**Overview**

A hair brain idea… with good intensions that ended with “some” scope creep and lots of learning along the way.

Well originally this started as a very simple idea, lets create some data, publish it onto some Kafka topics, sink that into a [MongoDB](http://mongodb.com/) Atlas database/collection and then utilize the new Mongo stream processing to extract some value via aggregations and push (emit) that back onto [Apache Kafka](https://kafka.apache.org/) topics… to be displayed in terminal windows via simple Python consumers… Well that was the original concept sold to MongoDB Creator Community.

First, I discovered realized that due to work that this will take significantly longer than the 1 month of free Confluent Cloud access/credits, so the plan pivoted to deploying Confluent Platform via docker-compose locally.

I wanted the data to be used to have association / relevance and not simple fake random data so a small [Golang](https://go.dev/) (picked the language just because) application was created that constructed the source data from provided seed data/options. Note to full time coders, I am aware of various improvements that could be made, I know it can be split into a basket creator and a separate payment creator and deployed as containers themselves… The app was not the core of the project so allow me some peace ;)

The idea, play a store, do the all to well-known shopping basket and payments game, create a basket (constructed at a store selected at random from set of stores defined in seed file), comprised from random number of items (selected from seed file), random quantity of each item, once constructed the basket is posted onto a salesbasket topic and then create a salespayment record, associated with the basket, posted onto a separate salespayment topic.

By [George Leonard](https://www.linkedin.com/in/george-leonard-945b502/) – 17 July 2024

**Sections**

Source data generator

Golang app…

Data sets:

Salesbasket

{

"InvoiceNumber": "1341243123341232",

"SaleDateTime\_Ltz": "2023-12-23T16:53:39.911+02:00",

"SaleTimetamp\_Epoc": "1718117619911",

"Store" : {

"Id": "1033",

"Name": "Derry"

},

"Clerk": {

"Id": "231",

"Name": "Martin",

"Surname": "Smith"

},

"TerminalPoint": "14",

"BasketItems":[

{

"Id": "234123412",

"Name": "Minty Frsh",

"Brand": "Colgate",

"Category": "Healthcare",

"Price":12412.00,

"Quantity":3

},

{

"Id": "234123421",

"Name": "All Bran",

"Brand": "Kellog's",

"Category": "Cereal",

"Price":12.00,

"Quantity":3

},

{

"Id": "534123412",

"Name": "Sugar Free",

"Brand": "Coke",

"Category": "Cool drinks",

"Price":112.00,

"Quantity":2

},

{

"Id": "224123412",

"Name": "Auto Wash",

"Brand": "OMO",

"Category": "Cleaning",

"Price":22.00,

"Quantity":4

}

],

"Net": 442.23,

"VAT":10.00,

"Total":452.23

}

Salespayments

{

"InvoiceNumber": "1341243123341232",

"PayDateTime\_Ltz": "2023-12-23-T16:55:39.000+02:00",

"PayTimetamp\_Epoc": "1718117619911",

"Paid": 452.23,

"FinTransactionID": "42dfgt245wsdg34231rfwfg234234"

}

Serialization Formats

Now first thing, the data is structured, either as a row with columns, or can be a xml message or a JSON structure, all of this is simple text at this point, what’s published onto the topic and read from the topic however is a byte stream, and this is where serialization comes in, the options being:

* [JSON](https://www.w3schools.com/js/js_json_intro.asp)
* [Protobuf](https://protobuf.dev/)
* [Apache Avro](https://avro.apache.org/)

Each have its Pro’s and Con’s as will be discussed later.

Project environment

Below we’re going to have a “quick” discussion about what we need to deploy to run the project. Allot of this was done locally, to safe cost, some of it cannot be done locally (MongoDB Atlas streams and Dashboards). Doing this locally yourself also happens to give you more insight into how the stack fit together, how to extend the stack/capabilities by importing additional jar files.

docker-compose.yml

Include project name

Include network name

Modify container names to be created as project name + container name

Beware “localhost” monster in a docker-compose/container environment, what’s the saying if all else fails “It’s always DNS”. When talking, referencing services in a container based environment, always think, when a instruction is given to a container, how does it know who to talk to, on your local machine you can point to different services simply by saying localhost:<port> but that same localhost in the container is the container itself, so always remember, refer to other services by their service name as defined in the docker-compose.yml file, it will safe you years of grieve.

[Confluent](https://www.confluent.io/) Platform Kafka environment

Broker

…

kSql

kStream

Connect

Custom Connect container with additional connectors installed

[Apache Flink](https://flink.apache.org/)

Little Note, all of Apache Flink stacks are actually build using a single image: flink-kafka:1.16.0-scala\_2.12-java11, this image contains all of: jobmanager, taskmanager and sql\_client.

Apache Flink is package by various groups, i.e. Ververita, Confluent:

It’s really helpful to just scan through: <https://nightlies.apache.org/flink/flink-docs-master/docs/deployment/resource-providers/standalone/docker/>

Note: Persistence is not configured by default (in a docker-compose lab is deployed when deploying the previous mentioned images). Well, what do I mean by this… surprise if you create Flink tables, and exit Flink, when you restart it, all your tables/jobs are gone and you will need to recreate them. See [Robin Moffatt](https://www.linkedin.com/in/robinmoffatt/)’s [Decodablecoe](https://github.com/decodableco/examples/tree/main/catalogs) Git repo for examples on how to configure persistence.

[MongoDB Atlas local](https://hub.docker.com/r/mongodb/atlas) & MongoDB Atlas cloud

[PostgreSql](https://www.postgresql.org/)

Let’s complete our environment for a future use case.

[MySql](https://www.mysql.com/)

Let’s complete our environment for a future use case.

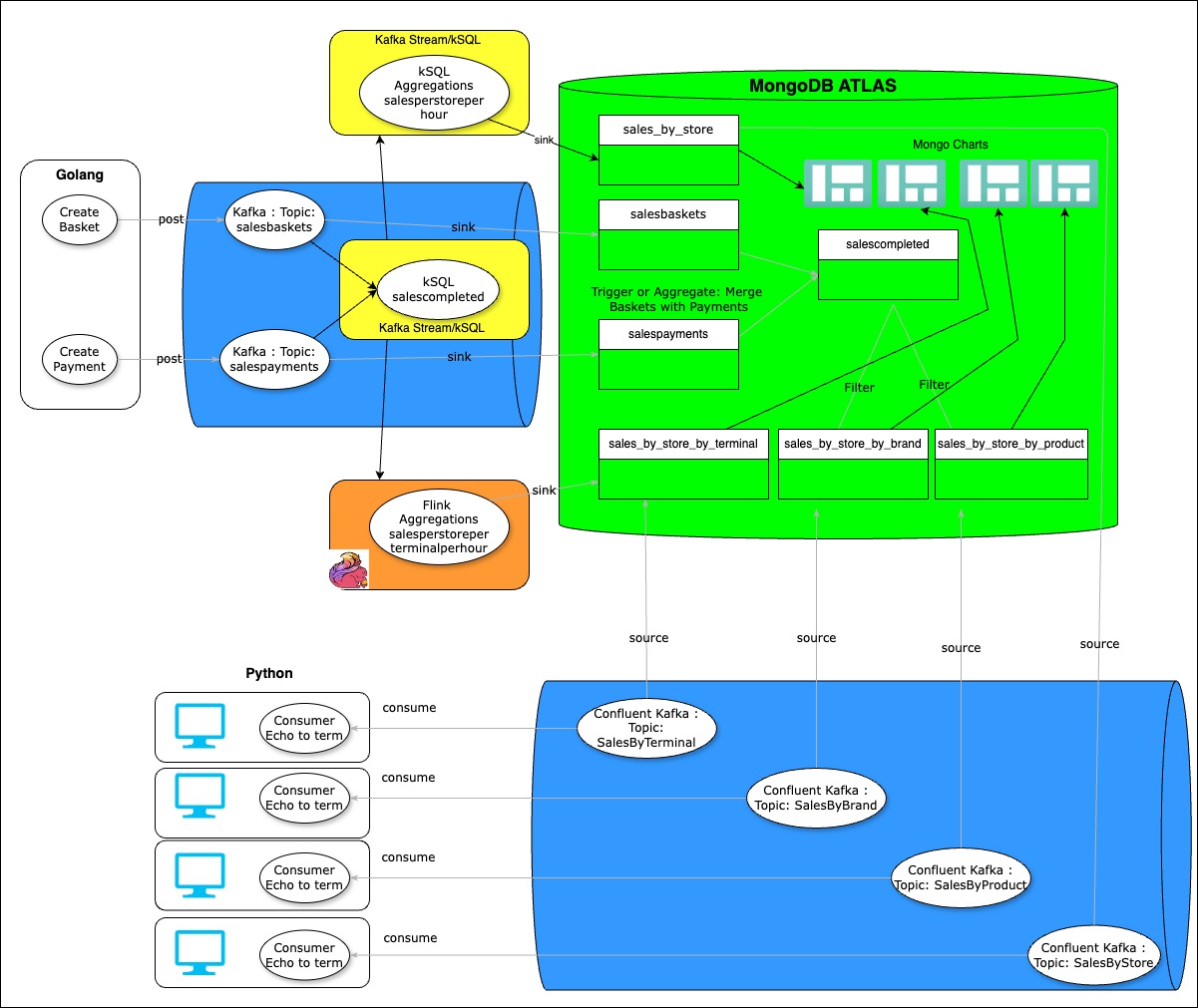
Kcat

Sometimes it helps to be able to peek at whats going on inside topics.

Note: confluent-consumer-…

Computations/Aggregations:

Well in the end this is what all this is about actually (everything above was scaffolding, but nevertheless important to know and understand). This is where we want to take our raw feeds (salesbaskets, salespayments and the derived salescomplete) and do analysis/aggregations on them to derive value/insight into what’s happening on the floor, how is business doing, really.



Aggregations via kSql and kTable

Aggregations via Apache Flink

Consider the difference in output that emit change vs emit final has.

Data serialization format

First there was Protobuf and then Avro came.

Avro Schema

A word of warning: Case Sensitivity between Flink Sql’s: Create table <>, Select <> From <> and Kafka schema registry entries matters.

Small pieces of work.

Aggregations via Mongo Stream Processing

Results:

Sink (push) to Collections

Sink (push) to Kafka Topics

Source (pull) from Mongo

Dashboards/Graphs in Mongo

Lesson’s

Credit’s

Repo’s

Originally, I started with a JSON structure, serialized version, with no schema registry. Performance was north of 10 000 txn/second.

Then a little devil whispered in my ear lets add schema registry as all good practices/papers advised. Well performance dropped to 500 txn/second.

So I’ve long since heard Protobuf’s are the dope, it’s fast, performing so let’s refactor app into Protobuf structured , using a Confluent Kafka serialization client, plugging into the and then the associated schema registry changes. Great, we’re back at 8000tx/second. Issue now… Support, by default included libraries inside Apache Flink for one is not… Work around, on Kafka cluster create a Avro serialized stream from Pb serialized, it works… but there must be a better way.

Ok, I’ve heard about this thing called Avro, lets refactor again, surprise, was allot more complicated to get working than expected… but eventually got it working, and we’re back at 8000+txn/second. So now we using what everyone says is the best serialization across streaming architecture and, we have rich support across various stacks.

Version 1: JSON payload

<https://github.com/georgelza/MongoCreator-GoProducer-json>

Version 2: JSON Protobuf

<https://github.com/georgelza/MongoCreator-GoProducer-pb>

Version 3: JSON Avro

<https://github.com/georgelza/MongoCreator-GoProducer-avro>

**About Me**

I’m a techie, a technologist, 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.

Recent years, pivoted to a more generic Technology Architect capable of full stack architecture.

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

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