Рамблер/

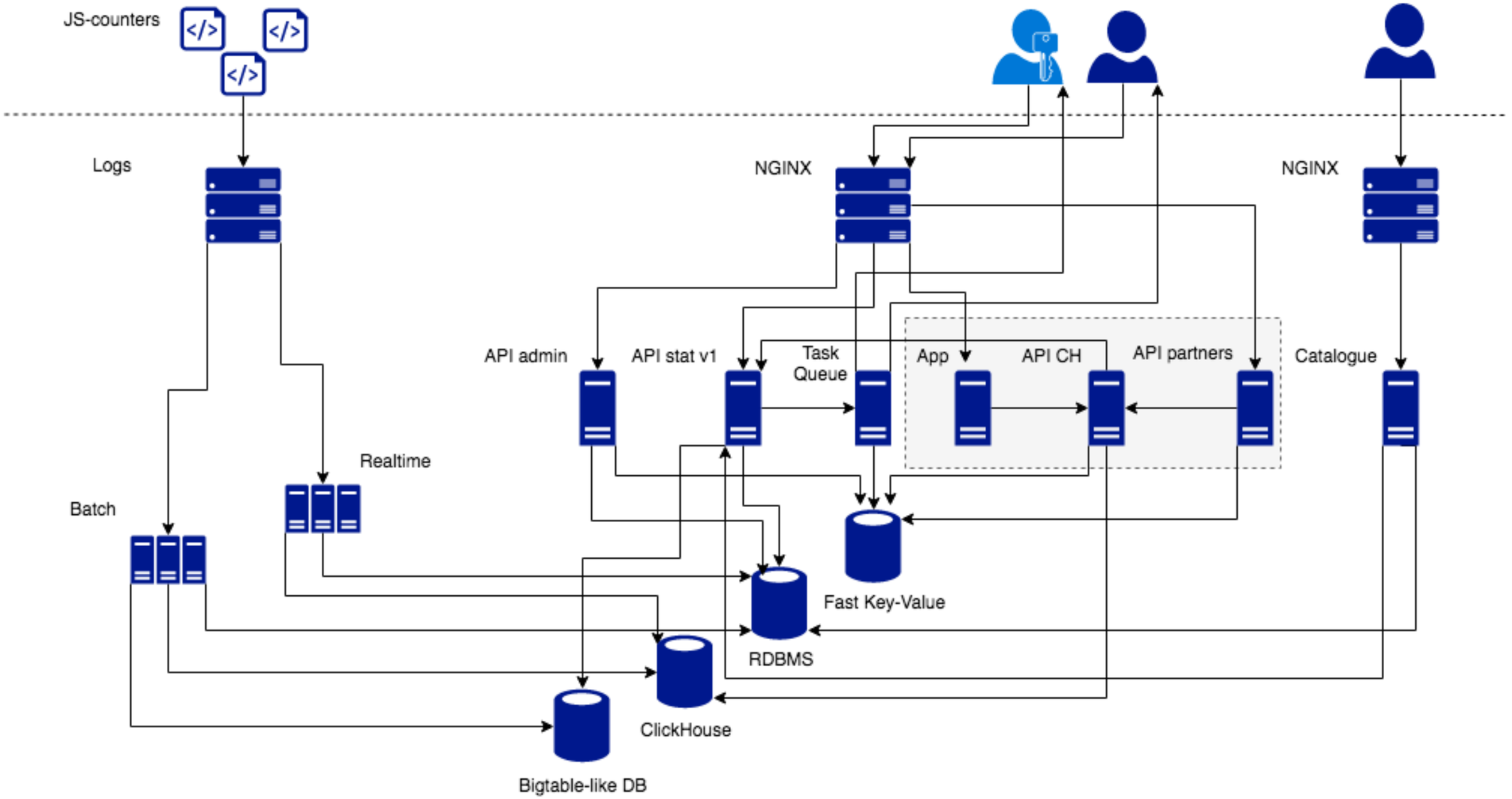
RAMBLER&Co

Разработка API ClickHouse

Виталий Самигуллин, группа разработки технологий Рамблер/топ-100

Введение

- Рамблер/топ-100 — сервис веб-аналитики
- Переход с batch на stream processing —> разработка API ClickHouse



Переход на stream processing

СУБД ClickHouse

1. Столбцовая (широкие таблицы)
2. Быстрая (столбцы, сжатие данных)
3. SQL-like синтаксис языка запросов
4. Широкие возможности трансформации и агрегации (перенос логики из приложения в запрос)
5. Аналитическая

Аналитические запросы

- Преимущественно чтение
- Транзакции не нужны
- Вставка данных пачками
- Данные не изменяются

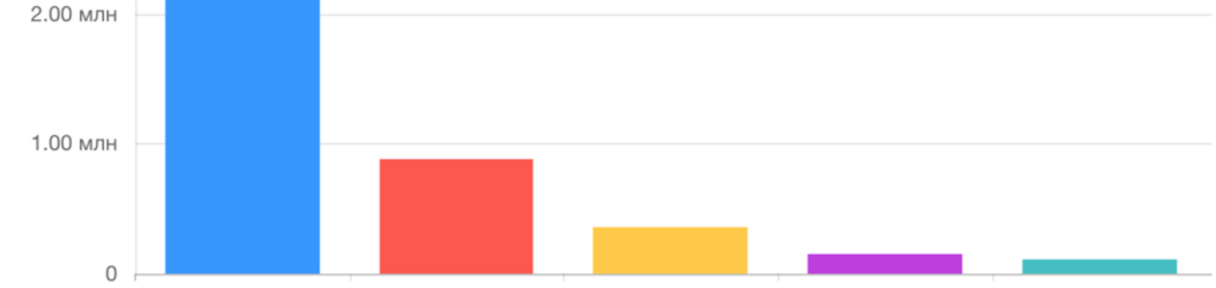
- Широкая таблица, читаются только нужные столбцы
- Хранение неагрегированных данных
- Чтение большого объема данных
- Запрос на основе части данных с приближенным результатом (sample)

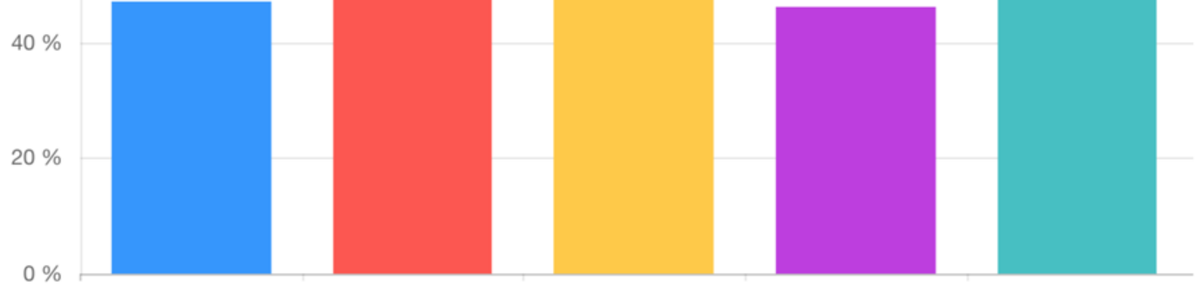
Продуктовый challenge

* Стандартные отчеты (шаблоны запросов, по 2 ручки на отчет)
* Конструктор отчетов (?)

Особенности API ClickHouse

* Единая точка входа (ручка)
* *Почти* ORM (генерация запроса вместо подстановки в шаблон)
* Кубики — сущности для генерации SQL-запросов
  - Размерности (что?)
  - Метрики (сколько?)





Размерности

Метрики

Россия Соединенные Штаты Украина Беларусь Казахстан

Страна, регион, город	Посетители ↓	Отказы	Глубина просмотра
Искать / Сбросить выделенные			
Итого и среднее	5.87 млн	49,14 %	2.75
Россия	3.93 млн	46,87 %	2.86
Соединенные Штаты	882 179	81,89 %	1.4
Украина	361 158	49,17 %	2.72
Беларусь	151 622	46,21 %	2.78
Казахстан	111 476	48,12 %	2.74
Германия	71 824	42,83 %	3.03

Отчет География

Пример кубиков — размерностей и метрик

Кубики

Кастомные отчеты

* Пользователь строит отчет из кубиков — размерностей и метрик

Рамблер/

От запроса к API к SQL-запросу

- Запрос к API
- SQL-запрос
- Как генерируется SQL-запрос (реализация кубиков в Python)
- На примере отчета “Технологии/Операционные системы”

```
{"dimensions": [
    {"name": "counter",
     "filters": [{"op": "eq", "val": 123}]},
    {"name": "os",
     "filters": [{"op": "nlike", "val": "Windows%"}]},
    {"name": "day",
     "filters": [{"op": "eq", "val": "2017-03-22"}]}
  ],
  "metrics": [
    {"name": "visitors",
     "filters": [{"op": "gt", "val": 100}]}
  ],
  "sort": [
      {"name": "visitors", "order": "desc"},
      {"name": "os", "order": "asc"}
  ],
  "offset": 0,
  "limit": 20,
  "sample": 1.0}
```

Кубики

JSON-запрос

* Размерности и их фильтры
* Метрики и их фильтры
* Сортировка
* Офсет/лимит
* Сэмплирование

Рамблер/

```
SELECT
  os_name AS os,
  uniqCombined(user_id) AS visitors
FROM hits
SAMPLE 1
WHERE
  (counter_id = 123) AND
  (dt = toDate('2018-03-22')) AND
  (os NOT LIKE 'Windows%')
GROUP BY
  os
HAVING
  visitors > 100
ORDER BY
  visitors DESC,
  os ASC
LIMIT 0, 20
```

Кубики

От SQL-запроса к кубикам

Размерности:

* Операционная система
* Счетчик
* Дата

Метрики:

* Посетители

Рамблер/

Свойства кубиков

* Колонка в БД
* Alias
* Видимость
* Фильтры
* Сортировка
* Выражения для ключевых слов запроса (SELECT, WHERE, …)

```
class Selectable(object):
    column = not_implemented
    alias = not_implemented

    def __init__(self, visible: bool=True,
        sortable: bool=False) -> None:
        self.visible = visible
        self.sortable = sortable
        self.filters = []
        ...
    def filter(self, operator: str, value: Any) -> None:
        ...
    def sort(self, ascending: bool=False,
        priority: int=0) -> None:
        ...
    @property
    def select(self) -> Optional[str]:
        ...
```

Кубики

Базовый класс

* Список операторов
* Механизм добавления фильтрации
* Механизм сортировки
* Механизм отображения выражения в зависимости от видимости кубика
* @property для необходимых ключевых слов (SELECT, WHERE, GROUP BY, …)

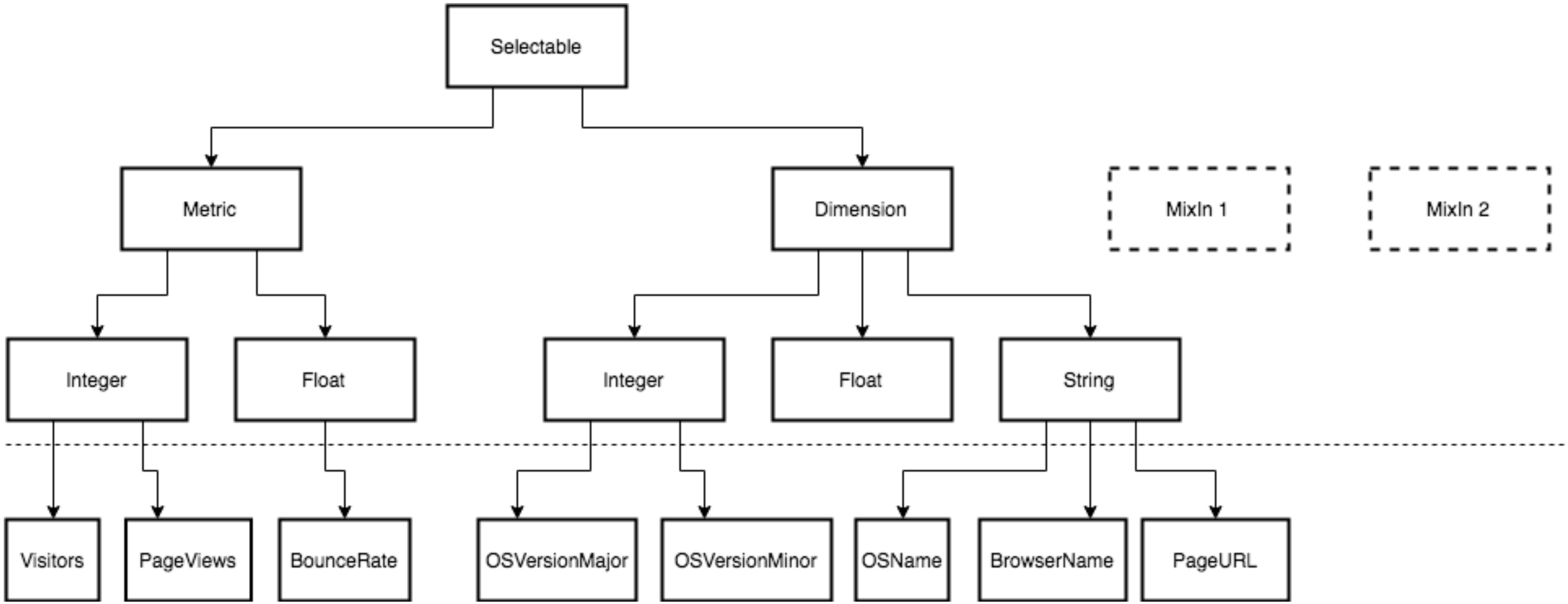
Рамблер/

```
class OSName(StringDimension):
    column = 'os_name'
    alias = 'os'
    functions = []
    …

class Visitors(IntegerMetric):
    column = 'user_id'
    alias = 'visitors'
    functions = [uniqCombined]
    …
```


Кубик как класс в Python

* Наследует от базовых классов
* Использует class variables
* Использует @property для выражений ключевых слов



Иерархия классов

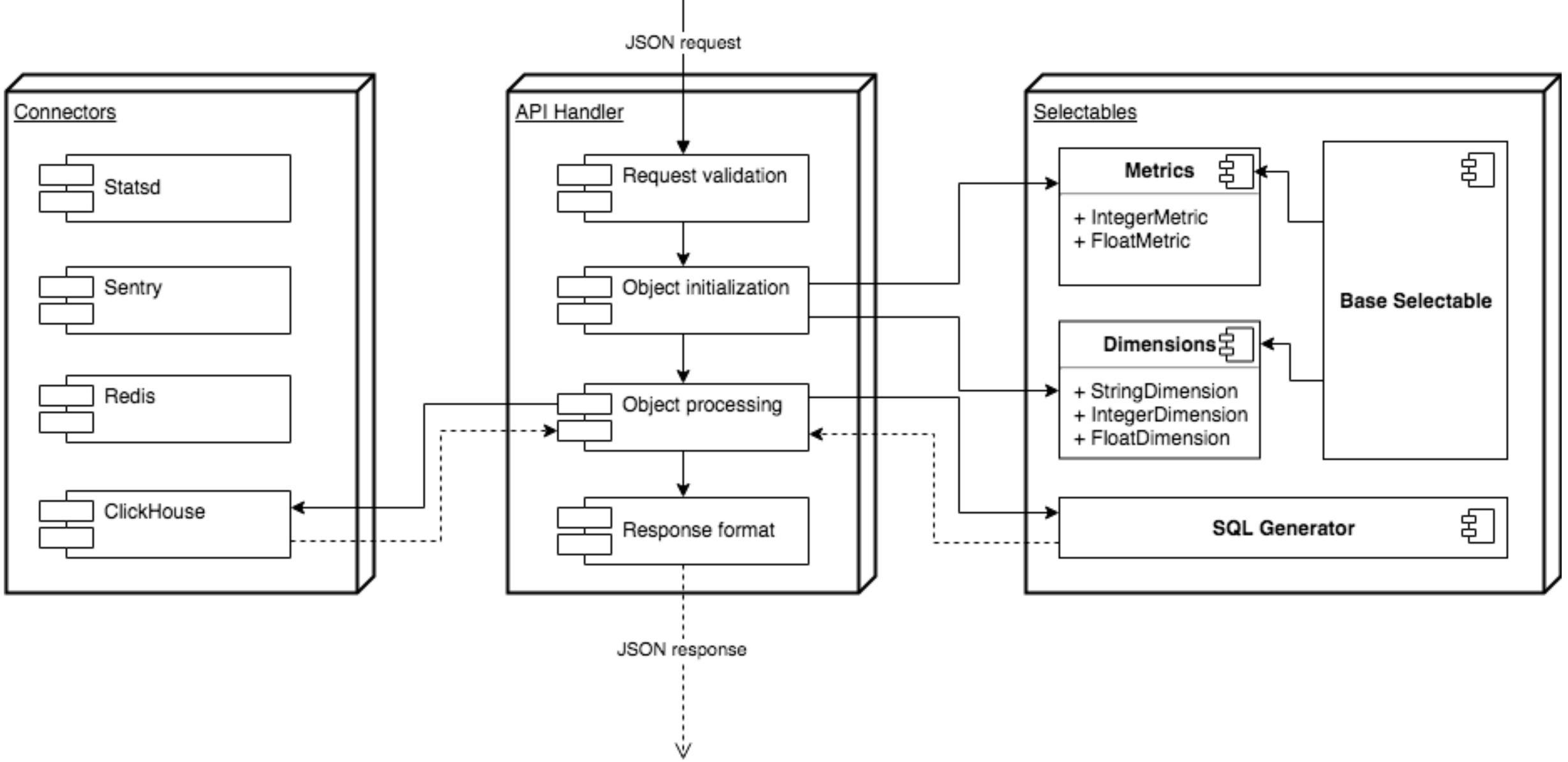
```
generator = SQLGenerator(table=‘hits’,
    sample=0.01, limit=[offset, limit])
generator.add([sel1, sel2, ...])
...
{
    'SELECT': [sel1.select, sel2.select],
    'FROM': 'hits',
    'SAMPLE': 1,
    'WHERE': [sel1.where, sel2.where],
    'GROUP BY': [sel1.groupby,
        sel2.groupby],
    'HAVING': [sel1.having, sel2.having],
    'ORDER BY': [sel1.orderby,
        sel2.orderby],
    'LIMIT': [0, 20],
}
```

Кубики

Генератор SQL-запросов

* Список инициализированных кубиков с фильтрами и сортировками
* Заполнение списков выражений для каждого ключевого слова
* Формирование строк с нужным разделителем (‘,’, ‘AND’)
* Формирование финальной строки SQL-запроса

Рамблер/



Внутреннее устройство API

Разработка API и Python 3.6+

- CPython 3.6.4 (f-string, ordered dict) ⇢ Быстрая разработка
- aiohttp (asyncio) ⇢ Асинхронность
- Type hinting (mypy) ⇢ Статический анализ
- pytest (параметризация) ⇢ Быстрое покрытие тестами

Рамблер/

RAMBLER&Co

Спасибо!

Виталий Самигуллин, группа разработки технологий Рамблер/топ-100

22 марта 2018

Вакансия в Рамблер/топ-100

- Сильный Python-разработчик
- Контакты: hr@rambler-co.ru