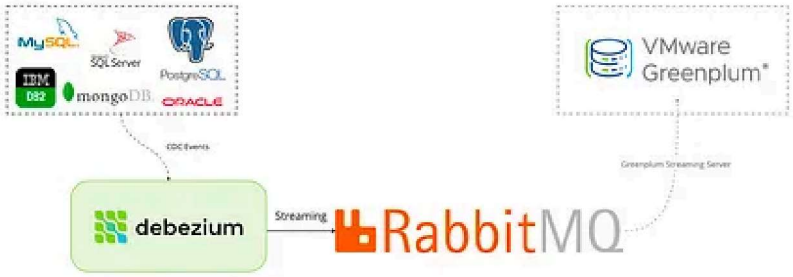
Streaming CDC events from Any Database to Greenplum Data Warehouse using RabbitMQ and Debezium

Ahmed Rachid Hazourli · Follow
Published in Greenplum Data Clinics
8 min read · Jul 23, 2023

Real-time analytics and seamless data integration are crucial for making informed business decisions in today's data-driven world. Change Data Capture (CDC) is a fundamental technology to capture and propagate data changes from various databases in real time. This blog post will explore how to achieve real-time CDC from any database to the Greenplum data warehouse using RabbitMQ Streams and Debezium.

In this post, we will explore how Greenplum, an innovative and high-performance data warehouse for big data analytics and machine learning, can unleash the power of big data in real time. Additionally, we will delve into its advanced streaming capabilities that enable real-time data ingestion, processing, and analytics.

We will walk through the technical configuration and step-by-step instructions to set up this data pipeline.



Understanding Change Data Capture (CDC)

Change Data Capture is a technique used to capture and record changes made to a database, allowing applications to respond quickly to those changes. It tracks insert, update, and delete operations on database tables and transforms them into events. Downstream systems can then ingest this event stream for various purposes, including real-time analytics, data warehousing, and data integration.

Introducing RabbitMQ (Streaming Events through RabbitMQ)

RabbitMQ is a widely adopted message broker that facilitates the reliable exchange of messages between applications.

Last year, streams were introduced, and today, RabbitMQ can be used to support many use cases, whether as a message broker, for message streams, or doing both in unison. It supports multiple messaging protocols and communication patterns, making it an ideal choice for streaming data from various sources to a destination like Greenplum.

Using the RabbitMQ streaming layer, we can decouple the data source from the data warehouse, ensuring better scalability, reliability, and fault tolerance.

Leveraging Debezium for Change Data Capture

Debezium is an open-source platform that specialises in CDC for databases. It supports various database management systems, including DB2, MySQL, PostgreSQL, MongoDB, Oracle, and SQL Server.

Debezium captures row-level changes from database transaction logs and transforms them into a CDC event stream. It also ensures data integrity and consistency during the capture process.

Greenplum Streaming Capabilities

Greenplum, a PostgreSQL-based massively parallel processing (MPP) data warehouse, offers a powerful streaming capability that enables real-time data ingestion from external sources.

With Greenplum Streaming Server (GPSS), organisations can efficiently process data streams and integrate them directly into the Greenplum database.

In our real-time CDC setup, GPSS plays a vital role in ingesting the data from RabbitMQ, as captured by Debezium, and populating CDC events to tables in Greenplum to INSERT, UPDATE or DELETE records.

As an integral part of Greenplum's ecosystem, GPSS delivers unmatched streaming capabilities that streamline data pipelines and supercharge analytical insights. It can handle massive volumes of streaming data with exceptional scalability (proven with 10 million events per second with more than 500 Billion rows per day in a Multi-Trillion Greenplum Database and Running Analytics and ML on top of this Database).

Real-time CDC from Any Database to Greenplum data warehouse

As a source database, we will use PostgreSQL and enable real-time CDC to Greenplum; the following components are used in the configuration:

- PostgreSQL 15 database
- Debezium Server 2.4
- RabbitMQ 3.12.2
- Greenplum 6.24 (+ Greenplum Streaming Server 1.10.1)

To facilitate the execution of this demo, all components will be deployed using docker-compose:

```
version: "3.9"
services:
  gpdb:
    image: docker.io/ahmedrachid/gpdb_demo:6.21
    depends_on:
      - rabbitmq
    privileged: true
    entrypoint: /usr/lib/systemd/systemd
    ports:
      - 5433:5432
    volumes:
      - ${PWD}/greenplum-db-6.24.0-rhel8-x86_64.rpm:/home/gpadmin/greenplum-db-
      - ${PWD}/gpss-gpdb6-1.10.1-rhel8-x86_64.gppkg:/home/gpadmin/gpss-gpdb6-1.
      - ${PWD}/script_gpdb.sh:/home/gpadmin/script_gpdb.sh
  rabbitmq:
    image: rabbitmq:3-management-alpine
    container_name: rabbitmq
    ports:
      - 5672:5672
      - 15672:15672
      - 5552:5552
    environment:
      RABBITMQ_DEFAULT_PASS: root
      RABBITMQ_DEFAULT_USER: root
      RABBITMQ_DEFAULT_VHOST: vhost
  postgres:
    image: quay.io/debezium/example-postgres:2.1
    container_name: postgres
    ports:
      - 5432:5432
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  debezium-server:
    image: quay.io/debezium/server:2.4
    container_name: debezium-server
    ports:
      - 8080:8080
    volumes:
      - ./conf:/debezium/conf
    depends_on:
      - rabbitmq
      - gpdb
      - postgres
```

The main configuration file for Debezium is conf/application.properties:

```
debezium.sink.type=rabbitmq
debezium.sink.rabbitmq.connection.host=rabbitmq
debezium.sink.rabbitmq.connection.port=5672
debezium.sink.rabbitmq.connection.username=root
debezium.sink.rabbitmq.connection.password=root
debezium.sink.rabbitmq.connection.virtual.host=vhost
debezium.sink.rabbitmq.connection.port=5672
debezium.source.connector.class=io.debezium.connector.postgresql.PostgresConnec
debezium.source.offset.storage.file.filename=data/offsets.dat
debezium.source.offset.flush.interval.ms=0
debezium.source.database.hostname=postgres
debezium.source.database.port=5432
debezium.source.database.user=postgres
debezium.source.database.password=postgres
debezium.source.database.dbname=postgres
debezium.source.topic.prefix=tutorial
debezium.source.table.include.list=inventory.customers
debezium.source.plugin.name=pgoutput
debezium.source.tombstones.on.delete=false
debezium.sink.rabbitmq.routingKey=inventory_customers
```

As you can see, we're using Debezium to stream CDC events from a PostgreSQL table called `inventory.customers` to our RabbitMQ cluster.

Deploy PostgreSQL database:

Run the following Docker command to install & deploy a preconfigured PostgreSQL container:

```
docker compose up postgres -d
```

You can now connect to the database and explore the table we're using to stream CDC events:

```
$ docker-compose exec postgres env PGOPTIONS="--search_path=inventory" bash -c
  id  | first_name | last_name |         email
------+------------+-----------+-----------------------
 1001 | Sally      | Thomas    | sally.thomas@acme.com
 1002 | George     | Bailey    | gbailey@foobar.com
 1003 | Edward     | Walker    | ed@walker.com
```

```
 1004 | Anne       | Kretchmar | annek@noanswer.org
(4 rows)
```

Setup & Configure RabbitMQ:

To deploy our RabbitMQ cluster, we'll use the same docker compose command as below:

```
docker compose up rabbitmq -d
```

Once you have deployed RabbitMQ using Docker Compose, you can configure it to suit your needs.

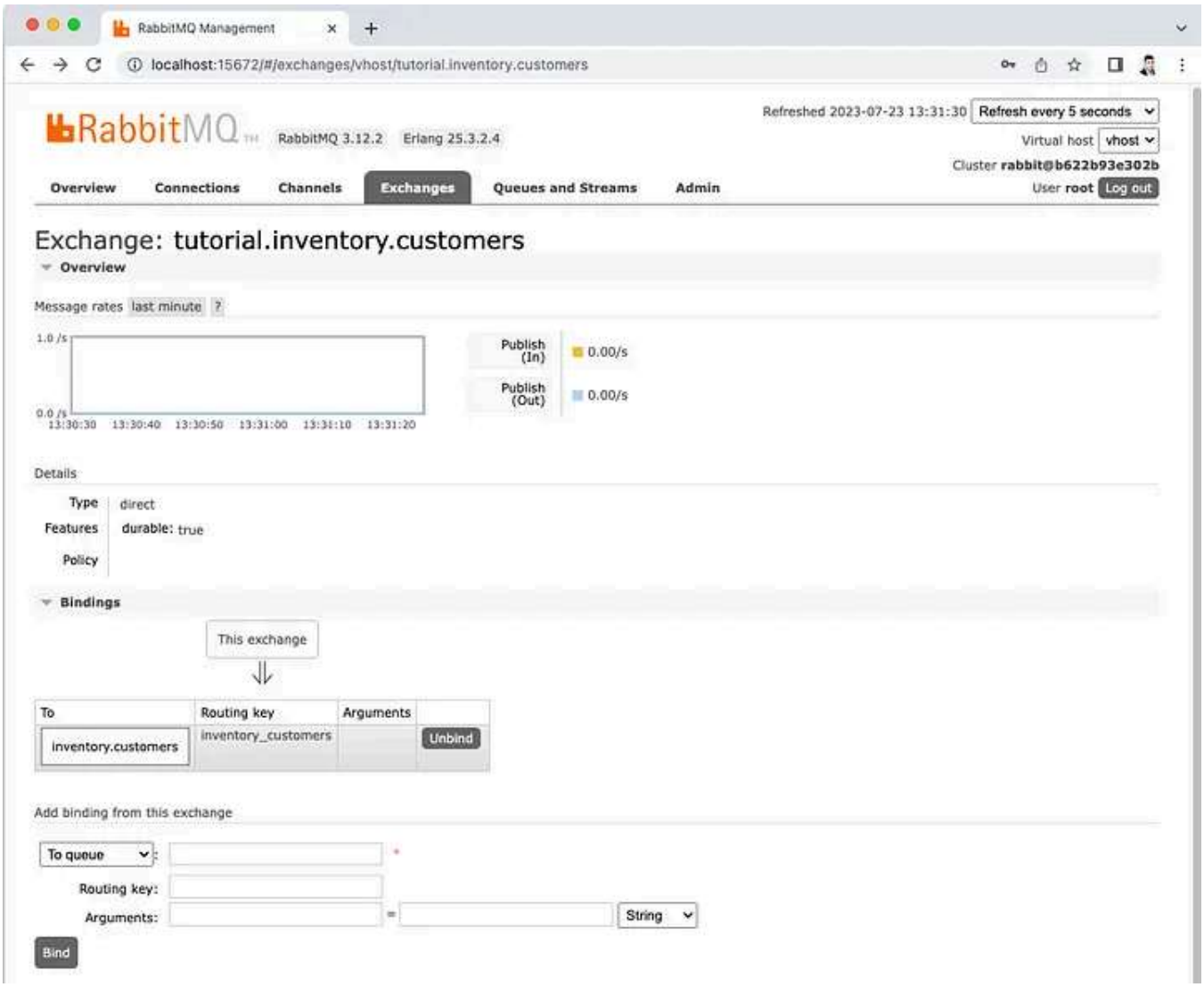
RabbitMQ provides a web-based management interface that you can use to configure various aspects of the RabbitMQ server.

To access the management interface, open a web browser and navigate to the IP address or domain name of your RabbitMQ instance, followed by the management port number (by default, `15672`). In this example, you can access the management interface by navigating to `http://localhost:15672`

Once you have accessed the management interface, you can log in with the default credentials (`root` / `root`) or with the username and password specified in the Docker Compose file.

From the management interface, you should create the following:

- A new **RabbitMQ Exchange** called `tutorial.inventory.customers`
- A new **RabbitMQ Stream** called `inventory.customers` and *bind* it to the `tutorial.inventory.customers` exchange using the **routingKey** `inventory_customers`

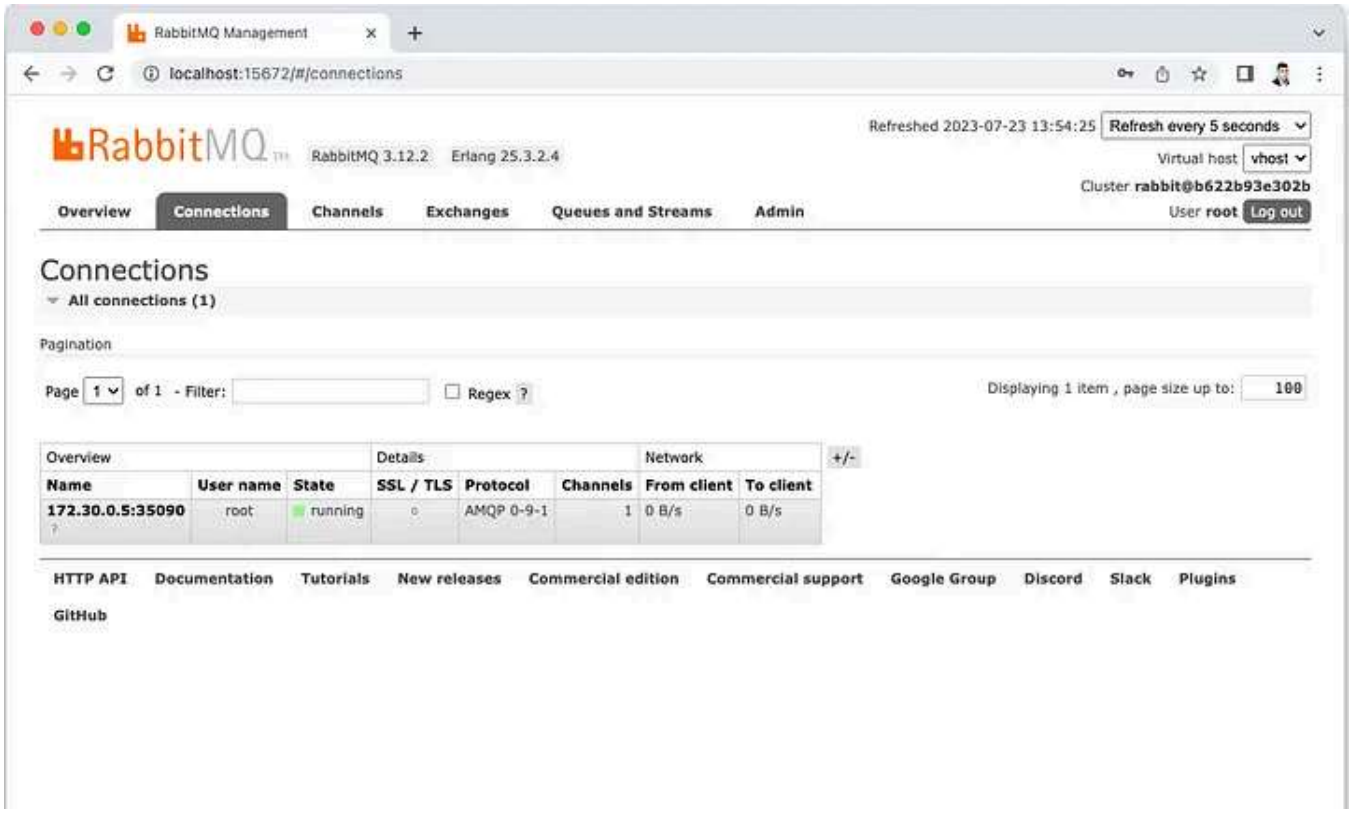


Setup & Configure Debezium Server:

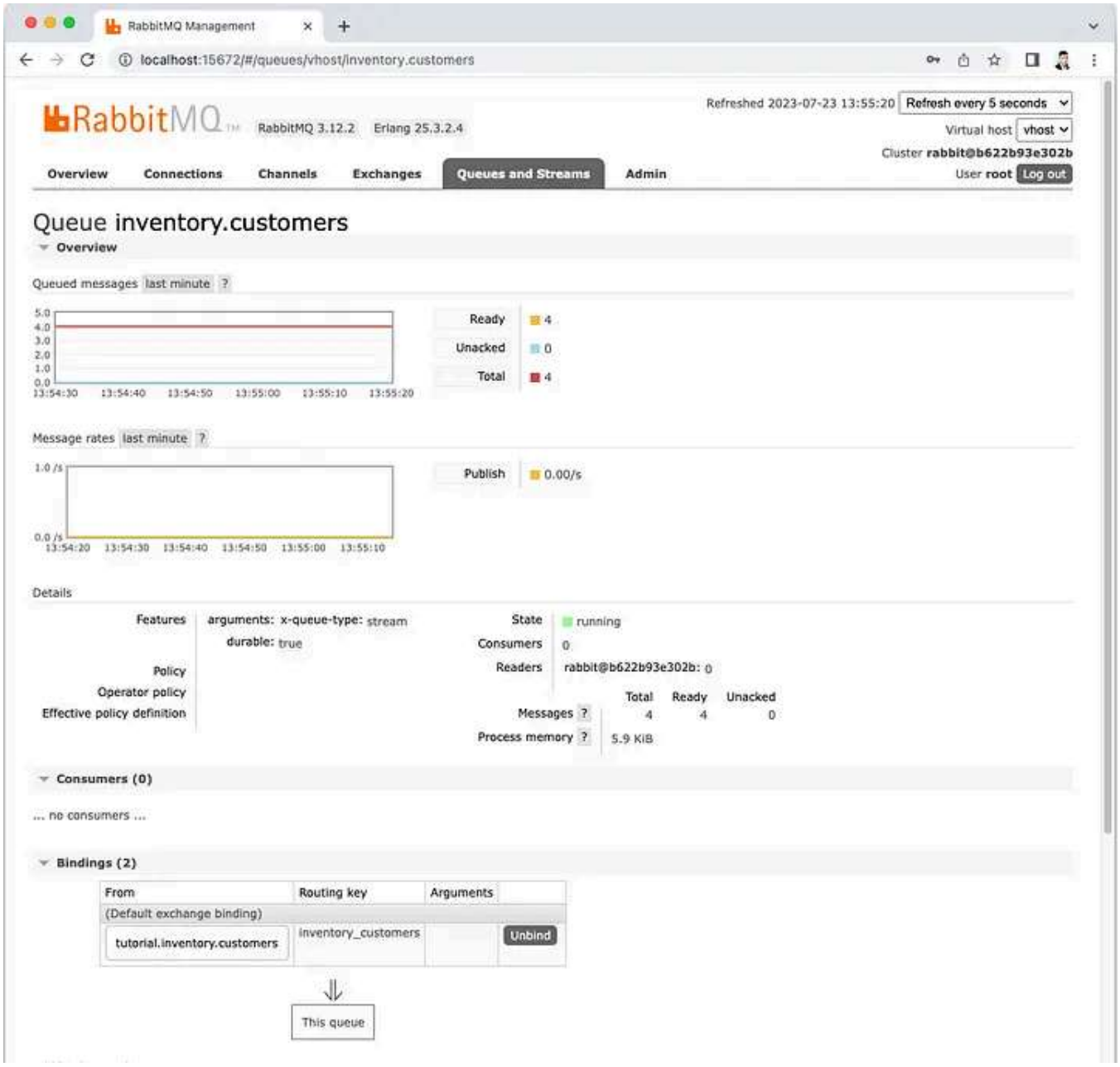
At this point, our PostgreSQL and RabbitMQ containers are running, but we haven't started our Debezium Server yet. To do so, we will run the following:

```
docker compose up debezium-server -d
```

Once done, you should see a new open connection to the RabbitMQ cluster:



You should also see new messages in the RabbitMQ Stream ready to be consumed:



Setup Greenplum to ingest real-time CDC events:

To enable real-time CDC from RabbitMQ to Greenplum, we have a Greenplum 6.24 instance, pre-configured with Greenplum Streaming Server 1.10.1:

Firstly, we need to deploy the Greenplum container:

```
docker compose up gpdb -d
```

Once done, we should now create a `customers` table to load CDC events:

```
$ docker compose exec -ti gpdb bash
$ su - gpadmin
$ psql postgres -c 'CREATE TABLE public.customers (id INT, first_name TEXT, las
```

GPSS or Greenplum Streaming Server efficiently consumes and processes data streams streaming from Kafka and RabbitMQ (from a RabbitMQ queue or a RabbitMQ stream) into Greenplum Database.

It provides a flexible, scalable architecture that enables high-throughput data ingestion with minimal latency. GPSS is designed to handle various data formats, including TEXT, CSV, JSON, Avro, and more, making it well-suited for real-time CDC scenarios.

To start loading CDC events from RabbitMQ Stream, we should start our GPSS process using the following:

```
nohup gpss &
```

Then, create our GPSS job with the configuration below:

```
DATABASE: postgres
USER: gpadmin
HOST: localhost
PORT: 5432
VERSION: 2
RABBITMQ:
  INPUT:
    SOURCE:
      SERVER: root:root@rabbitmq:5552
```

```
      STREAM: inventory.customers
      VIRTUALHOST: vhost
    DATA:
      COLUMNS:
        - NAME: j
          TYPE: json
      FORMAT: json
    ERROR_LIMIT: 25
  OUTPUT:
      TABLE: customers
      MODE: MERGE
      MATCH_COLUMNS:
        - id
      DELETE_CONDITION: ((j->>'payload')::json->>'op')='d'
      MAPPING:
           - NAME: id
             EXPRESSION: CASE WHEN ((j->>'payload')::json->>'op')='d' THEN (((j
           - NAME: first_name
             EXPRESSION: CASE WHEN ((j->>'payload')::json->>'op')='d' THEN (((j
           - NAME: last_name
             EXPRESSION: CASE WHEN ((j->>'payload')::json->>'op')='d' THEN (((j
           - NAME: email
             EXPRESSION: CASE WHEN ((j->>'payload')::json->>'op')='d' THEN (((j
```

Submit the GPSS job configuration:

```
$ gpsscli submit gpss_job.yaml
```

Start the GPSS job to begin capturing and processing data from RabbitMQ Stream:

```
$ gpsscli start gpss_job
```

Once started, you should now see `customers` job running:

```
[gpadmin@df7ac82e5633 ~]$ gpsscli list
```

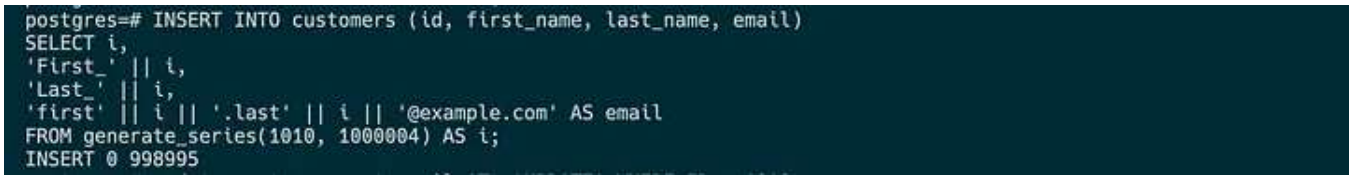
```
JobName                         JobID                                GPHost
gpss_job                        7f40703e01734ad570c31b028fdfc615     localhc
```

Insert data into the PostgreSQL `customers` table:

You should now see some data (4 records) coming into the Greenplum table, and you can also generate many CDC events by running INSERT/UPDATE or DELETEs on the source database.

Let's insert some data dynamically:

```
INSERT INTO customers (id, first_name, last_name, email)
SELECT i,
'First_' || i,
'Last_' || i,
'first' || i || '.last' || i || '@example.com' AS email
FROM generate_series(1010, 1000004) AS i;
```



Real-time CDC data being applied on Greenplum:

- After executing the *INSERT INTO* command, the CDC events are promptly applied to the Greenplum `customers` table in real time.

```
[gpadmin@df7ac82e5633 ~]$ psql postgres
psql (9.4.26)
Type "help" for help.

postgres=# select * from customers limit 10;
  id  | first_name | last_name |            email
------+------------+-----------+------------------------------
 1011 | First_1011 | Last_1011 | first1011.last1011@example.com
 1016 | First_1016 | Last_1016 | first1016.last1016@example.com
 1026 | First_1026 | Last_1026 | first1026.last1026@example.com
 1029 | First_1029 | Last_1029 | first1029.last1029@example.com
 1030 | First_1030 | Last_1030 | first1030.last1030@example.com
 1034 | First_1034 | Last_1034 | first1034.last1034@example.com
 1038 | First_1038 | Last_1038 | first1038.last1038@example.com
 1039 | First_1039 | Last_1039 | first1039.last1039@example.com
 1059 | First_1059 | Last_1059 | first1059.last1059@example.com
 1061 | First_1061 | Last_1061 | first1061.last1061@example.com
(10 rows)

postgres=# select count(*) from customers;
 count
--------
 998998
(1 row)
```

- On the other hand, you can ***UPDATE*** a record on the PostgreSQL source database:

```
postgres=# SELECT * FROM customers WHERE id = 1012;
  id  | first_name | last_name |             email
------+------------+-----------+-------------------------------
 1012 | First_1012 | Last_1012 | first1012.last1012@example.com
(1 row)

postgres=# UPDATE customers SET email='TestUPDATE' WHERE ID = 1012;
UPDATE 1
```

As a result of this process, you can observe that the ***UPDATE*** event has been successfully applied on the Greenplum side, ensuring data consistency and synchronisation.

```
postgres=# select * from customers where id = 1012;
  id  | first_name | last_name |             email
------+------------+-----------+-------------------------------
 1012 | First_1012 | Last_1012 | first1012.last1012@example.com
(1 row)

postgres=# select * from customers where id = 1012;
  id  | first_name | last_name |   email
------+------------+-----------+------------
 1012 | First_1012 | Last_1012 | TestUPDATE
(1 row)
```

Conclusion

Greenplum's streaming capabilities, RabbitMQ Streams and Debezium together, form a robust and efficient real-time CDC pipeline from any database to Greenplum.

This setup empowers organisations to harness the power of real-time data analytics, making informed decisions based on the latest data insights. By leveraging Greenplum's streaming capability, businesses can gain a competitive edge in today's data-driven landscape.

Authors

This blog post was **co-written** with my colleague: **Martin Visser**

📝 References :

1. Open-source Greenplum data warehouse: https://greenplum.org/
2. VMware Greenplum data warehouse: https://docs.vmware.com/en/VMware-Tanzu-Greenplum/index.html
3. Greenplum Streaming Server: https://docs.vmware.com/en/VMware-Greenplum-Streaming-Server/index.html
4. RabbitMQ: https://www.rabbitmq.com/
5. Debezium: https://debezium.io/

> Thanks for reading! Any comments or suggestions are welcome! Check out other Greenplum articles here.

Greenplum Rabbitmq Debezium Postgresql Change Data Capture

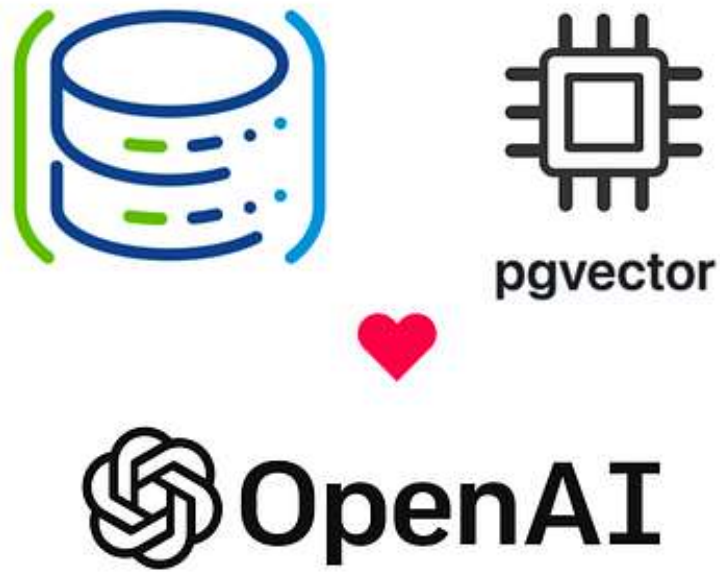
Follow

Written by Ahmed Rachid Hazourli

31 Followers · Writer for Greenplum Data Clinics

Sales Engineer @Snowflake ❄ https://www.linkedin.com/in/ahmed-rachid/

More from Ahmed Rachid Hazourli and Greenplum Data Clinics

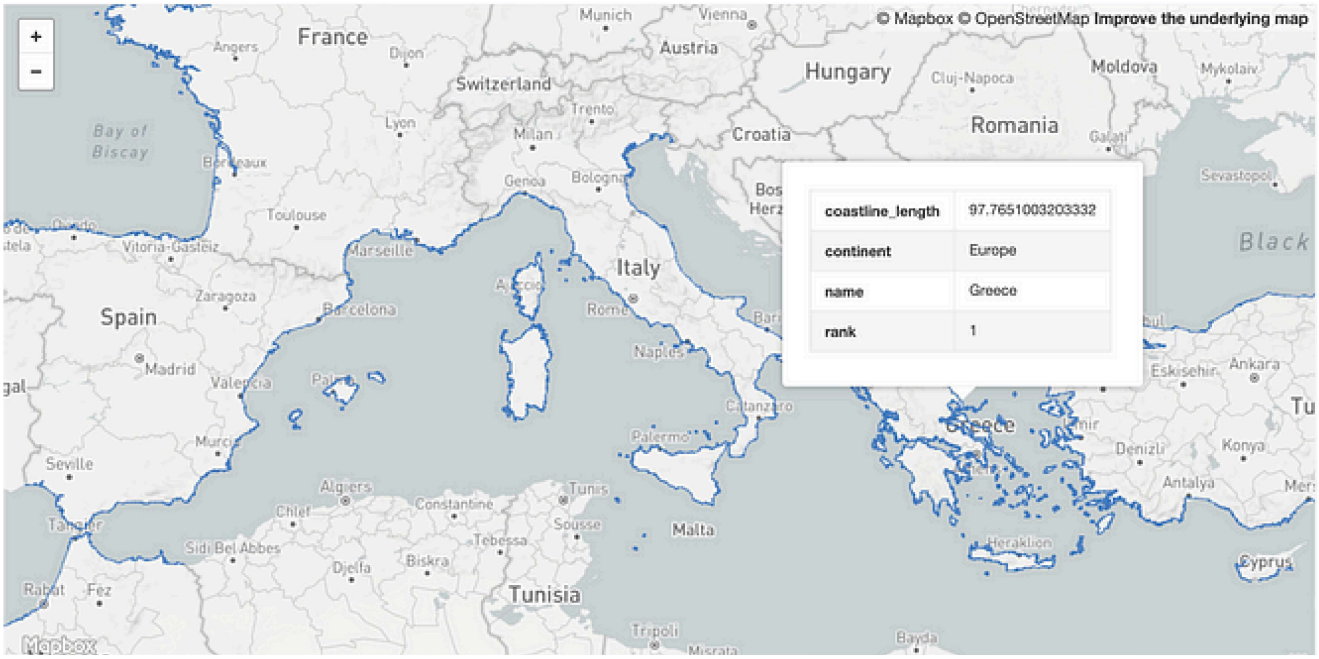


Ahmed Rachid Hazourli in Greenplum Data Clinics

Building large scale AI-powered search in Greenplum using pgvector and OpenAI

Learn how to build scalable AI-applications using the power of pgvector for vector similarity search within Greenplum data-warehouse

May 29, 2023 34 1

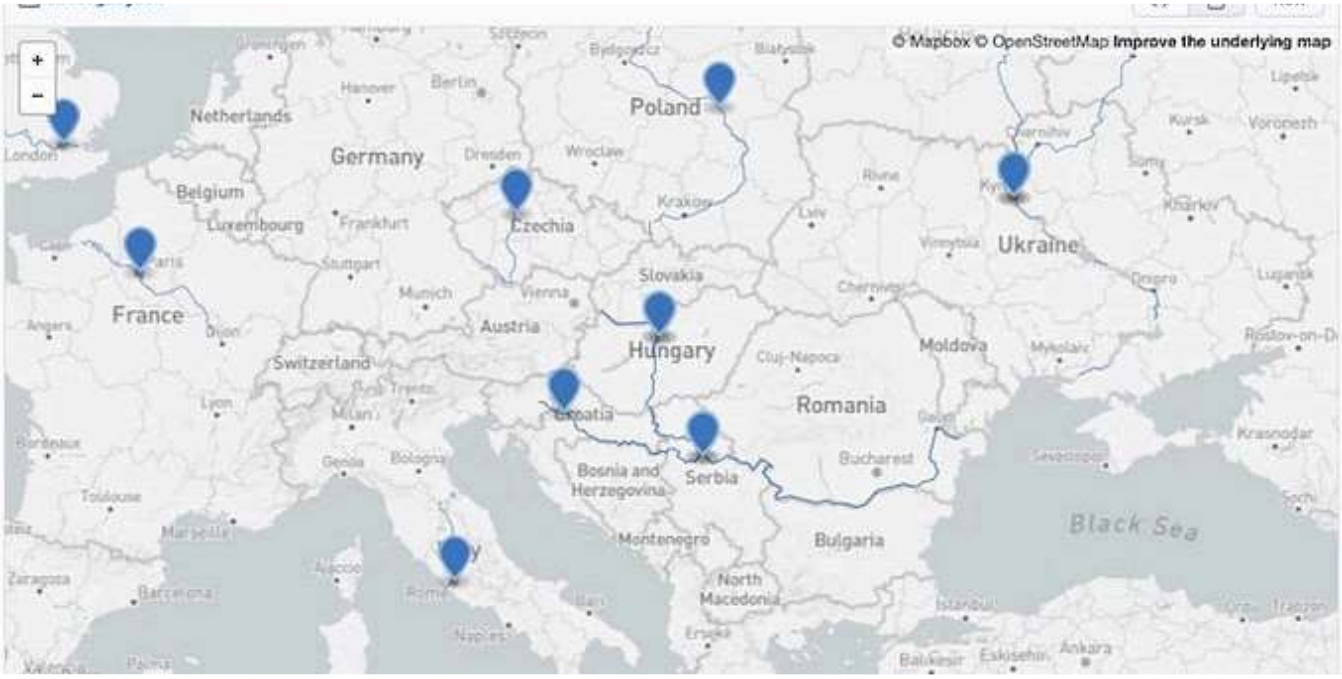


Constantinos Antzakas in Greenplum Data Clinics

Driving geospatial data insights with PostGIS

Using PostGIS functions with examples in PostgreSQL and Greenplum Databases

Feb 12, 2021 221

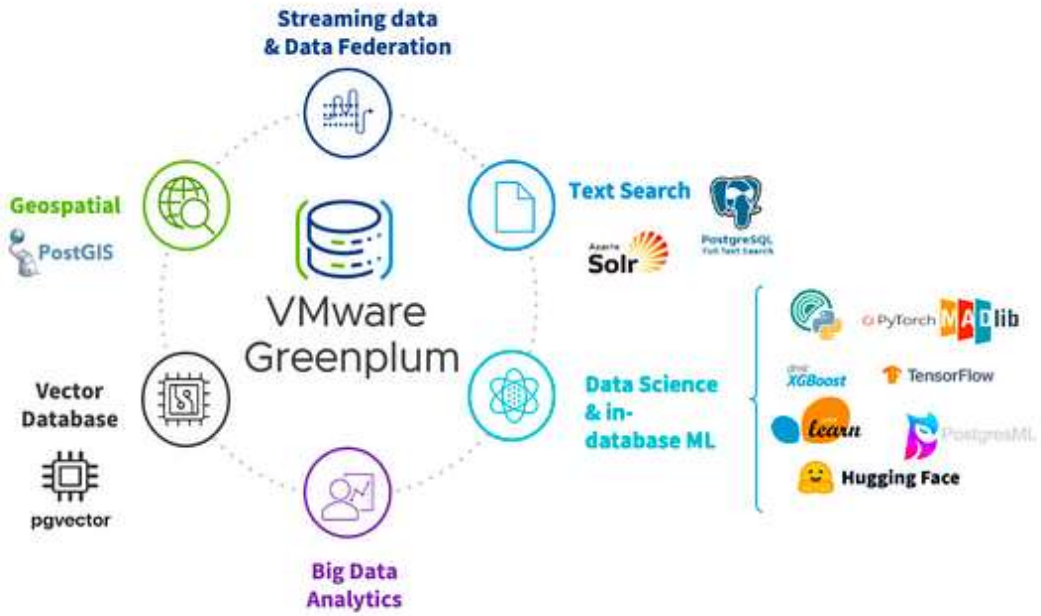


Constantinos Antzakas in Greenplum Data Clinics

Loading geospatial data with PostGIS

From shapefiles to Greenplum Database

Feb 8, 2021 139



Ahmed Rachid Hazourli in **AI monks.io**

Why VMware Greenplum as a Massive Vector Database for your Analytical needs ?

This blog post demonstrates why VMware Greenplum stands as a great choice for companies in search of a Data Platform for AI Analytical...

Aug 9, 2023 5

See all from Ahmed Rachid Hazourli

See all from Greenplum Data Clinics

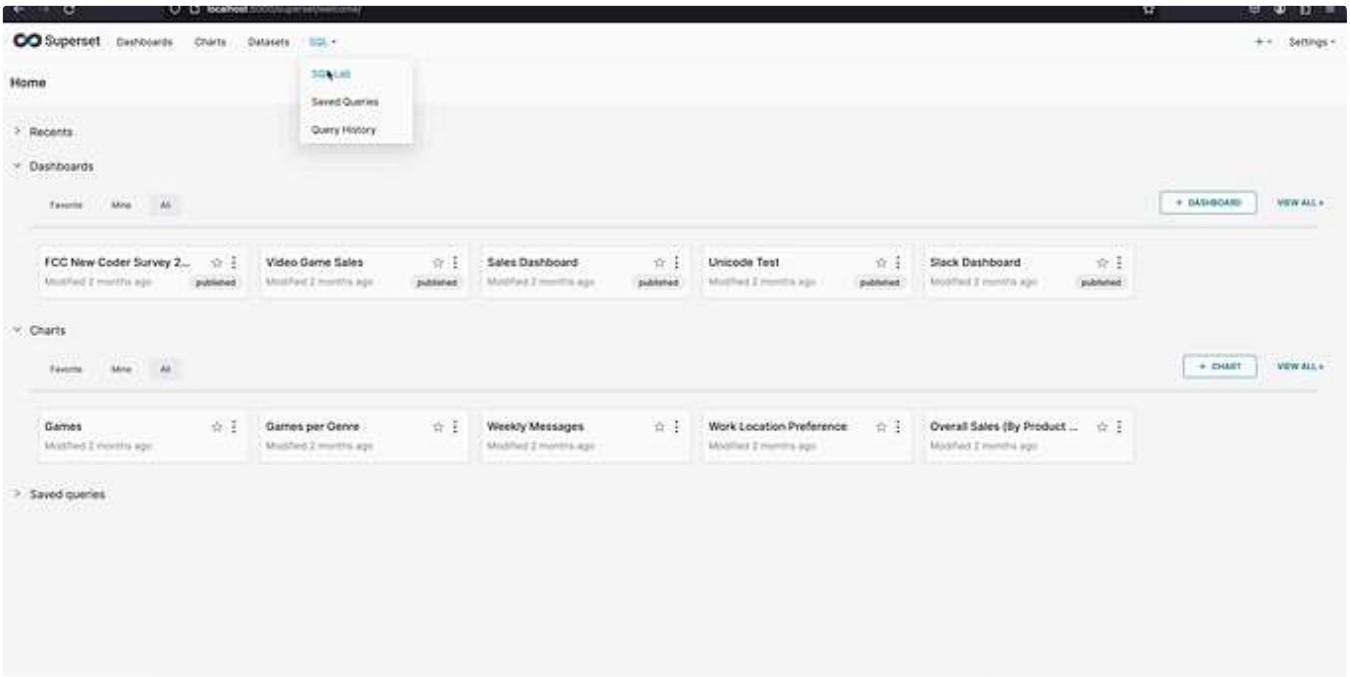
Recommended from Medium

Kareem Mohllal

Change Data Capture (CDC) with Debezium, Kinesis, and EventBridge

Capture data changes in source databases and deliver them to downstream systems in real-time using an event-driven approach.

Apr 14 26



CA Amit Singh in Free or Open Source software's

How to Connect Apache Superset with CSV Files via DuckDB

Learn to Connect Apache Superset Open Source Data Visualization with CSV via DuckDB (fast in-process analytical database).

Apr 14 4

Lists

Staff Picks

723 stories · 1267 saves

Stories to Help You Level-Up at Work

19 stories · 776 saves

Self-Improvement 101

20 stories · 2662 saves

Productivity 101

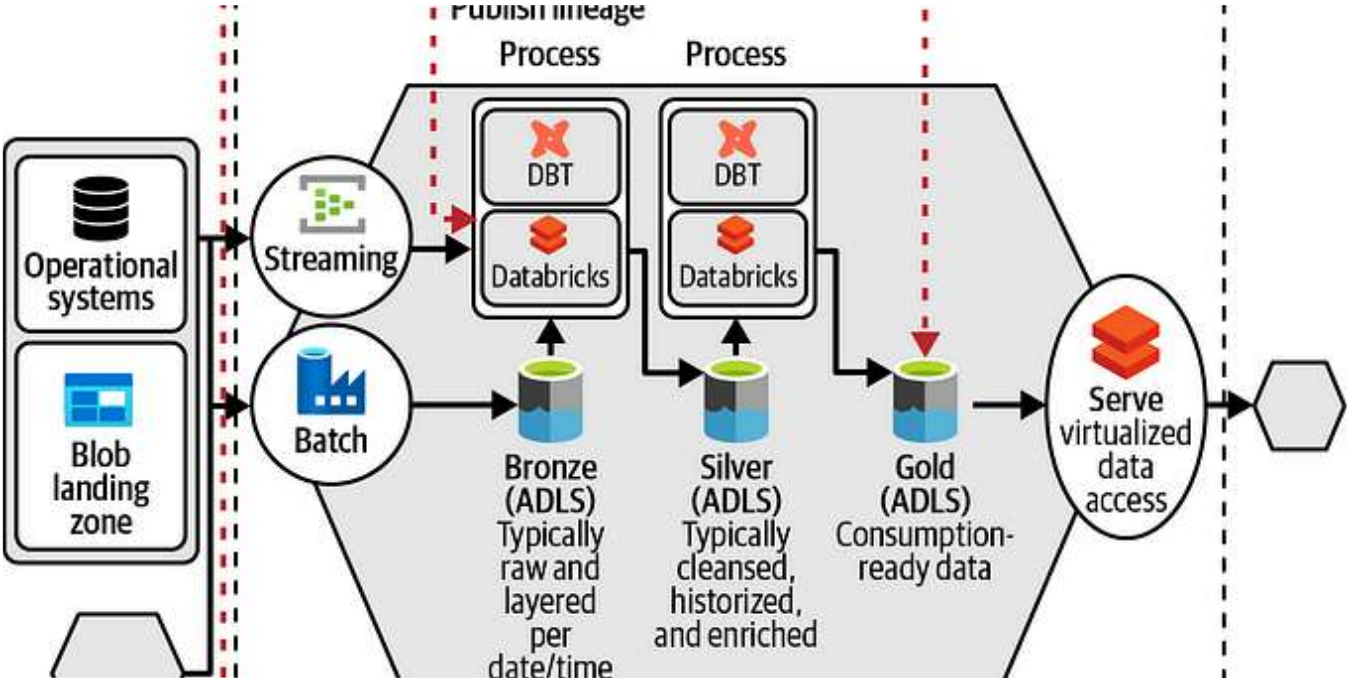
20 stories · 2285 saves

Sai Parvathaneni in Towards Dev

Building a Data Modeling Pipeline - dbt, Snowflake, and Airflow

In this article, we'll build a Data Modeling pipeline using dbt, Snowflake, and Airflow. By the end, you'll have a good understanding of...

Aug 25 28



Piethein Strengholt

Data Management at Scale

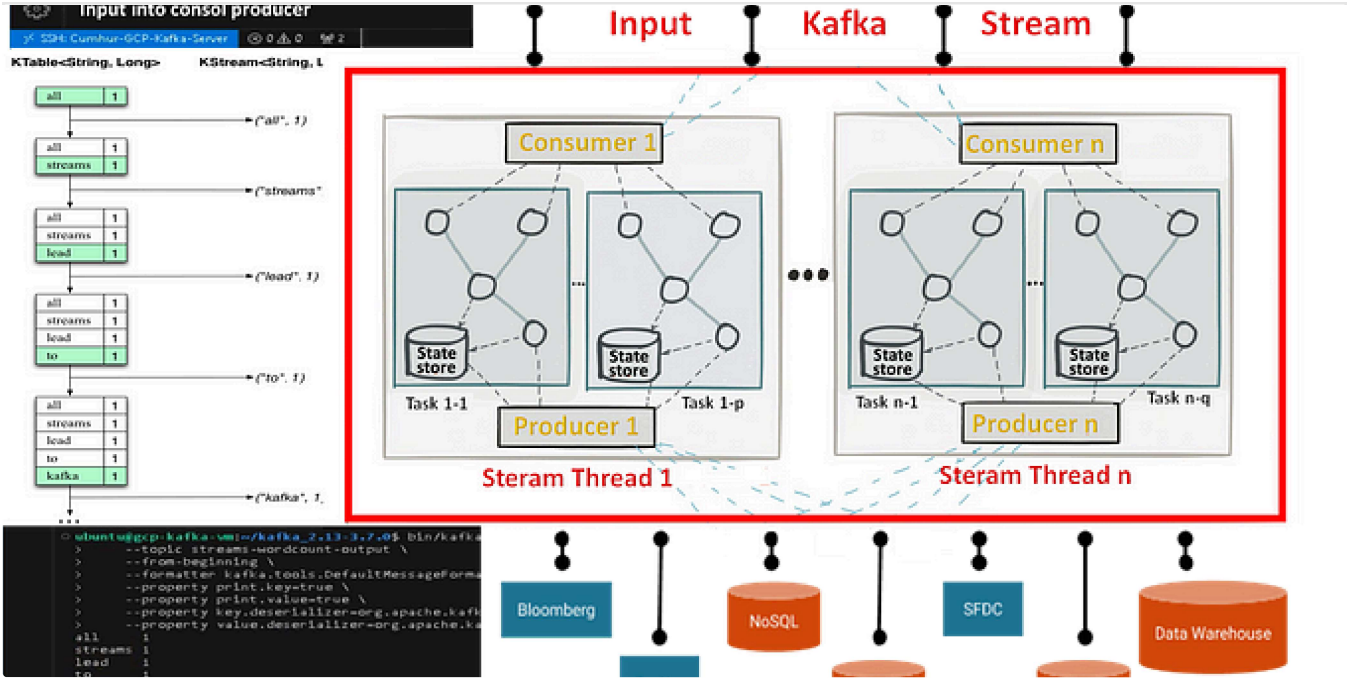
Jul 11, 2023 438 3

Mr.PlanB

Efficiently Scaling PostgreSQL with Kubernetes, Strategies and Best Practices

The integration of PostgreSQL, a leading open-source relational database management system, with Kubernetes, a powerful platform for...

Apr 18 1



Cumhur Akkaya

Message Queue(MQ) Systems-2: Installing and Using Apache Kafka as an Event Streaming Platform

From where we left off, we continue learning MQ Systems with “Apache Kafka”. We examined “Amazon SQS (Simple Queue Service)” in our...

Mar 11 33

See more recommendations