



Онлайн-образование

Проверить, идет ли запись!



Меня хорошо видно && слышно?

Ставьте +, если все хорошо

Напишите в чат, если есть проблемы

Elasticsearch



Дмитрий Кириллов

Технический директор

1С-Старт

@esteps_kirillov

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #канал-группы или #general



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара

01. Основы ES



02. Оптимизация индексов



03. Полнотекстовый поиск



04. Эксплуатация

Цели вебинара | После занятия вы сможете

1

Выполнять базовые запросы
для индексации и поиска данных

2

Настраивать индексы ES
для оптимизации поиска и хранения

3

Использовать ES
в составе Elastic Stack

Введение

Давайте познакомимся

- кто вы: разработчик или DBA?
- насколько хорошо вы знаете Elasticsearch?

Установка и инструменты

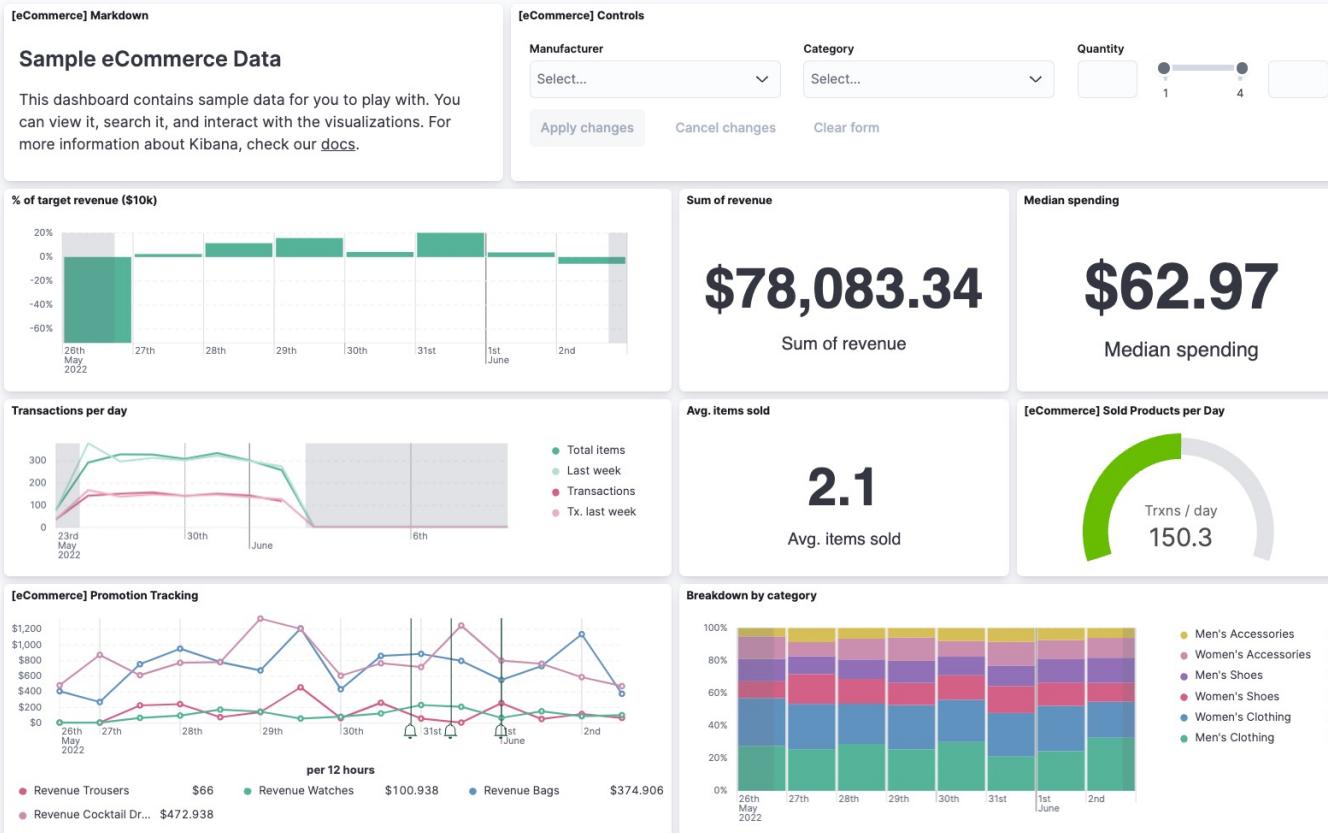
Docker

- на официальном сайте есть чёткие инструкции
- основной момент: не забыть скопировать пароль, который будет сгенерирован при установке
- для работы также потребуется http_ca.crt

Установка и инструменты

Kibana

- является частью Elastic Stack (ELK)
- работает в браузере
- интерфейс красивый, но не очень интуитивный



01. Основы ES

Основы ES

Особенности ES

- документо-ориентированная БД
- стек – Java + Apache Lucene
- взаимодействие – REST API + JSON
- гибкая конфигурация типов данных
(от автоматического вывода до строгих ограничений)
- огромные возможности поиска,
в т.ч. полнотекстового, в т.ч. на русском языке

Для чего можно использовать Elasticsearch

- сбор, хранение и анализ логов
- работа с метриками, в том числе бизнес-метриками
- полнотекстовый поиск по документам
- работа с картами и гео-данными

Компоненты Elastic Stack

Beats

Агенты – поставщики данных

Logstash

Парсер + буфер событий

Elasticsearch

Непосредственно база данных

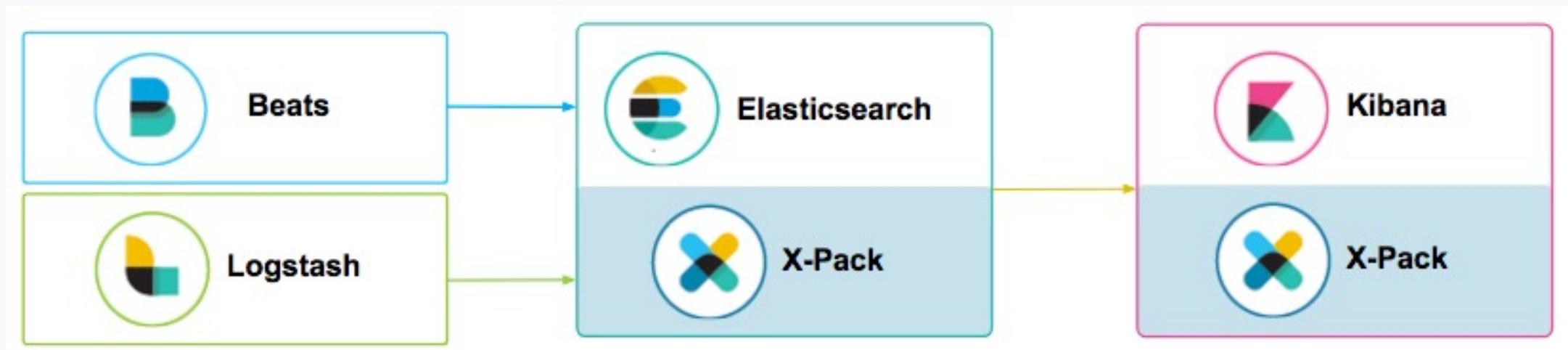
Kibana

Визуализация и анализ данных

X-Pack

Безопасность, оповещения, отчёты, ML...

Компоненты Elastic Stack



Ключевые концепции

Индекс

База данных

Документ

Запись в индексе

Mapping

Типы данных для конкретных полей документов

Вопросы на засыпку:

Elasticsearch написан на Java.

Какие минусы вы в связи с этим видите?

- работа в виртуальной машине
- сборщик мусора
- риски OutOfMemory



Кейс: интернет-магазин

Вы открыли ликёро-водочный интернет-магазин и хотите добавить поиск и фильтрацию по товарам.

```
{  
    "title": "Агдам",  
    "sku": "342-001",  
    "category": "Портвейн",  
    "price": 150,  
    "volume": 0.7,  
    "stock": [{ "shop": "Мира", "stock": 30 },  
              { "shop": "Ленина", "stock": 0 }]  
}
```

Создание нового индекса

```
PUT /otus-shop
```

- только нижний регистр + символы пунктуации
- в теле запроса можно передать настройки + mapping

Добавление нового документа в индекс

```
POST /otus-shop/_doc  
POST /otus-shop/_create/<_id>
```

- `_doc` присваивает автоматический id
- `_create` требует указания уникального id

По умолчанию в БД добавляются:

- поля документа по отдельности
- плюс его исходный JSON (`_source`)

Вопросы на засыпку:

Как вы думаете, нужно ли нам присваивать конкретные ID для товаров интернет-магазина?

Почему?

Получение документа из индекса

```
GET /otus-shop/_doc/<_id>
GET /otus-shop/_source/<_id>
```

- _doc возвращает мета-данные
- _source возвращает только исходный JSON
- если документ объёмный, можно запросить только часть полей из исходного JSON

Обновление документа в индексе

```
POST /otus-shop/_update/<_id>
{
  "doc": {
    "price": 2500
  }
}
```

- после обновления документ будет переиндексирован
- можно обновить только часть полей документа

Удаление документа из индекса

```
DELETE /otus-shop/_doc/<_id>
```

Вопросы на засыпку:

- вы начали выгружать **полную** базу товаров из 1С в Elasticsearch через REST API
- эта операция будет выполняться раз в день по расписанию
- какие подводные камни могут возникнуть?
- какие способы решения этих проблем вы знаете?

Массовое добавление/изменение документов

```
POST /_bulk
{ "create" : { "_index" : "otus-shop", "_id": "342-001" } }
{ ...JSON... }
{ "update" : { "_index" : "otus-shop", "_id": "342-001" } }
{ ...JSON... }
{ "delete" : { "_index" : "otus-bulk", "_id": "342-001" } }
```

- в запросе чередуются команды/данные
- на стороне ES есть доп.оптимизации для ускорения обработки
- кол-во строк в одном запросе подбирается опытным путём

Поиск: основы

- основной инструмент поиска – Query DSL
- простые запросы можно комбинировать в составные
- по умолчанию в результате поиска возвращаются исходные документы
- для каждого документа вычисляется score:
его релевантность поисковому запросу

Язык запросов **не очень очевидный**, с ним нужна практика.

Основы Elasticsearch

Поиск: все результаты

```
GET /otus-shop/_search
{
  "query": {
    "match_all": { }
  }
}
```

Основы Elasticsearch

Поиск: полнотекстовый поиск

```
GET /otus-shop/_search
{
  "query": {
    "match": {
      "title": "красное вино"
    }
  }
}
```

— запросы по умолчанию **регистронезависимые**

Поиск: точное совпадение

```
GET /otus-shop/_search
{
  "query": {
    "term": {
      "category": "Портвейн"
    }
  }
}
```

- запросы по умолчанию **регистрозависимые**
- если тип поля был выведен автоматически как text,
нужно использовать <field>.keyword

Основы Elasticsearch

Поиск: диапазон

```
GET /otus-shop/_search
{
  "query": {
    "range": {
      "price": {
        "gte": 1200,
        "lt": 1300
      }
    }
  }
}
```

— операции gt / gte / lt / lte комбинируются через AND

Основы Elasticsearch

Поиск: составные запросы

```
GET /otus-shop/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title": "красное" } }
      ],
      "filter": [
        { "range": { "price": { "gte": 9950 } } }
      ]
    }
  }
}
```

Поиск: составные запросы

must	Обязательно	Влияют на score
must not	Обязательно	Влияют на score
filter	Обязательно	Не влияют на score
should	Желательно	Влияют на score

Поиск: запросы и фильтры

filter

Попадает ли документ под условия запроса?

query

Насколько документ соответствует запросу?
(влияет на score)

Вопрос на засыпку:

Предложите сценарий поиска в интернет-магазине, когда одновременно потребуются и filter, и query?

Поиск: агрегация данных

```
GET /otus-shop/_search
{
  "query": {
    "match_all": {}
  },
  "aggs": {
    "min_volume": {
      "min": { "field": "volume" }
    },
    "max_price": {
      "max": { "field": "price" }
    }
  }
}
```

Поиск: выборка конкретных полей

```
GET /otus-shop/_search
{
  "query": {
    "match_all": {}
  },
  "_source": false,
  "fields": [
    "price"
  ]
}
```

Поиск: выборка только результатов вычислений

```
GET /otus-shop/_search
{
  "query": {
    "match_all": {}
  },
  "size
```

02. Оптимизация индексов

Как устроены реляционные БД

- у нас есть базы данных
- в базах есть таблицы
- в таблицах есть поля
- у каждого поля есть тип
- эти типы привязываются ко всей **таблице**

Вопрос на засыпку:

А зачем нужны эти типы?

Не проще ли хранить всё как строки?

Как устроен Elasticsearch

- у нас есть индексы
- в индексах есть документы
- у документов есть поля
- у каждого поля есть тип
- эти типы привязываются ко всему **индексу**
- такая привязка называется mapping

Оптимизация индексов

Что происходит при добавлении документа

- ES смотрит, есть ли в документе поля, для которых ещё не был указан mapping*
- если такие поля есть, для них **автоматически** выводится наиболее подходящий тип данных
- другими словами, нам не обязательно **заранее** описывать типы данных и поля для индекса

* если это самый первый документ в индексе – тогда анализируются все его поля

Оптимизация индексов

Вопросы на засыпку

- у нас есть список документов
- мы начинаем их поочерёдно добавлять в индекс
- ES выводит типы на основе самого первого документа из списка
- что может пойти не так?

Конфигурация mapping-а вручную

- можно заранее подобрать оптимальные типы данных для полей документов
- это можно сделать, только если таких полей ещё нет в индексе
- если mapping для поля уже установлен, изменить его **нельзя***
- основной способ решения проблемы — переиндексация

* но можно поменять некоторые параметры

Оптимизация индексов

Получение mapping-a

```
GET /otus-shop/_mapping
```

– результат можно брать за основу для своего варианта

Оптимизация индексов

Создание индекса с конкретным mapping-ом

```
PUT /otus-shop
{
  "mappings": {
    "title": { "type": "text" },
    "sku": { "type": "text" },
    "category": { "type": "keyword" },
    "price": { "type": "short" }
    ...
  }
}
```

Оптимизация индексов

Основные типы данных

Keyword	Теги, ID, HTTP-коды...	Точный поиск
Число	Целое / с плав.точкой	Точный / диапазон
Дата	Как строка / timestamp	Точный / диапазон
Текст	На любом языке	Полнотекстовый

Тип данных: ключевое слово (Keyword)

- типовой сценарий – когда одно и то же слово встречается во множестве документов*
- примеры: теги, категории, коды ошибок...
- при поиске всегда проверяется полное совпадение

* т.е. у ключевого слова низкая кардинальность

Оптимизация индексов

Тип данных: число (Numeric)

- доступны типы от 8 до 64 бит
- рекомендуется проанализировать данные и подобрать **наименьший** тип данных
- переполнение приведёт к ошибке,
документ не будет проиндексирован

Параметры, общие для скалярных типов данных

index	Будем ли мы искать по этому полю
doc_values	Будем ли мы сортировать по этому полю
ignore_malformed	Нужно ли отклонять весь документ, если поле "битое"
store	Нужно ли хранить это поле как дополнение к _source

Оптимизация индексов

Параметр `dynamic`

<code>true</code>	Новые поля добавляются в mapping
<code>runtime</code>	Новые поля загружаются из <code>_source</code> при поиске
<code>false</code>	Новые поля не добавляются в mapping
<code>strict</code>	Новые поля приводят к отклонению документа

Во всех случаях новые поля можно добавлять в mapping вручную.

Практика: разработка mapping-а

- 1** У нас есть документы для товаров с конкретными полями
- 2** Ваша задача: помочь подобрать для них подходящие типы данных
- 3** Критерии оценки: оптимизация размера и скорости поиска



Тайминг: 5–10 минут

Оптимизация индексов

Практика: разработка mapping-a

```
{  
    "title": "Агдам",  
    "sku": "342-001",  
    "category": "Портвейн",  
    "price": 150,  
    "volume": 0.7,  
    "stock": [{ "shop": "Мира", "stock": 30 },  
              { "shop": "Ленина", "stock": 0 }]  
}
```

Оптимизация индексов

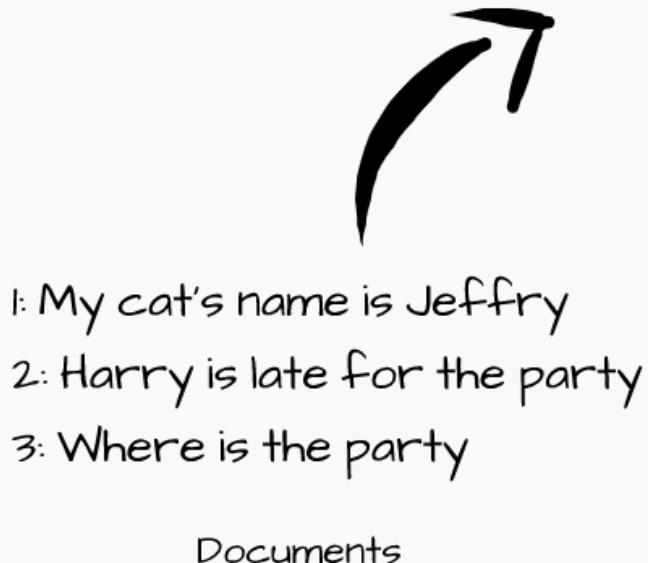
Проблема: медленный поиск по диапазону

- сценарий: пользователи часто ищут товары в разных ценовых категориях

Оптимизация индексов

Под капотом ES: Inverted Index

- берём term
- сразу видим список документов
- асимптотика близка к $O(1)$

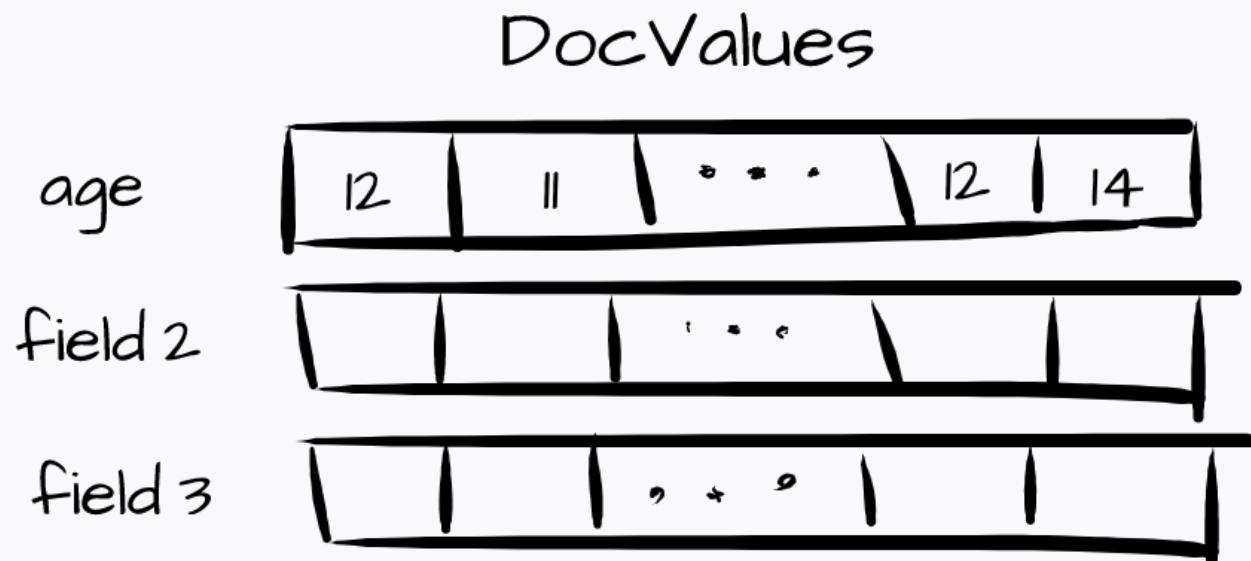


Inverted Index

Оптимизация индексов

Под капотом ES: DocValues

- берём value
- ищем в колонке индекс его документа
- занимают меньше места, чем index
- асимптотика $O(\log N) \dots O(N)$



Профилирование запросов

```
GET /otus-shop/_search
{
  "profile": true,
  "query": {
    "range": {
      "price": { "gte": 1200 }
    }
  }
}
```

Оптимизация индексов

Решение: денормализация + keywords

```
{  
    "title": "Агдам",  
    "sku": "342-001",  
    "category": "Портвейн",  
    "price": 150,  
    "price_range": "100-200",  
    "volume": 0.7,  
    "stock": [{ "shop": "Мира", "stock": 30 },  
              { "shop": "Ленина", "stock": 0 }]  
}
```

Оптимизация индексов

Проблема: поиск по вложенным массивам

```
GET /otus-shop/_search
{
  "query": {
    "bool": {
      "filter": [
        { "match": { "stock.shop": "Мира" } },
        { "range": { "stock.stock": { "gte": 15 } } }
      ]
    }
  }
}
```

Оптимизация индексов

Что происходит с вложенными массивами?

Исходник:

```
"stock": [{ "shop": "Мира", "stock": 30 },  
          { "shop": "Ленина", "stock": 0 }]
```

Результат:

```
"stock.shop": [ "Мира", "Ленина" ],  
"stock.stock": [ 30, 0 ]
```

В результате невозможно понять,
какие остатки относятся к какому магазину.

Оптимизация индексов

Решение: Nested Type

```
{  
  "mappings": {  
    ...  
    "stock": {  
      "type": "nested",  
      "properties": {  
        "shop": { "type": "keyword" }  
        "stock": { "type": "short" }  
      }  
    }  
    ...  
  }  
}
```

Оптимизация индексов

Поисковый запрос для Nested Type

```
GET https://localhost:9200/otus-shop/_search
{
  "query": {
    "nested": {
      "path": "stock",
      "query": {
        "bool": {
          ...
        }
      }
    }
  }
}
```

Шаблоны индексов (Index Templates)

- можно заранее создать шаблоны для типовых задач
- шаблон автоматически применяется,
если он соответствует имени нового индекса
(например, `otus-*`)
- если изменить шаблон, это **не повлияет**
на существующие индексы

03. Полнотекстовый поиск

Полнотекстовый поиск

Кейс: поиск по вопросам к юристу

Вы открыли сайт для юридических консультаций,
и вам нужен полнотекстовый поиск по вопросам пользователей

```
{  
  "content": "Дайте консультацию, как лучше всего  
  открыть ИП с двумя учредителями,  
  правда второй уже ООО, спасибо!"  
}
```

Полнотекстовый поиск

Поиск: полнотекстовый поиск

```
GET /otus-consulting/_search
{
  "query": {
    "match": {
      "content": "000"
    }
  }
}
```

Полнотекстовый поиск

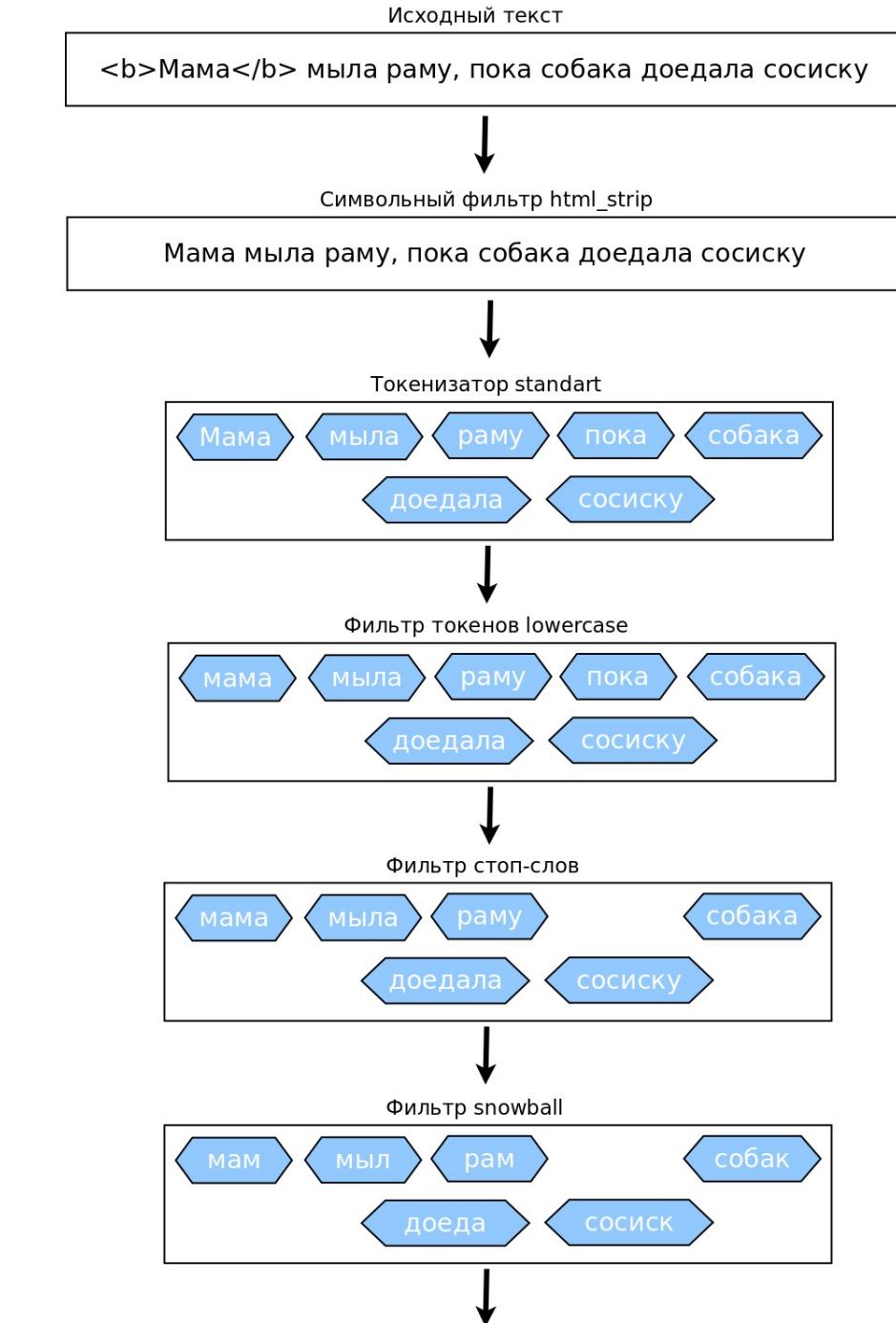
Проблема: поддержка русского языка

```
GET /otus-consulting/_search
{
  "query": {
    "match": {
      "content": "консультация"
    }
  }
}
```

Полнотекстовый поиск

Elasticsearch: анализ текста

- исходный текст последовательно проходит несколько этапов обработки
- для русской морфологии нужно явно указать фильтры
- поисковые запросы проходят через этот же набор этапов



Полнотекстовый поиск

Решение: анализаторы

```
{  
  "settings": {  
    "analysis": {  
      ...  
      "analyzer": {  
        "my_russian": {  
          "filter": [ "lowercase", "ru_stop", "ru_stemmer" ]  
        }  
      }  
    }  
  }  
}
```

Полнотекстовый поиск

Проблема: опечатки

```
GET /otus-consulting/_search
{
  "query": {
    "match": {
      "content": "кансультация"
    }
  }
}
```

Редакционное расстояние

- у нас есть два слова
- они отличаются друг от друга
- мы можем посчитать, сколько правок нужно сделать в одном из них, чтобы превратить его во второе
- это называется "Расстояние Дамерау–Левенштейна"

	M	O	N	K	E	Y	
	0	1	2	3	4	5	6
M	1	0	1	2	3	4	5
O	2	1	0	1	2	3	4
N	3	2	1	0	1	2	3
E	4	3	2	1	1	1	2
Y	5	4	3	2	2	2	1

Полнотекстовый поиск

Решение: нечёткий поиск

```
GET /otus-consulting/_search
{
  "query": {
    "match": {
      "content": {
        "query": "кансультация",
        "fuzziness": "auto"
      }
    }
  }
}
```

— значение fuzziness можно настроить под себя

Полнотекстовый поиск

Проблема: аббревиатуры и синонимы

```
GET /otus-consulting/_search
{
  "query": {
    "match": {
      "content": "индивидуальный предприниматель"
    }
  }
}
```

Полнотекстовый поиск

Решение: база синонимов

```
{  
  "settings": {  
    "analysis": {  
      "filter": {  
        "my_synonym_filter": {  
          "type": "synonym",  
          "synonyms": [  
            "ИП, индивидуальный предприниматель, ПБОЮЛ"  
          ]  
        }  
      }  
    }  
  }  
}
```

04. Эксплуатация

Особенности Elasticsearch

- повышенное потребление ресурсов
- требуются быстрые и ёмкие диски
- не гарантируется согласованность данных
- не рекомендуется использовать ES в качестве основной БД

Кластер Elasticsearch: виды узлов

data	Хранят данные
ingest	Выполняют пред-обработку новых документов
master	Управляют кластером
client	Маршрутизируют запросы

Кластер Elasticsearch: шардирование и репликация

- индекс можно разделить на нужное кол-во шардов
- шарды делятся на основные (primary) + реплики
- мы можем управлять их размером и количеством
- чтобы изменить кол-во шардов для существующего индекса, требуется создать новый через Split API / Reindex

Кластер Elasticsearch: резервное копирование

- ES позволяет создавать snapshot-ы
- они хранятся в репозитории за пределами кластера
- для экономии места выполняется дедупликация
- рекомендуется дополнительно делать резервные копии файловой системы кластера

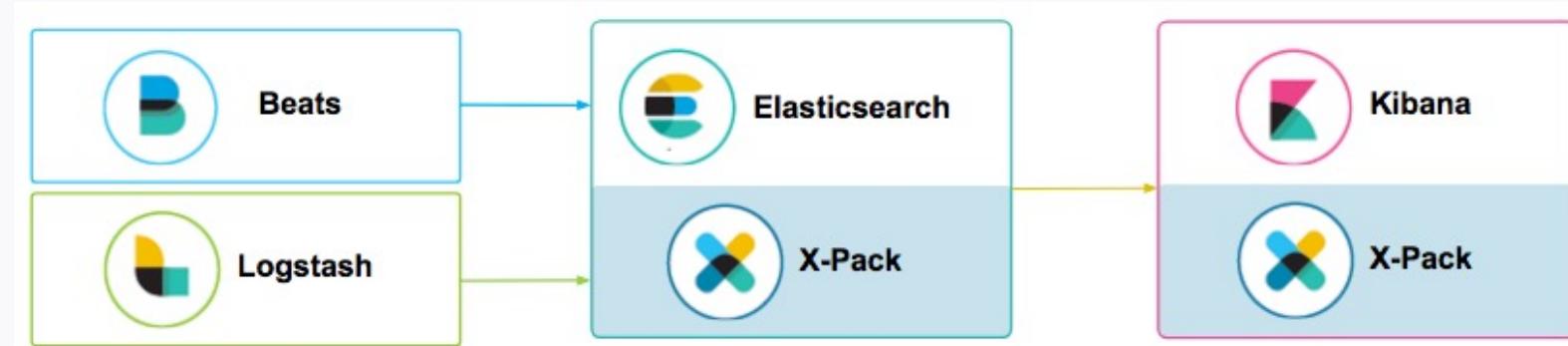
Кластер Elasticsearch: мониторинг и статистика

```
GET /_cluster/health  
GET /_cluster/stats  
GET /_nodes/stats
```

Эксплуатация

Elastic Stack: концепция

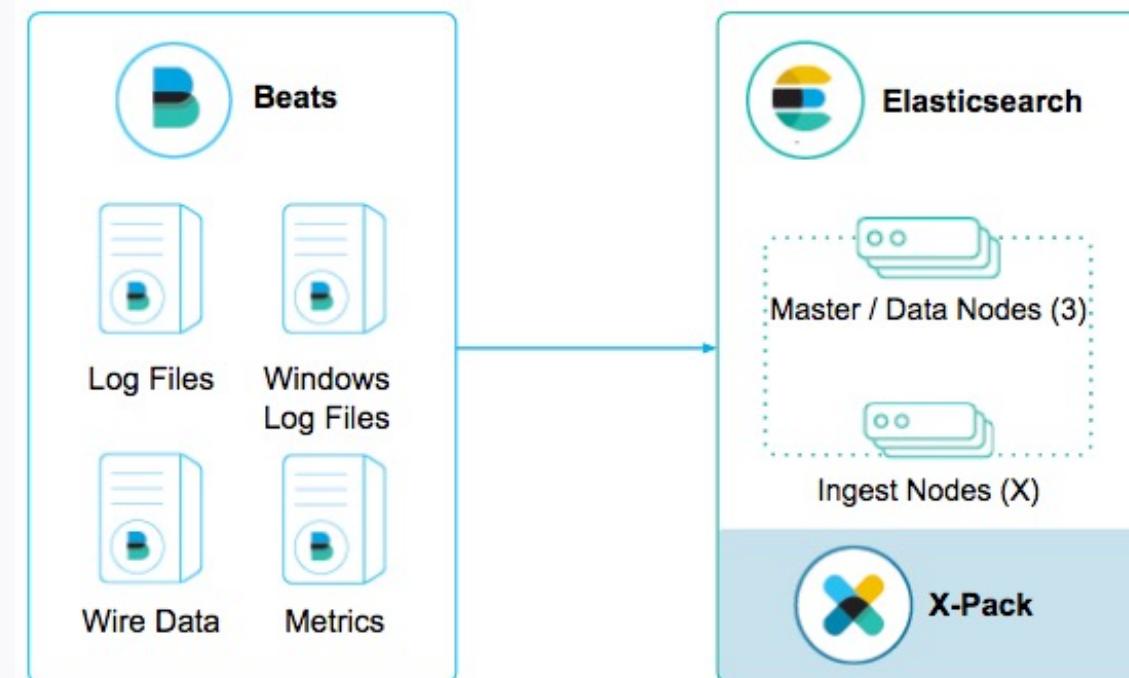
- Beats и Logstash поставляют данные
- ES работает как БД
- Kibana работает как front-end



Эксплуатация

Elastic Stack: базовый вариант

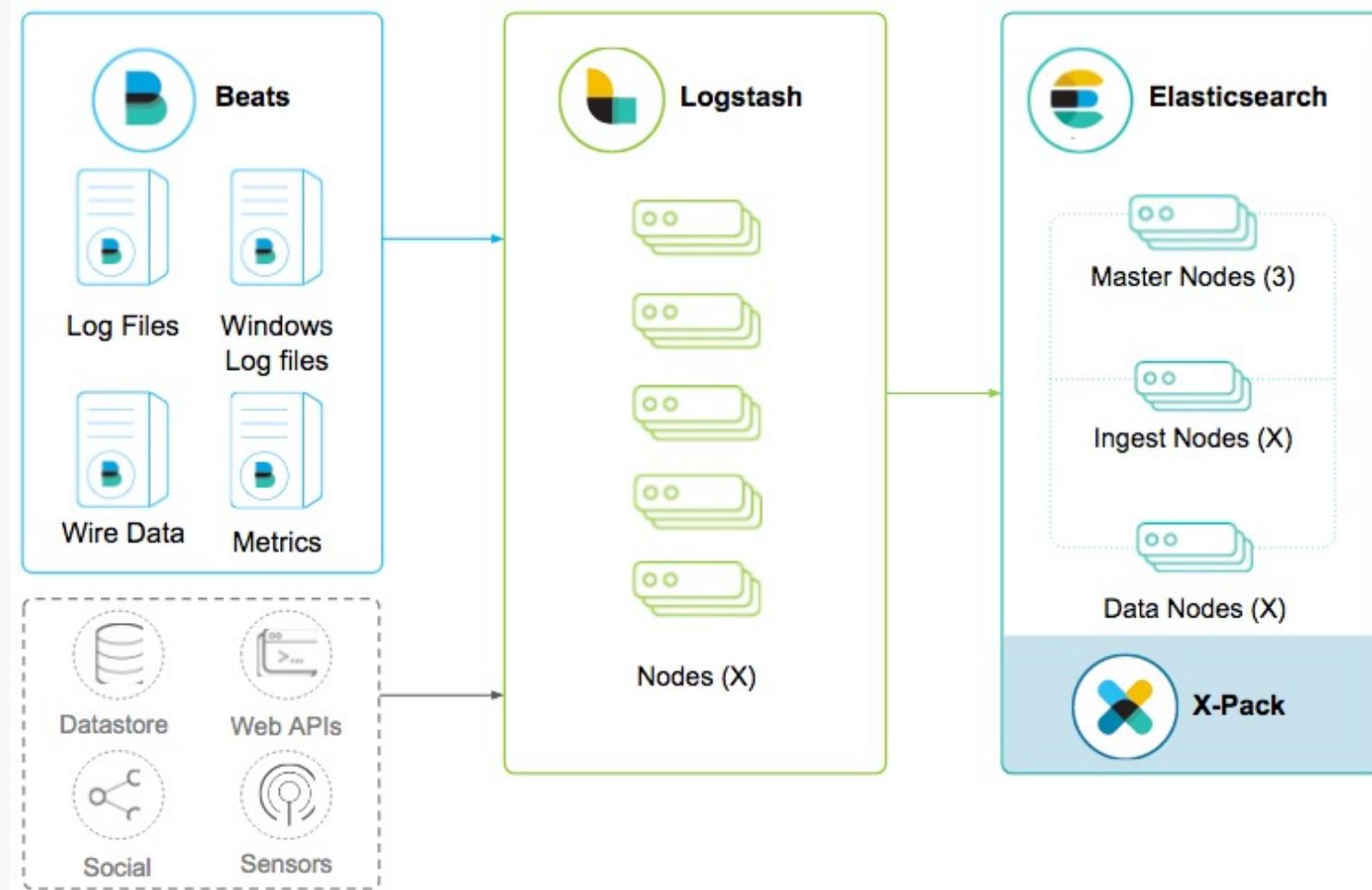
- Beats пишет данные напрямую в ES
- ES нормализует и обогащает данные (процессорами)



Эксплуатация

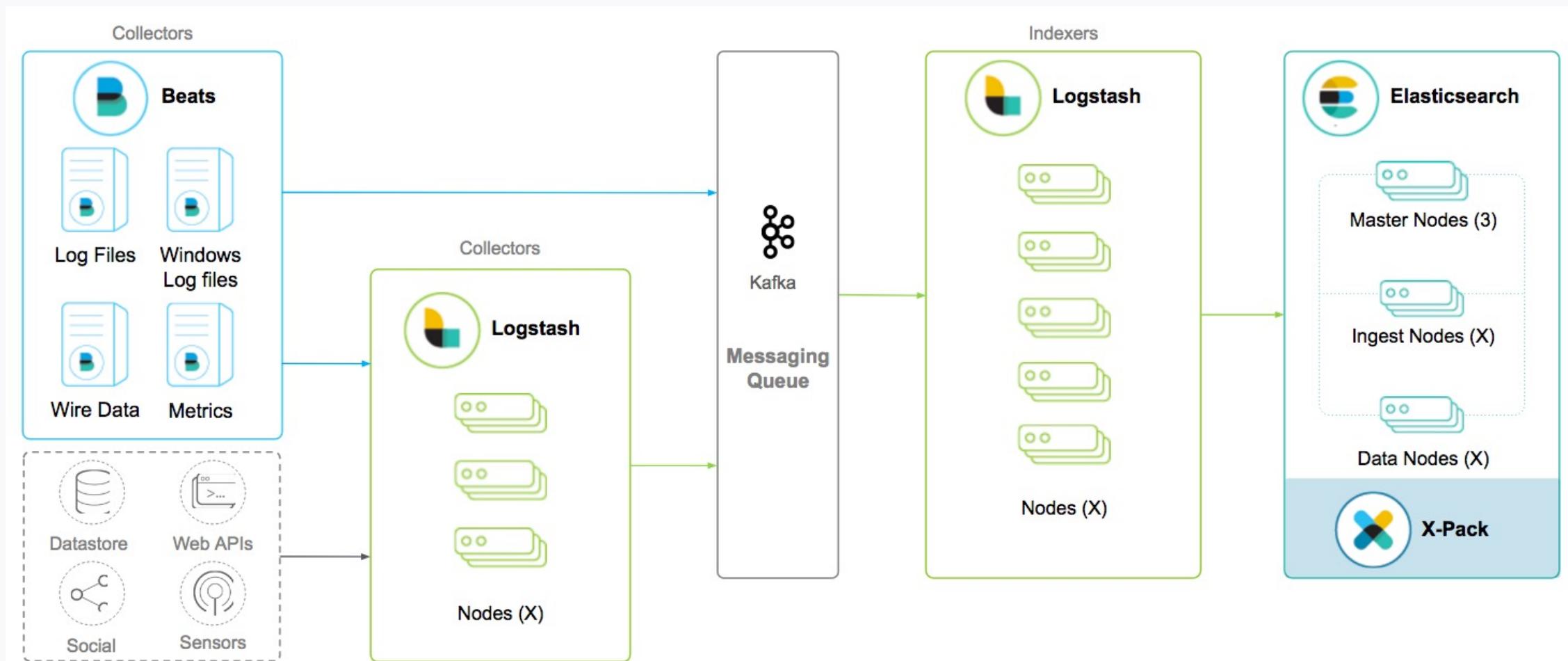
Elastic Stack: гибкий вариант

- разные источники данных пишут в LS в своих форматах
- LS десериализует и обогащает данные (плагинами)
- на стороне LS работает персистентный буфер



Эксплуатация

Elastic Stack: разделение индексации и сбора данных



Цели вебинара | Проверка достижения целей

- 1 Выполнять базовые запросы для индексации и поиска данных
- 2 Настраивать индексы ES для оптимизации поиска и хранения
- 3 Использовать ES в составе Elastic Stack

Домашнее задание

- 1 Развернуть ES и добавить в него индекс с текстовыми документами
- 2 Написать запрос для нечёткого поиска с русской морфологией
- 3 Детали в ЛК



Срок: желательно сдать до 20.12

Следующий вебинар

Тема: Cassandra: monitoring and problem solving



Дата: 14 декабря, среда, в 20:00



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию
в ЛК – можно
изучать



Обязательный
материал обозначен
красной лентой



Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие вебинары



Дмитрий Кириллов

Технический директор
1С-Старт
[@esteps_kirillov](https://www.instagram.com/@esteps_kirillov)