# From MongoDB via Flink CDC Action framework, persisted into Apache Paimon on S3.

IoT Data packaged in JSON format pushed into MongoDB, extracted using Flink CDC, using Flink action framework & Stored into Apache Paimon.

(25 February 2025)

**Overview**

In previous [blogs](https://github.com/georgelza/split-sources-in-streaming-world) we pushed our data either onto an Apache ([Confluent](https://www.confluent.io/)) [Kafka](https://kafka.apache.org/) based topic and additionally into a MySQL and/or PostgreSQL database. For the Kafka topic we then used a standard Flink / Kafka consumer to subscribe to the messages. In the case of [MySQL](https://www.mysql.com/) and [PostgreSQL](https://www.postgresql.org/), we utilized Apache Flink’s CDC. Well, that was then.

This time round, we’re going to use [Apache Flink’s flink\_paimon\_action](https://paimon.apache.org/docs/master/cdc-ingestion/overview/) framework to [CDC](https://en.wikipedia.org/wiki/Change_data_capture) source the payload directly out of our MongoDB collection and store it directly into [Apache Parquet](https://parquet.apache.org/) files stored in our [Apache Paimon](https://paimon.apache.org/) based table, written onto S3 Object storage provided via a [MinIO](https://min.io/) server.

Our data is generated via a Python IoT program (located in *app\_iot1/* and now containerized) created in the previous blog.

* In the first “part” we simply consume the data from our MongoDB where we specify a source collection and persisted that then into a defined Apache Paimon datastore/table.
* The second “part” we expand on this in that we will reconfigure the stream to source the full [MongoDB](https://www.mongodb.com/) and persist all collections into our [Apache Paimon](https://paimon.apache.org/) based database.
* In the third “part” we push one of our data streams back out to a Kafka topic, why, because we can.
* Somewhere along the line we will also modify our first IoT message / payload structure to demonstrate the ability to “handle” table structure/evolution, this will be part b of part 1… ;) now keep track of all of this.

For catalog services we will be using the [Apache Hive](https://hive.apache.org/)’s and their [Metastore](https://cwiki.apache.org/confluence/display/hive/design#Design-Metastore) functionality as created in a previous blog (but with a little version update applied recently).

For those that have been following my previous blogs, you will notice I’ve upgraded my [Confluent](https://www.confluent.io/) Kafka Cluster (now 7.7.1) and the [Apache Flink](https://flink.apache.org/) environment (now 1.19.1). The [Apache Paimon](https://paimon.apache.org/) stack has also been upgraded to 0.9.0

As always, all the code can be found in the [GIT repository](https://github.com/georgelza/DataPipeline-MongoDBPaimon), and yes, we’re still using a substantial amount of Makefiles, Docker-compose.yml and Dockerfile’s.

<Full disclosure, I am still working on the aggregation bits inside Flink, but then these have been covered in a previous blog with examples>.

Ok, Now, before we start with the working example. A Lesson Learned time, well the summary.

Initially when I started with this blog I took my basic framework, stripped out everything and then simply started adding the libraries from the [Mongo CDC example](https://paimon.apache.org/docs/master/cdc-ingestion/mongo-cdc/) and previous libraries, but upgraded for the new versions listed above.

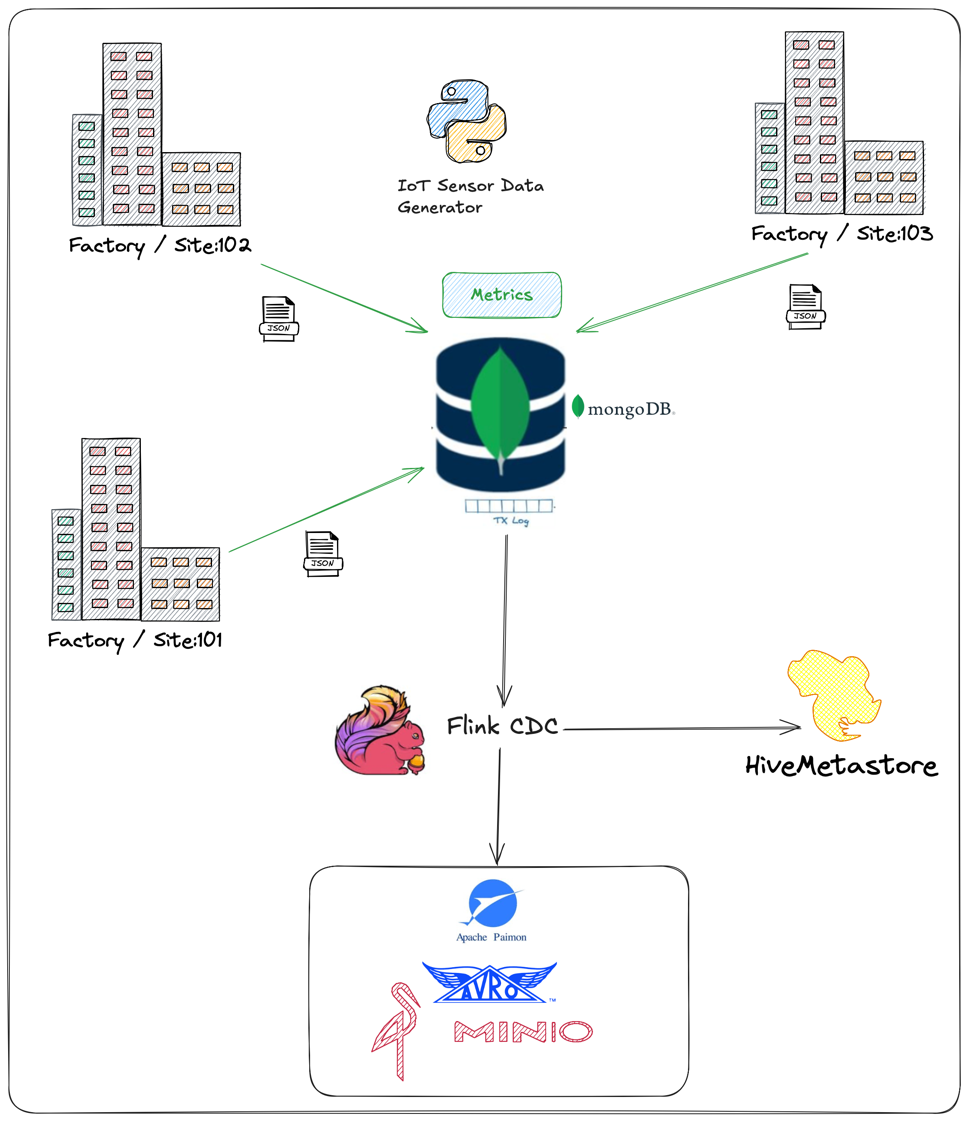
Well needless to say things did not work… I was stuck, what was I doing wrong… What did I break along the path of upgrading all my components, what wrong version of a jar file did I now have, or totally miss.

The lesson… lets upgrade the stack, then use a previously known working pipeline to first confirm things are still working, before introducing new functionality/methods of doing something.

To recover my sanity… I pulled in an app from a previous blog, and simply configured previously known CDC pipeline producing data into MySql table, consuming that via Apache Flink CDC into a Flink Table object and then sink’ing the stream into my Apache Paimon table in Parquet file format located on our MinIO S3 store… Once that worked, I knew that our base stack work… and we could now proceed to investigate the new “stuff”.

Ok, back to why we’re here.

We will use our Python application to create a IoT data stream as if we’re reading sensors in factories fitted to machinery. These “[JSON](https://www.json.org/)” documents will be pushed into a MongoDB collection called *telemetry*\_*north*. We will then use the [MongoDB CDC capability via the action frame](https://paimon.apache.org/docs/master/cdc-ingestion/mongo-cdc/) to replicate this directly into an Apache Paimon Table.



First up, lets replicate that collection.

/*opt/flink/bin/flink run \*

*-Dexecution.checkpointing.interval=10s \*

*-Dexecution.checkpointing.num-retained=5 \*

*-Dstate.checkpoints.num-retained=10 \*

*-Dpipeline.name='sync-mongodb-table-to-paimon-s3' \*

*/opt/flink/lib/paimon/paimon-flink-action-0.9.0.jar \*

*mongodb\_sync\_table \*

*--mongodb\_conf hosts=mongodb:27017 \*

*--mongodb\_conf username=root \*

*--mongodb\_conf password=dbpassword \*

*--mongodb\_conf database=Mongo0 \*

*--mongodb\_conf collection=telemetry\_north \*

*--catalog\_conf metastore=hive \*

*--catalog\_conf uri=thrift://metastore:9083 \*

*--warehouse s3a://warehouse/paimon/ \*

*--database iot \*

*--table 'telemetry\_north'*

Ok, so this is funky. And simple. Let test schema evolution. For this we change *iot1\_datagen* service in our docker-compose file and change the *0 -> 1* for the variable *STRUCTMOD*. What this will do is add “*location*” tag to our payload containing “*latitude*” and “*longitude*” values. We can now restart the service, and we should now be producing new payloads… What we expect to see here is records containing the additional tags flowing/appearing in our Apache Paimon table, with no changes by us required otherwise. Ok, so we can handle schema evolution.

Next up, we will reconfigure the setup (drop the above table replication job first) and then lets rather replicate the entire MongoDB.

/*opt/flink/bin/flink run \*

*-Dexecution.checkpointing.interval=10s \*

*-Dexecution.checkpointing.num-retained=5 \*

*-Dstate.checkpoints.num-retained=10 \*

*-Dpipeline.name='sync-mongodb-table-to-paimon-s3' \*

*/opt/flink/lib/paimon/paimon-flink-action-0.9.0.jar \*

*mongodb\_sync\_database \*

*--mongodb\_conf hosts=mongodb:27017 \*

*--mongodb\_conf username=root \*

*--mongodb\_conf password=dbpassword \*

*--mongodb\_conf database=Mongo0 \*

*--catalog\_conf metastore=hive \*

*--catalog\_conf uri=thrift://metastore:9083 \*

*--warehouse s3a://warehouse/paimon/ \*

*--database iot*

At this point we can restart our *iot1\_datagen* service (configured in our d*evlab/docker-compose.yml)* and we should see the *iot.db/telemetry\_north* table receiving data.

All good, we know everything still working.

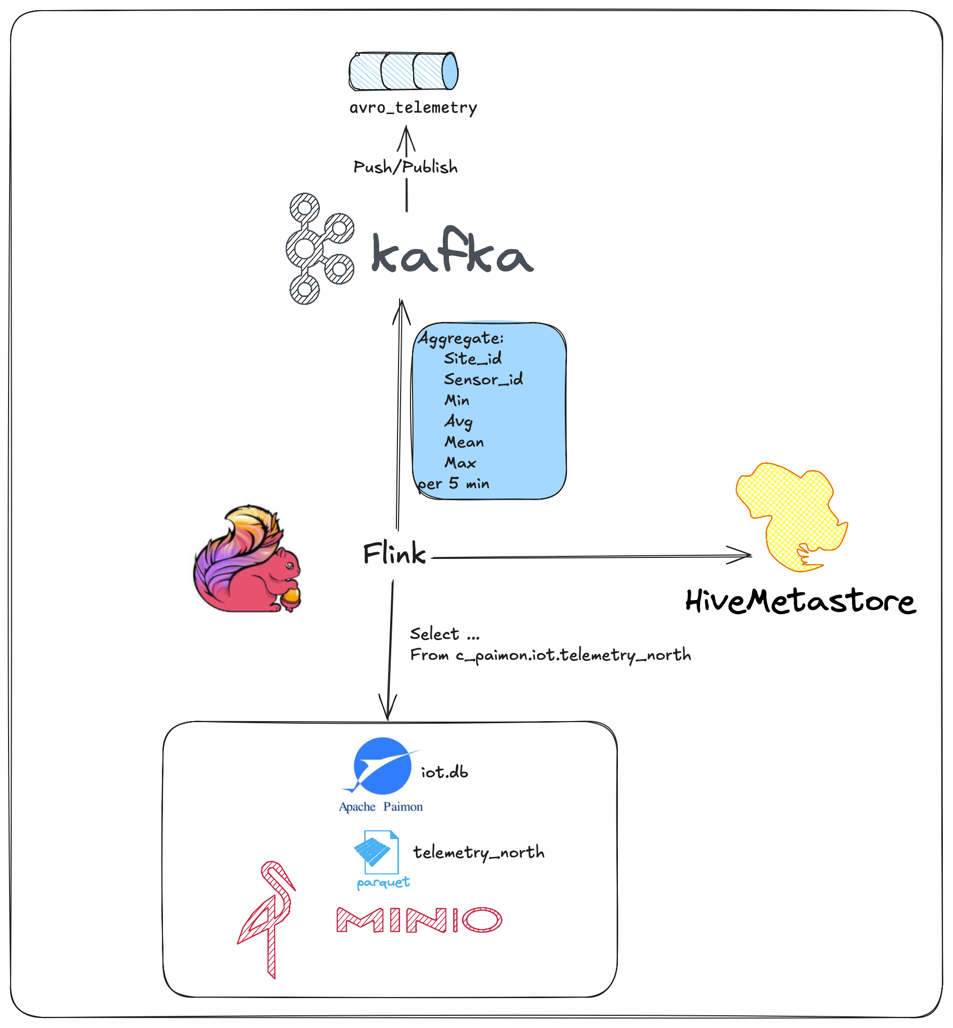
To test the dynamic capability of the stack to dynamically add extra collections created in the source database we will reuse our base iot generator container, but now produce data for different sites into a separate collection (*telemetry\_south*), this is all accomplished by environment variables. This is accomplished by running the *iot2\_datagen* service from our *devlab/docker-compose*.*yml* file

What we will see here is the CDC pipeline creating a new target table and start to push the new data set without us having touched anything.

Ok, now for the last bit… <As per above the below is not completed yet, do return to the blog/repo as I update it>.

Next up we will build a time-based rolling window on an aggregation. We want to see the per *site\_id,* *sensor\_id* the *min, avg, mean* and *max* measurement per 5 min window.

We lastly want this value pushed back onto our Kafka cluster onto a topic called *avro*\_*telemetry*.



First, we create our target table and define the backing topic

```

*CREATE OR REPLACE TABLE c\_hive.iot.t\_f\_avro\_telemetry\_5min (*

*`site\_id` STRING*

*,`sensor\_id` STRING*

*,`min\_val` DOUBLE*

*,`avg\_val` DOUBLE*

*,`mean\_val` DOUBLE*

*,`max\_val` DOUBLE*

*,`max\_val` DOUBLE*

*,`max\_val` DOUBLE*

*,`window\_start` DATETIME*

*,`window\_end` DATETIME*

*) WITH (*

*'connector' = 'kafka'*

*,'topic' = 'avro\_telemetry\_5min'*

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

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

*,'scan.startup.mode' = 'earliest-offset'*

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

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

*,'key.format' = 'raw'*

*,'key.fields' = 'site\_id'*

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

*);*

```

And next we select, aggregate and insert the values.

```

*INSERT INTO c\_hive.iot.t\_f\_avro\_telemetry\_5min (*

*`site\_id`*

*,`sensor\_id`*

*,`min\_val`*

*,`avg\_val`*

*,`mean\_val`*

*,`max\_val`*

*,`window\_start`*

*,`window\_end`*

*) SELECT*

*site\_id as site\_id,*

*sensor\_id as sensor\_id,*

*min(measurement) as min\_val,*

*avg(measurement) as avg\_val,*

*mean(measurement) as mean\_val,*

*max(measurement) as max\_val*

*from\_unixtime(WINDOWSTART) as window\_start,*

*from\_unixtime(WINDOWEND) as window\_end*

*FROM c\_paimin.iot.telemtry\_north*

*WINDOW TUMBLING (SIZE 5 MINUTE)*

*GROUP BY site\_id*

```

And that’s it…

So, in summary, we build a data pipeline from MongoDB database, that can handle schema evolution and collections being added, have that data flow into our Apache Paimon data store stored as Apache Parquet files.

We then build a windowing aggregation of this back onto an Apache Kafka topic.

I think this is pretty neat… Hope you enjoyed the exploration.

Good luck, this is all fraught with rabbit holes, as always, so many and you can disappear so easily… but then that’s ½ the fun.



**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)

<https://www.linkedin.com/in/george-leonard-945b502/>

<https://medium.com/@georgelza>