# An IoT Sensor Stream.

## ShadowTraffic -> Confluent Kafka -> Confluent Connect -> MongoDB Atlas Timeseries Collection

(? October 2024 - Part 1)

**Overview**

So, allot is written all over the interweb’s regarding moving various types of data around from sources into target data stores.

Thought I’d write a little blog about a very topical (underappreciated) data format, namely [Timeseries](https://en.wikipedia.org/wiki/Time_series) data.

There are countless applications of this format, [IoT](https://en.wikipedia.org/wiki/Internet_of_things) sensor data being the most spoken about.

But we can expand on this, anything that is a reading, a value that is associated with a point in time can be stored and analyzed/visualized as time series data.

Some examples:

Financial:

* Currency exchange rates
* Stock prices
* Stock levels

Any type of sensor reading, i.e.

* Weather / Environmental
  + Temperature
  + Wind speed
  + Wind direction
  + Humidity
  + Atmospheric pressure
  + Gas compositions/readings
* Any factory/mechanical sensor
  + Pressure
  + Flow
  + Volume
  + Temperature
  + Current
  + voltage

As can be seen we can go on and on, it is really up to you to have your imagination go…

Time series data is by nature normally constructed out of 3 components primarily, namely:

* A timestamp
* Metadata: One or more key/value pairs describing what was measured.
* The measured value.

Time series data normally also have an aging policy associated with it (to manage cost & size) via what’s defined as a roll up. At the start data will be stored based on per second values.

Depending on requirements a company might decide to retain this “level” of data for the first month, after which it’s rolled up into say a per hour averages for the next 12 months after which it’s rolled up/averaged into values per day.

Below is a very simple example:

{

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

“metadata” : {

“siteId” : 100

},

“measurement” : 1013.3997

}

Next is a bit more example, here we simply expand our metadata section with addition tags to filter on:

{

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

"metadata" : {

"siteId" : 100,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi"

},

"measurement" : 1013.3997

}

Now the following example can be the opening of a can of worms, as the saying goes, and I’m not going to promote or defend the following option, other than list it.

{

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

“metadata” : {

"siteId” : 100,

"deviceId” : 1042

},

“pressure” : 1013.3997,

“temperature”: 34.9

}

This can even be expanded to the following:

{

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

“metadata” : {

“siteId” : 100,

“deviceId” : 1042

},

“measurements: {[

{“sensorId”:22, “value”: 1013.3997},

{“sensorId”:33, “value”: 34.9}

]}

}

At this point we’re heading very much into the world of basic JSON data document.

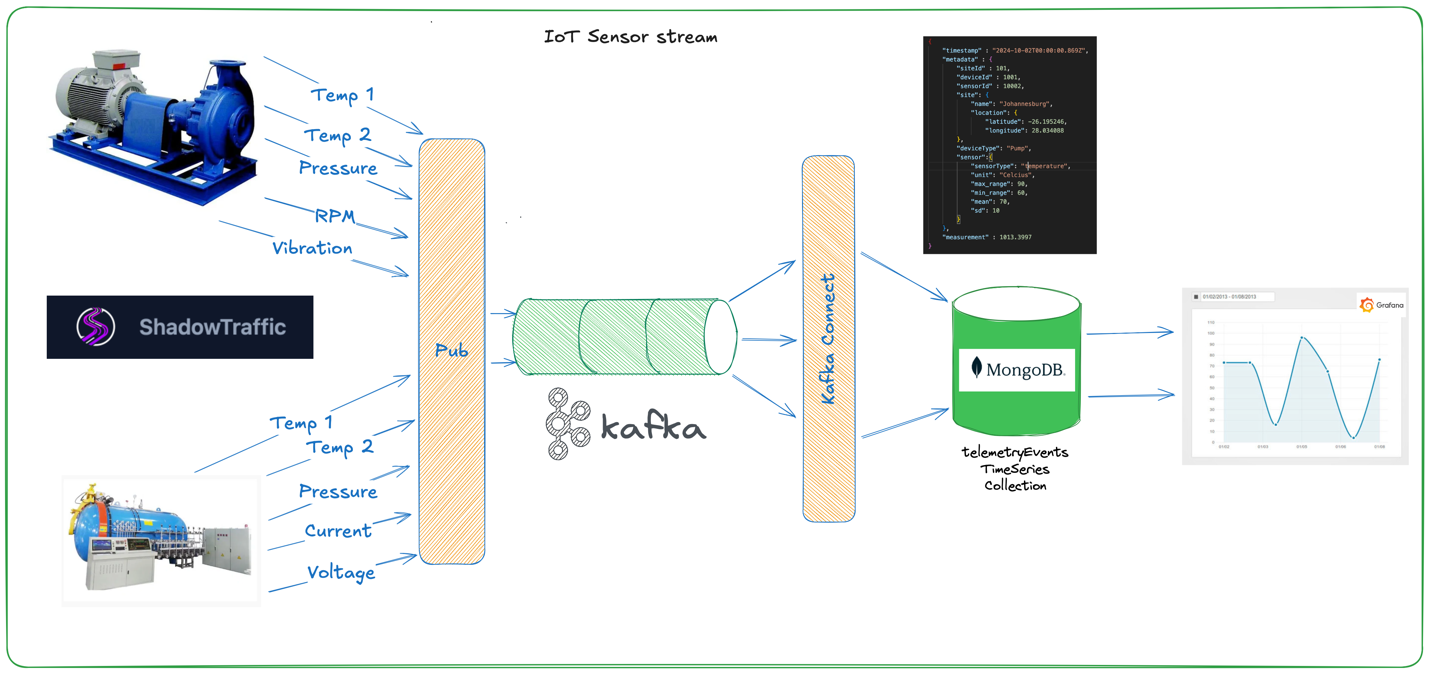
In the end the best way to proceed here is to test the various options at scale on your own data in your own environment and understand the various Pro’s and Con’s of each option.

Our little Demonstration scenario:

Below is a depiction of our imaginary manufacturer. The Company have multiple production facilities, each factory/location have multiple devices (pumps, motors, ovens etc.). Each of these devices has various types of sensors attached.

For our example we will look at 2 of these devices, a pump and a pressurized oven, known as a kiln, think of a company creating Carbon Fiber parts for Formula 1. ;)

The following diagrams depicts the above flow.



A solution like this would not normally have the sensors directly insert their measured values into the data store. The accepted standard is to push the values from source via a streaming architecture comprised out of technologies like [MQTT](https://en.wikipedia.org/wiki/MQTT) and [Kafka](https://kafka.apache.org/).

In our example the measurements, think of them as messages, or payloads, or readings are posted onto a Kafka topic “telemetryEvents”. We then use the Apache Kafka Connect framework to push/sink our [TimeSeries](https://www.mongodb.com/docs/manual/core/timeseries-collections/) specific data into a timeseries collection inside our [MongoDB Atlas](https://www.mongodb.com/lp/cloud/atlas/try4?utm_content=controldbaasterms&utm_source=google&utm_campaign=search_gs_pl_evergreen_atlas_core_prosp-brand_gic-null_emea-za_ps-all_desktop_eng_lead&utm_term=mongodb%20database%20service&utm_medium=cpc_paid_search&utm_ad=p&utm_ad_campaign_id=12212624560&adgroup=115749711543&cq_cmp=12212624560&gad_source=1&gbraid=0AAAAADQ1401r5oNkFLr-riVIionkqQ01S&gclid=Cj0KCQjwpP63BhDYARIsAOQkATY-mPMCGgUEzaUP8ZOaxxUC2-mBbowLGpzNHVZDXl8QC3ugW6L0dKQaAtGZEALw_wcB) datastore.

Once our data gets into our MongoDB Atlas environment we have access to the entire MongoDB suite of capabilities like [Atlas Stream Processing](https://www.mongodb.com/docs/atlas/atlas-stream-processing/), [Atlas Charts](https://www.mongodb.com/docs/charts/) etc.

To explore the above, I create a mini lab inside docker. All the code used is available via the following [GIT](https://github.com/georgelza/mongo-iot-timeseries-stream.git) repository.

The environment stood up consists out of:

* [Confluent Kafka Broker](https://www.confluent.io/)
* Kafka Schema Registry
* Kafka Connector
* [MongoDB](https://www.mongodb.com/)
* [Grafana Enterprise Cloud environment for visualization](https://grafana.com/)
* [Shadowtraffic](https://shadowtraffic.io/)

The environment is deployed using a docker compose.yaml file located in the devlab subdirectory.

The docker compose commands and all commands are executed using various make commands.

Once the environment is running you can run the shadowtraffic tool to create the fake telemetry events using the below command. Shadowtraffic is distributed as a docker image.

Shadowtraffic takes a configuration file as input and can then base on that produce many options of output data.

docker run --rm \

--name shadow \

-p 9400:9400 \

--network timeseries \

-v $(pwd)/conf:/workspace \

--env-file $(pwd)/conf/license.env \

--env-file $(pwd)/conf/params.env \

shadowtraffic/shadowtraffic:0.7.16 \

--config /workspace/config.json \

--watch --seed 341248291

The stack is started using the following commands (located in the devlab subdirectory):

* make run
* make deploy

Before executing make deploy make sure to edit the MongoDB Atlas credentials located in devlab/creConnect/.pwdmongoatlas.

Make deploy will first create our topic (using devlab/creTopics/creTopics.sh) on the Kafka cluster, followed by adding a schema definition record (as documented in devlab/creTopics/schema\_telemetryEvents.json) and lastly configure a Kafka Connect sink job (using devlab/creConnect/cremongoatlassinks.sh) which will store/send our data to our target MongoDB Database and collection.

Of course, before you can do this you will need to create a MongoDB account, and a Mongo Cluster. Following this create a timeseries based collection called “telemetryEvents”.

NOTE: the .pwdmongoatlas file is not in the git repo as it contains credentials so it’s listed inside the .gitignore file. The contents of the file are as follows:

export MONGO\_URL=mongodb+srv://user:password@mongoclusteraddress.net

export COMPOSE\_PROJECT\_NAME=timeseries

Note: The current configuration of the shadowtraffic job creates a much larger payload, which is currently matched to the larger schema entry created. The Kafka header record is also larger than required for a proper environment. This was all just done to show how information can be populated into different sections of the kafka message and eventually into our MongoDB collection.

Payload



Key

A screenshot of a computer

Description automatically generated

Header



For now I’ve configured a Grafana cloud account to visualize our timeseries values.



For now, this will be it… There are some ideas for a follow up blog.

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)