гипотезы

IVI — устойчивый лидер на быстрорастущем рынке OTT-сервисов в России

50млн

Среднее значение MAU во 2 квартале 2020 (рост +12% год к году)

>3млн

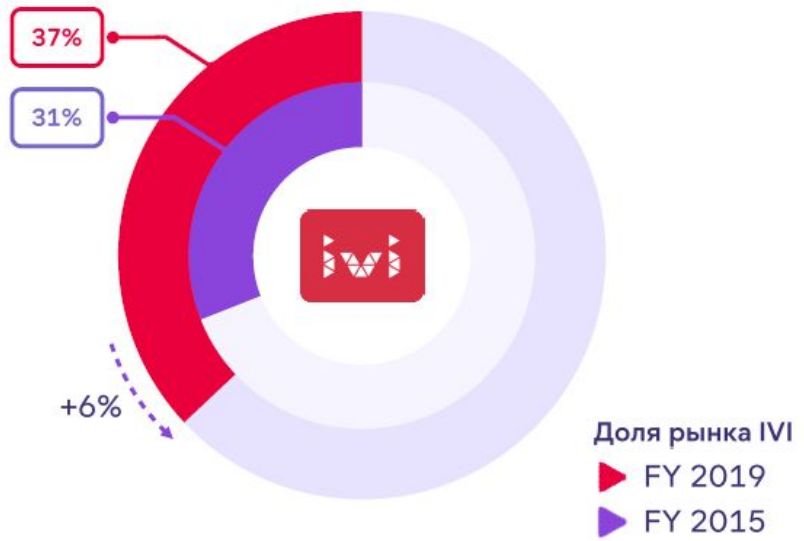
Ежемесячная база платящих пользователей

Онлайн-кинотеатр №1

Среди россиян

*согласно данным опроса потребителей

Структура российского рынка видеосмотрения в разбивке по выручке

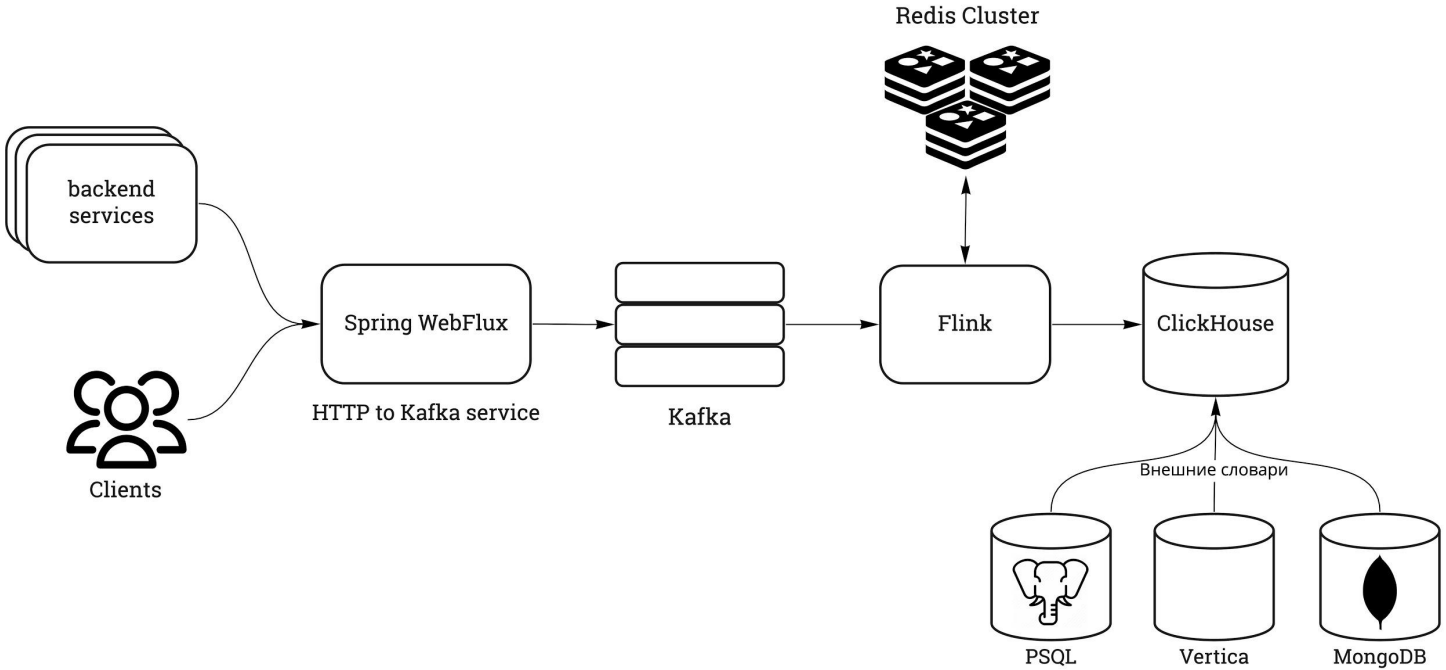


Источник: Company data, TMT Consulting

Кто мы - BigData Team

- Стриминг данных с этих 50 млн пользователей
- Spring - Kafka - Flink - Clickhouse
- Данные с 11 года и легаси хранилища

Стриминг данных в IVI



https://habr.com/ru/company/ivi/blog/347408/

Запросы бизнеса

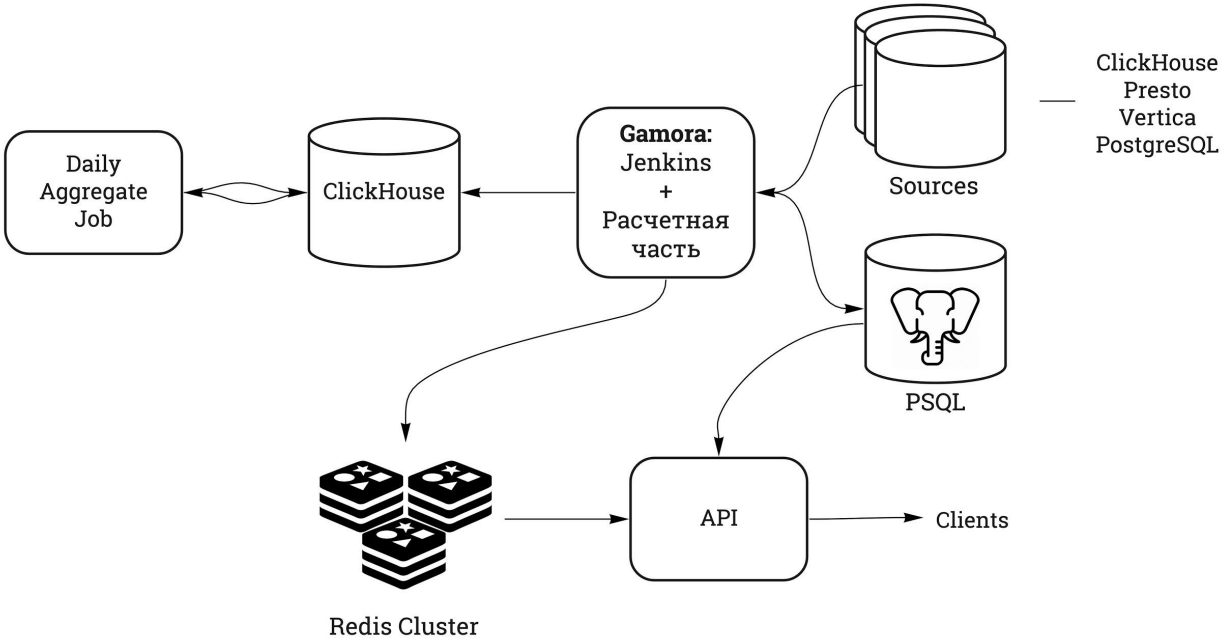
- Хочу пользователей у которых
  - 3 дня смотрения
  - минимум 120 минут смотрения за последние 7 дней
  - активный AdBlock
  - нет использованного ранее триала
  - и нет просмотров определенных страниц

MVP на коленке

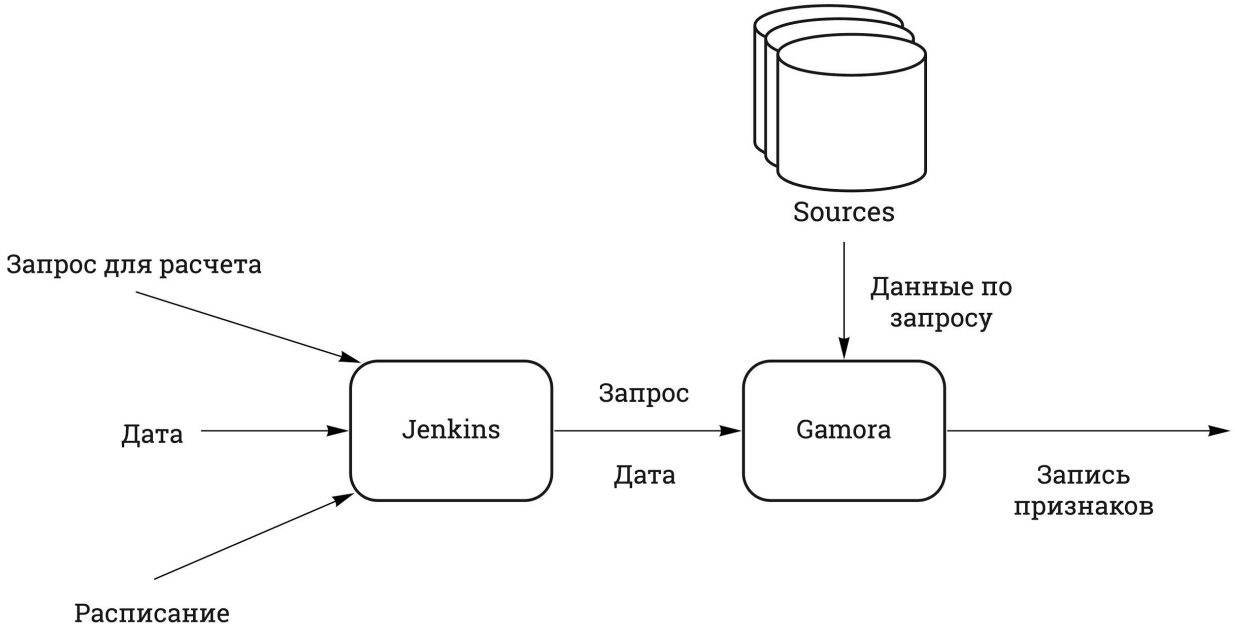
Основные понятия

- Признак - некоторое свойство пользователя
- Условие - правило выделения признака
  - SQL-запрос
  - Файл
  - Логическое выражение
- Джоба - “применитель” условий
- Фильтр - логическое выражение над признаками
- Сегмент - подмножество пользователей, соответствующих некоторому фильтру

MVP: общая архитектура

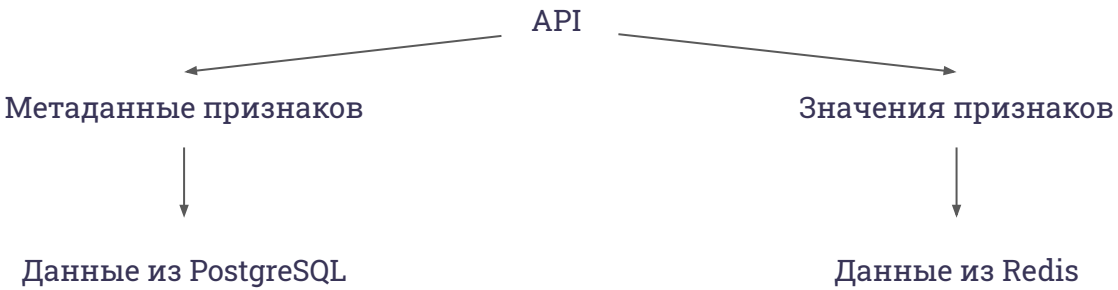


MVP: расчетная часть



MVP - API

Jetty-сервер с двумя сервлетами:



Ограничение API - 1000 rps (< 50ms)

MVP - схемы хранения данных

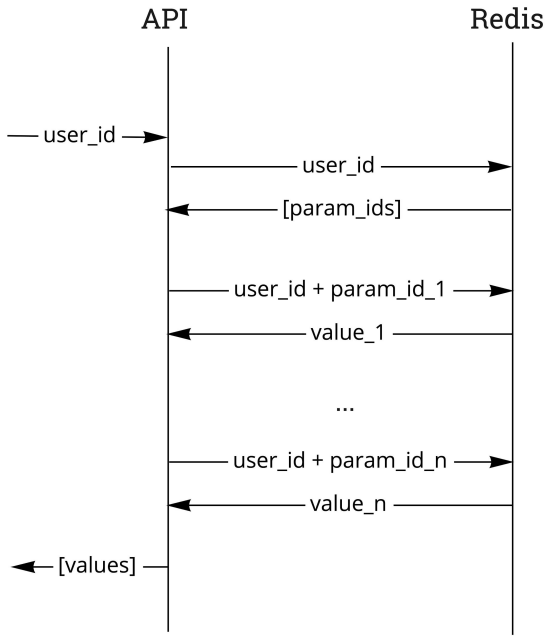
Redis

- Основное хранилище значений
- Используется для запросов API
- Записи в байтовом формате

Схема:

Ключ	Значение
`user_id + param_id`	`value`

`user_id`	`[`
	`  param_id_1,`
	`  param_id_2,`
	`  ...`
	`  param_id_n`
	`]`



MVP - схемы хранения данных

ClickHouse

- Данные для отладки
- На каждый признак своя колонка

Плюс: не замороченная схема

Минус: тяжело добавлять новые колонки

Схема таблицы:

user_id	Int64
date	Date
dt	DateTime
param_1_name	Param1Type
param_2_name	Param2Type
...	...
param_n_name	ParamNType

Важно было уметь

- Поддерживать набор баз-источников
- Поддерживать разные типы данных - bool, int, string
- Сохранять историю для отладки
- Отвечать довольно быстро

Важно было уметь

- Поддерживать набор баз-источников
- Поддерживать разные типы данных - bool, int, string
- Сохранять историю для отладки
- Отвечать довольно быстро
- Свободно расширять набор признаков (~2000)
- Расширять набор признаков без рук разработчиков

Кто в итоге подсел?

- Аналитика - универсальный агрегат
- Машинное обучение - история разнообразных фич
- Коммуникации - удобно набирать аудитории

- *А не промо-акции и размещение рекламы в видеоплеере*
- *И зачем нам API ?*

Сложности

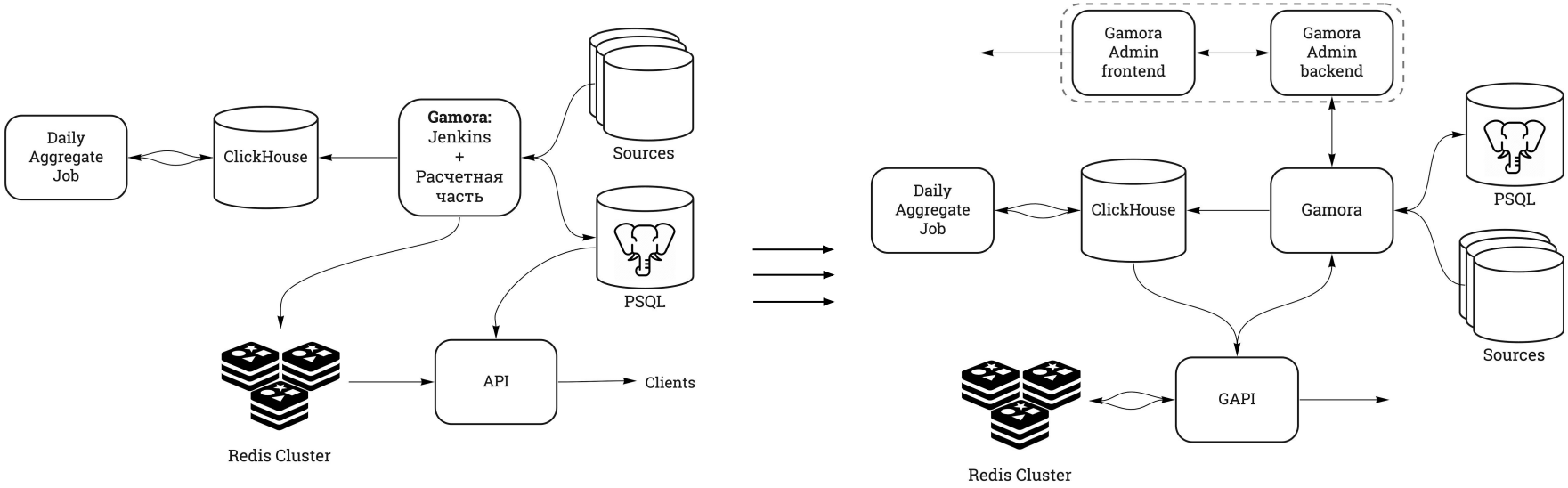
- Сомнения про нагрузку
- Clickhouse не любит когда постоянно изменяют схему
- Слишком простая админка без проверки на дурака
- Запись в ClickHouse однопоточна -> долго

Запросы бизнеса росли

- Хотим забирать аудитории по api
- Надо держать больше нагрузки
- Правда неограниченное количество признаков (> 10 000)
- Прозрачное управление признаками

2.0

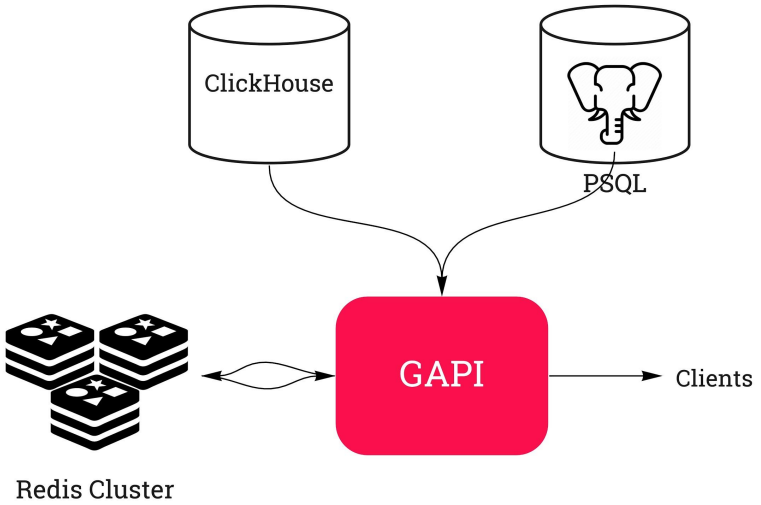
Архитектура 2.0



2.0
API

API

- Spring WebFlux
- Работа с Redis:
  - Spring Data Redis
  - driver - Lettuce (async & on netty)



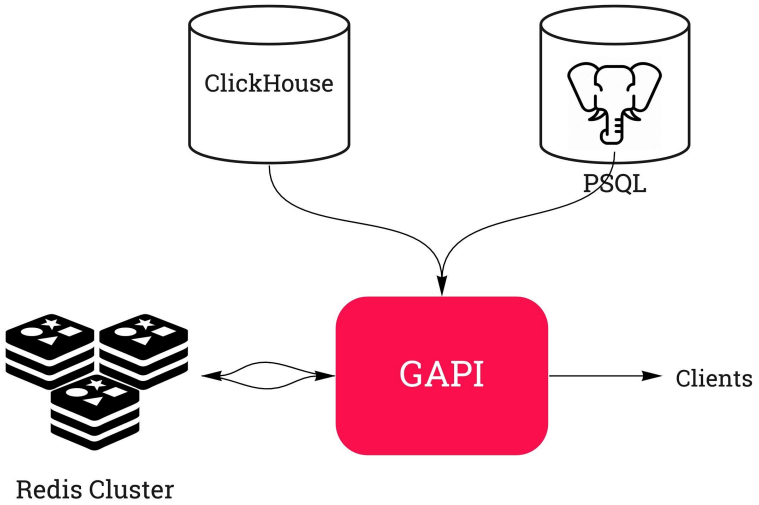
API

Блокирующие операции:

- Работа с ClickHouse - ClickHouse-JDBC
- Работа с PostgreSQL - Spring Data JPA

\+ Mono.fromCallable(...)
Flux.defer(...)

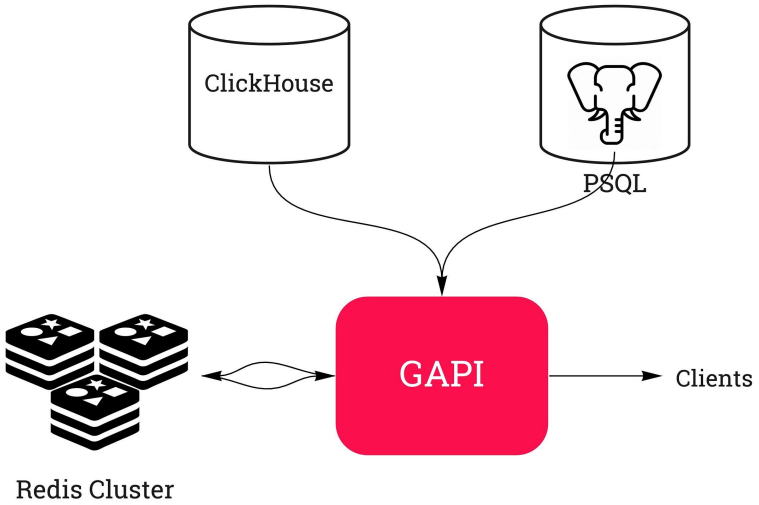
Неблокирующие операции!



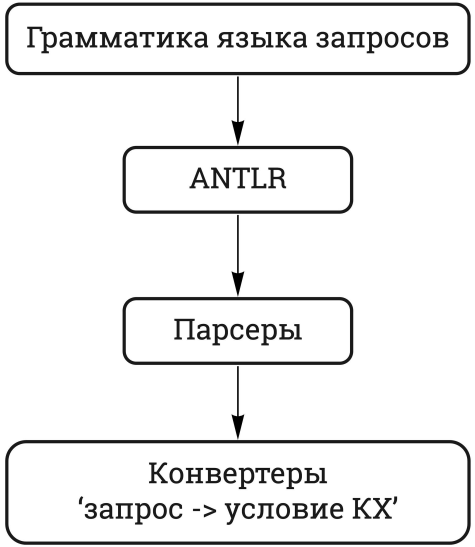
API

Что умеет:

- Отдавать метаданные
- Работать с быстрым хранилищем (Redis w/r)
- Читать из медленного хранилища:
  - самые частые значения признака
  - фильтрация юзеров



QL для фильтрации



ANTLR - ANother Tool for Language Recognition

QL для фильтрации

Простые условия:

EQ/INEQ:

```
{ type: "eq" | "not_eq" | "gt" | "lt" | "gte"
| "lte" | "contains" | "not_contains"
  property_id: int
  values: string[]
}
```

DATE:

```
{ type: "ago_delta" | "ahead_delta"
  property_id: int
  values: string[]
}
```

IN:

```
{ type: "in" | "not_in"
  property_id: int,
  values: string[]
}
```

NULL:

```
{ type: "is_null" | "is_not_null"
  property_id: int
}
```

QL для фильтрации

Составные условия:

OR/AND:

```
{ type: "or" | "and"
  conditions: Condition[]
}
```

NOT:

```
{ type: "not"
  condition: Condition
}
```

Condition - любое условие (простое или составное)

QL для фильтрации

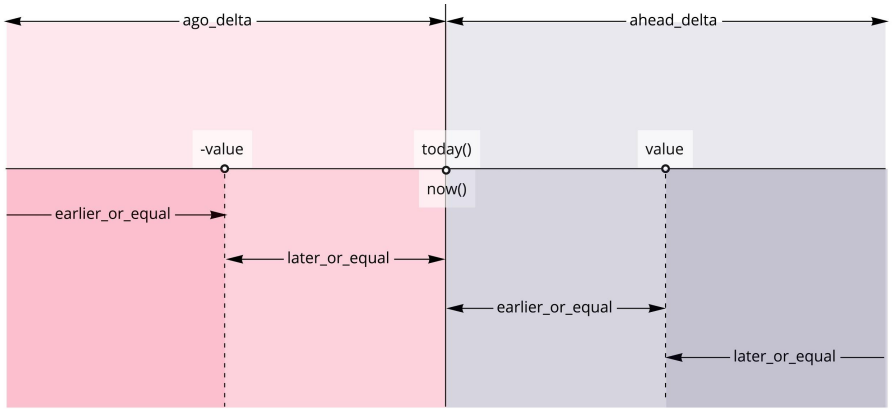
Пример

Достать всех пользователей, которые

- последняя территория активности - Россия

И

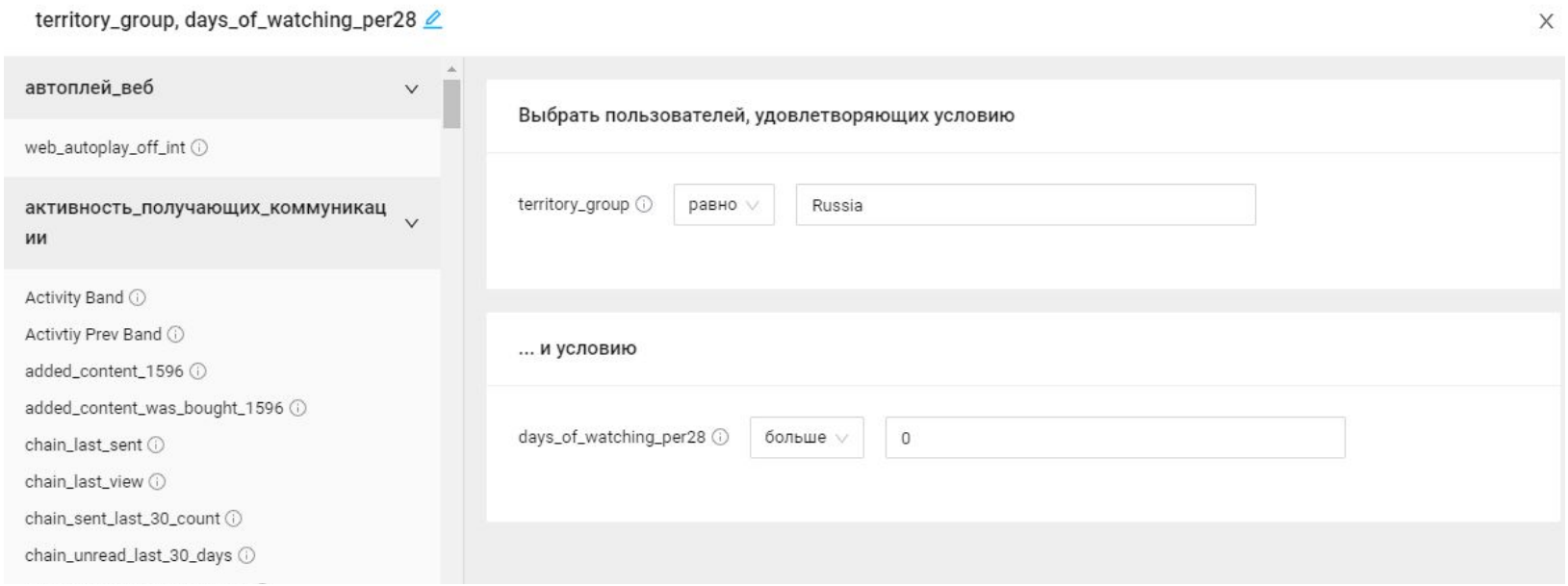
- пришел на ivi больше 6 месяцев назад



Условие

```
"type": "and",
"conditions": [
  { "type": "eq",
    "property_id": 1,
    "values": [ "Russia" ]
  },
  { "type": "ago_delta",
    "property_id": 2,
    "values": [ "6", "month", "earlier_or_equal" ]
```

Аналитика заменили на UI

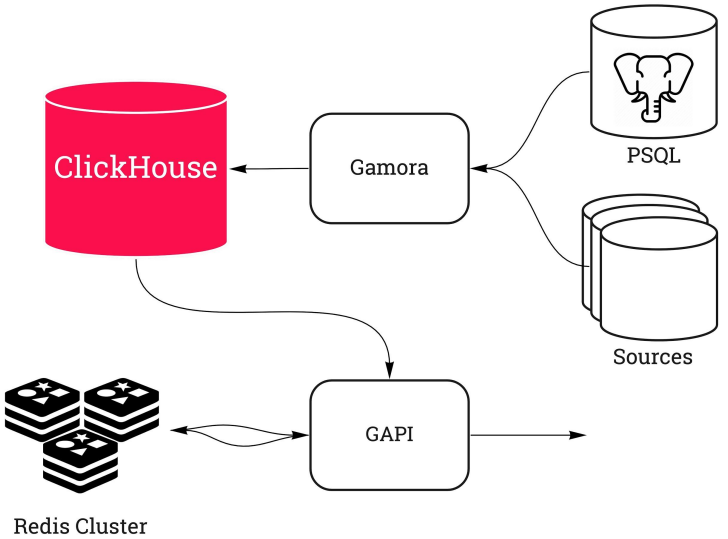


2.0
ClickHouse

ClickHouse

Проблема

- ALTER блокируется
- Инсерт в широкие таблицы - больно
- API начнет ходить в ClickHouse



ClickHouse - нет ALTERам

- запас колонок

user_id	Int64
date	Date
dt	DateTime
territory_group	String
days_at_ivi	Int
+	+
param_1_int	Int
param_2_int	Int
param_N_int	Int

ClickHouse - нет ALTERам

- entity-attribute-value model
- 1 строка = 1 признак

user_id	Int64
date	Date
dt	DateTime
param_name	String
param_type	String
...	...
param_int_value	Int64

ClickHouse - нет ALTERам

- “компактный” EAV
- 1 строка = 1 задача

user_id	Int64
date	Date
dt	DateTime
param_name	Array(String)
param_type	Array(String)
...	...
param_int_value	Array(Int64)

ClickHouse - массивы

user_id	100000	
date	2020-12-10	
dt	2020-12-10 10:30:00	
property.name	['last_content_rating',	'last_watched_content']
property.type	['Integer',	'String']
int8_value	[NULL,	NULL]
int64_value	[9,	NULL]
string_value	[NULL,	'Метод']

Немного боли аналитика

```
WHERE
territory_group = 'Russia'
```

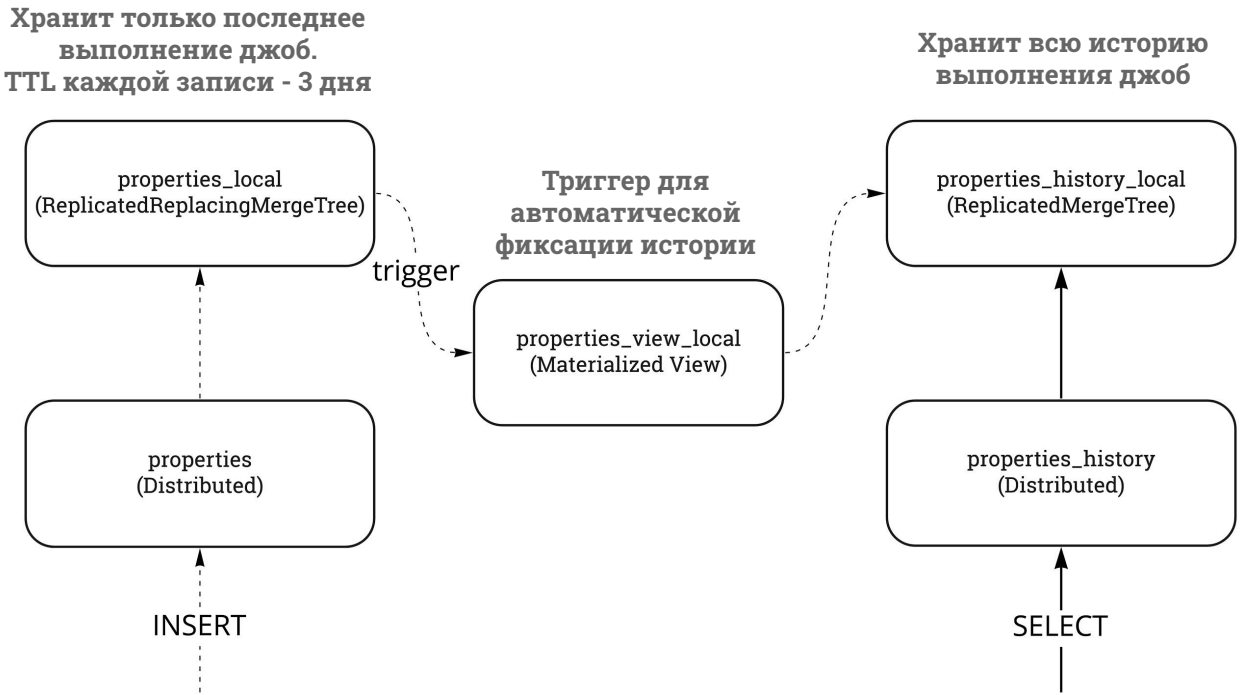
```
WHERE
has(property.name, 'territory_group') -- фильтруем строки
GROUP BY user_id
HAVING
argMaxIf( -- дедуплицируем
    arrayElement(property.string_value, -- из метаданных знаем тип
    indexOf(property.name, 'territory_group')), -- ищем индекс
    dt,
    has(property.name, 'territory_group')) = 'Russia'
```

Движки в Clickhouse

Семейство MergeTree

- репликация
- сэмплирование
- ключи сортировки
- дедупликация и агрегация
- TTL
- партиции

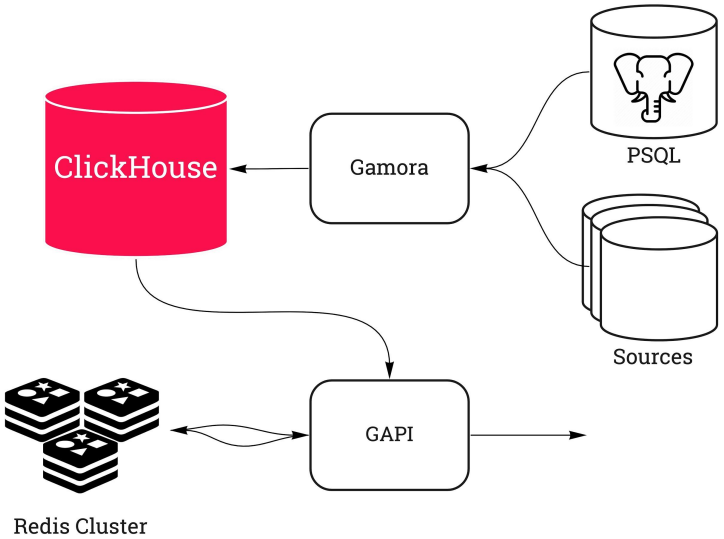
ClickHouse - автоистория



Gamora ClickHouse

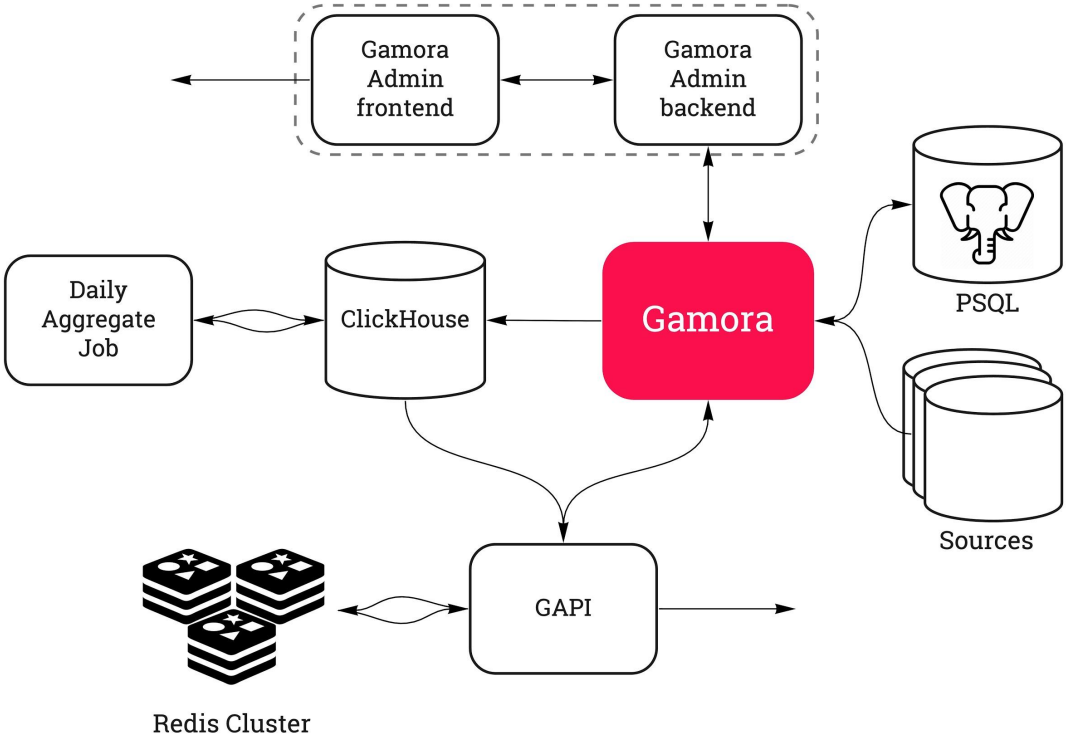
Как живёт:

- Отдельный кластер
- 4 сервера
- 2 кластера-”зеркала” (только Distributed- таблицы)



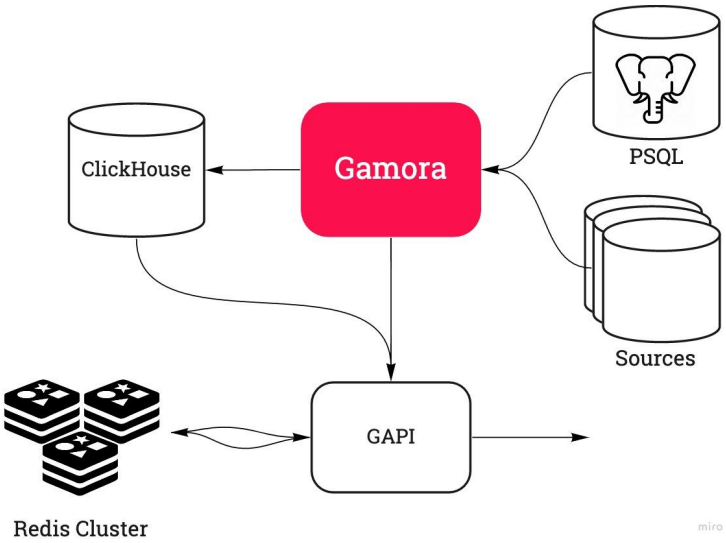
2.0
Основное приложение

Gamora


Gamora
Admin
frontend
Gamora
Admin
backend
Daily
Aggregate
Job
ClickHouse
Gamora
PSQL
Sources
GAPI
Redis Cluster

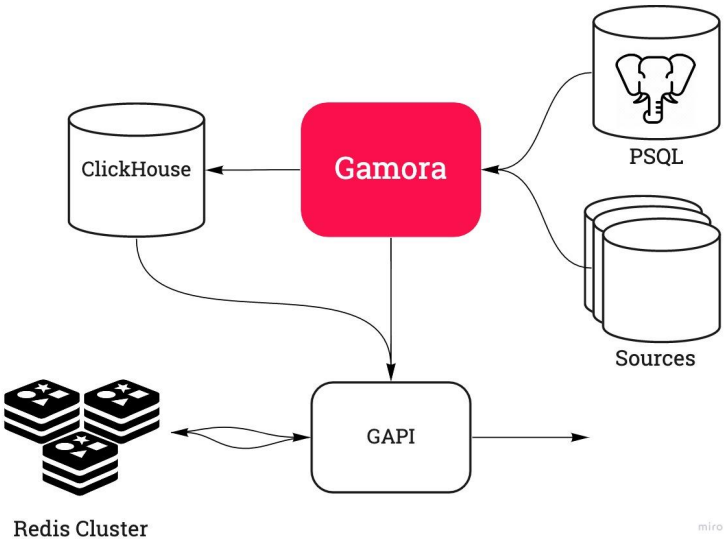
Gamora - план перевода

- Сделать самостоятельным сервисом
- Отказаться от Jenkins
- Написать свою админку
- PostgreSQL - только из Gamora
- Взаимодействие между сервисами



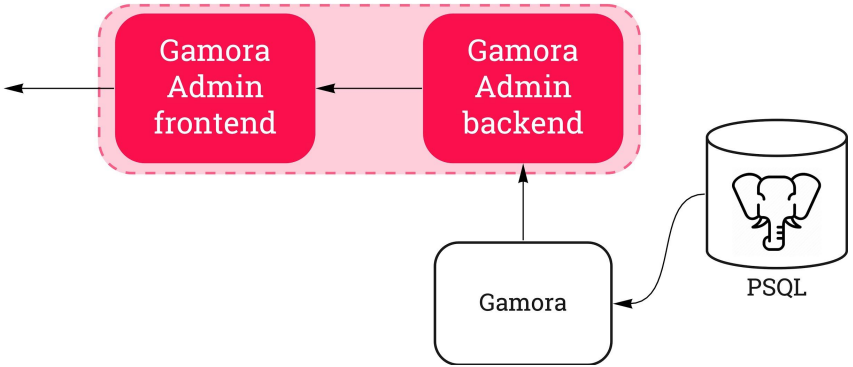
Gamora - как она сделана

- Spring WebFlux
- Jenkins -> Quartz:
  - интеграция со Spring
  - PostgreSQL для метаданных джоб
  - возможно сделать кластерным



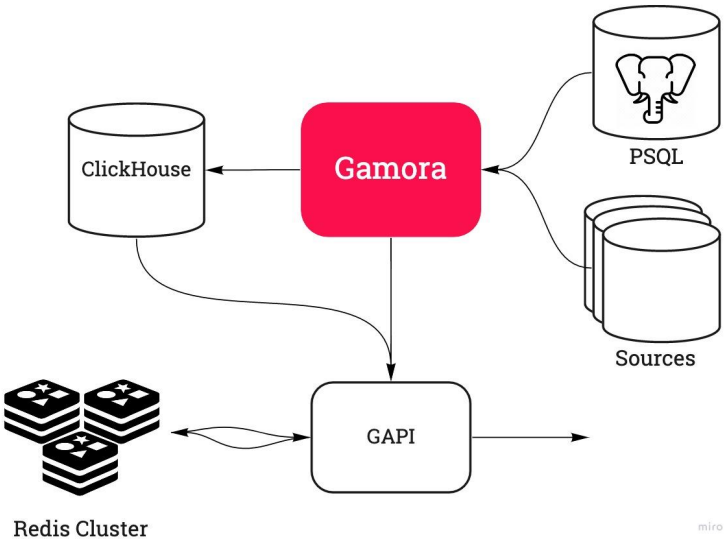
G.admin

- Админка:
  - frontend: react-admin
  - backend: Spring WebFlux



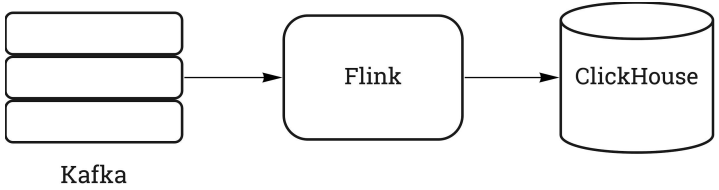
Gamora - как она сделана

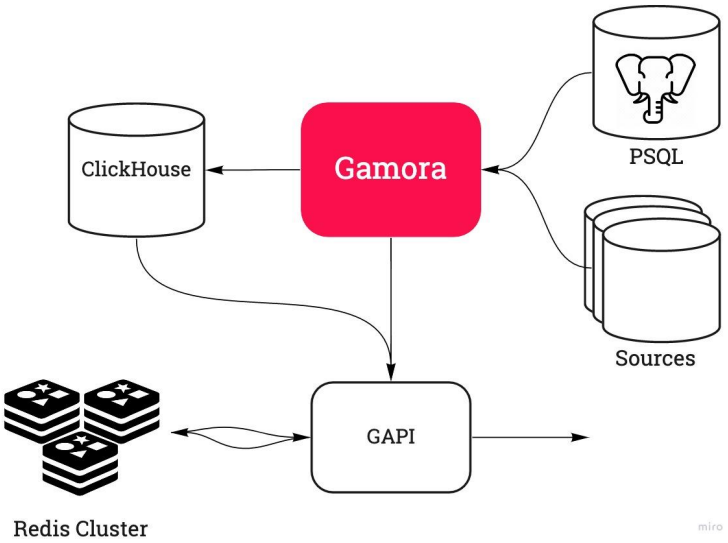
- Межсервисное взаимодействие:
  - RSocket
  - ProtoBuf



Gamora - как она сделана

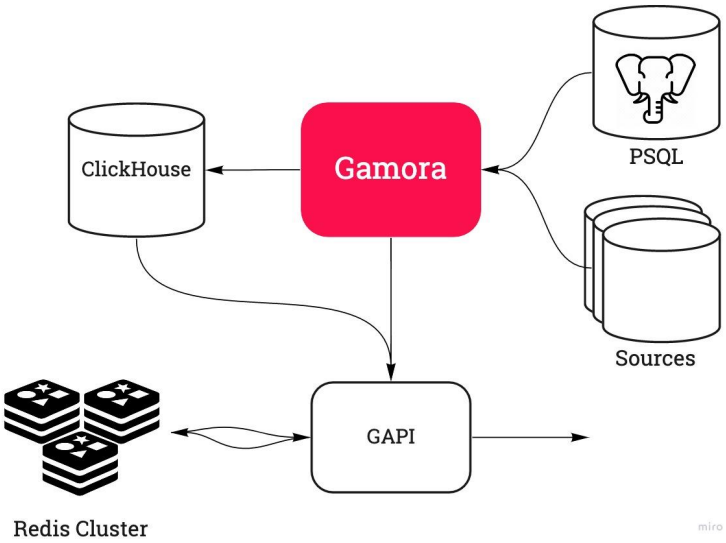
- github.com/ivi-ru/flink-clickhouse-sink



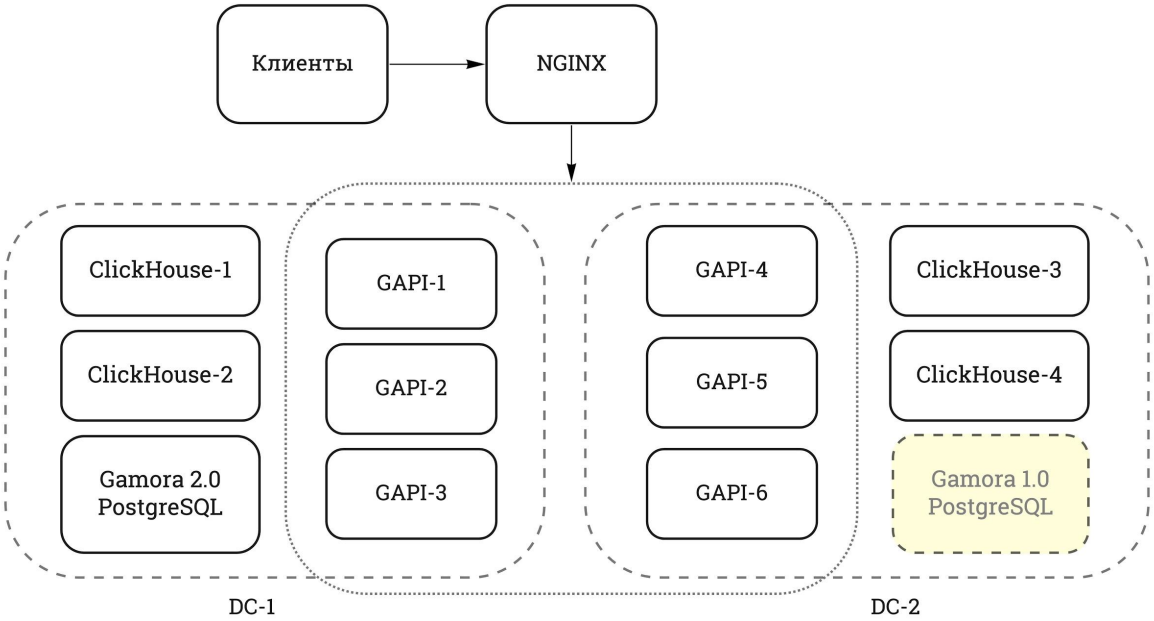


Gamora - что умеет

- Рассчитывает признаки пользователей
- Источники данных:
  - ClickHouse
  - Presto
  - Vertica
  - PostgreSQL
  - Файлы
- Расчет по установленному расписанию (Quartz scheduler) или по требованию (через GAPI или админку)



Gamora - Отказоустойчивость



Что теперь умеем?

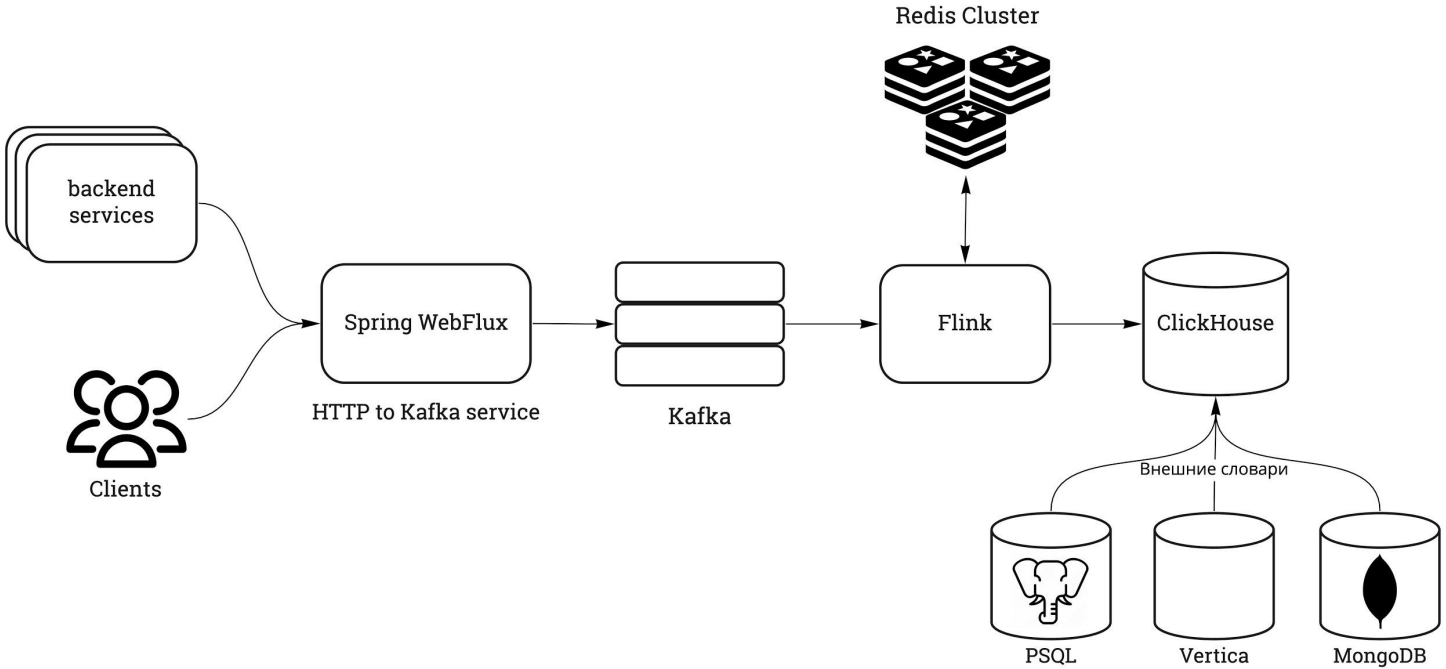
- Собирать признаки из разных баз данных
- Считать признаки по расписанию и запросу и записывать всё это в ClickHouse и Redis
- Отвечать на:
  - “Какие есть признаки?”
  - “Что насчиталось по этому пользователю?”
  - “Вот все пользователи, такие что …”
  - “Таких пользователей нашлось N”
  - “Топ N значений признака”
- Собирать из всего посчитанного дневной агрегат

Кто в итоге подсел?

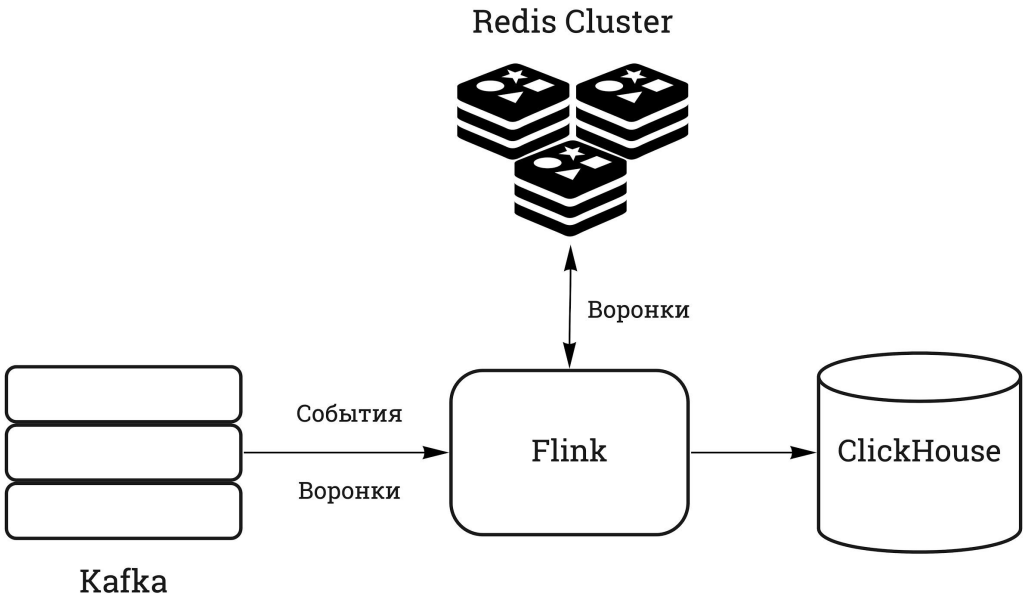
- Поддерживаем crm
- Готовим сырье для машинного обучения
- Продаем таргетированную видеорекламу
- Персонализируем лендинги
- Персонализируем блоки приложения
- Включаем закрытые запуски
- Снимаем нагрузку с микросервисов

Признаки almost real-time

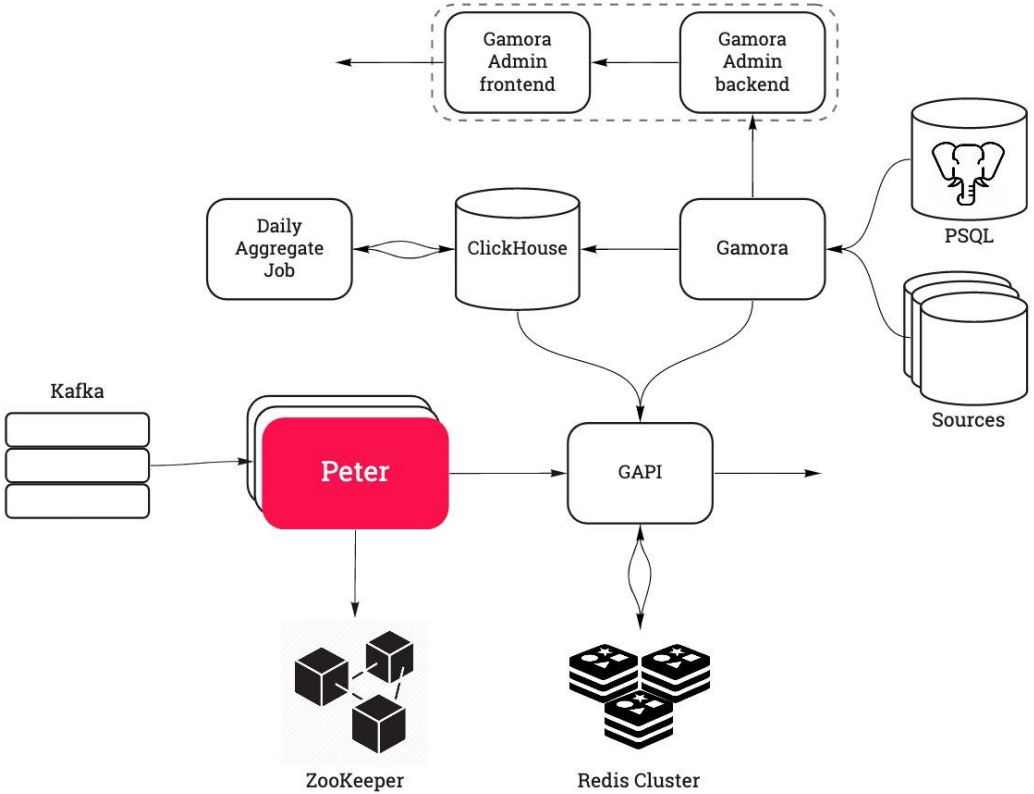
Как вообще мы собираем данные



Картинка про воронки

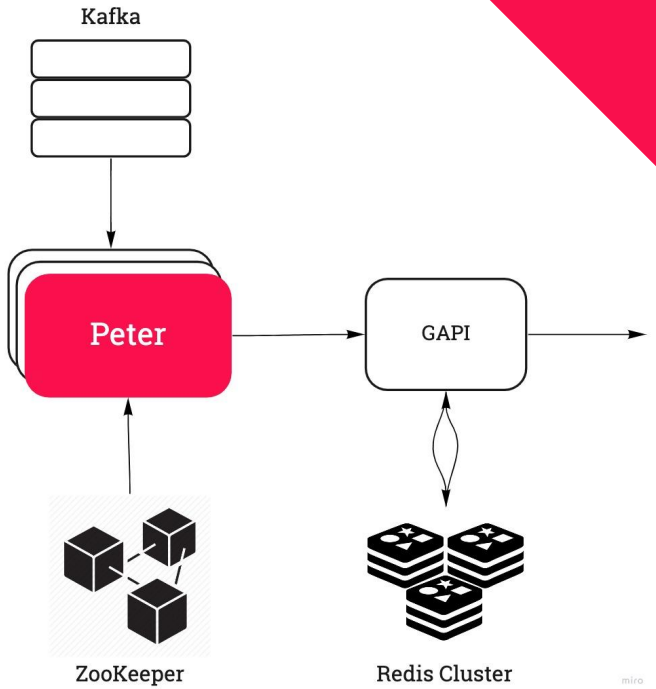


Peter - быстрые признаки



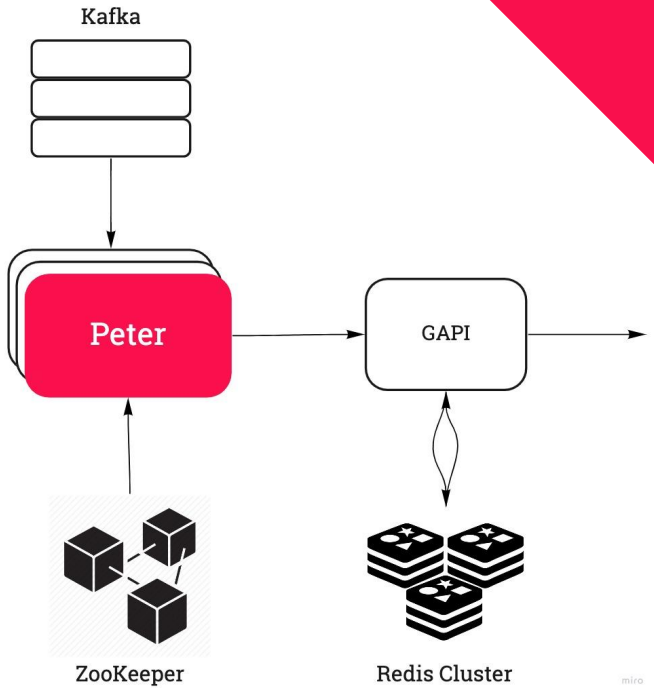
Peter - быстрые признаки

- Как устроен:
  - Flink
  - Обновление близко к реальному времени
  - Для конфигураций признаков - ZooKeeper + Curator



Peter - быстрые признаки

- Ограничения текущей версии:
  - Только bool-значения признаков
  - Источник: топик backend-событий
  - Обработка только первого уровня json



Peter - условия

```
(event == purchase)
        and
 (success == true)
```



```
a and b
```

```
{ "name": "has_any_purchase",
  "property_id": 1,
  "conditions":
    { "expression": "a b AND",
      "variables": [
        { "name": "event",
          "value": "purchase",
          "operand": "a",
          "predicate": "eq"
        },
        { "name": "success",
          "value": "true",
          "operand": "b",
          "predicate": "eq"
```

Peter - условия

```
(event == purchase)
        and
(success == true)
```



```
a and b
```

```
{ "name": "has_any_purchase",
  "property_id": 1,
  "conditions":
    { "expression": "a b AND",
      "variables": [
        { "name": "event",
          "value": "purchase",
          "operand": "a",
          "predicate": "eq"
        },
        { "name": "success",
          "value": "true",
          "operand": "b",
          "predicate": "eq"
```

Peter - условия

```
(event == purchase)
        and
 (success == true)
```



```
a and b
```

```
{ "name": "has_any_purchase",
  "property_id": 1,
  "conditions":
    { "expression": "a b AND",
      "variables": [
        { "name": "event",
          "value": "purchase",
          "operand": "a",
          "predicate": "eq"
        },
        { "name": "success",
          "value": "true",
          "operand": "b",
          "predicate": "eq"
```

Peter - условия

```
(event == purchase)
        and
 (success == true)
```



```
a and b
```

```
{ "name": "has_any_purchase",
  "property_id": 1,
  "conditions":
    { "expression": "a b AND",
      "variables": [
        { "name": "event",
          "value": "purchase",
          "operand": "a",
          "predicate": "eq"
        },
        { "name": "success",
          "value": "true",
          "operand": "b",
          "predicate": "eq"
```

Peter - условия

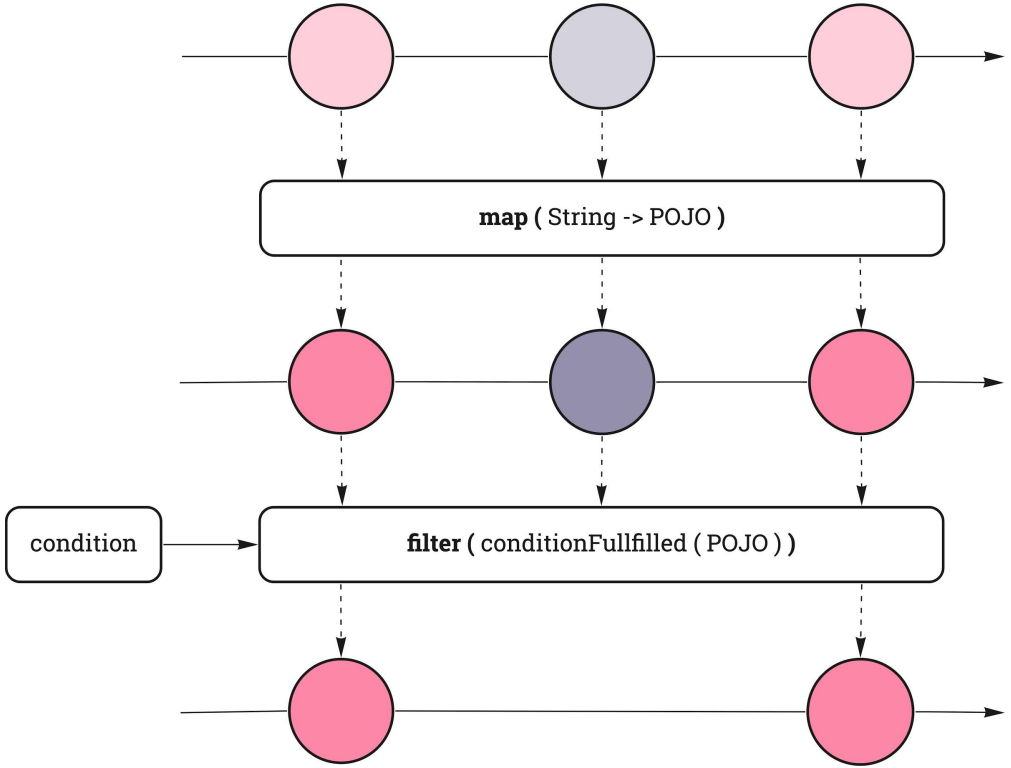
```
(event == purchase)
        and
 (success == true)
```



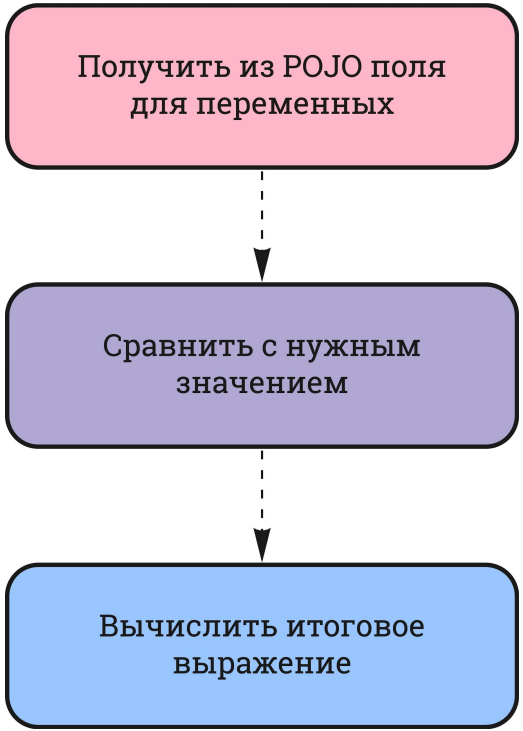
```
a and b
```

```
{ "name": "has_any_purchase",
  "property_id": 1,
  "conditions":
    { "expression": "a b AND",
      "variables": [
        { "name": "event",
          "value": "purchase",
          "operand": "a",
          "predicate": "eq"
        },
        { "name": "success",
          "value": "true",
          "operand": "b",
          "predicate": "eq"
```

Путешествие события



Путешествие события




```
"expression": "a b AND",
"variables": [
  { "name": "event",
    "value": "purchase",
    "predicate": "eq",
    "operand": "a"
  },
  { "name": "success",
    "value": "true",
    "predicate": "eq",
    "operand": "b"
  }
```

Peter для бизнеса

- Признаки из событий стриминга
- Большинство - “пользователь хоть раз делал X”

Что дальше?

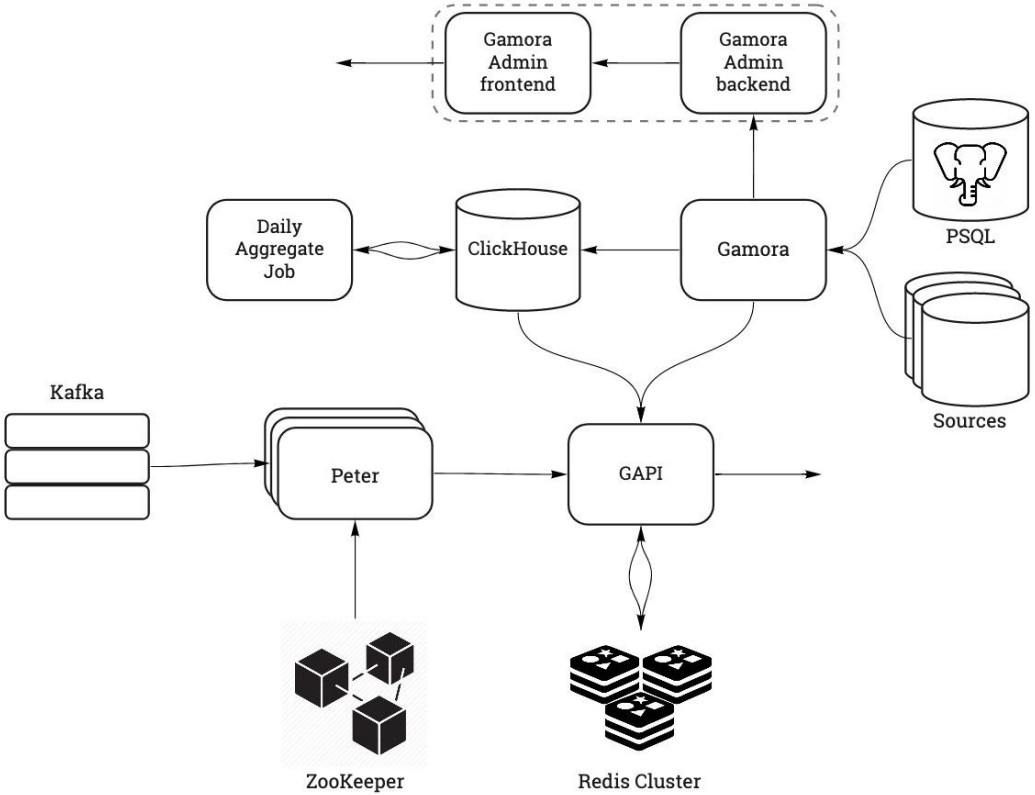
Чему мы научились

- DWH + Scheduler + KVstorage + Jetty = норм сегментатор
- Свой язык запросов - не страшно
- Clickhouse не тормозит, если его использовать без насилия

To improve

- rSocket модный и молодежный, но возможно уйдём в grpc
- Расширение функционала быстрых признаков
- Встроить проверки готовности данных

Вопросы?


Gamora Admin frontend
Gamora Admin backend
Daily Aggregate Job
ClickHouse
Gamora
PSQL
Sources
Kafka
Peter
GAPI
ZooKeeper
Redis Cluster

Приглашение к дискуссии

- Есть ли у вас “единое окно”? Как оно устроено?
- Как вы следите за хаосом и мусором в гибких системах?

Спасибо за внимание :)