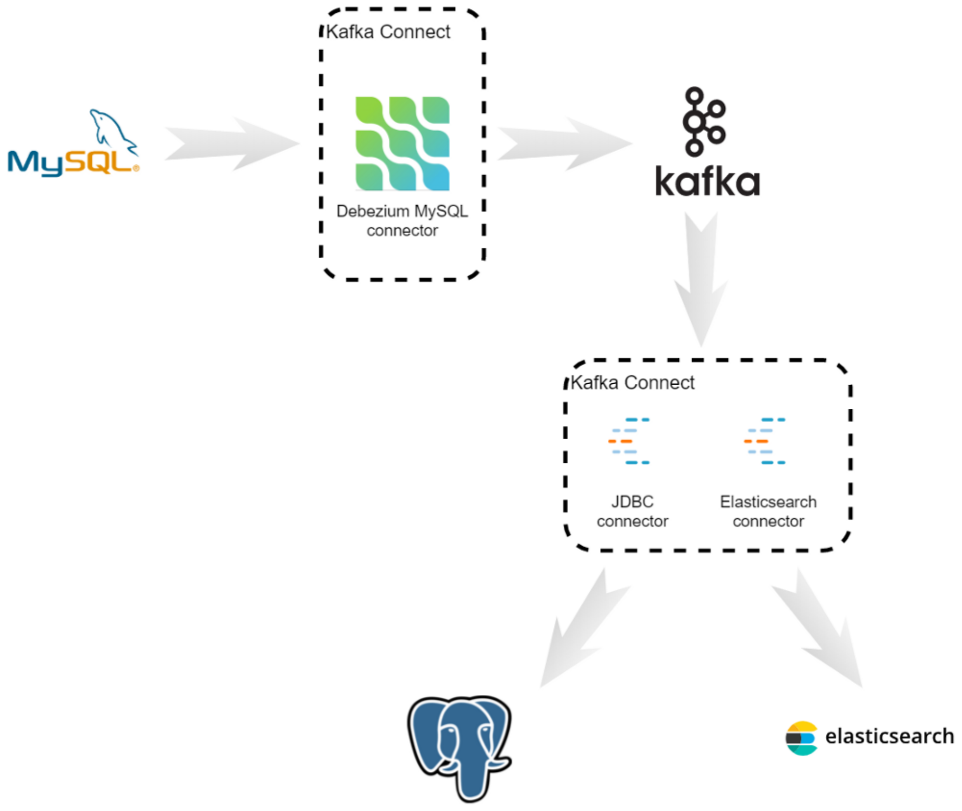
MySQL CDC with Apache Kafka and Debezium

Real-time change replication with Kafka and Debezium

Kushal Yellam
Aug 19, 2019 · 5 min read



MySQL CDC with Apache Kafka and Debezium Architecture

Overview

Debezium is a CDC (Change Data Capture) tool built on top of Kafka Connect that can stream changes in real-time from MySQL, PostgreSQL, MongoDB, Oracle, and Microsoft SQL Server into Kafka, using Kafka Connect.

Debezium records historical data changes made in the source database to Kafka logs, which can be further used in a Kafka Consumer. We can then easily capture the changes via the Kafka Consumer. In case of abrupt stopping or crashing of application or

Debezium, when restarted the data gets consumed to form the last recorded offset, resulting in zero data loss.

- Debezium's MySQL connector reads MySQL's binary log to understand the changes in data sequentially. A *change event* for every row-level operation like insert, update and delete is produced and recorded to a separate Kafka topic.

Kafka Connect is used for streaming data between Apache Kafka and other external systems. It is used to define *connectors* that move large collections of data in and out of Kafka.

The software versions used here are:

- Apache Kafka 2.1
- Zookeeper 3.4
- Debezium 0.9.2
- MySQL 5.7

Install Debezium

Download ***debezium-connector-mysql-0.9.2-plugin.tar.gz jar*** from link.

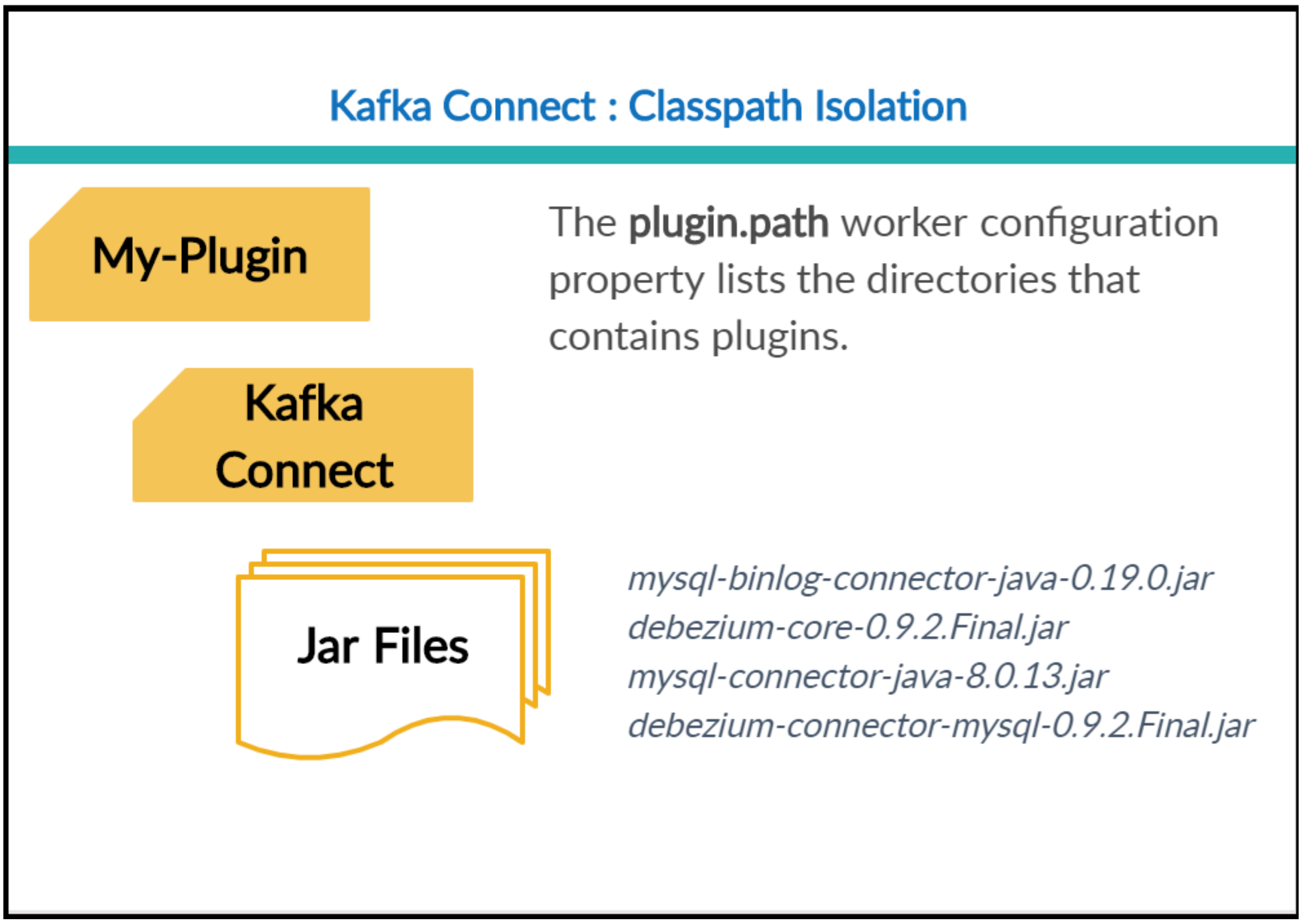
```
Unpack the .tar.gz into a separate folder, for eg:
/user/connector/plugins so that you have:

/user/connector/plugins/debezium-connector-mysql/mysql-binlog-
connector-java-0.19.0.jar

/user/connector/plugins/debezium-connector-mysql/debezium-core-
0.9.2.Final.jar

/user/connector/plugins/debezium-connector-mysql/mysql-connector-
java-8.0.13.jar

/user/connector/plugins/debezium-connector-mysql/debezium-connector-
mysql-0.9.2.Final.jar
```

The Kafka Connect worker config should be updated in order to put the required Debezium jars in Kafka Connect classpath.

Edit $KAFKA_HOME/config/connect-distributed.properties and update the plugin.path property with the location of the Debezium jar. eg:

```
plugin.path=/user/connector/plugins
```

plugin.path is based on this structure:


Kafka Connect : Classpath Isolation
My-Plugin
Kafka Connect
Jar Files
The **plugin.path** worker configuration property lists the directories that contains plugins.
mysql-binlog-connector-java-0.19.0.jar
debezium-core-0.9.2.Final.jar
mysql-connector-java-8.0.13.jar
debezium-connector-mysql-0.9.2.Final.jar

Kafka Connect : Classpath Isolation

After updating the plugin.path, restart Kafka and Kafka Connect so that it picks up the new Debezium jars.

MySQL config

Debezium uses MySQL's bin-log to track all the events, hence the bin-log should be enabled in MySql. Here are the steps to enable bin-log in MySQL.

1. Check the current state of bin-log activation:

```
$ mysqladmin variables -uroot|grep log_bin

| log_bin | OFF
```

To enable the bin-log we need to modify my.cnf, which is the configuration file for MySQL. The location for my.cnf may vary from system to system, for me it is /usr/local/opt/mysql/my.cnf.

2. Add/ Modify the below properties in my.cnf.

```
[mysqld]

server-id = 42

log_bin = mysql-bin

binlog_format = row

binlog_row_image = full

expire_logs_days = 10
```

3. Restart MySQL in order for the changes to take place.

```
sudo service mysql restart
```

We can verify the bin-log status via the following command.

```
$ mysqladmin variables -uroot|grep log_bin

| log_bin | ON
```

In order for Debezium to track all the changes in bin-log, it needs a user with SELECT, RELOAD, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT permissions. Grant them to the user you intend to use for Debezium.

```
$ mysql -uroot
```

```
mysql> GRANT SELECT, RELOAD, SHOW DATABASES, REPLICATION SLAVE,
REPLICATION CLIENT ON *.* TO ‘debezium’ IDENTIFIED BY ‘dbz’;
```

Kafka Connect setup

The connector configuration can be loaded into Kafka Connect via the REST API:

```
curl -k -X POST -H “Accept:application/json” -H “Content-
Type:application/json” https://localhost:8083/connectors/ -d
‘{“name”:”mysql-connector-
demo”,“config”{“connector.class”:”io.debezium.connector.mysql.MySqlC
onnector”,“database.hostname”:”localhost”,“database.port”:”3306",“da
tabase.user”:”debezium”,“database.password”:”********”,“database.ser
ver.id”:”1",“database.server.name”:”MySQL-Database-
Docker”,“database.history.kafka.bootstrap.servers”:”localhost:9092",
“database.history.kafka.topic”:”dbhistory.demo”,“include.schema.chan
ges”:”true”,“table.whitelist”:”<database_name>.<table_name>”}}’
```

Now check that the connector is running successfully:

```
List all active connectors: curl -k
https://localhost:8083/connectors/

Check connector specific status: curl -k
https://localhost:8083/connectors/<CONNECTOR_NAME>/status
```

Use JQ for beautified output.

```
curl -s “http://localhost:8083/connectors" | jq ‘.[]’ | \

xargs -I{connector_name} curl -s “http://localhost:8083/connectors/"
{connector_name}"/status" | \

jq -c -M ‘[.name,.connector.state,.tasks[].state] | \

join(“:|:”)’| column -s : -t| sed ‘s/\”//g’| sort

mysql-connector | RUNNING | RUNNING
```

If the connector is found to be in a FAILED state, then we can check the Connector worker logs for errors. One of the common reasons for this error can be the incorrect path to the JAR files. The unavailability of Kafka Connect brokers may also result in a failure.

If the connector is in RUNNING state, we can see the Connect worker logs as follows, which indicates the successful ingestion of data from MySQL to Kafka.

```
[2019–08–09 16:27:40,268] INFO Starting snapshot for
jdbc:mysql://localhost:3306/?
useInformationSchema=true&nullCatalogMeansCurrent=false&useSSL=false
&useUnicode=true&characterEncoding=UTF-8&characterSetResults=UTF-
8&zeroDateTimeBehavior=convertToNull with user 'debezium'
(io.debezium.connector.mysql.SnapshotReader:220)

[…]

[2019–08–09 16:27:57,297] INFO Step 8: scanned 97354 rows in 24
tables in 00:00:15.617
(io.debezium.connector.mysql.SnapshotReader:579)

[2019–08–09 16:27:57,297] INFO Step 9: committing transaction
(io.debezium.connector.mysql.SnapshotReader:611)

[2019–08–09 16:27:57,299] INFO Completed snapshot in 00:00:17.032
(io.debezium.connector.mysql.SnapshotReader:661)
```

THE NAMING CONVENTION FOR KAFKA TOPICS

All the change events are stored in a separate topic per table by Debezium. If the connector was configured using the following parameters:

```
"database.dbname": "database_name",

"database.server.name": "database_server_name",
```

The change events for the **cdc_test** table will be stored in a Kafka topic which is of the format `<hostname>.<databasename>.<tablename>`. i,e in this case database_server_name.database_name.cdc_test.

We can list all the Kafka topics using the following command:

```
kafka–topics –zookeeper localhost:2181 –list
```

Each **table** in the database becomes a separate **topic** in Kafka containing one partition by default:

```
MySQL–Database–Docker.database_name.user

MySQL–Database–Docker.database_name.address

MySQL–Database–Docker.database_name.category
```

Now let's look at the messages. Each **event** becomes a **message** on a table-specific Kafka topic.

We can listen to these messages using the kafka-console-consumer.sh

```
$KAFKA_HOME/bin/kafka–console–consumer.sh –bootstrap–server
"loclahost:9092" –topic "MySQL–Database–Docker.database_name.user" –
from–beginning
```

After running the console consumer, insert a record in the MySQL table to see the messages in the console consumer :

```
mysql> INSERT INTO `users` VALUES (2, 'John Doe');
```

The records from Debezium have the following JSON structure:

```
{"schema":{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"user_id"},
{"type":"string","optional":false,"field":"user_name"}],"optional":t
rue,"name":"_0.0.0.160.Debezium_test.users.Value","field":"before"},
{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"user_id"},
{"type":"string","optional":false,"field":"user_name"}],"optional":t
rue,"name":"_0.0.0.160.Debezium_test.users.Value","field":"after"},
{"type":"struct","fields":
[{"type":"string","optional":true,"field":"version"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"int64","optional":false,"field":"ts_sec"},
{"type":"string","optional":true,"field":"gtid"},
{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},
{“type”:”int32",”optional”:false,”field”:”row”},
{“type”:”boolean”,”optional”:true,”default”:false,”field”:”snapshot”
},{“type”:”int64",”optional”:true,”field”:”thread”},
{“type”:”string”,”optional”:true,”field”:”db”},
{“type”:”string”,”optional”:true,”field”:”table”},
{“type”:”string”,”optional”:true,”field”:”query”}],”optional”:false,
”name”:”io.debezium.connector.mysql.Source”,”field”:”source”},
{“type”:”string”,”optional”:false,”field”:”op”},
{“type”:”int64",”optional”:true,”field”:”ts_ms”}],”optional”:false,”
name”:”_0.0.0.160.Debezium_test.users.Envelope”},”payload”:
{“before”:null,”after”:{“user_id”:2,”user_name”:”John
Doe”},”source”:
{“version”:”0.8.2",”name”:”10.0.0.160",”server_id”:0,”ts_sec”:0,”gti
d”:null,”file”:”bin.000035",”pos”:120,”row”:0,”snapshot”:true,”threa
d”:null,”db”:”database”,”table”:”users”,”query”:null},”op”:”c”,”ts_m
s”:1549018495125}}
```

Note the structure of the message — we have a before and after field of the record, plus a bunch of metadata (source, op, ts_ms). Depending on what you’re using the CDC events for, you’ll want to retain some or all of this metadata.

A message is a combination of two metadata: Schema and Payload. The payload consists of 2 parts: before and after, which contains the previous and current record in case of an update operation in MySQL.

We can pick up the current record from the ‘after’ payload.

```
{“after”:{“user_id”:2,”user_name”:”John Doe”},”source”:
{“version”:”0.8.2",”name”:”10.0.0.160",”server_id”:0,”ts_sec”:0,”gti
d”:null,”file”:”bin.000035",”pos”:120,”row”:0,”snapshot”:true,”threa
d”:null,”db”:”database”,”table”:”users”,”query”:null},”op”:”c”,”ts_m
s”:1549018495125}
```

In the case of SSL encrypted Kafka and Kafka Connect environment, the following command is used to register the connector.

```
curl –k –X POST –H “Accept:application/json” –H “Content-
Type:application/json” https://localhost:8083/connectors/ –d
‘{“name”:”DEMO-SSL–MySQL–CONNECTOR”,“config”:
{“connector.class”:”io.debezium.connector.mysql.MySqlConnector”,“dat
abase.hostname”:”localhost”,“database.port”:”3306",“database.user”:”
debezium”,“database.password”:”********”,“database.server.id”:”1",“d
atabase.server.name”:”DEMO-
CONNECTOR”,“database.history.kafka.bootstrap.servers”:”localhost:909
2",“database.history.kafka.topic”:”dbhistory.demo”,“include.schema.c
hanges”:”true”,“table.whitelist”:”database_name.table_name”,“databas
e.history.producer.sasl.mechanism":"PLAIN","database.history.produce
r.security.protocol":"SASL_SSL","database.history.producer.ssl.key.p
assword":"********","database.history.producer.ssl.keystore.location
":"/tmp/client.jks","database.history.producer.ssl.keystore.password
":"******","database.history.producer.ssl.truststore.location":"/tmp
/client.jks","database.history.producer.ssl.truststore.password":"**
******"}}'
```

Kafka commands to invoke the console producer and consumer in case of SSL enabled Kafka Cluster.

```
 — Create Consumer

$KAFKA_HOME/bin/kafka-console-consumer.sh —bootstrap-server
localhost:9092 —topic DEMO-CONNECTOR.database_name.table_name —
consumer.config ~/ssl.properties —from-beginning

 — Create Producer

$KAFKA_HOME/bin/kafka-console-producer.sh —broker-list
localhost:9092 —topic DEMO-CONNECTOR.database_name.table_name —
producer.config ~/ssl.properties
```

Conclusion

We set up a Streaming data pipeline to get the data in real-time from a MySQL database to Kafka, which can be achieved using Kafka Connect and Debezium MySQL connector. As this connector works in a streaming fashion, the changes committed to MySQL database are captured in real-time.

Thank you!

References

Tutorial for Debezium 0.9

2017-09-21 07:24:01,628 INFO MySQL|dbserver1|snapshot Step 0: disabling auto-commit and enabling repeatable read…

debezium.io

Debezium Connector for MySQL

Debezium and Kafka Connect are designed around continuous streams of event messages, and the structure of these events…

debezium.io

Kafka Connect - Confluent Platform

Kafka Connect, an open-source component of Kafka, is a framework for connecting Kafka with external systems such as...

docs.confluent.io

Big Data Apache Kafka Debezium Kafka Connect Kafka Streams