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

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.

Revision #1
Created 2022-10-13 14:14:30 UTC by gasick
Updated 2022-10-13 14:14:51 UTC by gasick