Если вы видите что-то необычное, просто сообщите мне.

Databases

- Команды для работы с бд neo4j
- Новая страница
- Подключение Kafka к PostgreSQL
- Потоковая Передача данных PostgreSQL + Kafka + Debezium: часть 1
- Работа с elasticsearch

Команды для работы с бд neo4j

Создание записи

```
CREATE (user:User {<<Ключ>>:<<Значение>>,})
```

Получение всех пользователей

```
MATCH (user:User)
```

Получение одного или несколько пользователей по какому либо значению

```
MATCH (user:User {<<Ключ>>: <<Значение>>})
MATCH (user_parent:User {<<Ключ>>: <<Значение>>})
MATCH (user_children:User {<<Ключ>>: <<Значение>>})
```

Создание связи между пользователями

```
CREATE (user_children)-[:ИМЯ СВЯЗИ(Указывается капсом)]->(user_parent)
```

Получение детей у пользователя

```
MATCH p=(user:User {Ключ:'Значение'})<-[:ИМЯ СВЯЗИ*]-(:User) RETURN p
```

Получение родителей у пользователя

```
MATCH p=(user:User {Ключ:'Значение'})-[:ИМЯ СВЯЗИ*]->(:User) RETURN p
```

Выгрузка из файла

LOAD CSV WITH HEADERS FROM "ТУТ ПУТЬ К ФАЙЛУ(Если он внутри neo4j то указывается через file:// можно брать с сервера через http/https" AS row MERGE (user:User {{Ключ: row.<<значение из файла>>}})

`MERGE` работает как `get_or_create`(Тоесть если он нашел в данном случае пользователя то будет использовать его, если не нашел он его создаст)
В случае выгрузки из файла, не получилось найти старые объекты. Создавал новые

Изменение какого-либо поля у пользователя(ей) сохраняет сам

```
MATCH (user:User)
SET user.<<имя ключа>> = 10 + 10
```

Для того, что бы проверить почему тормозит запрос и с целью оптимизации, перед запросом указать `PROFILE` внутри neo4j
Детально опишет запрос.

```
CREATE INDEX ON :<Имя>(<Ключ>) <-- Создает индекс в бд
CALL db.indexes <-- Выведет индексы которые созданы
```

Новая страница

Когда на слейве делаешь su - postgres -c "su - postgres -c "pg_basebackup --host=... --username=... --pgdata=/var/lib/postgresql/data ..." ты указываешь pgdata - путь куда будут писаться бекапы мастера и он должен соответствовать вот тому, что я там написал (/var/lib..), т.к. это главная папка постгри, откуда он читает. Так вот. Перед тем, как выполнять эту команду, папку надо почистить. Причем эта папка неявно доступа по энв переменной $PGDATA и я соответственно так и делал (я уж умный дохуя) rm -r $PGDATA/data/* , и он ругался на resource busy и прочее, надо бы из под докера делать su, делать rm -r, и после этого выполнять команду на бекап и все заебись

Еще если вдруг понадобится тебе на будущее - ставь в postgresql.conf wal_keep_segments не нулевой, побольше (он в МБайтах) - размер хранимого журнала WAL - журнал записей транзакций наскок я понял (если ты знаешь лучше то ок), так вот, если он нулевой, то если в мастере есть записи, а слейв стартануть позже - то WAL обнулится и будет ошибка кароче, вот как-то так

Подключение Kafka к PostgreSQL

Инструкция поможет вам взять на себя ответственность без проблем и без потери эффектисновсти. Цель статьи в создании процесса экспорта данных настолько гладко, насколько это возможно.

В конце статьи вы сможете успешно подключать Kafka к PostgreSQL, плавно передавать данные потребителю по выбору, для полноценного анализа в реальном времени. В дальнейшем это повзолит пострить гибкий ETL(дословно «извлечение, преобразование, загрузка») конвеер для вашей организации. Из стати вы узнаете более глубокое понимание инструментов и техник и таким образом оно поможет вам отточить ваши умения дальше.

Требования

Для лучшего пониманиния статьи, требуется понимание следующего списка тем:

- Знания PostgreSQL.
- Знания Kafka
- Kafka и PostgreSQL установленны на хосте.

Введение в Kafka

Apache Kafka это продукт с открытым исходным кодом, который помогает публиковать и подписываться на большие по объему сообщения в распределенной системе. Kafka использует идею лидер-последователь, позволяя пользователя копировать сообщения в независимые от падения, и в дальнейшем позволядет делить и хранить сообщения в Kafka топиках в зависимости от темы сообщения. Kafka позволяет настраивать в реальном времеи потоки данных и приложения для изменения данных и потоков от источника к цели.

Ключевые особенности Kafka:

- Масштабируемость: Kafka имеет исключительную масштабируемость и может быть отмасштабированно без времени простоя.
- Изменение данных: Kafka предлагает KStream и KSQL(в случае Confluent Kafka) для изменению данных на лету.
- Отказоустойчивость: Kafka использует посредников для копирования данных и постоянства данных, для создания отказоустойчивых систем.
- Безопасность: Kafka может быть объеденина с различными метриками безопасности такими как Kerberos, для передачи информации конфиденциально.
- Производительность: Kafka распределенна, разделена и имеет очень высокую пропускную способность для публикации и подписки на сообщения.

Для более подробного описания, можно обратиться на официальный сайт разработчиков Kafka

Введение в PostgreSQL.

PostgreSQL это мощное, производственного класса, с открытым исходным кодом СУБД которая использует стандартные SQL запросы связанных данных и JSON для запросов несвязанных данных хранящихся в базе данных. PostgreSQL имеет отличную поддержку для всех операционных систем. Он поддерживает расширенные типы данных и оптимизацию операций, которые можно найти в комерческих проектах каа Oracle, SQL Server и т.д.

Ключевые особенности PostgreSQL:

- Имеет расширенную поддержку для сложных запросов.
- Предоставляет отличную поддержку для географических объектов и следовательно он может быть использован для географической информационной системы и сервисе на основе положения.
- Предоставляет поддержку для клиент-серверной сетевой технологии
- Упреждающая журнализация(write-ahead-logging (WAL)) позвляет быть базе данных отказоустойчивой.

Для большей информации по PostgreSQL, можно посмотреть официальный вебсайт.

Процесс ручной настройки Kafka и PostgreSLQ интеграции

Kafka поддерживает подключение с PostgreSQL и различными другими базами данных с помощью различных встроенных подключений. Эти коннекторы помогают передавать данные от источника в Kafka и затем передать потоком в целевой сервис с помощью выбора топиков Kafka. Так же, есть множество подключений для PostgreSQL, которые помогают установить подключение к Kafka.

1. Установка Kafka

Чтобы подключить Kafka к PostgreSQL, для начала нужно скачать и установить Kafka.

2. Старт Kafka, PostgreSQL и Debezium сервер

Confluent предоставляется пользователям с различным набором встроенных подключений которые действуют как источники и сток данных, и помогает пользователям передавать их данные через Kafka. Один из таких подключений/образов которые позволяют подключать Kafka к PostgreSQL - Debezium PostgreSQL Docker образ.

Чтобы установить Debezium Docker который поддерживает. подключение к PostgreSQL с Kafka, обратимся к официальному проекту Debezium Docker и склониурем проект на нашу локальную систему.

Как только вы склонировали проект вам нужно запустить Zookeper сервис который хранит настройки Kafka, настройки топиков, и упревление нодами Kafka. Это всё запускается следующей командой:

docker run -it --rm --name zookeeper -p 2181:2181 -p 2888:2888 -p 3888:3888 debezium/zookeeper:0.10

Теперь с работающим Zookeper, вам нужно запустить Kafka сервер. Чтобы сделать это откройте консоль и выполните следующую команду:

docker run -it --rm --name kafka -p 9092:9092 --link zookeeper:zookeeper debezium/kafka:0.10

Как только вы запустили Kafka и Zookeeper, теперь запускаем PostgreSQL сервер, его мы будем подключать к Kafka. Это можно выполнить следующей командой:

docker run — name postgres -p 5000:5432 debezium/postgres

Теперь стартуем Debezium. Для этого выполним следующую команду:

docker run -it — name connect -p 8083:8083 -e GROUP_ID=1 -e CONFIG_STORAGE_TOPIC=my-connect-configs -e OFFSET_STORAGE_TOPIC=my-connect-offsets -e ADVERTISED_HOST_NAME=$(echo $DOCKER_HOST | cut -f3 -d’/’ | cut -f1 -d’:’) — link zookeeper:zookeeper — link postgres:postgres — link kafka:kafka debezium/connect

Как только вы запустили все эти сервера, логинимся в командную оболочку PostgreSQL используя следующие команды

psql -h localhost -p 5000 -U postgres

3. Создаем базу данных в PostgreSQL

Как только вы вошли в PostgreSQL, вам необходимо создать базуданных. Для примера если вы хотите создать базуданных с именем `emp`, вы можете использовать следующую команду:

CREATE DATABASE emp;

В готовой базе, создадим таблицу, которая будет хранть информацию. Для этого выполним:

CREATE TABLE employee(emp_id int, emp_name VARCHAR);

Теперь нужно добавить данные или несколько записей в таблицу. Для этого выполните выполните команды как указано ниже:

INSERT INTO employee(emp_id, emp_name) VALUES(1, 'Richard') INSERT INTO employee(emp_id, emp_name) VALUES(2, 'Alex') INSERT INTO employee(emp_id, emp_name) VALUES(3, 'Sam')

Таким образом вы можете создать postgreSQL базу данных и вставить в неё значение, для того чтобы настроить подключение между Kafka и PostgreSQL.

4. Поднятие подключения Kafka-PostgreSQL

Как только вы настроили PostgreSQL базу данных, вам нужно поднять Kafka-Postgres подключение, которое позволить вам тянуть данные из PostgreSQL в Kafka топик. Для этого вы можете создать Kafka подключение используя следующий скрипт:

curl -X POST -H “Accept:application/json” -H “Content-Type:application/json” localhost:8083/connectors/ -d ‘ { “name”: “emp-connector”, “config”: { “connector.class”: “io.debezium.connector.postgresql.PostgresConnector”, “tasks.max”: “1”, “database.hostname”: “postgres”, “database.port”: “5432”, “database.user”: “postgres”, “database.password”: “postgres”, “database.dbname” : “emp”, “database.server.name”: “dbserver1”, “database.whitelist”: “emp”, “database.history.kafka.bootstrap.servers”: “kafka:9092”, “database.history.kafka.topic”: “schema-changes.emp” } }’

Чтобы проверить что подключение прошло успешно воспользуйтесь командой:

curl -X GET -H “Accept:application/json” localhost:8083/connectors/emp-connector

Для того, чтобы проверить что Kafka получил данные из PostgreSQL или нет, нужно ключить Kafka Console Consumer, используя следующую команду:

docker run -it -name watcher -rm - link zookeeper:zookeeper debezium/kafka watch-topic -a -k dbserver1.emp.employee

Команда выше теперь отобразит вашу базу данных PostgreSQL в консоли. После того как убедимся что данные полученны в Kafka верно, можно воспользоваться KSQL/KStream или Spark поток для произвдения действий ETL над данными.

Потоковая Передача данных PostgreSQL + Kafka + Debezium: часть 1

В этой инструкции мы будем использовать Postgres, Kafka< Kafka Connect, Debezium и Zookeeper для создание маленького api, который отслеживает магазины и крипто попуки во времени.

Введение

Платформа потоковой передачи данных, как Kafka позволяет вам строить ситемы которые обрабатывают данные в реальном времени. Эти системы имеют мириады случаем использования, проекты могут ранжироваться от простой обработки данных для ETL систем до проектов требующих высокую скорость координации микросервисов требуемое все виды Kafka как приемлимое решение.

Один из моих любимых примеров использования Kafka происходит от New Relic инженерный блог. New Relic помогает разработчикам отслеживать производительность их приложений. Их свойства работают в реальном времени, что может быть важно так как множество разработчиков полагаются на него в качестве системы опвещение, когда что-то идет не так. New Relic сереьзно использует Kafka для координирования микросервисов и связывать их в реальном времени друг с другом.

В то время как Kafka слишком сложна в качестве простого инструменка, который мы собираемся построить сегодня, эта инструкция покажет как настроить Kafka. Мы не будем делить наш API на множество сервисов, но мы будем использовать Kafka внутри нашего сервиса что того, чтобы просчитать и создать допольнительные данные доступные через API.

Что такое Kafka?

Kafka очень мощный платформа потока событий, которая позволяет обрабатывать массинвый набор данных в реальном времени. В добавок, можно сказать, kafka масштабируемы и отказоустойчива, делает её популярным выбором для проектов которые требуют скорость обработки данных.

Что такое Debezium?

Реляционная SQL база данных в сердце бесчетного количество программный проектов. Для примена, если вы хотите использовать Kafka, но часть (или всё) ваших данных существует в Postgres базе данных, Debezium - это инструмент который подключается к Postgres и потоковым образом передает данные в Kafka. Запускается на сервере с базой данных.

Что такое Zookeper?

ZooKeeper - еще один кусок програмного обеспечения от Apache, который использует Kafka для хранения и управления конфигурацией. Для базовой настройки, которую мы будем использовать не требуется глубокое понимание Zookeeper.

Если вы уже закончили установку проекта как этот в боевом окружении, вы захотите узнать гораздо больше о том, как оно работает и как его настроить. В будущем, Kafka не потребует Zookeeper.

Что такое Kafka Connect?

Kafka Connect работает как мост для входящих и исходищх потоковых данных. Вы можете подключить вашу Kafka к различным источникам баз данных. В этой иснтрукции, мы будем использовать для подключения Debezium, Postgres, но это будет не единственный источник данных для которых Connect может быть полезен. Есть бесконечное количество

коннекторов написанных для того, чтобы манипулировать различными данными в Kafka.

Так же экосистема Kafka может быть полезна, вы сможете получить большую отдачу отдачу от Kafka в последствии если вложитесь в Kafka:

Использвоание Docker для настройки Postgres, Kafka и Debezium

Эта инструкция будет состоять из несколких частей. Первая, мы настроим маленкий API сервер, который позволит вам хранить записи. Затем, используя данные цен, покупок/продаж, данные будут проходить через Kafka и расчитывать различные общие метрики. Мы так же поэкспериментируем используя Debezium `sink` для потока данных из Kafka обратно в SQL базу даннхы.

В этой части мы поднимем и запустим Kafka и Debezium. В конце инструкции, у вас будет проект который передает потоковым образом события из таблицы в топик Kafka.

Мы будем исполльзовать Docker и docker-compose чтобы помочь нам запустить Postgres, Kafka и Debezium. Если вы не знакомы с этими инструментами, возможно будет полезно прочитать про инструменты прежде чем продолжить.

Созадим Postgres контейнера с помощь Docker

Первое, настроим базвый Postgres контейнер.

version: '3.9'

services: db: image: postgres:latest ports: - "5432:5432" environment: - POSTGRES_PASSWORD=arctype

После запуска docker-compose, мы должны иметь рабочую базу данных

db_1 | 2021-05-22 03:03:59.860 UTC [47] LOG: database system is ready to accept connections

Теперь, проеверим, что она работает.

$ psql -h 127.0.0.1 -U postgres Password for user postgres:

postgres@postgres=#

После подключения нас приветствует psql консоль.

Добавим Debezium Kafka, Kafka Connect, и Zookeeper образы

Теперь добавим другие образы необходимые для Kafka. Debezium предлагет образы Kafka, Kafka Connect и Zookeeper, которые предназначены специально для работы с Debezuim. Поэтому использовать мы будем их.

version: '3.9'

services: db: image: postgres:latest ports: - "5432:5432" environment: - POSTGRES_PASSWORD=arctype

zookeeper: image: debezium/zookeeper ports: - "2181:2181" - "2888:2888" - "3888:3888"

kafka: image: debezium/kafka ports: - "9092:9092" - "29092:29092" depends_on: - zookeeper environment: - ZOOKEEPER_CONNECT=zookeeper:2181 - KAFKA_ADVERTISED_LISTENERS=LISTENER_EXT://localhost:29092,LISTENER_INT://kafka:9092 - KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=LISTENER_INT:PLAINTEXT,LISTENER_EXT:PLAINTEXT - KAFKA_LISTENERS=LISTENER_INT://0.0.0.0:9092,LISTENER_EXT://0.0.0.0:29092 - KAFKA_INTER_BROKER_LISTENER_NAME=LISTENER_INT

connect: image: debezium/connect ports: - "8083:8083" environment: - BOOTSTRAP_SERVERS=kafka:9092 - GROUP_ID=1 - CONFIG_STORAGE_TOPIC=my_connect_configs - OFFSET_STORAGE_TOPIC=my_connect_offsets

- STATUS_STORAGE_TOPIC=my_connect_statuses depends_on: - zookeeper - kafka

Настройки переменного окружения для Kafka позволяют нам указать различные сети и протоколы безопаснотсти если у вашей сети есть различные правила для внутреннего брокера подключения в отличии от внешних клиентов подключающихся к Kafka. Наша простая настройка самостоятельна с созданное сетью внутри Docker.

Kafka Connect создает топик в Kafka и использует их для хранения настроек. Вы можете указать имя, которое он будет использовать для топик с переменными окружением. Если у вас есть множетсво Kofka Connect нод, они могут выполнять работу паралельно когда они имеют одну и ту же `GROUP_ID` и `_STORAGE_TOPIC` потоковые события PostgreSQL

Создадим таблицу чтобы проверить потоковые события.

create table test (id serial primary key, name varchar);

Настроим Debezium Connector для PostgreSQL.

Если мы запустим наш Docker проект, Kafka, Kafka Connect, Zookeeper и Postgres он прекрасно работает. Однако, Debezium требует конкретной настройки коннектора для запуска потоковых данных от Postgres.

Совместный SQL редактор

Прежде чем мы активируем Debezium, нам нужно подготовить Postgres сделав необольшие конфигурационные изменения. Debezium использует нечто встроенное в POstgres, под названием WAL, или упреждающую журнализацию. Postgres использует этот лог чтобы проверить целостность данных и управлять версиями ячеек и транзакций. WAL в Postgres имеет несколько режимов, которые можно настроить, и для работы Debezium WAL режим должен быть указан как `replica`. Давайте это настроим.

psql> alter system set wal_level to 'replica';

Возможно понадобится рестарт Postgres контейнера для применения настройки.

Есть еще один плагин Postgres не включенный в образ который мы используем, поэтому нам понадобится wal2json. Debezium может работать и с wal2json и с protobuf. Для этой инструкции, мы будем использовать wal2json. Так как он согласно имени переводит Postgres

WAL лог в JSON формат.

С помощью запущенного Docker, в ручном режиме установим wal2json исполльзуя aptitude. Чтобы добраться до косноли Postgres контейнера, для начала найдем ID контейнера и выполним следующий набор команд:

$ docker ps

CONTAINER ID IMAGE c429f6d35017 debezium/connect 7d908378d1cf debezium/kafka cc3b1f05e552 debezium/zookeeper 4a10f43aad19 postgres:latest

$ docker exec -ti 4a10f43aad19 bash

Теперь, когда мы внутри контейнера давайте поставим wal2json:

$ apt-get update && apt-get install postgresql-13-wal2json

Активируем Debezium

Мы можем общаться с Debezium делая HTTP запросы. Для этого нужен POST запрос данные которого отформатированны в JSON формате. JSON определяет параметры коннектора который мы пытаемся создать. Поместим данные в файл и будем его использовать с `cURL`.

У нас есть несколько конфигурационных опций на данный момент. Тут можно использовать белый или черный списки если вы хотите чтобы Debezium отображал только определенные таблицы(или для избежания определенных таблиц)

$ echo ' { "name": "arctype-connector", "config": { "connector.class": "io.debezium.connector.postgresql.PostgresConnector", "tasks.max": "1", "plugin.name": "wal2json", "database.hostname": "db", "database.port": "5432", "database.user": "postgres", "database.password": "arctype", "database.dbname": "postgres", "database.server.name": "ARCTYPE", "key.converter": "org.apache.kafka.connect.json.JsonConverter", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter.schemas.enable": "false", "value.converter.schemas.enable": "false", "snapshot.mode": "always" } } ' > debezium.json

Теперь можно отправить эту конфигурацию в Debezium

```
$ curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json" 127.0.0.1:8083/connectors/ --data "@debezium.json"
```

Ответ должен быть со следующим содержанием JSON если это уже не настроенный коннектор.

```
{ "name": "arctype-connector", "config": { "connector.class": "io.debezium.connector.postgresql.PostgresConnector", "tasks.max": "1", "plugin.name": "wal2json", "database.hostname": "db", "database.port": "5432", "database.user": "postgres", "database.password": "arctype", "database.dbname": "postgres", "database.server.name": "ARCTYPE", "key.converter": "org.apache.kafka.connect.json.JsonConverter", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter.schemas.enable": "false", "value.converter.schemas.enable": "false", "snapshot.mode": "always", "name": "arctype-connector" }, "tasks": [], "type": "source" }
```

Проверим настройку потоковой передачи Kafka

Теперь после вставки обновления или удаления записей мы будем использовать изменения как новое сообщение в Kafka топике связанной с таблицей. Kafka Connect создаст 1 топик для SQL таблицы. Чтобы проверить что всё работает верно, нам нужно мониторить Kafka топик.

Kafka идет с shell скриптами которые помогают вам вставлять ваши настройки Kafka. Это удобно когда вы хотите проверить вашу конфигурацию и её удобно включать в Docker образ который мы используем. Первый, который мы будем использовать список всех топиков в нашем Kafka кластере. Давайте запустим и проверим что мы видим топик для нашей `test` таблицы.

```
$ docker exec -it $(docker ps | grep arctype-kafka_kafka | awk '{ print $1 }') /kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --list ARCTYPE.public.test __consumer_offsets
```

my_connect_configs my_connect_offsets my_connect_statuses

Встроенный в инструмент Kafka требует указания `--bootstrap-server`. Он ссылается на bootstrap потому, что вы обычно запускаете Kafka как кластер с несколькими нодами, и вам нужно один из них, который "выставлен наружу" чтобы зайти в кластер. Kafka обрабатывает все остальное самостоятельно.

Вы можете увидеть нашу `test` таблицу в списке `ARCTYPE.public.test.` Первая часть, ARCTYPE - это префикс который мы настроили для `[database.server.name](http://database.server.name)` поле в настройках JSON. Вторая часть отражает схему Postgres таблицы в ней, в последней части название таблицы. При добавлении Kafka производителей и приложений с потоковыми данными, количество топиков будет увеличиваться, поэтому удобно указывать префиксы, чтобы проще идентифицировать какой из топиков относится к таблице в бд.

Теперь монжо использовать другой инструмент называемым консольый потребитель для слежения за топиками в реальном времени. Называется он "console consumer" потому, что это типа потребителя kafka - утилита которая постребляет сообщения из топика и что-нибудь делает с ним. Потребитель может делать что угодно с данными которые он потребяет и консоль потребителя ничего не делает кроме как выодит эти сообщения в консоль.

$ docker exec -it $(docker ps | grep arctype-kafka_kafka | awk '{ print $1 }') /kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic ARCTYPE.public.test

По умолчанию, консольный потребитель, потребляет только сообщения у него уже не было. Если вы хотите увидеть все сообщения в топике нужно добавить ключ `--from-beginning` в команду запуска .

Теперь наш потребитель следить за новыми сообщенямии в топике, а мы запустим `INSERT` и посмотрим вывод.

postgres=# insert into test (name) values ('Arctype Kafka Test!'); INSERT 0 1

Вернемся к нашему Kafka потребителю:

$ docker exec -it $(docker ps | grep arctype-kafka_kafka | awk '{ print $1 }') /kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic ARCTYPE.public.test ... { "before":

null, "after": { "id": 8, "name": "Arctype Kafka Test!" }, "source": { "version": "1.5.0.Final", "connector": "postgresql", "name": "ARCTYPE", "ts_ms": 1621913280954, "snapshot": "false", "db": "postgres", "sequence": "["22995096","22995096"]", "schema": "public", "table": "test", "txId": 500, "lsn": 22995288, "xmin": null }, "op": "c", "ts_ms": 1621913280982, "transaction": null }

В месте с мета данными вы можете увидеть главный ключ поля `name` записки которую вы добавили.

Выводы

Давайте скоординируемся, так как мы имеем Postgres для передачи данных в Kafka кластер. Во второй части, мы построим SQL схему чтобы улучшить наше приложение, для вычисления данных.

Работа с elasticsearch

Создание индекса

```
curl -u ПОЛЬЗВАТЕЛЬ:ПАРОЛЬПОЛЬЗОВАТЕЛЯ -X GET "IPАДРЕС:5002/samples/"
```

Добавление

```
curl -u ПОЛЬЗВАТЕЛЬ:ПАРОЛЬПОЛЬЗОВАТЕЛЯ -XPOST --header 'Content-Type: application/json'
IPАДРЕС:5002/sample/\_doc -d '{
   "school" : "asdasaTEST", "@timestamp" : "'"$(date +%Y-%m-%dT%H:%M:%S)"'"
}'
```

Измнение

```
curl -u ПОЛЬЗВАТЕЛЬ:ПАРОЛЬПОЛЬЗОВАТЕЛЯ -XPUT --header 'Content-Type: application/json'
IPАДРЕС:5002/samples/\\\_doc/4 -d '{

   "school" : "asaTEST", "@timestamp" : "'"$(date +%Y-%m-%dT%H:%M:%S)"'"

}'
```

> Вот несколько распространненых примеров команд ElasticSearch используя `curl` ElasticSearch часто сложен. Тут мы постараемся сделать его легче.

Удаление индексов.

Ниже индекс назван *sample*

```
curl -X DELETE 'http://localhost:9200/samples'
```

Показать все индексы

```
curl -X GET 'http://localhost:9200/\_cat/indices?v'
```

Показать все докуменыт в индексах

```
curl -X GET 'http://localhost:9200/sample/\_search'
```

Запрос используя параметры URL.

Тут мы используем Lucene запрос формат для написания: *q=school:Harvard*

```
curl -X GET http://localhost:9200/samples/\_search?q=school:Harvard
```

Запрос с JSON aka DSL для запросов в Elasticsearch.

Вы можете использовать параметры для URL. Но вы можете так же использоват JSON, как показано в следующем примере. JSON будет легче для чтения и отладки, когда у вас сложный запрос, чем один длинный запрос в виде URL.

```
curl -XGET --header 'Content-Type: application/json' http://localhost:9200/samples/\_search -d
'{
      "query" : {
        "match" : { "school": "Harvard" }
    }
}'
```

Показать список индексов.

Все поля индексов. Выведет все поля и их типы в каждом индексе.

```
curl -X GET http://localhost:9200/samples
```

Добавить данные

```
curl -XPUT --header 'Content-Type: application/json' http://localhost:9200/samples/\_doc/1 -d
'{
   "school" : "Harvard"
}'
```

Обновление документа.

Вот как добавить поле к существующему документу. Для начала создадим его, затем обновим.

Копирование

```
curl -XPUT --header 'Content-Type: application/json' http://localhost:9200/samples/\_doc/2 -d
'
{
    "school": "Clemson"
}'
```

```
curl -XPOST --header 'Content-Type: application/json'
http://localhost:9200/samples/\_doc/2/\_update -d '{
"doc" : {
               "students": 50000}
}'
```

Бэкап для индкса.

```
curl -XPOST --header 'Content-Type: application/json' http://localhost:9200/\_reindex -d '{
  "source": {
    "index": "samples"
  },
  "dest": {
    "index": "samples\_backup"
  }
}'
```

Объем загруженных данных в формате JSON:

```
export pwd="elastic:"

curl --user $pwd  -H 'Content-Type: application/x-ndjson' -XPOST
'https://58571402f5464923883e7be42a037917.eu-central-1.aws.cloud.es.io:9243/0/\_bulk?pretty'
--data-binary @<file>
```

Показать здоровье кластера

```
curl --user $pwd  -H 'Content-Type: application/json' -XGET
https://58571402f5464923883e7be42a037917.eu-central-
1.aws.cloud.es.io:9243/\_cluster/health?pretty
```

Сбор

Для nginx веб сервера это произведет подсчет пользователей по городам.

```
curl -XGET --user $pwd --header 'Content-Type: application/json'
 https://58571402f5464923883e7be42a037917.eu-central-
1.aws.cloud.es.io:9243/logstash/\_search?pretty -d '{
        "aggs": {
             "cityName": {
                    "terms": {
                     "field": "geoip.city\_name.keyword",
                                "size": 50

        }
   }
  }
}'
```

Это расширит на код ответа количества городов в nginx логах веб сервера.

```
curl -XGET --user $pwd --header 'Content-Type: application/json'
 https://58571402f5464923883e7be42a037917.eu-central-
1.aws.cloud.es.io:9243/logstash/\_search?pretty -d '{
        "aggs": {
          "city": {
```

```
                "terms": {
                        "field": "geoip.city\_name.keyword"
                },
        "aggs": {
          "responses": {
                "terms": {
                     "field": "response"
                 }
           }
         }
      },
      "responses": {
                "terms": {
                     "field": "response"
                 }
        }
   }
}'
```

Использование ElasticSearch с базовой авторизацией.

Если у вас включена безопасность в ElasticSearch, тогда вам необходимо предоставить пользователя и пароль, как показано ниже для всех команд-запросов:

```
curl -X GET 'http://localhost:9200/\_cat/indices?v' -u elastic:(password)
```

Красивый вывод.

Добавьте ?pretty=true к любому поиску чтобы вывести причесаный JSON:

```
 curl -X GET 'http://localhost:9200/(index)/\_search'?pretty=true
```

Запрос на получение только определенных полей.

Вернет только определенные поля поместив их в массив *_source*

```
GET filebeat-7.6.2-2020.05.05-000001/\_search
 {
    "\_source": \["suricata.eve.timestamp","source.geo.region\_name","event.created"\],
    "query":      {
        "match" : { "source.geo.country\_iso\_code": "GR" }
    }
}
```

Запрос по дате.

В случае когда поле типа дата вы можете использовать математику дат:

```
GET filebeat-7.6.2-2020.05.05-000001/\_search
 {
    "query": {
        "range" : {
            "event.created": {
                "gte" : "now-7d/d"
            }
        }
}
}
```