# An IoT Sensor Stream using timeseries based data.

## 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 constructed out of 3 primary components, namely:

* A timestamp, The When…
* Metadata: The What, one or more key/value pairs describing what was measured.
* The measured value.

Most timeseries data stores will normally also have an aging policy associated with it (to manage cost & size) via what’s store for how long at what granular level. At the start data might be stored based on per second values.

Depending on requirements a company might decide to retain this granularity of data for the first month, after which it’s rolled up into say a per minute averages for the next 12 months after which it’s rolled up/averaged into values per hour. My personal view… rolling up to a per day value removes to much Insite into what was measure, what happened, you lose too much value…

Below is a very simple example of a time series (IoT device) payload:

{

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

“metadata” : {

“deviceId” : 1023

},

“measurement” : 1013.3997

}

Or a financial example, depicting the RoE for a currency against the USD at a specific time,

{

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

“metadata” : {

“currency” : “Pound”

},

“measurement” : **1.31454**

}

Next is a bit more complex example, here we simply expand our metadata section with addition tags to filter on, as can be seen, we still have the when, what and measure value sections:

{

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

"metadata" : {

"siteId" : 100,

"deviceId" : 1023,

"sensorId" : 10180,

"unit" : "Psi"

},

"measurement" : 1013.3997

}

Now the following examples 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, which can be stored in a normal collection.

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.

So, my mind, the hamster (as I jokingly refer to it) work in strange ways… overnight decided to return here and expand with some thoughts, not defending any of the options, only going to list some considerations.

Let’s start with some simple numbers, we’re monitoring our below 2 devices, each with multiple sensors. As these are physical industrial machines, we’re going to say for that first month we want a 1 second view.

60s x 60min x 24hour x 31days = 2,678,400 messages per month.

If we were to roll this up to a per minute value/average per sensor for months 2-12:

60min x 24hour x 31days = 44,640 messages per month = 535,680 / year

Let’s just have that sink in for a second there. That can either be one message with multiple readings transmitted or that number for each sensor… if say 3 of the values for each reading is say, siteId, deviceId and timestamp, which is repeated, that’s allot of extra bits to be repeated.

Now consider the message payload size. We can send for each sensor it’s own per second message or, if we have a say local [PLC (Programmable Logic Controller](https://en.wikipedia.org/wiki/Programmable_logic_controller)) solution in place it might allow us to combine all the sensors readings per device per second into one payload, which is then send across the wire, (reducing traffic, number of messages and network bandwidth requirements).

Even if we have a great network, the more you transfer the higher the potential failure rate and contention for resources become.

A option could be, combine the multiple sensor readings per device into one message per second, transmit that and then split this up using [Atlas Stream Processing](https://www.mongodb.com/docs/atlas/atlas-stream-processing/) on our MongoDB Atlas cluster into individual records per sensorId and store them in our timeseries collection/s separately.

If everything is local, it becomes less of an issue, but give some consideration if you have multiple sites, some remote using expensive satellite communication, or even devices at the edge, say readings from a moving object, a car, boat, plane, train, which at times have “connectivity challenges” be that low bandwidth even simply expensive GSM based.

Basically, for IoT/time series-based data, due to the Velocity & Volume give some consideration to what is transferred and the payload shape/size.

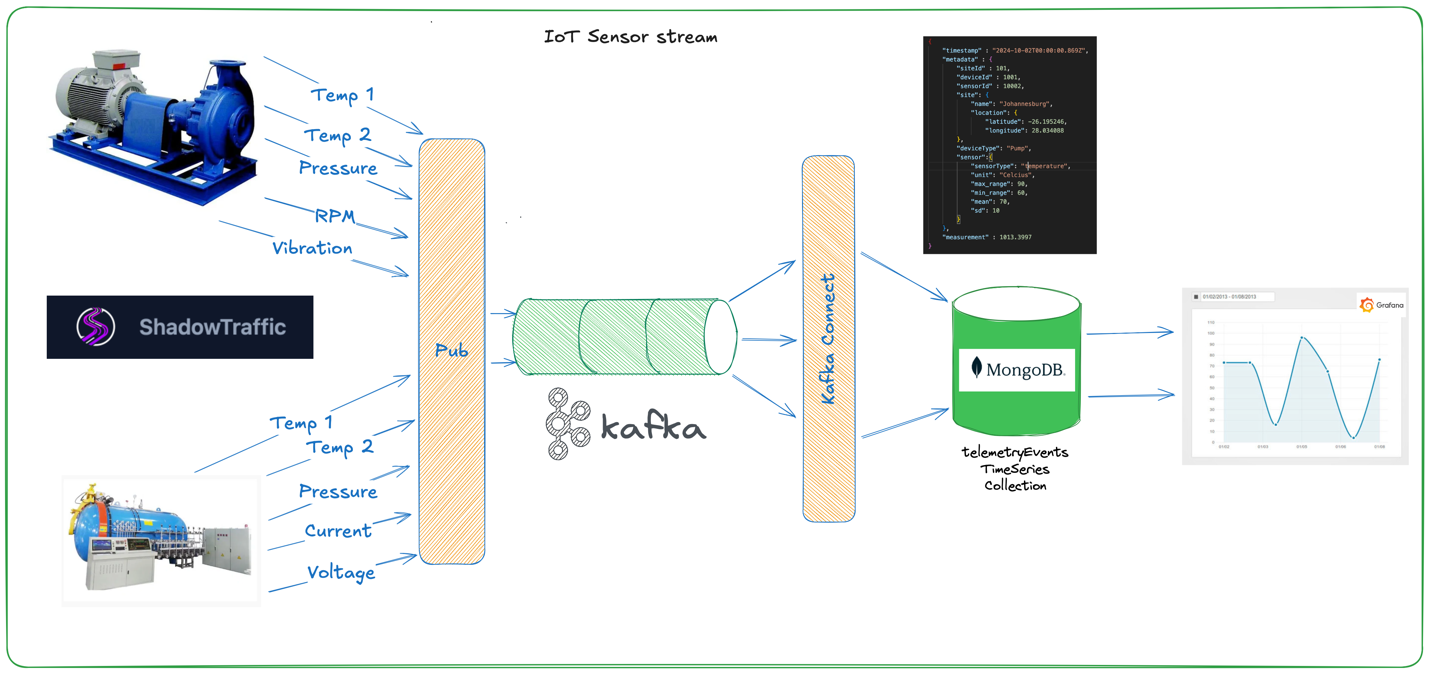
For consideration, the message structure/design at source and during transmission does not necessarily have to match how you store it and then analyze it.

Our little Demonstration scenario:

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

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.

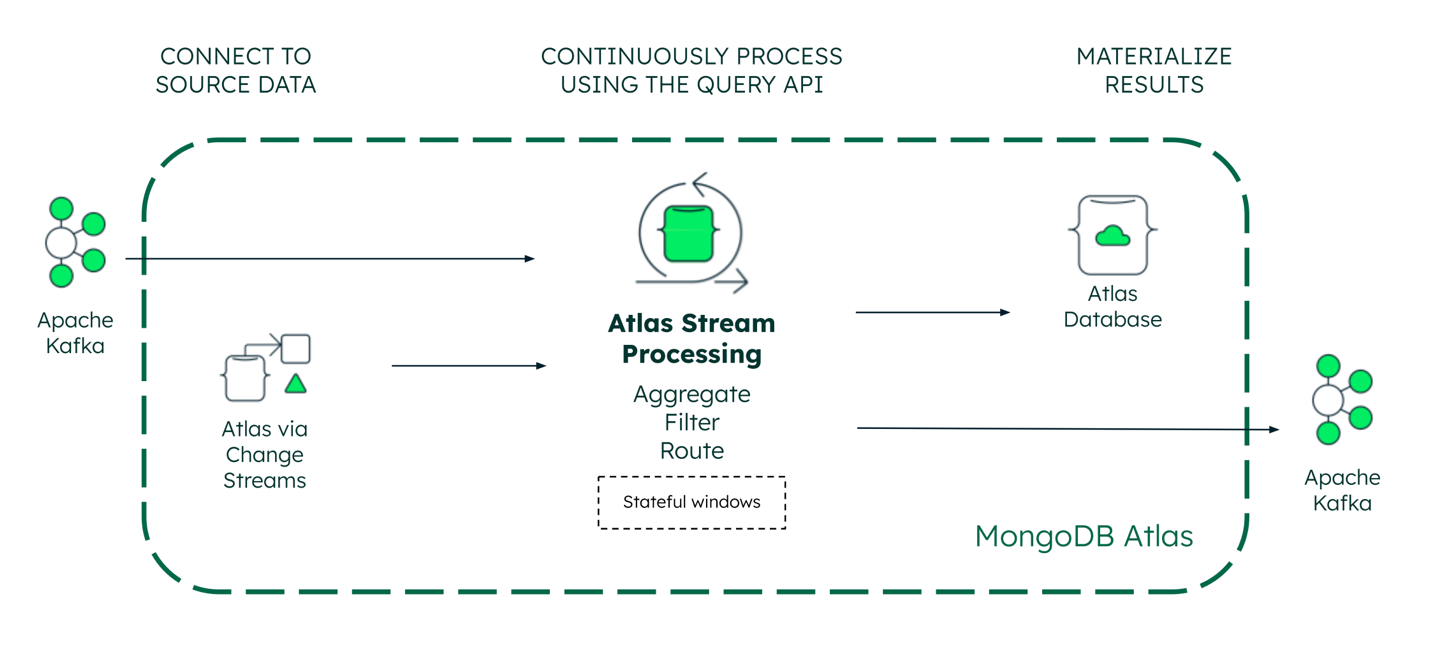


A solution like this would not normally have the sensors directly insert their measured values into the data store. The accepted standard is to push the values from source via a streaming architecture comprised out of technologies like [MQTT](https://en.wikipedia.org/wiki/MQTT) and [Kafka](https://kafka.apache.org/).

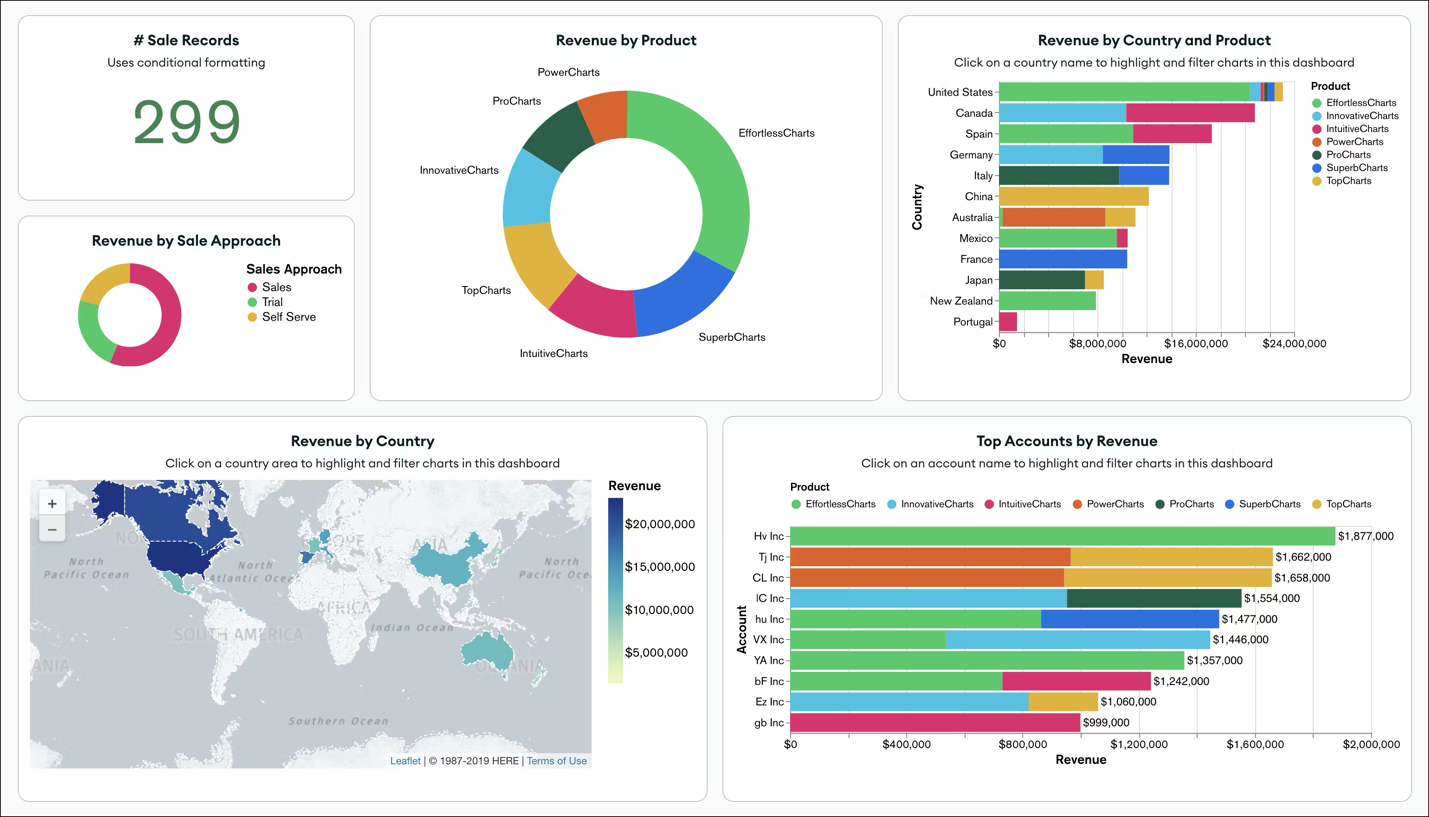
In our example the measurements, think of them as messages, or payloads, or readings are posted onto a Kafka topic “telemetryEvents”. We then use the Apache Kafka Connect framework to push/sink our [TimeSeries](https://www.mongodb.com/docs/manual/core/timeseries-collections/) specific data into a timeseries 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.

[Atlas Stream Processing](https://www.mongodb.com/docs/atlas/atlas-stream-processing/)



[Atlas Charts](https://www.mongodb.com/docs/charts/)



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/)
* [Shadowtraffic](https://shadowtraffic.io/)

External to the local Docker environment:

* [Grafana Enterprise Cloud environment for visualization](https://grafana.com/)

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](https://shadowtraffic.io/) is a creation by [Michael Drogalis](https://www.linkedin.com/in/michael-drogalis/) and distributed as a docker image, available from [hub.docker.com](https://hub.docker.com/r/shadowtraffic/shadowtraffic).

Shadowtraffic takes a configuration file as input and can then produce data based on a configuration file provided outputting to various options.

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

First, we need to build our Kafka connector with the required MongoDB sink connector/libraries. This can be accomplished by going into the devlab directory and executing:

(the image build will be tagged with the repo\_name value specified in the .env file in the devlab/ directory)

* Make build

Following the above build step the stack can be started using the following commands (to be executed 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 entry (as documented in devlab/creTopics/schema\_telemetryEvents.json) and lastly configure a Kafka Connect sink job (using devlab/creConnect/cremongoatlassinks.sh) which will sink/send our data to our target MongoDB Atlas 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 specific collection called “telemetryEvents”.

NOTE: the .pwdmongoatlas file is not in the git repo as it contains credentials and as such to keep GIT happy it is listed inside the .gitignore file. The contents of the file are as follows however:

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

export COMPOSE\_PROJECT\_NAME=timeseries

Note: The current configuration of my 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 done simply to show how information can be populated into different sections of the kafka message and eventually into our MongoDB collection.

**Payload**



**Key**

Currently I’ve configured to key to be the sensorId, ensuring all sensor readings end on the same Kafka topic partition, in the order they were created.

A screenshot of a computer

Description automatically generated

**Header**



For now, I’ve created a Grafana cloud account to visualize our timeseries values. The below depiction is of the raw measure data points, in a real-life environment the charts might be configured to rather show the mean value of the sensor readings per second.

Also of course accepting with a pinch of salt that a random data generator might not produce data exactly representing a physical device as it will behave. But then this is more than likely my ability to configure Shadowtraffic optimally at the moment still.



For now, this will be it…

There are some ideas in development for a follow up blog, looking at introducing a Machine Leaning for predictive analysis.

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)

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

<https://www.mongodb.com/community/creators/>

Annexure

Regarding [Shadowtraffic](https://shadowtraffic.io/) and what Michael has built here. There is allot I can say about this tool having been playing with it the last couple of weeks. The capabilities that are built into it is amazing, but what’s even more amazing is the simplicity of being able to build very complex data flows. It’s easy, well sort of too built a data generator, but to build something that’s extremely powerful and yet simple to configure use is a very different story all together.

I think [this](https://www.youtube.com/watch?v=-cxd8zEXm7c) example can speak for itself.