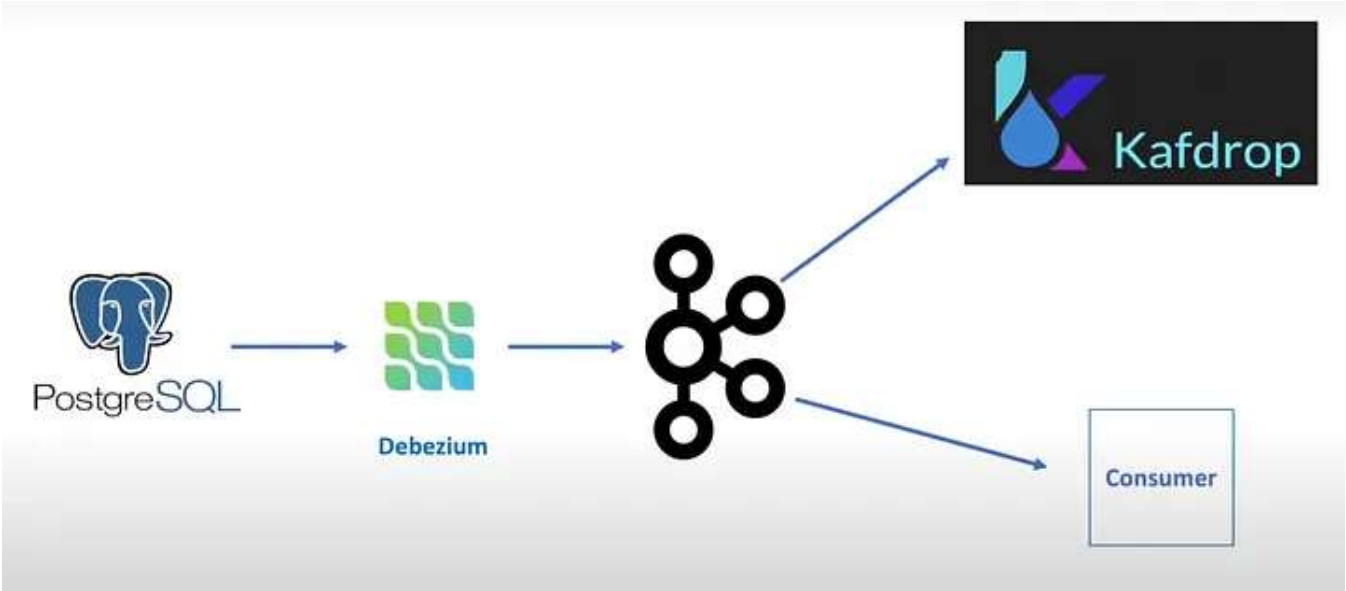
Run Debezium Kafka Kafka | Zookeeper | K Docker Compose

Cloud_Geek · Follow
4 min read · Dec 29, 2022

Listen Share

In this article, we will see how Data Engineering teams Capture Data Change events and how data is published to kafka topics etc.



Step1: Provision Docker containers

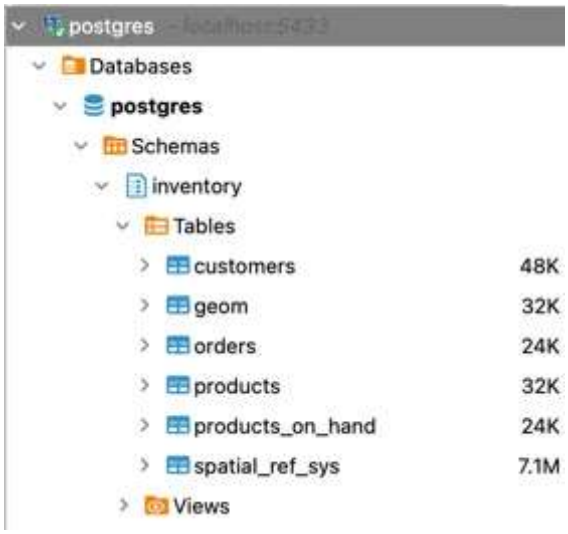
We will provision below 6 (4+2) docker containers using a docker-compose file.

Zookeeper: ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. We will bring up zookeeper node on standard 2181 port, TICK_TIME is 2 seconds which is the heartbeat for zookeeper. Ports 2888 and 3888 are used by zookeeper peers to talk to each other.

Kafka Cluster: A Kafka cluster is a system that consists of several Brokers, Topics, and Partitions for both. The key objective is to distribute workloads equally among replicas and Partitions. We will provision 2 kafka nodes first node runs on port 9092 and second on 9093. Internal listener ports are 29092, 29093 and external are 9092, 9093 . This article explains more about kafka listeners.

Kafdrop: Kafdrop is a web UI for viewing Kafka topics and browsing consumer groups. The tool displays information such as brokers, topics, partitions, consumers, and lets you view messages. We will run kafdrop on default 9000 port.

Postgres: We need a database for which we want to capture Data Change Events. We will use *debezium/example-postgres:1.9* image to provision postgres database since it has got some default schema & tables created. Postgres docker runs on port 5433



Default schema & tables in postgres docker image

Debezium Connector/Kafka Connect: The Debezium PostgreSQL connector captures row-level changes in the schemas of a PostgreSQL database. This container runs on port 8083.

docker-compose-debezium.yaml:

```
version: '3.3'
services:
  postgres:
    image: debezium/example-postgres:1.9
    container_name: postgres
    ports:
      - 5433:5432
```

```
      environment:
        - POSTGRES_USER=postgres
        - POSTGRES_PASSWORD=postgres

  # Zookeeper, single node
    zookeeper:
      image: wurstmeister/zookeeper:latest
      environment:
        ZOOKEEPER_CLIENT_PORT: 2181
        ZOOKEEPER_TICK_TIME: 2000
      ports:
        - 2181:2181
        - 2888:2888
        - 3888:3888

   # kafka multi node
    kafka1:
      image: wurstmeister/kafka:latest
      restart: "no"
      links:
        - zookeeper
      ports:
        - 9092:9092
      environment:
        KAFKA_BROKER_ID: 1
        KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
        KAFKA_LISTENERS: INTERNAL://:29092,EXTERNAL://:9092
        KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka1:29092,EXTERNAL://localhost:
        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTE
        KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
        #https://github.com/wurstmeister/kafka-docker/issues/553

    kafka2:
      image: wurstmeister/kafka:latest
      restart: "no"
      links:
        - zookeeper
      ports:
        - 9093:9093
      environment:
        KAFKA_BROKER_ID: 2
        KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
        KAFKA_LISTENERS: INTERNAL://:29093,EXTERNAL://:9093
        KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka2:29093,EXTERNAL://localhost:
        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTE
        KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
        #https://github.com/wurstmeister/kafka-docker/issues/553

  #kafdrop for topic/msg visualization
    kafdrop:
      image: obsidiandynamics/kafdrop
      restart: "no"
      environment:
```

```
      KAFKA_BROKERCONNECT: "kafka1:29092,kafka2:29093"
      JVM_OPTS: "-Xms16M -Xmx512M -Xss180K -XX:-TieredCompilation -XX:+UseStrin
    ports:
      - 9000:9000
    depends_on:
      - kafka1
      - kafka2

  # debezium connector
  kconnect:
    image: debezium/connect:1.9
    ports:
      - 8083:8083
    environment:
      CONFIG_STORAGE_TOPIC: my_connect_configs
      OFFSET_STORAGE_TOPIC: my_connect_offsets
      STATUS_STORAGE_TOPIC: my_connect_statuses
      BOOTSTRAP_SERVERS: kafka1:29092,kafka2:29093
    links:
      - zookeeper
      - postgres
    depends_on:
      - kafka1
      - kafka2
```



```
# 2. kafdrop --> docker
# 3. postgres --> docker
# 4. debezium connector --> docker
# 5. pgAdmin/DBeaver --> local system
# 6. Postman --> local system
```

Spin-up dockers:

```
docker-compose -f docker-compose-debezium.yaml up -d
```

Step2: Register Kafka Connect

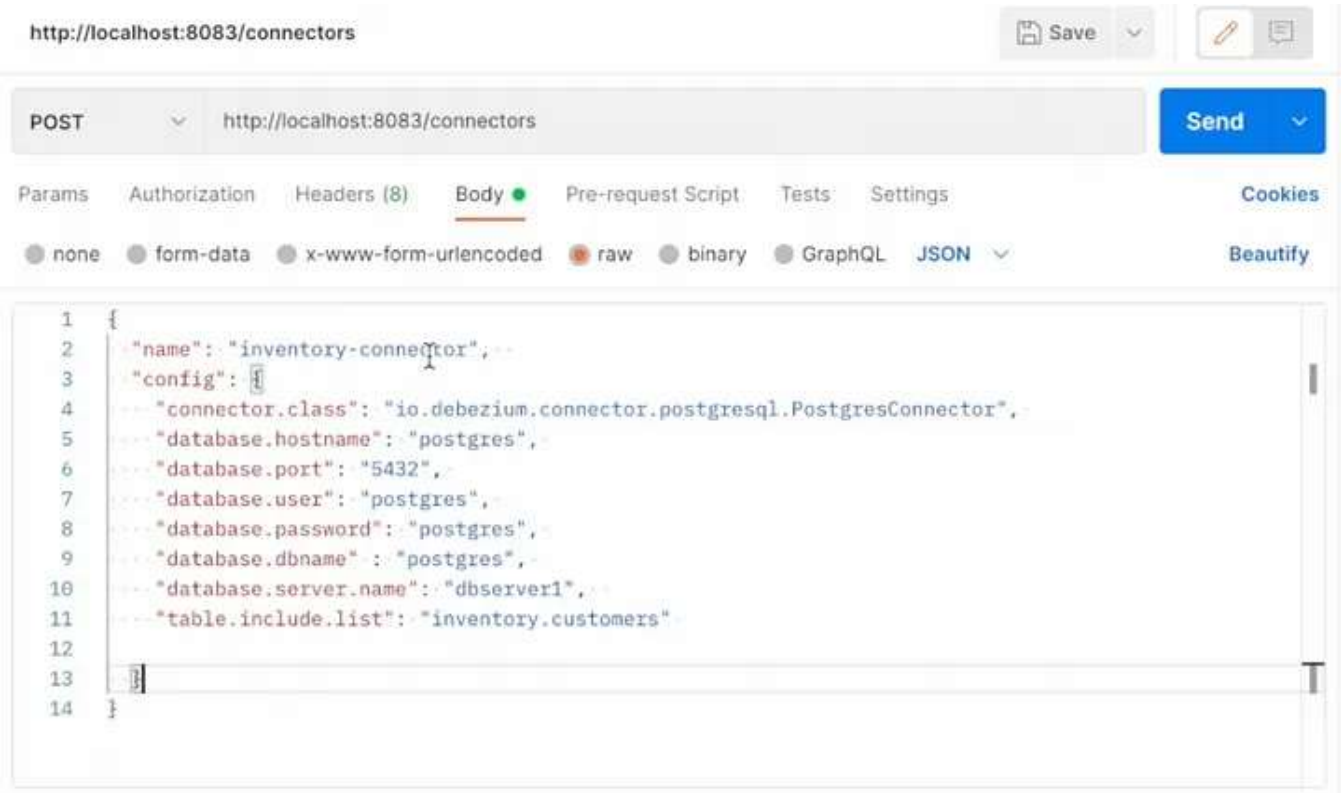
Once all the containers are up and running, open postman and make a POST call to http://localhost:8083/connectors with below request body.

Note: We are passing hostname as '**postgres**' which is the container name in docker-

compose file for postgres container and the port is internal port **5432** not the exposed one 5433.

```
{
    "name": "inventory-connector",
    "config": {
        "connector.class": "io.debezium.connector.postgresql.PostgresConnector'
        "database.hostname": "postgres",
        "database.port": "5432",
        "database.user": "postgres",
        "database.password": "postgres",
        "database.dbname": "postgres",
        "database.server.name": "dbserver1",
        "table.include.list": "inventory.customers"
    }
}
```

for example:



Postman POST request

Here we want to get all change events (INSERT, UPDATE, DELETE) for ***customers*** table in ***inventory*** *schema*. All the changes will be published to our kafka topic.

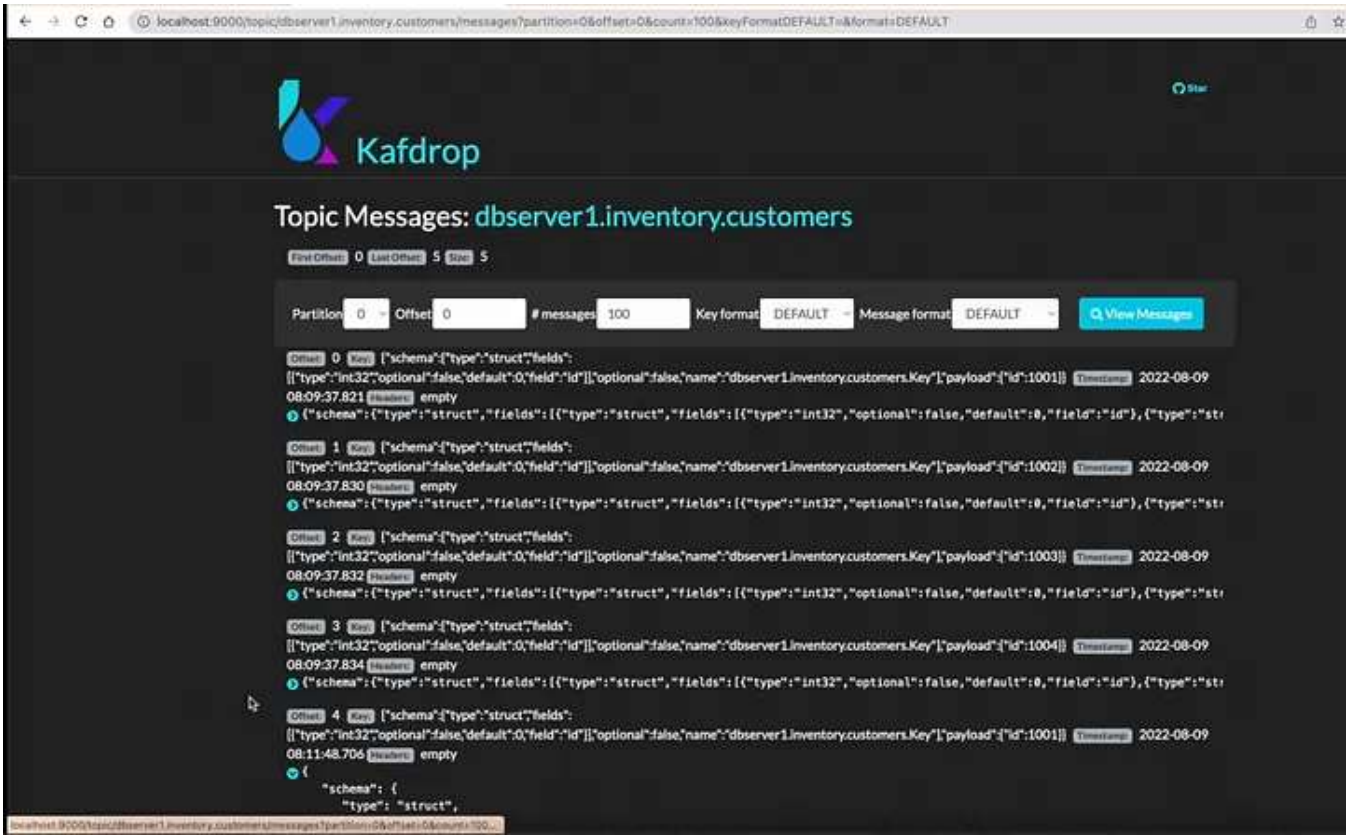
To learn more about postgres connector, you can read the official documentation here.

Step3: Visualize change events

Now that we have kafka connect successfully registered, we are good to make some changes in database.
We can use any database client eg. pgAdmin or DBeaver and connect to our database *(***host***: localhost,* **post***: 5433,* **user***: postgres,* **password***: postgres)* and we can make an **update/insert** to existing row in ***customers*** table.

```
update inventory.customers set first_name = 'Cloud_Geek' where id = 1001
```

After the update statement is executed, we can see in kafdrop UI under topic: '**dbserver1.inventor.customers**' a new message would be published (offset 4 below):



UPDATE change event in Kafdrop UI

Similarly if we make an INSERT/DELETE statement, we will see new messages getting published to the same topic.

Conclusion

This is how CDC (Capture Data Change) is used by data engineering teams. This was just a basic example.

We can write consumers for the above topic (**dbserver1.inventor.customers**) and trigger various actions(emails etc) based on change events (insert, update, delete). This is how we get automated messages for various stages of our delivery items (Amazon, Flipkart, Delhivery)

. . .

Hope this article was helpful to you. If you liked it, please upvote, follow and share.

!! Happy Learning !! ✌️

Debezium | Kafka Connect | Postgres | Kafka | Data Engineering

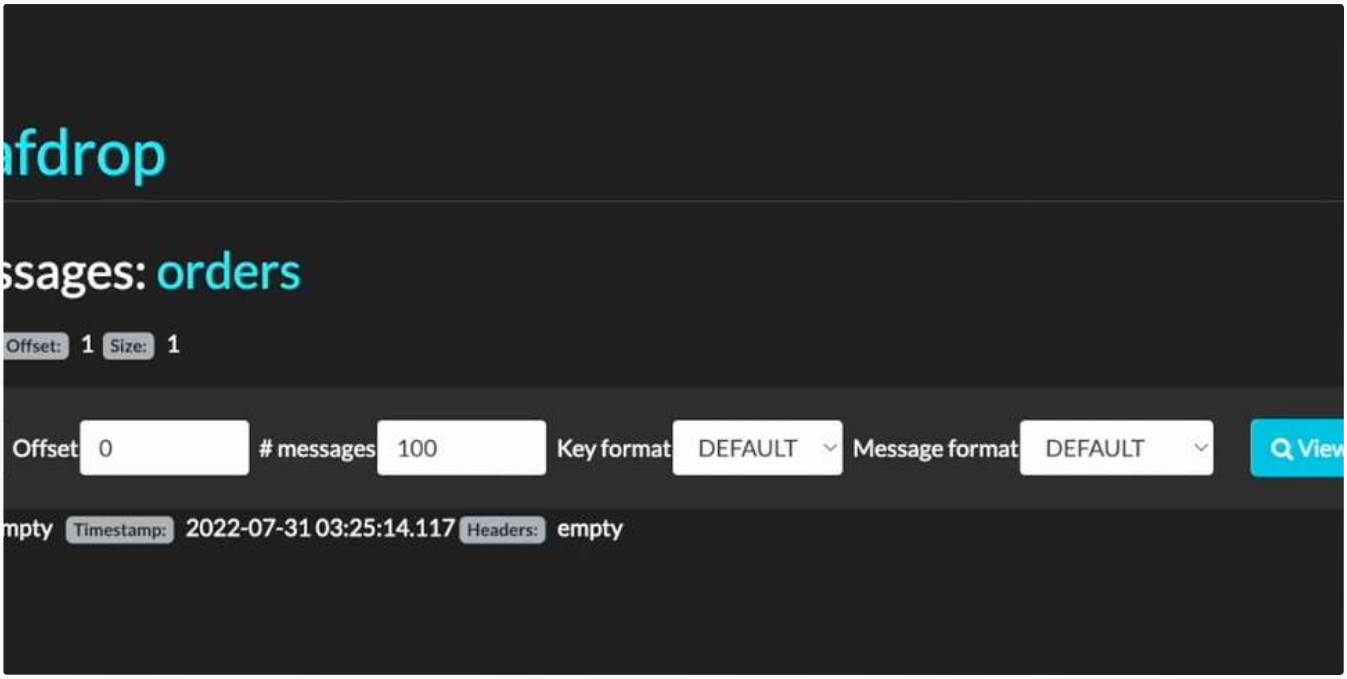
Follow

Written by Cloud_Geek

8 Followers

Passionate about technology, Problem solver |
https://www.youtube.com/channel/UC_cw8htcKIDxeMux3zo7eXw

More from Cloud_Geek

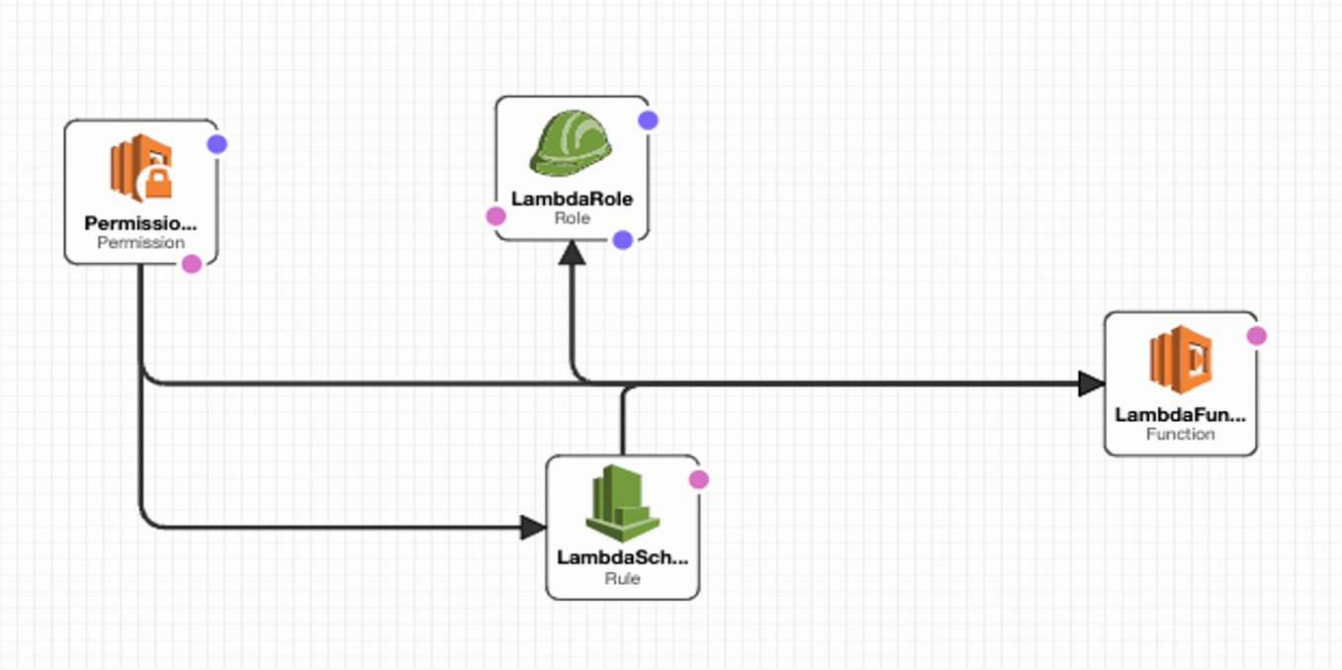


Cloud_Geek

Kafka Producer and Consumer in Python | Dockerized kafka & zookeeper

In this article, we will learn to write a producer and a consumer for Kafka Topic in python.

2 min read · Dec 29, 2022

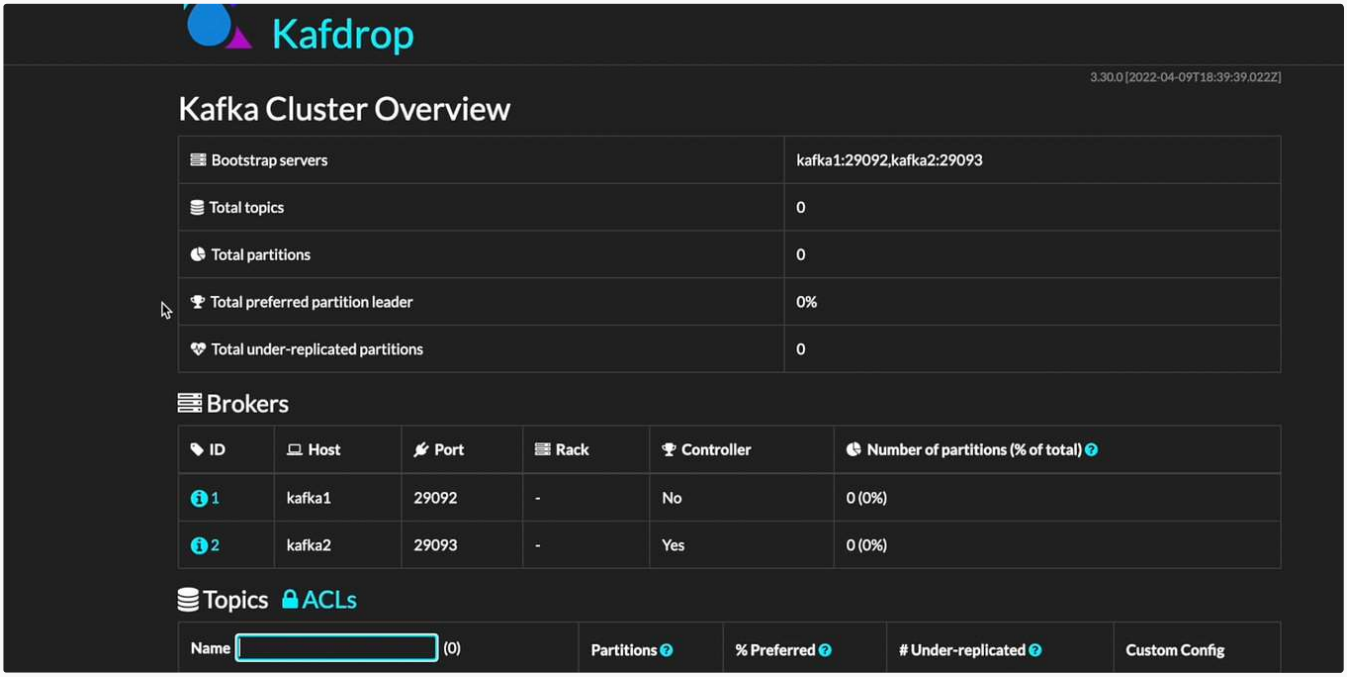


Cloud_Geek

Cleanup Redundant Lambda Log Groups | Automation | Save AWS Cost

If you are working in a company which uses AWS as cloud provider, you must have a mechanism to cleanup stale lambda log groups because...

3 min read · Sep 24, 2022



Cloud_Geek

Run Kafka cluster & Zookeeper using Docker | Kafdrop | docker compose

I always found spinning up kafka and zookeeper nodes little painful. And most of the times, if I have to see message of a particular...

2 min read · Dec 28, 2022

See all from Cloud_Geek

Recommended from Medium

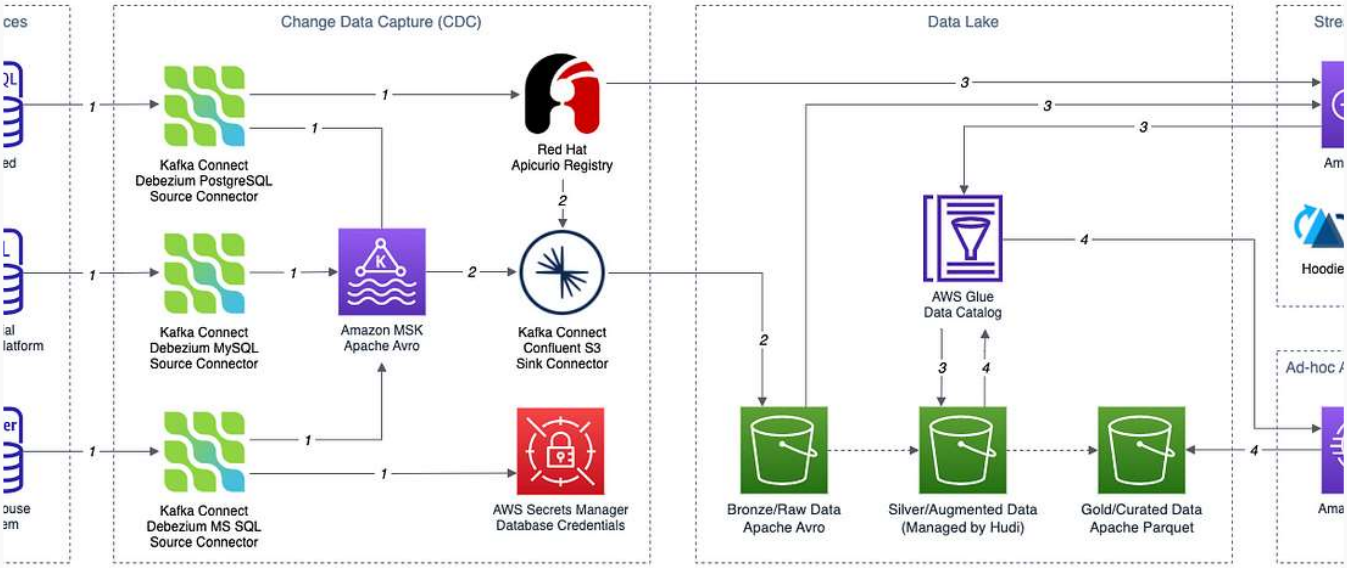
Thomas Reid in Level Up Coding

Streaming change data capture (CDC) data between databases using Kafka

This article will briefly introduce Kafka, how to connect database sources to it using the Kafka SQL client ksqlDB and create and...

21 min read · Mar 26

86



Gary A. Stafford in ITNEXT

Building Data Lakes on AWS with Kafka Connect, Debezium, Apicurio Registry, and Apache Hudi

Learn how to build a near real-time transactional data lake on AWS using a combination of Open Source Software (OSS) and AWS Services

· 21 min read · Feb 27

Lists

New_Reading_List

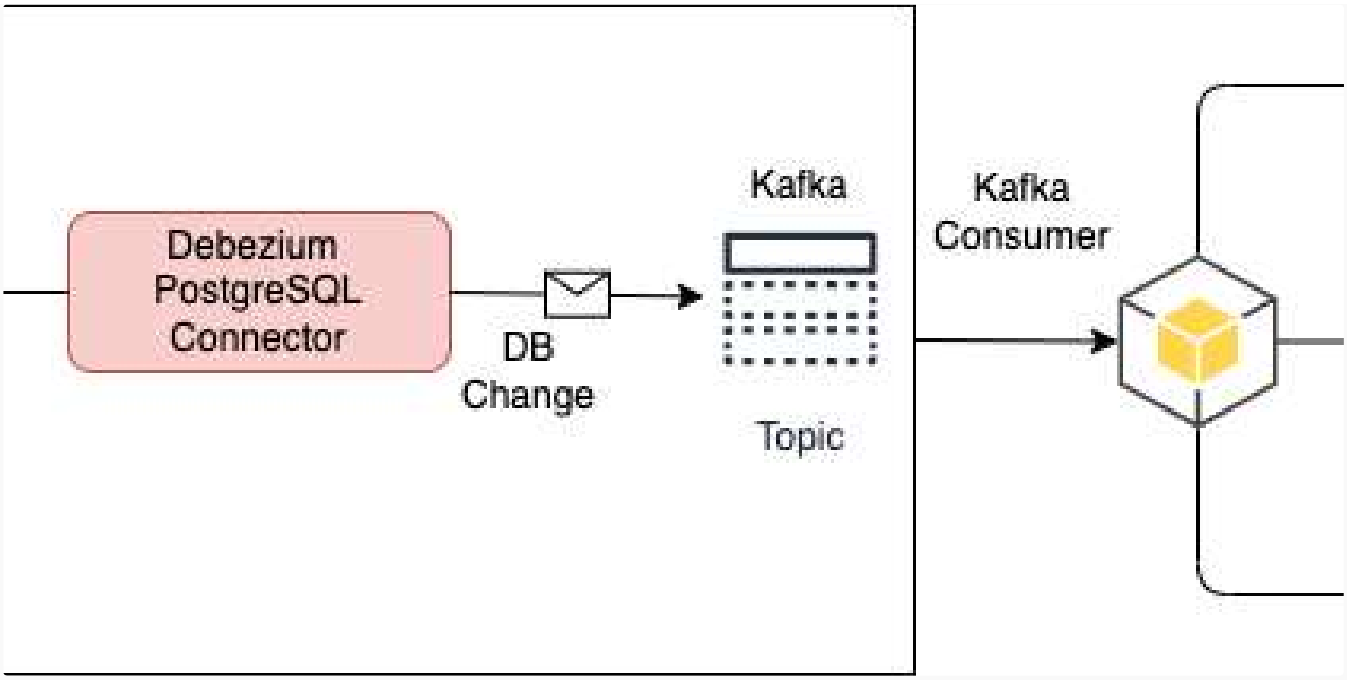
174 stories · 41 saves

Now in AI: Handpicked by Better Programming

260 stories · 57 saves

Natural Language Processing

451 stories · 82 saves



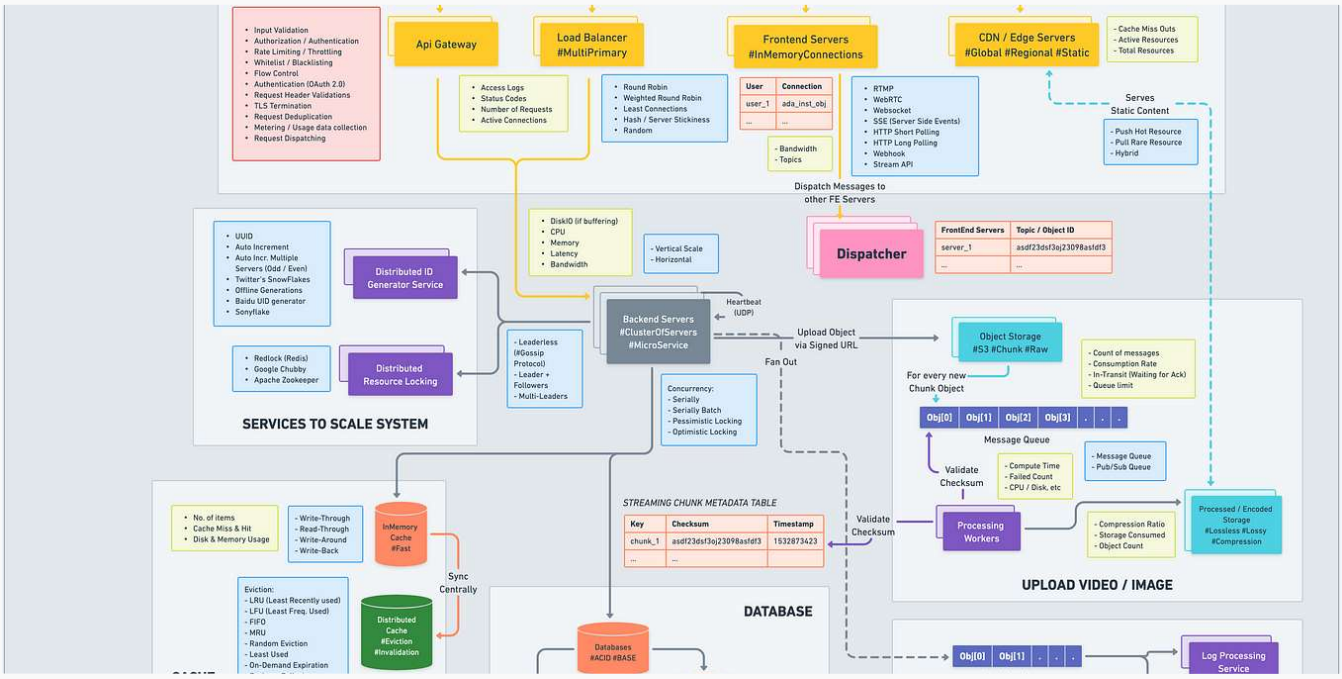
Joao Paulo Alvim

Streaming data from PostgreSQL to s3 using Debezium, Kafka and Python

How to use Change Data Capture (CDC) to move data in near real-time from a relational database to any other application

12 min read · Jan 30

42 6



Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

· 9 min read · Apr 20

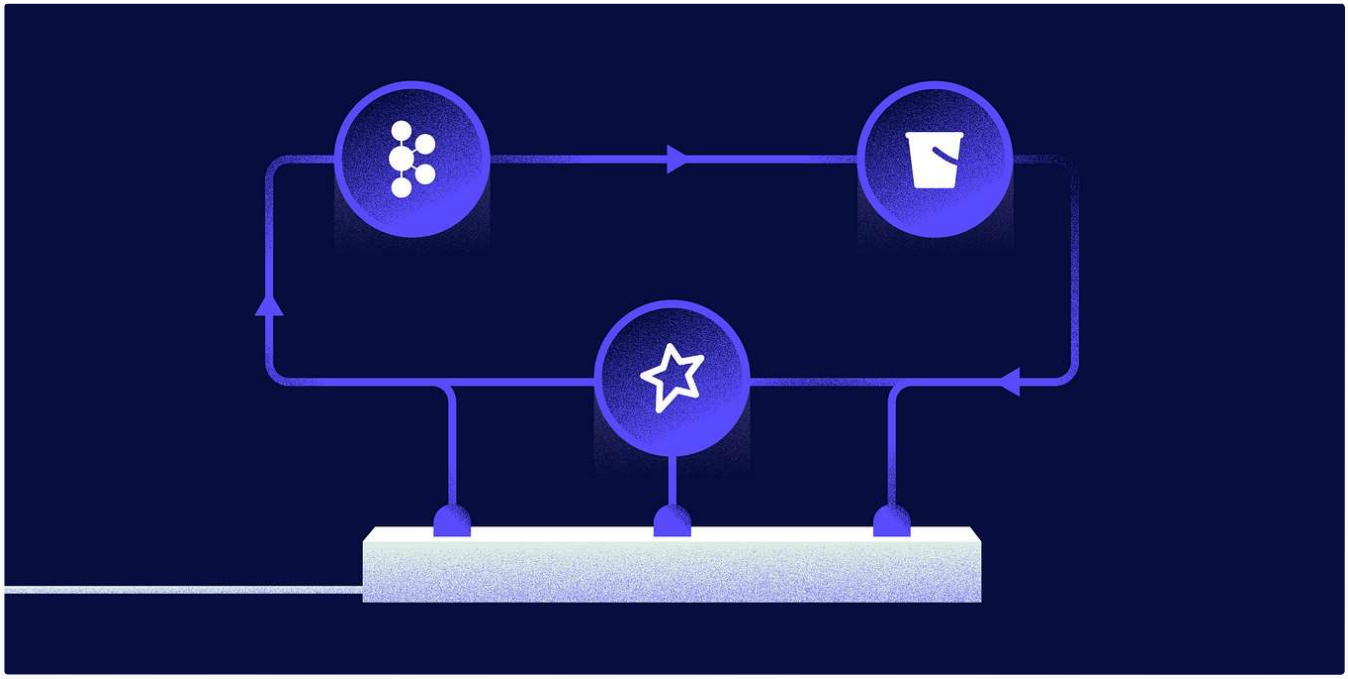
6.9K 55

Gabriel Nascimento

Get better performance Insights by using ElasticSearch and K6 through Docker for Load Testing...

Using Kibana with K6 can be a powerful combination for monitoring and analyzing the results of load and performance tests performed with...

6 min read · 3 days ago



Tal Wanish in Riskified Tech

Dockerizing Spark Structured Streaming with Kafka And LocalStack

How to integrate Kafka and S3 with Spark Structured Streaming using Docker Compose

6 min read · Feb 6

291

See more recommendations