# An exercise in Discovery, Streaming data in the analytical world.

## Confluent Kafka, Apache Flink + Flink CDC from PostgreSql & MySql => Apache Paimon, Apache Avro with a Apache Hive Metastore

(? Sept 2024 - Part 6+1)

**Overview**

So, this is a continuation of the previous series. Totally unplanned but honestly not unexpected.

I had a request from a good friend to mix up the inbound data source a bit…

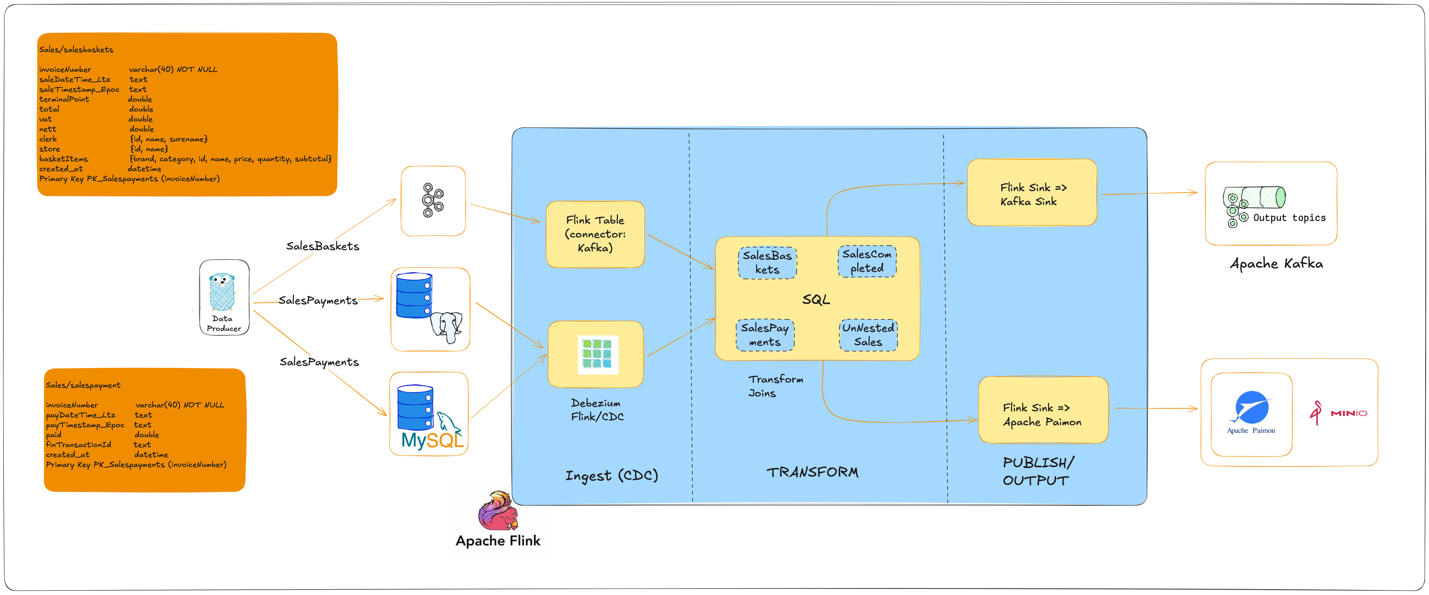
Instead of generating all the data on the [Golang](https://go.dev/) Application and then posting that onto a Kafka topic from where it goes into all directions… Let’s face it, not all your data is going to come from one nicely shrink-wrapped source, it is all over the place… so ye we’re simulating that with this slight modification.

What we’re doing here is splitting that inbound stream. We still have our imaginary shop, creating SalesBaskets and SalesPayments, however the SalesBaskets will go onto a [Confluent](https://www.confluent.io/) Kafka Topic and the SalesPayments will now go into either [Oracle MySql](https://www.mysql.com/) or [PostgreSql](https://www.postgresql.org/) database/table.

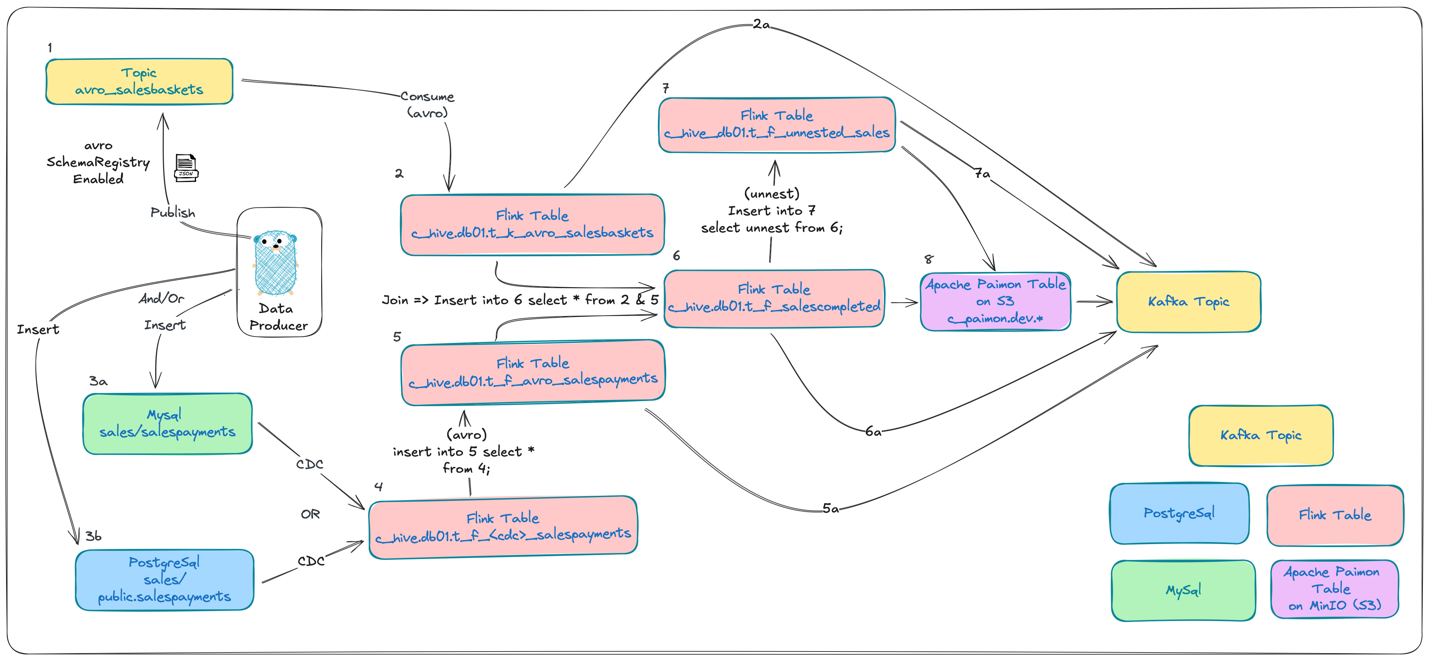
From the [Confluent](https://www.confluent.io/) Kafka Topic (avro\_salesbaskets) it will be pulled into [Apache Flink](https://flink.apache.org/) via a Apache Flink table definition and from either of the [Mysql](https://www.mysql.com/) or [PostgreSql](https://www.postgresql.org/) database, (the database called sales and the table: salespayments) the data will be source via [Apache Flink CDC](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/get-started/introduction/) (Change Data Capture) process utilizing the build in [Debezium](https://debezium.io/) capability.

The following two diagrams depicts the above flow described.

High Level



And a more detail view:



Two new componets that come into play in this blog series, CDC processing and [Apache Paimon.](https://paimon.apache.org/)

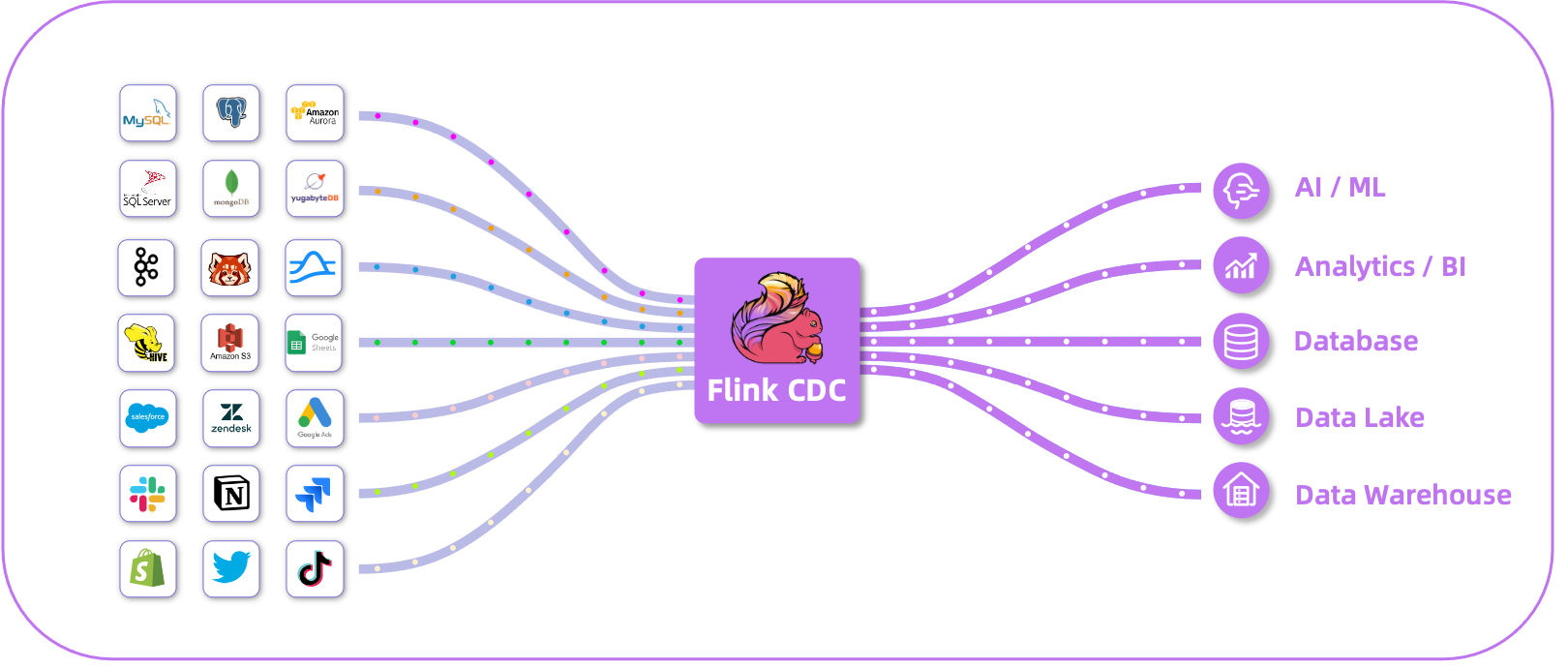
See below for CDC and see part 4 for a discussion regarding [Apache Paimon](https://paimon.apache.org/).

**But you ask, What is (CDC) Change Data Capture?**

Change data capture (CDC) refers to the process of identifying and capturing changes made to data in a database and then delivering those changes in real-time to a downstream process or system.

And as per [Wikipedia](https://en.wikipedia.org/wiki/Change_data_capture).

The [below diagram](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/connectors/flink-sources/overview/) at a very simple level shows how rich this [Apache Flink CDC](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/get-started/introduction/) Source/Target capability is in fact.

[](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/connectors/flink-sources/overview/)

**NOTE**: As discovered, the CDC tech is very very dependent on the various versions of the databases used and the associated JAR libraries deployed inside the $FLINK\_HOME/lib directory enabling all of this. See [Supported Connectors](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/connectors/flink-sources/overview/) for a Compatibility matrix.

Where you find it will be determined by what version of the Blog you get to, ha ha ha… Originally it was planned to be [MySql](https://www.mysql.com/), simply because there was already allot of [PostgreSql](https://www.postgresql.org/) databases in the blog.

But then I discovered one of the alternate tooling utilities I wanted to potentially also show does not produce/output to MySql database currently so in comes an PostgreSql datastore, yet another one, the 3rd Pg database, but honestly, I do think it has merit to be standalone.

The Original blog series can be found [here](https://medium.com/@georgelza/an-exercise-in-discovery-streaming-data-in-the-analytical-world-part-1-e7c17d61b9d2), and the associated GIT repo is [here](https://github.com/georgelza/MongoCreator-GoProducer-avro.git).

This rabbit hole will stand on its own from a code viewpoint and the GIT repo is [here](https://github.com/georgelza/split-sources-in-streaming-world.git).

NOTE: I added a field (subtotal) to the Salesbaskets document in the basketitems array of documents, should you be comparing it to the original basket,

The Stack is again comprised out of:

* Confluent Kafka stack
* Apache Flink
* Apache Flink CDC
* Apache Hive Metastore as Catalog
* Apache Paimon as Open table Format
* Apache Avro as Open File Format
* MySql as datasource for Flink CDC
* PostgreSql as Datastore for Hive Metastore & as Datasource for Flink CDC
* MongoDB as a possible Kafka Sink destination via defined Connector
* Docker
* Docker Compose
* Lots of Makefiles

If you keen to explore Docker, Docker Compose and Makefiles see: [An exercise in Discovery, Building Docker Images, using Makefiles & Docker Compose](https://medium.com/@georgelza/an-exercise-in-discovery-building-docker-images-using-makefiles-docker-compose-part-1-ed89f3546da7)

NOTE: In this project I further mixed it up a bit in the Dockerfiles. I introduced a repo\_name variable pulled in using the arguments primitive, allowing the entire build to be pulled into a personal hub.docker.com account more easily, see the various .env files for the repo\_owner variable.

As it stands this will be a 5-part posting, but it’s by no means complete.

Good luck, this is all fraught with rabbit holes, as always, so many and you can disappear so easily…



**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

------------------------------------------------------------------------------------------------------------------

# An exercise in Discovery, Streaming data in the analytical world.

## Confluent Kafka, Apache Flink + Flink CDC from PostgreSql & MySql => Apache Paimon, Apache Avro with a Apache Hive Metastore Catalog

(? Sept 2024 - Part 2)

**Overview**

See devlab/sql-client/Dockerfile for which jar files to include in image build.

I then build my flink image with the same JAR files. See: devlab/flink/Dockerfile.

Once our above two images are built, we can turn to preparing our database/s. To accomplish this, see the sql/mysqlcdc and sql/postgrescdc directories for the 2 scripts (mysql-init.sql & postgresql-init.sql) executed during the database create during the spin up of the mysqlcdc and postgrescdc containers.

NOTES:

General

* it’s important to include a primary key on your tables in both MySql, PostgreSQL and inside your Apache Flink table definitions.

MySqlDB

* For MySql make sure your database is in binary log mode:
  + See: [General FAQ](https://nightlies.apache.org/flink/flink-cdc-docs-master/docs/faq/faq/)
* Make sure to set: SET GLOBAL binlog\_expire\_logs\_seconds = 60000;
  + Ignore the comments to set the following 2 settings, they being depreciated:

show variables like 'expire\_logs\_days';

set global expire\_logs\_days=7;

* Permissions (for our flinkcdc user defined in the create table executed via mysql-init.sql script is defined when we created the user/role via the mysql-init.sql script at container create time) and postgresql-init.sql for PostgreSql.

I originally used the below:

GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'flinkcdc'@'sales';

Discovered I needed to change to (basically the flinkcdc user need access to the database where the data is “sales” and also the “postgres” database as it needs to get to the write ahead logs and other postgres internals):

GRANT SELECT, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'flinkcdc'@'%';

[Additional note re MySqlDB setup for Replication](https://dev.mysql.com/doc/refman/8.4/en/replication-howto-repuser.html)

* And a BIG one… case matters…, let’s just say that again, CASE MATTERS. Your MySql Table/column names case need to match the case of the [Apache Flink](https://flink.apache.org/) table being created. – Lesson learned, maybe don’t put the column names in quotes, allowing the database to either lower case or upper case, and then match that in the Create Table in ‘creFlinkFlows/creCdc.sql’.

PostgreSQL

* Re your replication role/user, see [PostgreSql DB setup for Replication.](https://www.alibabacloud.com/help/en/flink/developer-reference/postgresql-cdc-connector/):
* Make sure ‘wal\_level='logical' is configured, this can be confirmed via the below steps.
  + First make sure you are starting up your database with the desired postgresql.conf file as per the configs section, to do this enter the container:

“docker compose exec -it <name> bash”

Ie:

docker compose exec -it postgrescdc bash

* + Then execute plsql cli utility:

psql -h localhost -p 5432 -U <user> -d <database>

SHOW config\_file;

It should respond with /etc/postgresql/postgresql.conf, we specify this in our docker-compose.yml using the configs section of the postgrescdc service.

If not then you can use the command primitive syntax in the postgrescdc service section to override the postgresql.conf file used by default and instruct postgres binary on startup using our desired config file.

[command: -c config\_file=/etc/postgresql/postgresql.conf](https://stackoverflow.com/questions/30848670/how-to-customize-the-configuration-file-of-the-official-postgresql-docker-image)

See the above URL embedded, scroll down to “With Docker Compose” on how this works.

Once you are now using the correct postgresql.conf file you can confirm the setting wal\_level setting by issuing ‘show wal\_level;’ in plsql cli tool as we used above.

[How to use the PostgreSql Docker Official image:](https://www.docker.com/blog/how-to-use-the-postgres-docker-official-image/)

PostgreSql was a bit more complicated in the end than I expected.

Was surprised when google told me the solution to some errors… but I was not using the tech as far as I was concerned.

IE: The advise was to configure “decoding.plugin.name”. But I nowhere saw [Debezium](https://debezium.io/) or configured anything. Then SLACK came to the saving and was informed that Apache Flink CDC does use [Debezium](https://debezium.io/) engine internally and that you can pass Debezium configuration values using debezium. as a prefix: See below.



The PostgreSQL permissions required is “complicated” to say the least.

See: devlab/sql/postgrescdc/postgresql-init.sql for the SQL used to create the flinkcdc user and the replicator role in the end to enable all of this to work.

Hopefully I will be able to figure out how to reduce the permissions signed to flinkcdc and replicator role. Not a fan of assigning “superuser” as an example.

Looking at the Apache Flink CDC site, ([How to create a Postgres CDC table](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/connectors/flink-sources/postgres-cdc/)) they show a source table.

CREATE TABLE shipments (

shipment\_id INT,

order\_id INT,

origin STRING,

destination STRING,

is\_arrived BOOLEAN

) WITH (

'connector' = 'postgres-cdc',

'hostname' = 'localhost',

'port' = '5432',

'username' = 'postgres',

'password' = 'postgres',

'database-name' = 'postgres',

'schema-name' = 'public',

'table-name' = 'shipments',

'slot.name' = 'flink',

-- experimental feature: incremental snapshot (default off)

'scan.incremental.snapshot.enabled' = 'true'

);

My working example ended as:

CREATE TABLE c\_hive.db01.t\_f\_pgcdc\_salespayments (

invoicenumber STRING

,paydatetime\_ltz STRING

,paytimestamp\_epoc STRING

,paid DOUBLE

,fintransactionid STRING

,created\_at TIMESTAMP

,PRIMARY KEY(invoicenumber) NOT ENFORCED

) WITH (

'connector' = 'postgres-cdc',

'hostname' = 'postgrescdc',

-- NOTE: this is the port of the db on the container, not the external docker exported port via a port mapping.

'port' = '5432',

'username' = 'flinkcdc',

'password' = <password>,

'database-name' = 'sales',

'schema-name' = 'public',

'table-name' = 'salespayments',

'slot.name' = 'flinkcdc',

-- experimental feature: incremental snapshot (default off)

'scan.incremental.snapshot.enabled' = 'true',

'scan.startup.mode' = 'latest-offset',

'decoding.plugin.name' = 'pgoutput'

);

Primarily look at the bits in RED, the most important differences being the addition of decoding.plugin.name=pgoutput

And there we thought this was a block about CDC… Just a little side track on how to setup MySql and PostgreSQL databases.

Also useful during this trip down this rabbit hole:

* [PostgreSQL: When wal\_level to logical](https://www.dbi-services.com/blog/postgresql-when-wal_level-to-logical/)
* [Understanding Replication in PostgreSQL – How to Set Up PostgreSQL Streaming Replication](https://www.percona.com/blog/setting-up-streaming-replication-postgresql/#:~:text=PostgreSQL%20streaming%20replication%20is%20a,mirror%20the%20primary%20database%20accurately.)
* [PostgreSQL CDC Connector](https://www.alibabacloud.com/help/en/flink/developer-reference/postgresql-cdc-connector/)

At this point if I insert records into either the MySql: sales/salespayment table or PostgreSQL: sales/public.salespayments table I am able to see that inside flink-sql-client when I run:

select \* from c\_hive.db01.t\_f\_msqlcdc\_salespayments

or

select \* from c\_hive.db01.t\_f\_pgcdc\_salespayments;

We will stop here for today. The biggest part of this is actually now done.

We have two working CDC streams flowing into our [Apache Flink](https://flink.apache.org/) environment. See you in the next instalment.

**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

------------------------------------------------------------------------------------------------------------------

# An exercise in Discovery, Streaming data in the analytical world.

## Confluent Kafka, Apache Flink + Flink CDC from PostgreSql & MySql => Apache Paimon, Apache Avro with a Apache Hive Metastore Catalog

(? Sept 2024 - Part 6+3a)

**Overview**

At this point we’re going to go back to what we did in the original article.

First, as we created salespayments table as a CDC source above we will now push this data stream back into a Kafka by creating a new [Apache Flink](https://flink.apache.org/) table configured to output to a Kafka topic. The data will be pushed into this table utilizing a “select <columns> from <the source cdc table>.

And here is where things got interesting….

This is our MySQL CDC sourced table:

CREATE TABLE c\_hive.db01.t\_f\_msqlcdc\_salespayments (

`invoiceNumber` STRING

,`payDateTime\_Ltz` STRING

,`payTimestamp\_Epoc` STRING

,`paid` DOUBLE

,`finTransactionId` STRING

,`created\_at` TIMESTAMP

,PRIMARY KEY(`invoiceNumber`) NOT ENFORCED

) WITH (

'connector' = 'mysql-cdc'

,'hostname' = 'mysqlcdc'

,'port' = '3306'

,'username' = 'flinkcdc'

,'password' = 'flinkpw'

,'database-name' = 'sales'

,'table-name' = 'salespayments'

,'scan.startup.mode' = 'latest-offset'

);

We then create the target Apache Flink table as:

CREATE OR REPLACE TABLE c\_hive.db01.t\_f\_avro\_salespayments (

`invoiceNumber` STRING

,`payDateTime\_Ltz` STRING

,`payTimestamp\_Epoc` STRING

,`paid` DOUBLE

,`finTransactionId` STRING

,`created\_at` TIMESTAMP(3)

,`payTimestamp\_WM` AS TO\_TIMESTAMP(FROM\_UNIXTIME(CAST(`payTimestamp\_Epoc` AS BIGINT) / 1000)),

WATERMARK FOR `payTimestamp\_WM` AS `payTimestamp\_WM`

,PRIMARY KEY (`invoiceNumber`) NOT ENFORCED

) WITH (

'connector' = 'upsert-kafka'

,'topic' = 'avro\_salespayments'

,'properties.bootstrap.servers' = 'broker:29092'

,'properties.group.id' = 'testGroup'

,'value.format' = 'avro-confluent'

,'value.avro-confluent.schema-registry.url' = 'http://schema-registry:9081'

,'key.format' = 'avro-confluent'

,'key.avro-confluent.schema-registry.url' = 'http://schema-registry:9081'

,'value.fields-include' = 'ALL'

);

And now populate the [Apache Flink](https://flink.apache.org/) table: salespayments with the data from the CDC stream using the below statement.

Couple of things that we had to changed.

* The connector changed to upsert-kafka
* We needed to add a primary key not enforced.
* We needed to add the two key.\* variables
* We needed to remove “scan.startup.mode”
* And oh… we needed to check our target topic schema, in this version we added the created\_at column, so ye… had to add that to our avro schema salespayments.avsc file.

NOTE, in our code you will notice that the PostgreSQL table columns have been created all as lower case… as said previously, CASE matters, which had the ripple effect of impacting the select <columns> from PostgreSQL CDC source vs the MySQL CDC source. This is a exploration on discovery… Showing lessons learn (well those I remembered to write down and which won’t make the blog to confusing), things to think about and why. I could very easily just have gone and lower cased everything and not have demonstrated the “small” impact CASE has.

I chose to show how decisions or lack of knowledge/agreement/standard early on can have impact later on.

* MySqlDB

INSERT INTO c\_hive.db01.t\_f\_avro\_salespayments (

`invoiceNumber`

,`payDateTime\_Ltz`

,`payTimestamp\_Epoc`

,`paid`

,`finTransactionId`

,`created\_at`

) SELECT

invoiceNumber

,payDateTime\_Ltz

,payTimestamp\_Epoc

,paid

,finTransactionId

,created\_at

FROM

c\_hive.db01.t\_f\_msqlcdc\_salespayments;

* PostgreSql

INSERT INTO c\_hive.db01.t\_f\_avro\_salespayments (

`invoiceNumber`

,`payDateTime\_Ltz`

,`payTimestamp\_Epoc`

,`paid`

,`finTransactionId`

,`created\_at`

) SELECT

invoicenumber

,paydatetime\_ltz

,paytimestamp\_epoc

,paid

,fintransactionid

,created\_at

FROM

c\_hive.db01.t\_f\_pgcdc\_salespayments;

And at this point I’m going to post this image...

Play on words… Jar Jar Binx… Jar files…



Next up (was planned) , we’re going to join the salesbasket stream and the salespayments stream to create salescompleted. You will notice this table, when we created it we already configured it to be pushing data back to Kafka. We will be using out trusty insert into <column list> select <column list> from <source table> as previous.

As hinted in 3a, planned on showing how to do the join of the 2 tables into salescompleted, but as sarcastically depicted, [Jar Jar Binks,](https://en.wikipedia.org/wiki/Jar_Jar_Binks) jars jars jars…came visting.

As the character use to (in Star Wars movie series)… a simple turn of the head and the world comes tumbling down, due to those ears. For us those ears are jar’s… a small change and ye, things are tumbling…

At this point, I was eventually able to execute the Insert statement, and got my data flow appearing in my Kafka topic: avro\_salespayments. Good…

**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

------------------------------------------------------------------------------------------------------------------

# An exercise in Discovery, Streaming data in the analytical world.

## Confluent Kafka, Apache Flink + Flink CDC from PostgreSql & MySql => Apache Paimon, Apache Avro with a Apache Hive Metastore Catalog

(? Sept 2024 - Part 6+3b)

**Overview**

As hinted, next up we’re showing how to do the join of the 2 tables into salescompleted, but as sarcastically depicted, [Jar Jar Binks,](https://en.wikipedia.org/wiki/Jar_Jar_Binks) jars jars jars…

As the character use to (in Star Wars movie series)… a simple turn of the head and the world comes tumbling down, due to those ears. For us those ears are jar’s… a small change and ye, things are tumbling…

Following this we created out output join table salescompleted and then inserted records using the below.

CREATE OR REPLACE TABLE c\_hive.db01.t\_f\_avro\_salescompleted (

`invoiceNumber` STRING

,`saleDateTime\_Ltz` STRING

,`saleTimestamp\_Epoc` STRING

,`terminalPoint` STRING

,`nett` DOUBLE

,`vat` DOUBLE

,`total` DOUBLE

,`store` row<`id` STRING, `name` STRING>

,`clerk` row<`id` STRING, `name` STRING, `surname` STRING>

,`basketItems` array<row<`id` STRING, `name` STRING, `brand` STRING, `category` STRING, `price` DOUBLE, `quantity` INT, `subtotal` DOUBLE>>

,`payDateTime\_Ltz` STRING

,`payTimestamp\_Epoc` STRING

,`paid` DOUBLE

,`finTransactionId` STRING

,`payTimestamp\_WM` AS TO\_TIMESTAMP(FROM\_UNIXTIME(CAST(`payTimestamp\_Epoc` AS BIGINT) / 1000))

,`saleTimestamp\_WM` AS TO\_TIMESTAMP(FROM\_UNIXTIME(CAST(`saleTimestamp\_Epoc` AS BIGINT) / 1000))

,WATERMARK FOR `saleTimestamp\_WM` AS `saleTimestamp\_WM`

,PRIMARY KEY (`invoiceNumber`) NOT ENFORCED

) WITH (

'connector' = 'upsert-kafka'

,'topic' = 'avro\_salescompleted'

,'properties.bootstrap.servers' = 'broker:29092'

,'value.format' = 'avro-confluent'

,'value.avro-confluent.schema-registry.url'= 'http://schema-registry:9081'

,'key.format' = 'raw'

,'properties.group.id' = 'mysqlcdcsourced'

,'value.fields-include' = 'ALL'

);

INSERT INTO c\_hive.db01.t\_f\_avro\_salescompleted

SELECT

b.invoiceNumber

,b.saleDateTime\_Ltz

,b.saleTimestamp\_Epoc

,b.terminalPoint

,b.nett

,b.vat

,b.total

,b.store

,b.clerk

,b.basketItems

,a.payDateTime\_Ltz

,a.payTimestamp\_Epoc

,a.paid

,a.finTransactionId

FROM

c\_hive.db01.t\_f\_avro\_salespayments a,

c\_hive.db01.t\_k\_avro\_salesbaskets b

WHERE a.invoiceNumber = b.invoiceNumber

AND a.created\_at > b.saleTimestamp\_WM

AND b.saleTimestamp\_WM > (b.saleTimestamp\_WM - INTERVAL '1' HOUR);

For various aggregations we want to run we need to unnest the items in the sales basket first. As always, 2 steps, first we create our target table and then we use a insert statement.

CREATE OR REPLACE TABLE c\_hive.db01.t\_unnested\_sales (

`invoicenumber` STRING

,`store\_id` STRING

,`category` STRING

,`brand` STRING

,`product` STRING

,`subtotal` DOUBLE

,`saleDateTime\_Ltz` STRING

,`saleTimestamp\_Epoc` STRING

,`saleTimestamp\_WM` AS TO\_TIMESTAMP(FROM\_UNIXTIME(CAST(`saleTimestamp\_Epoc` AS BIGINT) / 1000))

,WATERMARK FOR `saleTimestamp\_WM` AS `saleTimestamp\_WM`

,PRIMARY KEY (`invoicenumber`, `store\_id`, `category`, `brand`, `product`) NOT ENFORCED

) WITH (

'connector' = 'upsert-kafka'

,'topic' = 'avro\_unnested\_sales'

,'properties.bootstrap.servers' = 'broker:29092'

,'value.format' = 'avro-confluent'

,'value.avro-confluent.schema-registry.url'= 'http://schema-registry:9081'

,'key.format' = 'avro-confluent'

,'key.avro-confluent.schema-registry.url' = 'http://schema-registry:9081'

,'properties.group.id' = 'mysqlcdcsourced'

,'value.fields-include' = 'ALL'

);

Now for those with a keen eye, you would have noticed for t\_f\_avro\_salescompleted above we used key.format of raw, but now here we’re using avro-confluent, why?

That’s because t\_f\_avro\_salescompleted primary key is based on a single column. When the primary key is a composite index based multiple columns we aren’t able to use raw, but rather need to use avro-confluent as per above.

INSERT INTO c\_hive.db01.t\_unnested\_sales

SELECT

`invoiceNumber` AS `invoicenumber`

,`store`.`id` AS `store\_id`

,bi.`category` AS `category`

,bi.`brand` AS `brand`

,bi.`name` AS `product`

,bi.`subtotal` AS `subtotal`

,`saleDateTime\_Ltz` AS saleDateTime\_Ltz

,`saleTimestamp\_Epoc` AS saleTimestamp\_Epoc

FROM c\_hive.db01.t\_f\_avro\_salescompleted -- assuming avro\_salescompleted is a table function

CROSS JOIN UNNEST(`basketItems`) AS bi;

At this point we have all the information we want, in Apache Flink and being pushed back to our Confluent Kafka cluster into topics.

**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

------------------------------------------------------------------------------------------------------------------

# An exercise in Discovery, Streaming data in the analytical world.

## Confluent Kafka, Apache Flink + Flink CDC from PostgreSql & MySql => Apache Paimon, Apache Avro with a Apache Hive Metastore Catalog

(? Sept 2024 - Part 6+4)

**Overview**

In this part we now turn to persisting the data.

Let’s face it, everything we have done up to now is of no value if we can’t do this… As there are enough written about [Apache Iceberg](https://iceberg.apache.org/) as an Open Table Format I decided to change it up again and rather push the data to [Apache Paimon](https://paimon.apache.org/).

[Apache Paimon](https://paimon.apache.org/) originates out of the [Apache Flink](https://flink.apache.org/) environment originally as known as the [Apache Flink tablestore](https://nightlies.apache.org/flink/flink-table-store-docs-master/docs/development/overview/), before being split out as its own [Apache](https://www.apache.org/) project.

*Anecdotal, it’s the datastore used by* [*Alibaba*](http://alibabacloud.com/) *behind their online store, driving sales/recommendations, so ye it can handle any load you can imagine.*

[Apache Paimon](https://paimon.apache.org/) can persist data in either:

* [Apache Hadoop (HDFS)](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)
* [AWS S3](https://aws.amazon.com/pm/serv-s3/?trk=c8974be7-bc21-436d-8108-722e8ab912e1&sc_channel=ps&ef_id=CjwKCAjw6c63BhAiEiwAF0EH1LA0JwlNpmLmXHogsLFp7K5zew60UdpC2QZz-lpoFv2A6JPowvtLFxoCBg0QAvD_BwE:G:s&s_kwcid=AL!4422!3!645125274431!e!!g!!aws%20s3!19574556914!145779857032&gbraid=0AAAAADjHtp_ui_GZQySpr62ifWS_Wj-9n&gclid=CjwKCAjw6c63BhAiEiwAF0EH1LA0JwlNpmLmXHogsLFp7K5zew60UdpC2QZz-lpoFv2A6JPowvtLFxoCBg0QAvD_BwE)

using either of the following Open File formats:

* [Apache Avro](https://avro.apache.org/),
* [Apache Parquet](https://parquet.apache.org/)
* [Apache ORC](https://orc.apache.org/)

In our case we’re going persist the data onto our [MinIO](https://min.io/) S3 backend store. This is primarily as [AWS S3](https://aws.amazon.com/pm/serv-s3/?trk=c8974be7-bc21-436d-8108-722e8ab912e1&sc_channel=ps&ef_id=CjwKCAjw6c63BhAiEiwAF0EH1LA0JwlNpmLmXHogsLFp7K5zew60UdpC2QZz-lpoFv2A6JPowvtLFxoCBg0QAvD_BwE:G:s&s_kwcid=AL!4422!3!645125274431!e!!g!!aws%20s3!19574556914!145779857032&gbraid=0AAAAADjHtp_ui_GZQySpr62ifWS_Wj-9n&gclid=CjwKCAjw6c63BhAiEiwAF0EH1LA0JwlNpmLmXHogsLFp7K5zew60UdpC2QZz-lpoFv2A6JPowvtLFxoCBg0QAvD_BwE) is becoming the standard for high volume, low cost storage in the Cloud, notwithstanding that most Enterprise storage platforms are now also providing [S3 Object](https://aws.amazon.com/s3/) storage functionality, enabling the same design for on Premise.

And off we go… ☺

Below I show two methods, the first is doing what’s referred to as CTAS (Create Table as Select)

CREATE TABLE c\_paimon.dev.t\_salesbaskets WITH (

'file.format' = 'avro'

) AS SELECT

`invoiceNumber`

,`saleDateTime\_Ltz`

,`saleTimestamp\_Epoc`

,`terminalPoint`

,`nett`

,`vat`

,`total`

,`store`

,`clerk`

,`basketItems`

FROM c\_hive.db01.t\_k\_avro\_salesbaskets;

CREATE TABLE c\_paimon.dev.t\_salespayments WITH (

'file.format' = 'avro'

) AS SELECT

`invoiceNumber`

,`payDateTime\_Ltz`

,`payTimestamp\_Epoc`

,`paid`

,`finTransactionId`

FROM c\_hive.db01.t\_f\_avro\_salespayments;

And as a second option we first create our output, target table, i.e.

CREATE OR REPLACE TABLE c\_paimon.dev.t\_unnested\_sales (

`invoicenumber` STRING

,`store\_id` STRING

,`category` STRING

,`brand` STRING

,`product` STRING

,`subtotal` DOUBLE

,`saleDateTime\_Ltz` STRING

,`saleTimestamp\_Epoc` STRING

,PRIMARY KEY (`invoicenumber`, `store\_id`, `category`, `brand`, `product`) NOT ENFORCED

) WITH (

'file.format' = 'avro'

);

And then we push the data from the c\_hive.db01. object into the above created table utilising a Insert into <target table> select <column list> from <source table> query.

INSERT INTO c\_paimon.dev.t\_unnested\_sales

SELECT

`invoicenumber`

,`store\_id`

,`category`

,`product`

,`brand`

,`subtotal`

,`saleDateTime\_Ltz`

,`saleTimestamp\_Epoc`

FROM c\_hive.db01.t\_unnested\_sales;

And that’s it for this Blog series. As always, there are bits I still want to explore/switch out, i.e.: I’d like to split the single container [Apache Hive Metastore Catalog](https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/table/hive/hive_catalog/) into a two server build where we have a separate [HiveServer2 and Metastore](https://hive.apache.org/). Potentially without the need for a S3 object store, as we will use PostgreSQL as the backend data store. But that’s for another day and another Blog.

Thanks for sticking around, Hope the article was of value.

Till next time.



**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

Some more References:

[Apache Flink](https://flink.apache.org/) originally by [Ververica](https://docs.ververica.com/) + [Flink CDC](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/get-started/introduction/)

* [Welcome to Flink CDC](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/get-started/introduction/)
* [Connectors: Flink CDC sources](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/connectors/flink-sources/overview/)
* [Core Concepts: Data Pipeline](https://nightlies.apache.org/flink/flink-cdc-docs-release-3.1/docs/core-concept/data-pipeline/)
* Flink Forward: [How-to guide: Build Streaming ETL for MySQL and Postgres based on Flink CDC](https://www.ververica.com/blog/how-to-guide-build-streaming-etl-for-mysql-and-postgres-based-on-flink-cdc) (Good place to start).

Apache Flink CDC project on Github

* [CDC Connectors for Apache Flink](https://github.com/apache/flink-cdc/tree/release-2.3.0)

Funky Funky, you can define Sources, Targets and Pipelines (with transformation) via YAML definitions… See the data Pipeline link above.

**Docker Image Ancestry**

The below is an Ancestry diagram of sort, showing how I layered the images, how one deployment uses an image created previously. This all sits inside the infrastructure directory of the GIT repo.

