Change Data Capture with Apache Kafka, PostgreSQL, Kafka Connect and Debezium

By Carbon Rider in Articles

November 16, 2019 0 Comment

Spread the love

0
Shares

Post Views: 2,553

I always wondered how Enterprise Systems are able to perform analytics on continuously increasing data. I have read articles which talk about Map Reduce jobs to perform bulk of data processing and later on utilize tools like Kafka, Flink etc. This enables to perform continuous data streaming and keep analytics in sync with latest changes. I admit I am still noob to all this stuff, But my curiosity doesn't let me rest. I definitely wanted to get started somewhere. Recently I started working on a problem which needed a solution to build reliable batch system.

I have worked with various solutions in batch jobs including home grown libraries, sophisticated frameworks like Quartz etc. But then, this time I wanted to try out something new. And so, I came across a term called as CDC (Change Data Capture).

With CDC, system utilizes tools which get notified for every change in the Data. For e.g. If you create a new record in database or insert an entry into a log file. Remember the concept of data is not just limited to Persistent storage like Database or File. I found this idea very appealing and thought of using it to build Scalable Batch system.

Building Blocks

My system is built on JDK 13, Spring Boot 2.X, Postgres 11.5 and lot of other stuffs. The idea to build batch was pretty simple. Create a single table with a key to identify Job and a JSON column to hold required data. This was pretty easy.

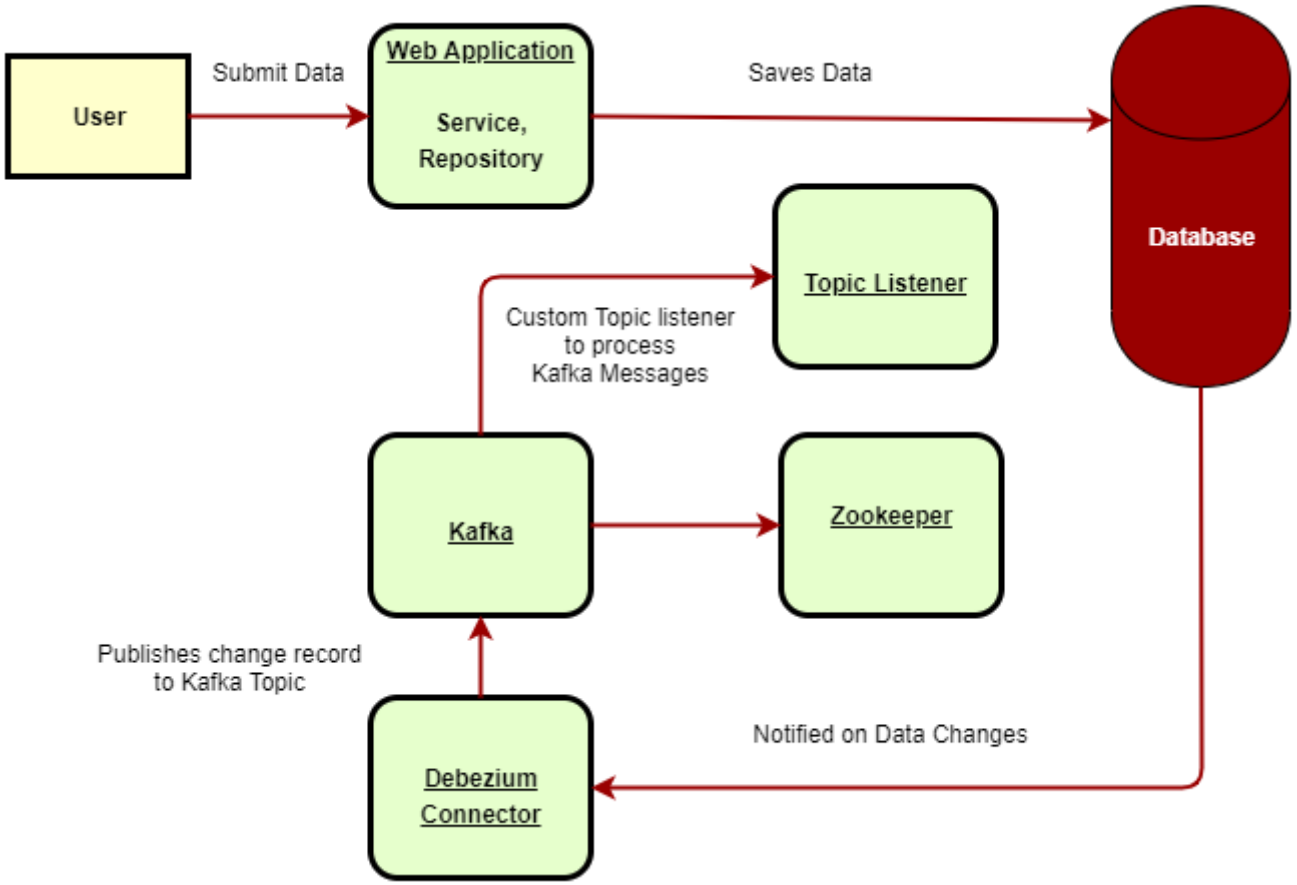
Software Setup

There are multiple options to setup the entire environment. You can either use Docker to kickstart entire stack within few minutes. In my case, I had to install each component individually. So we are going to follow plain vanilla installation.

This is the most simplest part of the entire setup ☺

1. Download JDK from – https://jdk.java.net/13/
2. Download kafka binary from – https://kafka.apache.org/downloads
3. Download Debezium connector plugin (PostgreSQL) – https://debezium.io/releases/
4. Download or follow PostgreSQL setup from – https://www.postgresql.org/download/

Unzip all the files at an appropriate place and make sure to configure JAVA_HOME environment variable pointing to JDK location.



Logical representation of Kafka, Debezium and PostgreSQL deployment

Configuration

The next step will focus on creating configuration files. The configuration includes feeding database details to Debezium connector, connect standalone mode, kafka and zookeeper. To begin with, lets have a look at Debezium connector for Postgres.

connector-postgres.properties

Create a folder inside your home directory as **conf**. We will create all of configuration files under this folder. This will help us to keep configurations at one place.

Create a properties file and name it as **connector-postgres.properties**. Copy below properties in the file and change it as per your environment.

```
name=postgres-connector
connector.class=io.debezium.connector.postgresql.PostgresConnector
database.hostname=xxx.xxx.xxx.xxx
database.port=5432
database.user=my_db_user
database.password=my_db_password
database.dbname=my_db_name
database.server.name=db_logical_name
plugin.name=pgoutput
table.whitelist=my_schema.table_name
errors.log.enable=true
errors.logs.include.messages=true
```

Some of the notable properties are

- **connector.class** – Defines which connector to be used.
- **plugin.name** – In With PostgreSQL 9.5 onwards you can leverage pgoutput instead of decoderbuf. If you are using older version of PostgreSQL, please refer to the Debezium documentation.
- **table.whitelist** – Defines the list of tables (comma separated) which should be observed for data changes.

connector-standalone.properties

Create a properties file to start Kafka in standalone mode. Following property defines the location of file, which Postgres Connector will use to store DB record offset.

offset.storage.file.filename=/tmp/connect.offsets

Add one more property to indicate where the Debezium Postgres connector can be located. Note that this folder should contain all the extracted files form Debezium zip file.

plugin.path=/home/ubuntu/software/connect-plugins

Zookeeper

We will use pre-packaged zookeeper available inside Kafka and there is no need to change the default properties. Refer to the following script to start Zookeeper with default options.

~/kafka/bin/zookeeper-server-start.sh ~/kafka/config/zookeeper.properties

Kafka

Next we start Kafka with default options and redirect the output stream to a log file.

~/kafka/bin/kafka-server-start.sh ~/kafka/config/server.properties > ~/kafka/logs/kafka.log

Kafka Connect (Standalone)

Once zookeeper and Kafka starts successfully, we must proceed with Kafka connect. Kafka connect can be bootstrapped either as a Standalone or in a Distributed mode. Following command can be used to start Kafka connect in standalone mode.

~/kafka/bin/connect-standalone.sh ~/sconf/connect-standalone.properties ~/conf/kafka-postgres.properties

The first argument indicates the shell script to start Kafka connect. You will find another script to start Kafka in distributed mode as well. The second argument defines the configuration to be applied while starting Kafka connect in standalone mode. The last argument provides details about the Debezium connector.

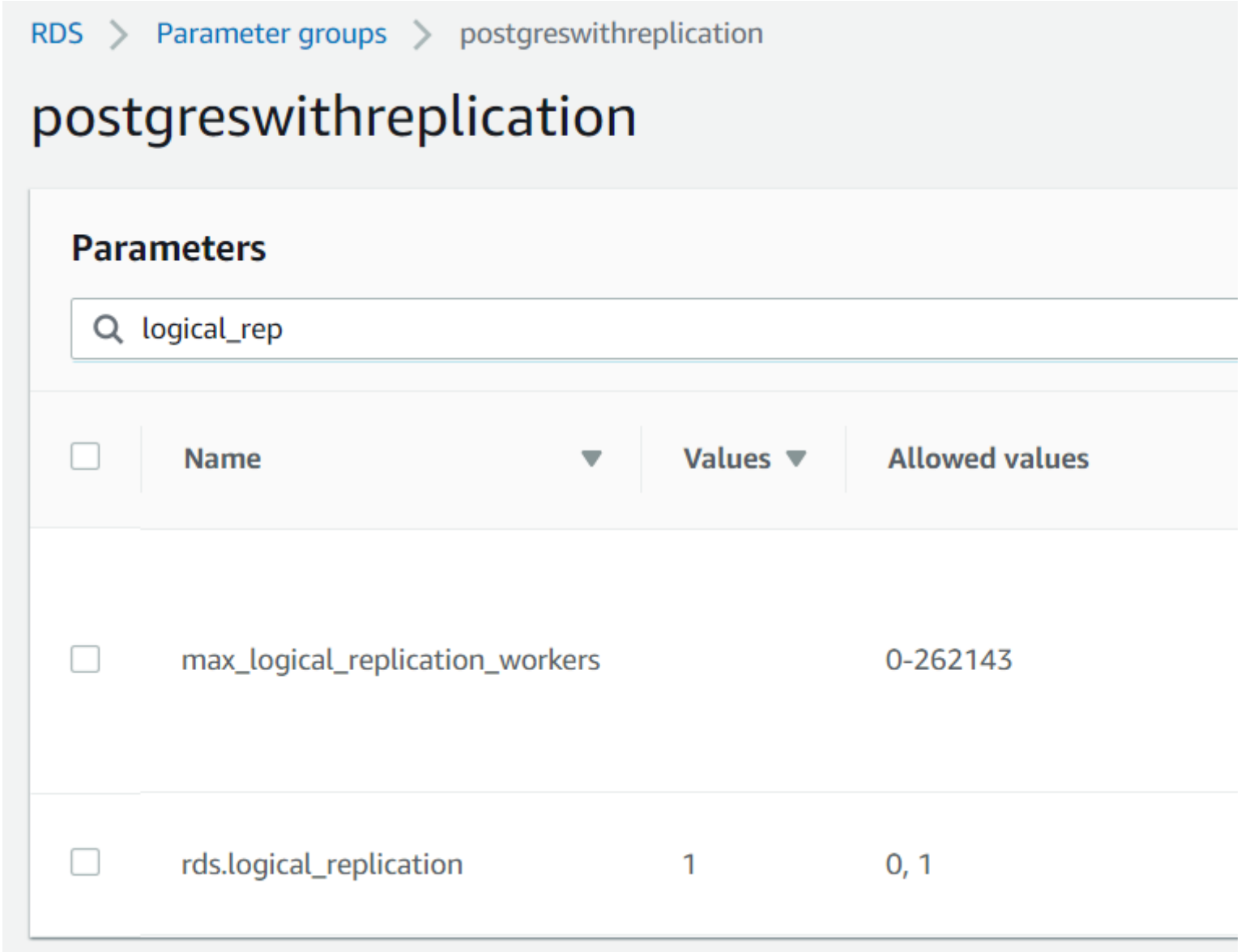
Kafka Topic Listener

This is the final step and purely depends on how you wish to process the Kafka messages. Following code snippet demonstrates Java Spring Boot Component to listen Kafka Topic

```
@Component
public class GlobalJobHandler {
@KafkaListener(topics = “db_logical_name.my_schema.table_name”, groupId = “GROUP_01”)
public void listen(@Payload(required = false)String message) {
}
}
```

AWS RDS – Postgres (Optional)

In case you are using AWS RDS – PostgreSQL service for managing your database, the steps are pretty simple. Before creating a database, create a new Parameter group and change the logical_replication value to 1.



Use this parameter group while creating a database. Also ensure that the database authentication is configured to use IAM roles. Since RDS doesn't allow superuser setup, you will have to provide credentials of main user in Postgres-connector.properties file (check above).