# A Factory IoT instrumentation data stream, from source to Fluss into a lakehouse tier.

IoT JSON documents stream from source via Kafka topics, consumed by Apache Flink, flattened and pushed into Fluss with deep storage via HDFS.

(1 June 2025)

**Overview**

In a previous [blog](https://medium.com/@georgelza/from-mongodb-via-flink-cd-action-framework-persisted-into-apache-paimon-on-s3-008976cdbfff) we were inserting our JSON packaged IoT generated data into a MongoDB Collection and then using [Apache Flink’s flink\_paimon\_action](https://paimon.apache.org/docs/master/cdc-ingestion/overview/) framework to CDC source the data and sink (insert) it into a [Apache Paimon](https://paimon.apache.org/) database…

In this [blog](https://github.com/georgelza/DataPipeline-Kafka_Fluss_Paimon-Basic.git) our datastream will now make a slight turn out of the [Flink](https://alibaba.github.io/fluss-docs/) environment into Fluss as our near real time streamhouse datastore (referred to as lakehouse tier in Fluss).

We again have our three regions, each with 2 factories (sites), each factoriy have many machines (devices) and each machine is instrumented with multiple sensors. They are distributed as:

* North *<root>/app\_iot1*
  + siteId 101
  + siteId 104
* South <*root>/app\_iot2*
  + siteId 102
  + siteId 105
* East <*root>/app\_iot3*
  + siteId 103
  + siteId 106

The primary idea for me for this blog was to simply connect the dots, figure out what settings have what impact and as such this is not going to be anything fancy or long.

This is sort of [part 1](https://github.com/georgelza/DataPipeline-Kafka_Fluss_Paimon-Basic.git) of a [2-part](https://github.com/georgelza/DataPipeline-Kafka_Fluss_Paimon-PyFlink.git) series.

In this first blog, we’re simply going to publish the same as the previous blog onto a 3 [Confluent](https://www.confluent.io/) Kafka Topics,

* *factory\_iot\_north,*
* *factory\_iot\_south and*
* *factory\_iot\_east)*

We will then source the topics via [Apache Flink](https://flink.apache.org/) into 3 Flink tables. This will be done by creating the tables with the *connector=kafka* specified.

Our Flink tables are cataloged via *hive\_catalog* and an *iot* database created, whereas our Fluss tables are cataloged via *fluss\_catalog.* Inside the Fluss catalog a *fluss* database will be used to host the fluss tables. This *fluss\_catalog* is also instrumental in that it defines how Flink communicates with Fluss environment via the specified “*coordinator-server:9123”* service.

* For the first example we will flatten the structure and push each topic into a dedicated Fluss table for Real Time analytics, with Tier 3 storage as [Apache Paimon](https://paimon.apache.org/) tables ([Apache Parquet](https://parquet.apache.org/) files) on [HDFS](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).
* The Second example we will again flatten the data structure, but this time round we push the datastream into a central table, with Fluss again providing table storage.

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 blogs (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 7.9.1) and the [Apache Flink](https://flink.apache.org/) environment (now 1.20.1). The [Apache Paimon](https://paimon.apache.org/) stack has also been upgraded to 0.9.0. We’re using Fluss 0.6.0 at this stage.

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

*Note: to execute this blog start with README.md located in the <root> folder and work from there, it will tell you exactly what to execute in which order to download all the dependencies and build everything.*

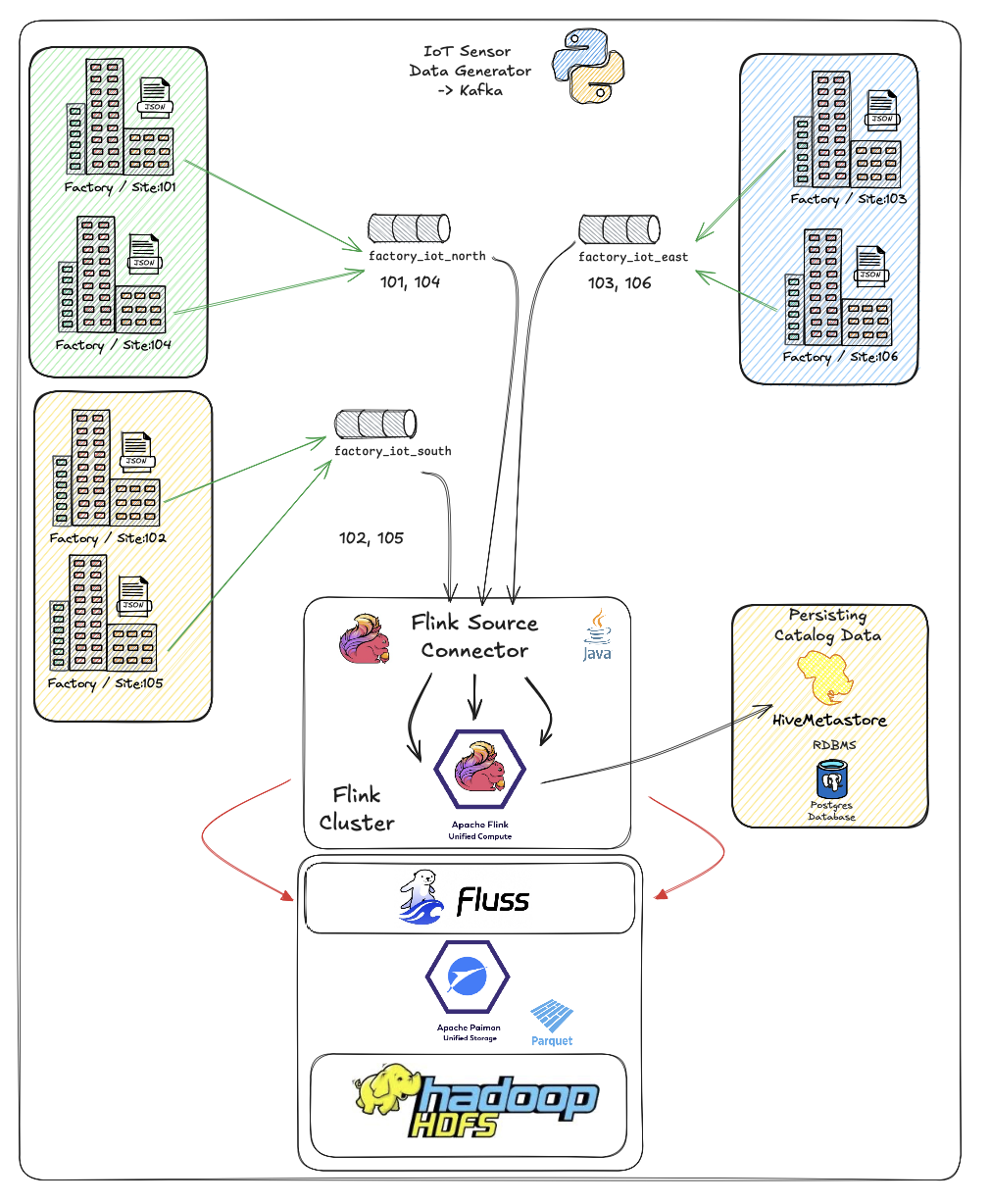
I’ve decided not to digress as usual and do allot of other “things” like aggregation.

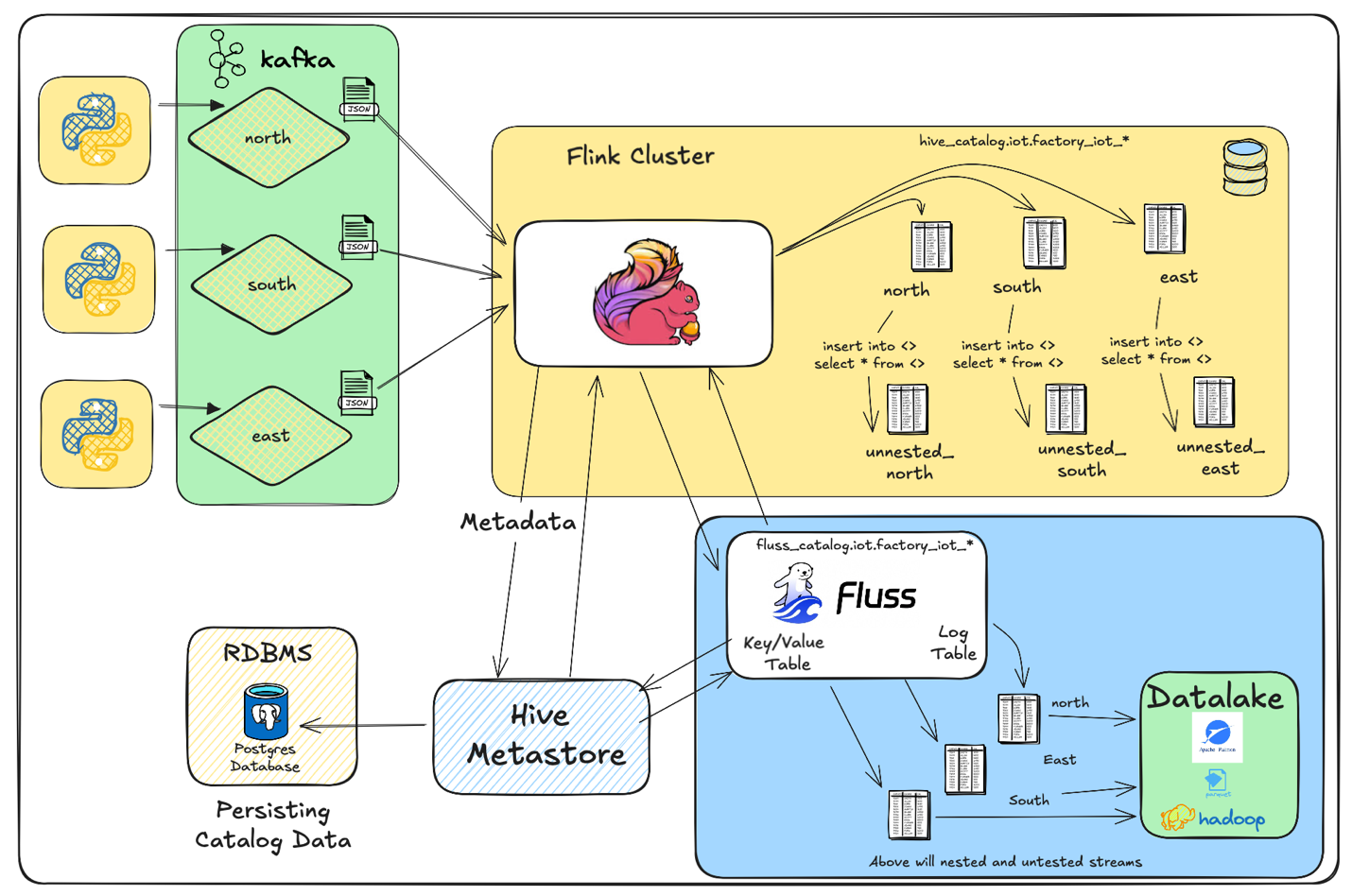
Ok, back to why we’re here.

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 onto our Kafka Topic called *factory\_iot\_\**. The topic “*key”* is set to the *siteId* of the various factories.

I grouped the factories as per above into 3 sets, North, South and East. (simulating a sort of regional distribution).

Overview of our environment:





To start navigate to the *<root>* folder and read the *README.md* file. This will give a similar overview as per above, that will then direct you to build the basic scaffolding (<*root>/infrastructure/),* after which you will be directed to *<root>/devlab0/README.md* instructing you how to build and run the various examples.

Our first region has been naughty, and not all the required detail is populated into the IoT JSON document.

Executing */<root>/app\_iot1/site1.sh>* will thus produce the below document.

{

"ts" : 123421452622,

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi"

},

"measurement" : 1013.3997

}

Below we have the second region, they seem to have read the documentation/expectations a bit more and populated more of the fields of our IoT JSON document.

Region South data is generated using *<root>/app\_iot2/site2.sh* application. This as you can see below adds the *ts\_human* and *location* object to the payload.

{

"ts" : 123421452622,

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi",

"ts\_human" : "2024-10-02T00:00:00.869Z",

"location": {

"latitude": -26.195246,

"longitude": 28.034088

}

},

"measurement" : 1013.3997

}

And lastly as we’re feeling very lucky, Region east paid attention and they are providing us a complete IoT Document. To generate their payload, execute *<root>/app\_iot3/site3.sh* application which will add the *deviceType* field to the metadata tag.

{

"timestamp" : "2024-10-02T00:00:00.869Z",

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi",

"ts\_human" : "2024-10-02T00:00:00.869Z",

"location": {

"latitude": -26.195246,

"longitude": 28.034088

},

"deviceType" : "Oil Pump",

},

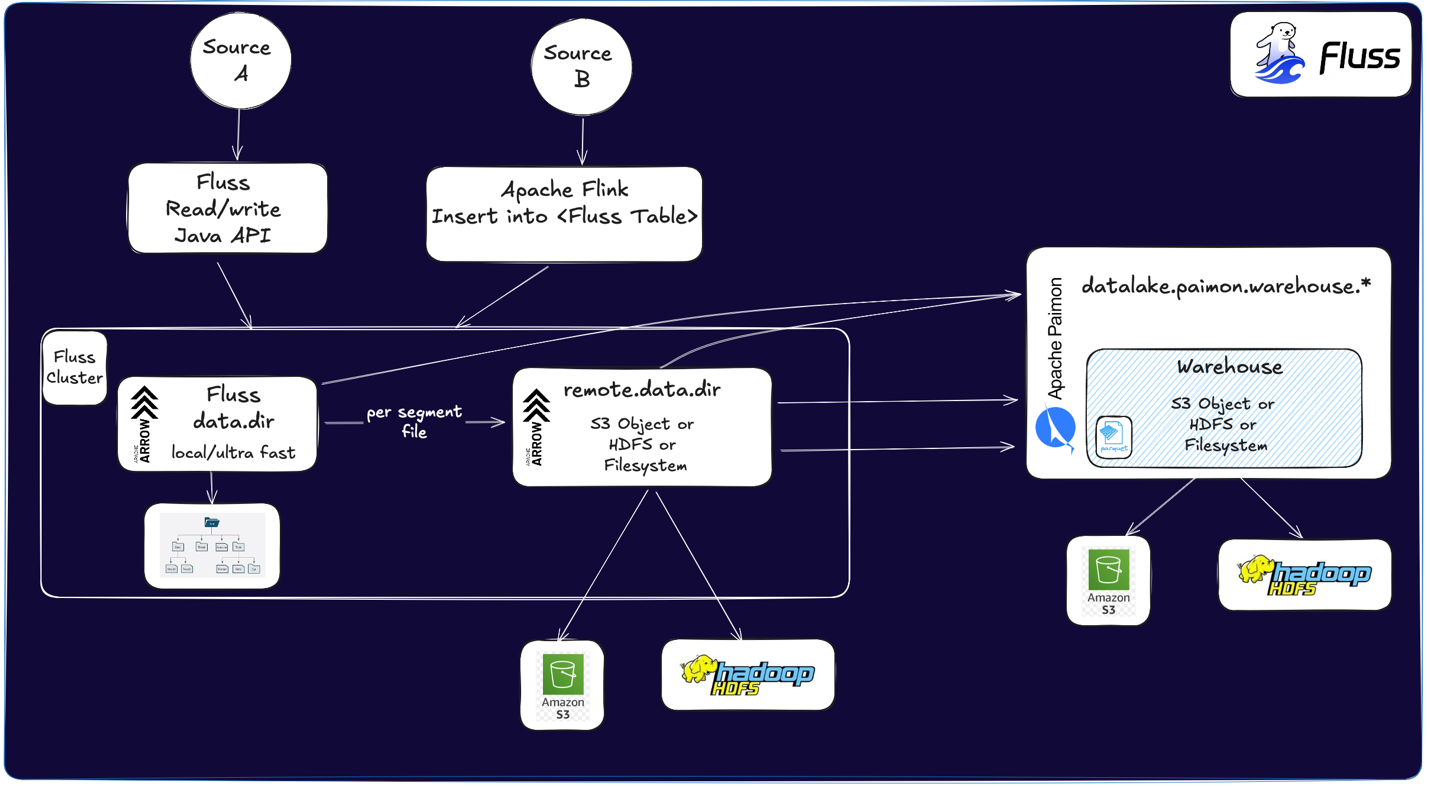
"measurement" : 1013.3997

}

So, with the 3 different versions of the IoT JSON document we demonstrate the very good fit of JSON for IoT and the ability along our entire path from source to data store to accommodate the dynamic data structure.

So, in summary, we build a data pipeline from our source factory, into Apache Kafka Topic, flattened by Apache Flink, pushed into Fluss from where it is then tiered down into the defined lakehouse which is based on Apache Paimon tables, in Apache Parquet file format, all stored on our HDFS stack.

Below is a over view of Fluss’s Tiered storage and stack;



Overview of the storage Tiers in a Fluss deployment.

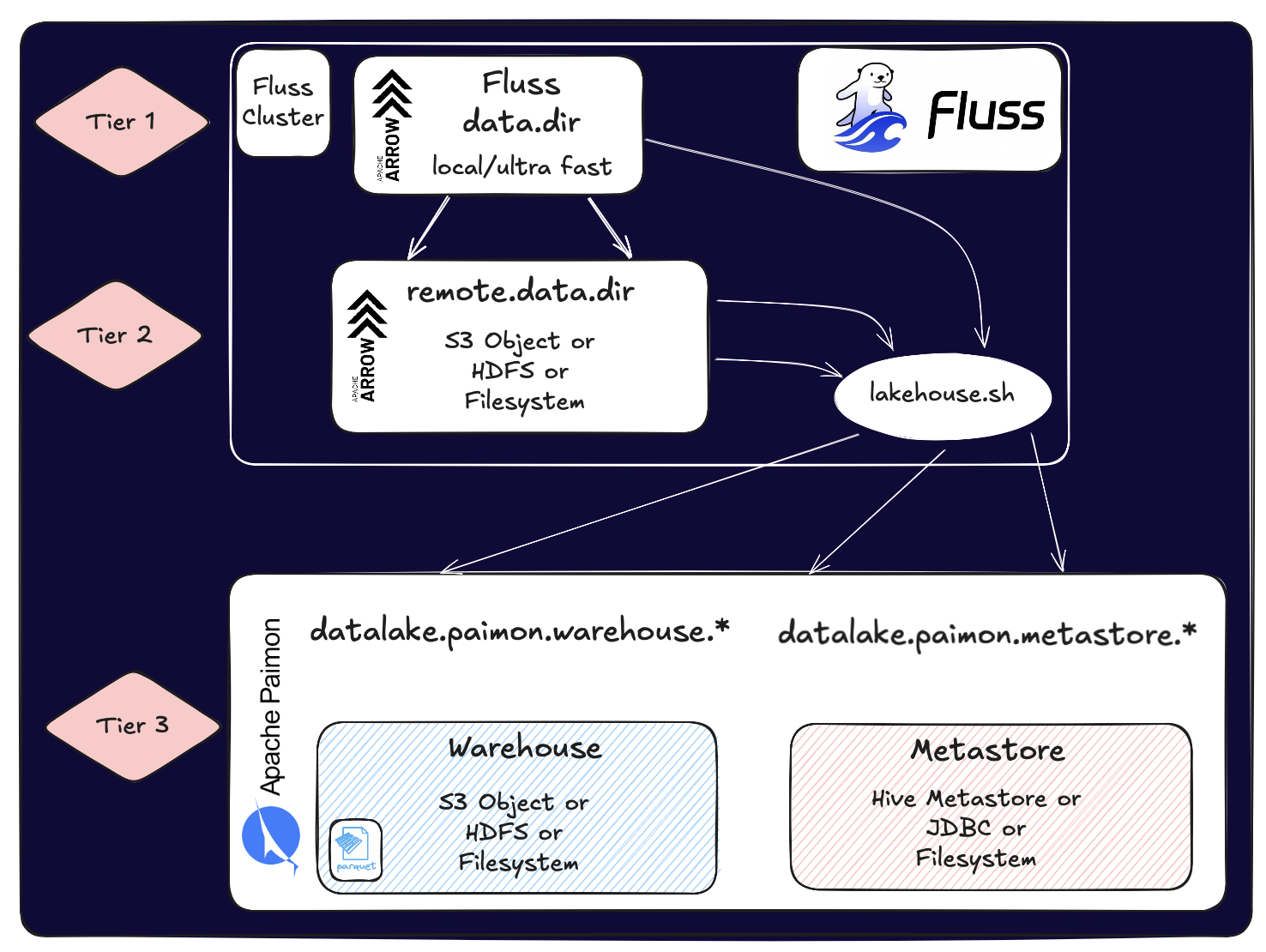
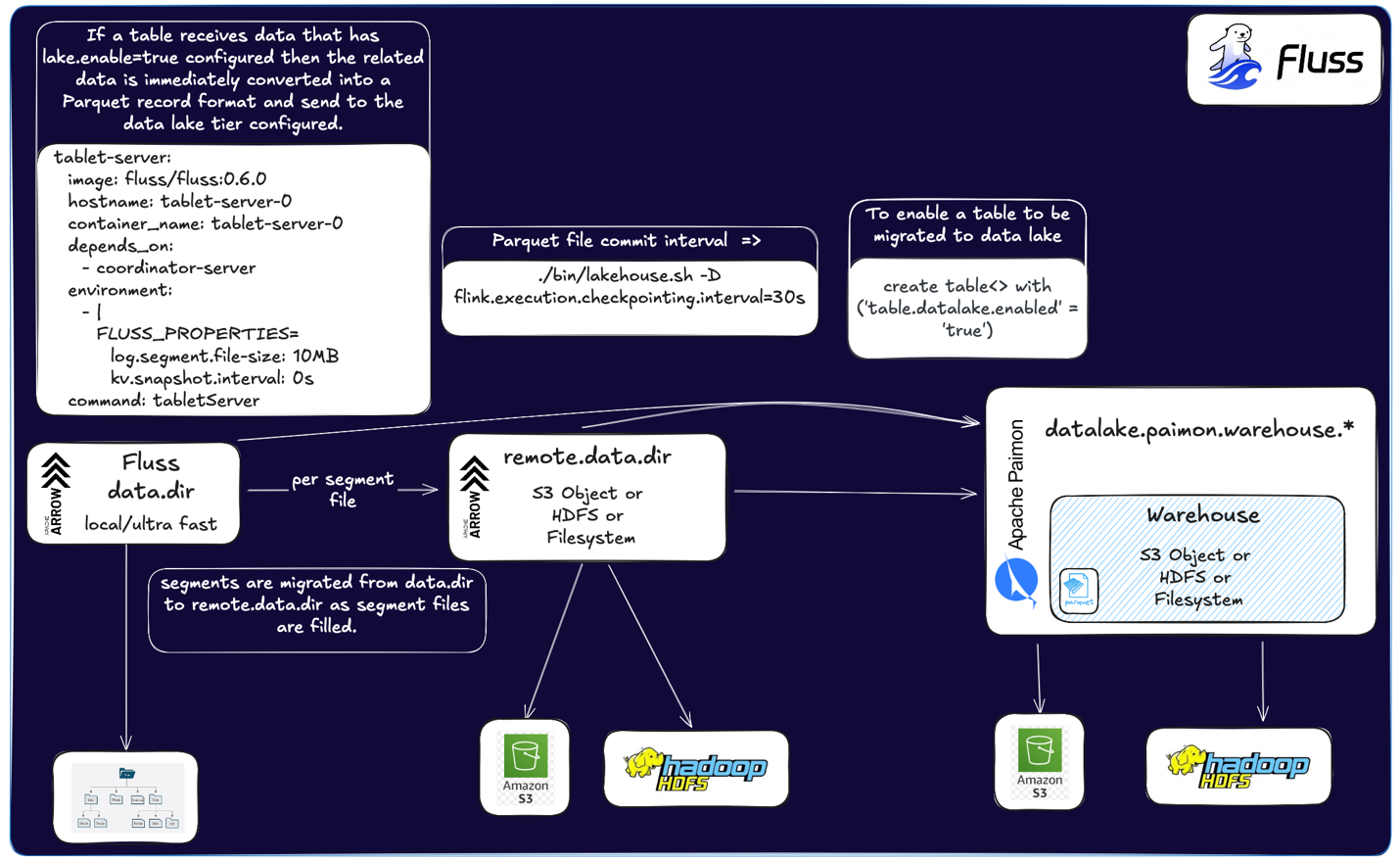


Diagram depicting the data migration Triggers:



NOTE: for this blog I pushed our IoT data stream into Apache Kafka topics, going forward Fluss will be able to present a Apache Kafka compatible endpoint allowing for data to be published directly into Fluss tables (using Kafka protocol) which will simplify our stack and result in much “fresher” data for analytics and lower cost as we will have less technology involved.

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

See you in part 2.

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>