Why we chose timescaleDB over InfluxDB

Jeremy Theocharis

## Introduction

The introduction and implementation of an Industrial IoT strategy is already complicated and tedious and there is no need to put unnecessary obstacles in the way through lack of stability, new programming languages or more databases than necessary. You need a piece of software that you can trust the most important data in your company.

We are often asked why we chose [timescaleDB](https://www.timescale.com/) instead of [InfluxDB](https://www.influxdata.com/). Both are time-series databases suited for large amount of machine and sensordata (e.g. vibration or temperature).

We started with InfluxDB (probably due to its strong presence in the home automation and Grafana communities) and then ended up with timescaleDB based on three arguments. In this article we would like to explain our decision and provide background information on why timescaleDB makes the most sense for the United Manufacturing Hub.

## Argument 1: Reliability & Scalability

A central requirement for a database: it cannot lose or corrupt your data. Furthermore, as a central element in an Industrial IoT stack, it must scale with growing requirements.

### TimescaleDB

TimescaleDB is built on PostgreSQL, which has been continuously developed for over 25 years and has a central place in the architecture of many large companies like [Uber, Netflix, Spotify or reddit](https://stackshare.io/postgresql). This has created a fault-tolerant database that can scale horizontally across multiple servers. In short: it is boring and works.

### InfluxDB

In contrast, InfluxDB is a relatively young startup that is funded at [**119.9 M USD**](https://www.crunchbase.com/organization/influxdb) (as of 2021-05-03), but still doesn’t have 25+ years of expertise to fall back on.

On the contrary: Influx has completely rewritten the database twice in the last 5 years [[1]](#footnote-26) [[2]](#footnote-27). Rewriting a softwae can improve fundamental issues or add new exciting features, but is almost always associated with breaking changes in the API and new unintended bugs. This results in additional migration projects, which take time and risk system downtime or data loss.

Due to the massive funding we get the impression that they add quite a lot exciting new features and functionalities (e.g. [an own visualization tool](https://www.influxdata.com/how-to-visualize-time-series-data/)). However, after testing we noticed that the stability suffers under these new features.

In addition, Influx offers the horizontally scalable version of the database only in the [paid version](https://docs.influxdata.com/influxdb/v1.8/high_availability/clusters/), which will scare off companies wanting to use it on a larger scale as you will be fully dependent on the provider of that software (vendor lock-in).

### Summary

With databases the principle applies: **Better boring and working, than exciting and unreliable.**

*We can also strongly recommend here the article by* [*timescaleDB*](https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/)*.*

## Argument 2: SQL is better known than flux

The second argument refers to the query language, i.e. the way information can be retrieved from the database.

### SQL (timescaleDB)

TimescaleDB, like PostgreSQL, relies here on SQL, the de facto standard language for relational databases. Advantages: A programming language established over 45 years, which almost every programmer knows or has used at least once. Any problem? No problem, just google it and some smart person has already solved it on StackOverflow. Integration with PowerBI? Standard interface already integrated!

SELECT time, (memUsed / procTotal / 1000000) as value  
FROM measurements  
WHERE time > now() - '1 hour';

*Example SQL code to get the average memory usage for the last hour.*

### flux (InfluxDB)

*InfluxDB*, on the other hand, relies on the homegrown [**flux**](https://www.influxdata.com/products/flux/), which is supposed to simplify time series data queries. It sees Time series data as a continuous stream upon which are applied functions, calculations and transformations[[3]](#footnote-36).

Problem: as a programmer you have to rethink a lot, because the language is [flow-based](https://www.influxdata.com/blog/why-were-building-flux-a-new-data-scripting-and-query-language/) and not based on relational algebra. It takes some time to get used to it, but it is still an unnecessary hurdle for the not so tech savvy companie, who is already struggling with the Industrial IoT.

In addition, we can say from some experience that the language quickly reaches its limits and in the past we had worked with additional Python scripts that extract the data from InfluxDB via Flux, then process it and then play it back again.

// Memory used (in bytes)  
memUsed = from(bucket: "telegraf/autogen")  
 |> range(start: -1h)  
 |> filter(fn: (r) =>  
 r.\_measurement == "mem" and  
 r.\_field == "used"  
 )  
  
// Total processes running  
procTotal = from(bucket: "telegraf/autogen")  
 |> range(start: -1h)  
 |> filter(fn: (r) =>  
 r.\_measurement == "processes" and  
 r.\_field == "total"  
 )  
  
// Join memory used with total processes and calculate  
// the average memory (in MB) used for running processes.  
join(  
 tables: {mem:memUsed, proc:procTotal},  
 on: ["\_time", "\_stop", "\_start", "host"]  
 )  
 |> map(fn: (r) => ({  
 \_time: r.\_time,  
 \_value: (r.\_value\_mem / r.\_value\_proc) / 1000000  
 })  
)

*Example Flux code for the same SQL code.*

### Summary

**In summary, InfluxDB puts unnecessary obstacles in the way of not-so-tech-savvy companies with flux, while PostgreSQL relies on SQL, which just about every programmer knows.**

*We can also strongly recommend the* [*blog post by timescaleDB\* on exactly this topic*](https://blog.timescale.com/blog/sql-vs-flux-influxdb-query-language-time-series-database-290977a01a8a/)*.*

## Argument 3: relational data

Finally the argument that is especially important for production: Production data is more relational than time series based.

Relational data is, simply put, all table-based data that you can store in Excel in a meaningful way, for example shift schedules, orders, component lists or inventory.

{{< imgproc Relational Fit “1200x1200” >}}Relational data. Author: AutumnSnow, License: CC BY-SA 3.0{{< /imgproc >}}

timescaleDB provides this by default through the PostgreSQL base, whereas with InfluxDB you always have to run a second relational database like PostgreSQL in parallel.

If you have to run two databases anyway you can reduce complexity and directly use use PostgreSQL / timescaleDB directly.

## Not an argument: Performance for time-series data

Often the duel timescaleDB vs InfluxDB is fought on the performance level. Both databases are efficient and 30% better or worse does not matter if both databases are 10x-100x faster [[4]](#footnote-43) than classical relational databases like PostgreSQL or MySQL.

Even if it is not important there is strong evidence that timescaleDB is actually more performant. Both databases regularly compare their performance against other databases and InfluxDB never compared themselves to timescaleDB, but timescaleDB has provided [a detailed performance guide of influxDB](https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/).

## Summary

**Who do you trust more? The nerdy and boring, or the good-looking accountant, with 25 new exciting tools?**

The same goes for databases: **Boring is awesome.**

1. https://www.influxdata.com/blog/new-storage-engine-time-structured-merge-tree/ [↑](#footnote-ref-26)
2. https://www.influxdata.com/blog/influxdb-2-0-open-source-beta-released/ [↑](#footnote-ref-27)
3. https://www.influxdata.com/blog/why-were-building-flux-a-new-data-scripting-and-query-language/ [↑](#footnote-ref-36)
4. https://docs.timescale.com/latest/introduction/timescaledb-vs-postgres [↑](#footnote-ref-43)