

**BRAINE - Big data pRocessing and Artificial Intelligence at the Network Edge**

|  |  |
| --- | --- |
| **Project Title:** | **BRAINE - Big data pRocessing and Artificial Intelligence at the Network Edge** |
| **Contract No:** | 876967 – BRAINE |
| **Instrument:** | ECSEL Research and Innovation Action |
| **Call:** | H2020-ECSEL-2019-2-RIA |
| **Start of project:** | 1 may 2020 |
| **Duration:** | 36 months |

Technical Report for WP3

***Comparison of databases to receive and store telemetry data.***

|  |  |
| --- | --- |
| **Due date of Internal Report:** | 31 Feb 2021 |
| **Actual submission date:** | ???? |
| **Version:** | V1.0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | https://www.braine-project.eu/wp-content/uploads/2020/02/ecsel-1.png | **Project funded by the European Community under the H2020 Programme for Research and Innovation.** |  | |  | |  |

|  |  |
| --- | --- |
| Project ref. number | 876967 |
| **Project title** | BRAINE - Big data pRocessing and Artificial Intelligence at the Network Edge |

|  |  |
| --- | --- |
| Deliverable title | **Work Package 43Technical report** |
| **Deliverable number** | **Technical Report for WP3** |
| Deliverable version | Version 1.0 |
| Previous version(s) |  |
| Deliverable filename | *BRAINE\_telementry\_storage\_report.docx* |
| Nature of deliverable | Report |
| Dissemination level | PP = Restricted to other programme participants (including the Commission Services) |
| Number of pages | 25 |
| Workpackage | WP3 |
| Task(s) | T3.4 |
| Partner responsible | LUH |
| Author(s) | John Rothman (LUH) |

|  |  |
| --- | --- |
| Abstract | Technical report on different monitoring storage systems. Covering several general topics to asses which technology will be a better fit for the BRAINE project. |

**Copyright**

© Copyright 2020 BRAINE Consortium

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the BRAINE Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

**Table of Contents**

[1. Executive Summary 6](#_Toc65668142)

[2. Overview of Technologies 7](#_Toc65668143)

[2.1 Prometheus Database 8](#_Toc65668144)

[2.2 InfluxDB 9](#_Toc65668145)

[2.3 TimescaleDB 10](#_Toc65668146)

[2.4 Graphite 11](#_Toc65668147)

[3. Comparing Databases 13](#_Toc65668148)

[3.1 Data Structure 14](#_Toc65668149)

[3.2 General Information 15](#_Toc65668150)

[3.3 Server-Side Script Info 16](#_Toc65668151)

[3.4 High Availability 17](#_Toc65668152)

[3.5 Distributed Environment 18](#_Toc65668153)

[3.6 API and Other Aaccess Methods 19](#_Toc65668154)

[3.7 Data Types 20](#_Toc65668155)

[3.8 Resolution 21](#_Toc65668156)

[3.9 Performance 22](#_Toc65668157)

[4. Conclusion 25](#_Toc65668158)

# Executive Summary

This is a technical report, that in the context of WP3, which overviews and compares four time series databases. We first give a basic overview of the following technologies: Prometheus, InfluxDB, TimescaleDB, and Graphite. Then a comparison is made between the some of the technologies, and a final conclusion is made for which technology is a best fit for the BRAINE project.

# Overview of Technologies

This section will overview four different potential technologies.

# Prometheus Database

Prometheus is application used for monitoring events and alerting. The service provides a local disk storage for a time series database

The method Prometheus uses to collect data is with a HTTP [pull model](https://en.wikipedia.org/wiki/Pull_technology). Prometheus uses this pull based model which means that the Prometheus system collects the data and feeds it to the Prometheus database (local storage)

As of now, the Prometheus database is not easy to directly communicate with the Prometheus database. Looking at Fig 1. You can see that the Prometheus server pulls metrics from an endpoint, and stores it in the Prometheus database. As of right now there is no practical method to directly import data to the database. This alone makes the Prometheus database an impractical use for the BRAINE project. Another large point against the Prometheus database is that it only supports millisecond resolution timestamps, while most other databases support micro/nanosecond. For these reasons I'll exclude the Prometheus database from future comparisons.

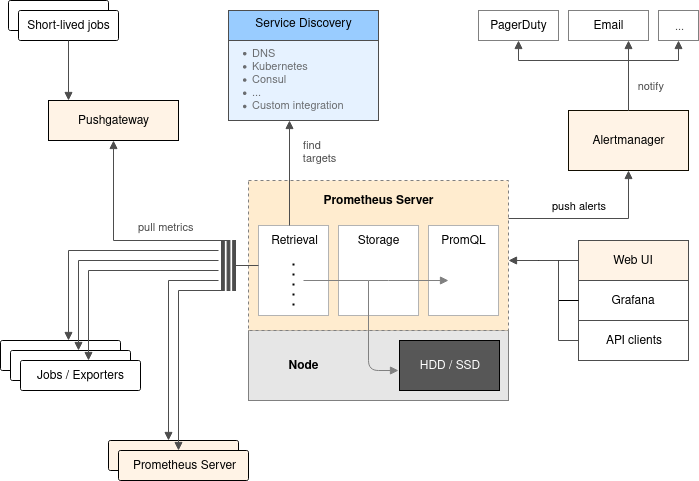


Fig 1. Prometheus high level architecture (<https://github.com/prometheus/prometheus>)

# InfluxDB

While InfluxDB has a free version, the enterprise version has some useful features locked down. During later sections we will only compare the free version. The data entries of influx are can be compared to SQL tables. The timestamp is always the primary index in InfluxDB. Additional information can be put in tags.

“InfluxDB is an open-source time series platform created in the language GO. This includes APIs for storing and querying data, processing it in the background for ETL or monitoring and alerting purposes, user dashboards, and visualizing and exploring the data and more.”[[1]](#footnote-2)

“If your InfluxDB performance requires any of the following, a single node (InfluxDB OSS) may not support your needs:

* more than 750,000 field writes per second
* more than 100 moderate queries per second ([see Query guides](https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/#query-guidelines))
* more than 10,000,000 [series cardinality](https://docs.influxdata.com/influxdb/v1.8/concepts/glossary/#series-cardinality)

If you want a single node instance of InfluxDB that’s fully open source, requires fewer writes, queries, and unique series than listed above, and do not require redundancy, we recommend InfluxDB OSS”. [[2]](#footnote-3)

# TimescaleDB

TimescaleDB is an open-source database created in the C language. It built upon PostgreSQL, and makes SQL scalable for time-series. The PostgreSQL provides partitioning across space and time, as well as full SQL support.

It is a great option if a SQL environment is preferred. TimescaleDB was developed with scalability in mind, so it is able to work well in distributed environments. The authors of Timescale claim:

**Easy to Use**

* + **Full SQL interface** for all SQL natively supported by PostgreSQL (including secondary indexes, non-time based aggregates, sub-queries, JOINs, window functions).
  + **Connects** to any client or tool that speaks PostgreSQL, no changes needed.
  + **Time-oriented** features, API functions, and optimizations.
  + Robust support for **Data retention policies**.

**Scalable**

* + **Transparent time/space partitioning** for both scaling up (single node) and scaling out (forthcoming).
  + **High data write rates** (including batched commits, in-memory indexes, transactional support, support for data backfill).
  + **Right-sized chunks** (two-dimensional data partitions) on single nodes to ensure fast ingest even at large data sizes.
  + **Parallelized operations** across chunks and servers.

**Reliable**

* + **Engineered up** from PostgreSQL, packaged as an extension.
  + **Proven foundations** benefiting from 20+ years of PostgreSQL research (including streaming replication, backups).
  + **Flexible management options** (compatible with existing PostgreSQL ecosystem and tooling).[[3]](#footnote-26346)

# Graphite

Graphite has a much simpler architecture than Prometheus because of its reduced scope, and is split up into 3 distinct parts:

* **Receive datapoints**: Graphite has 3 carbon daemons which handle receiving metric datapoints, with options to transform and aggregate datapoints into new metrics, or to provide a relay to pass data to multiple storage backends. Carbon passively receives metrics in various protocols, and has no requirements for them other than a simple string format. That allows metrics to be generated easily within an application and sent immediately and quickly by UDP, more reliably by TCP, or batched and bundled as line-separated datapoints or in a python pickle object.
* **Store datapoints**: vanilla Graphite stores metrics in the Whisper database format, which can be run locally or remotely. Carbon-relay handles replication or consistent hashing for sharded metric storage for redundancy and increased capacity. Whisper requires the total storage timeframe to be determined up front, including rules for rolling up resolutions over time. Once the storage criteria are determined the file is created, so the total amount of space used is taken up immediately and doesn’t change.
* **Present data**: Graphite uses a simple django app to display graphs of metrics, to query and transform them using Graphite functions, and to provide a render API to give access to external services like Grafana.

[[4]](#footnote-4)

Graphite is similar to Prometheus in that it is a push-based system, so you cannot store data with a high precision. Infact Graphite’s database manager “Whisper” can only support up to a one second resolution. Due to the lack of support of event logging, and the previously stated precision resolution, I will not include Graphite for future comparisons.

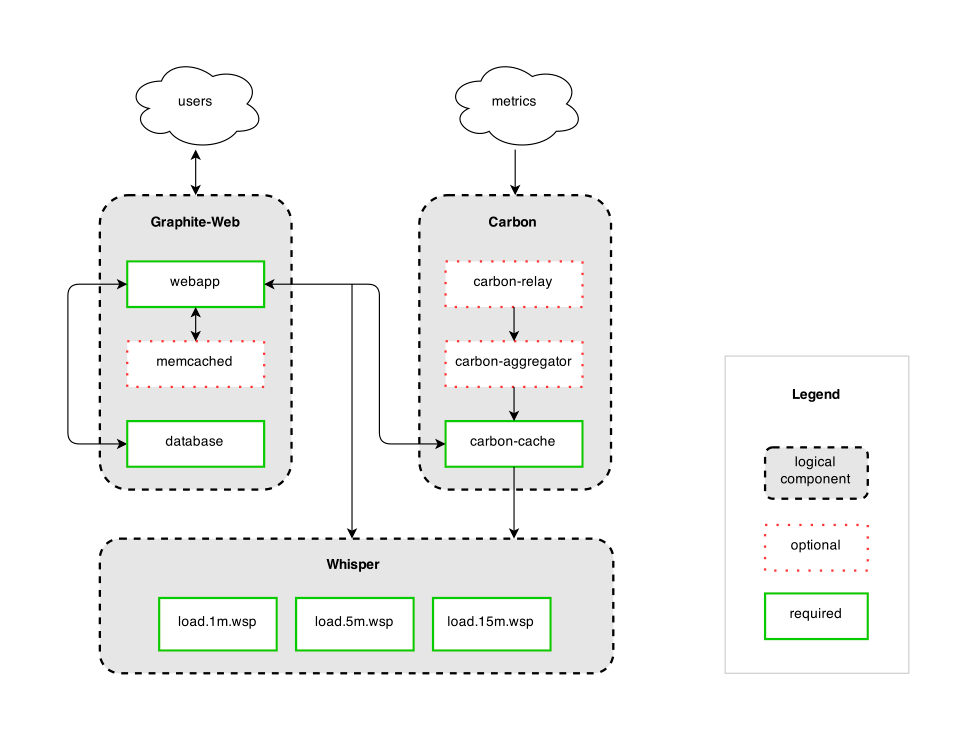


Fig. 2 Graphite High level Architecture overview (<https://cdn.buttercms.com/0ZVCntdXQlKuqRlU2HCr>)

# Comparing Databases

In this section we will only compare InfluxDB and TimescaleDB. As previously stated Prometheus database, and Graphite would not be good fit for the BRAINE project.

# Data Structure

|  |  |  |
| --- | --- | --- |
| Name | InfluxDB | TimescaleDB |
| Secondary database models | - | Relational DBMS |
| Data scheme | schema-free | yes |
| Foreign keys | no | yes |

Table 1.

In Table 1 we can see that InfluxDB supports a schema-free environment. “This allows the developers to have several benefits such as a greater flexibility over data types, no pre-defined database schemas, no data truncation, suitable for real-time analytics functions, and enhanced scalability and flexibility” [[5]](#footnote-5)

TimescaleDB uses PostgreSQL as it’s backend, which allows for High Availability and in general a larger ecosystem for support.

# General Information

|  |  |  |
| --- | --- | --- |
| Name | InfluxDB | TimescaleDB |
| Server operating systems | [Linux, OS X](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Linux, Windows, OS X](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| XML support | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| SQL support | [SQL-like query language](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| Triggers | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| Transaction concepts | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [ACID](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| Durability | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| In-memory capabilities | [Yes (depending on engine used)](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| User concepts | [Simple rights management via user accounts](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Fine grained access rights according to SQL-standard](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |

Table 2

The server operating systems show us which operating systems the software can be installed/ran on. XML support allows some form of processing of XLM formatted data, some examples could include XQuery or XPath. SQL support indicates if the software provides the ability to run SQL commands. Triggers represent the support for code that is executed in response to specific events. Transaction concepts support allows for the data integrity to be maintained after non-atomic manipulations of data. Durability support represents the ability to make data persistent. In-memory capabilities are the ability to have structures (some or all) to be contained in memory only. User concepts support gives access control of the database, and subsections of the database

Looking at Table 2, InfluxDB supports in-memory capabilities. The rest of the features are supported by TimescaleDB

# Server-Side Script Info

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| Server-side scripts (Stored procedures) | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [user defined functions, PL/pgSQL, PL/Tcl, PL/Perl, PL/Python, PL/Java, PL/PHP, PL/R, PL/Ruby, PL/Scheme, PL/Unix shell](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |

Server-side scripts represent the ability to call executable code (procedures) located on the server. In this case TimescaleDB allows for several options to execute scripts on the server, while InfluxDB is lacking in that regard.

# High Availability

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| Disaster/Crash Recovery | [Yes, all versions](https://docs.influxdata.com/influxdb/v1.8/administration/backup_and_restore/) | [Yes, uses postgresql](https://www.postgresql.org/docs/8.1/gist-recovery.html\) |
| Recovery Time Objective | No | Yes, see table below |
| Recovery Point Objective | No | Yes, see table below |
| Replication methods info | No | [Source-replica replication with hot standby and reads on replicas](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| Incremental Backup | [No](https://docs.influxdata.com/influxdb/v1.8/administration/backup_and_restore/) | [Yes (Postgresql)](https://wiki.postgresql.org/wiki/Incremental_backup) |
| Full Backup | [Yes](https://docs.influxdata.com/influxdb/v1.8/administration/backup_and_restore/) | [Yes (Postgresql)](http://https/docs.timescale.com/latest/using-timescaledb/backup) |
| Detection of failures | No | [Yes (Postgresql)](https://www.postgresql.org/docs/9.3/wal-reliability.html) |

The Disaster/crash recover represents the ability for the system to recover from a crash. Recovery Time Objective represents the amount of time the system can operate normally after network downtime. Recovery Point Objective represents the amount of data that would be loss during network downtime. Replication methods is the support for data redundancy across multiple nodes. Incremental backups support means the system is able to keep track of which files that have been changed since the last backup was made, while full backup is just a total copy of the entire dataset. Detection of failures is the systems ability to detect if a sub-part of the dataset goes down.

With respect to High availability, it looks like PostGreSQL offers many features that TimescaleDB is able to use. InfluxDB does offer High availability in the enterprise version, but the open-source version does not support this.

InfluxDB doesn’t appear to support RTO and RPO, but I was able to find the table below for TimescaleDB in a single server typical workload setting

|  |  |  |  |
| --- | --- | --- | --- |
| Capability | Basic | General Purpose | Memory optimized |
| Point in Time Restore from backup | Any restore point within the retention period  RTO - Varies  RPO < 15 min | Any restore point within the retention period  RTO - Varies  RPO < 15 min | Any restore point within the retention period  RTO - Varies  RPO < 15 min |
| Geo-restore from geo-replicated backups | Not supported | RTO - Varies  RPO < 1 h | RTO - Varies  RPO < 1 h |
| Read replicas | RTO - Minutes\*  RPO < 5 min\* | RTO - Minutes\*  RPO < 5 min\* | RTO - Minutes\*  RPO < 5 min\* |

(https://docs.microsoft.com/en-us/azure/postgresql/concepts-business-continuity)

# Distributed Environment

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| Partitioning methods | [No](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | [Yes, across time and space attributes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |

Partitioning methods support means the software is able to store information across multiple nodes. InfluxDB free verion does not support storing data on different nodes, while TimescaleDB supports Hash Partitioning.

# API and Other Access Methods

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| HTTP API | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | - |
| JSON | [Yes (over UDP)](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) | - |
| ADO.NET | - | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| JDBC | - | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| ODBC | - | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| native C library | - | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |
| streaming API for large objects | - | [Yes](https://db-engines.com/en/system/Graphite%3BInfluxDB%3BKdb%2B%3BPrometheus%3BTimescaleDB) |

With access methods we can see that InfluxDB and TimescaleDB cover completely different topics of access.

# Data Types

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| integer | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| float | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| string | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| boolean | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| date | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| arrays | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| JSON blobs | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| geospatial dimentions | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| currencies | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |
| binary data and customized data types | [No](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Yes](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |

Above we can see that TimescaleDB supports more datatypes.

# Resolution

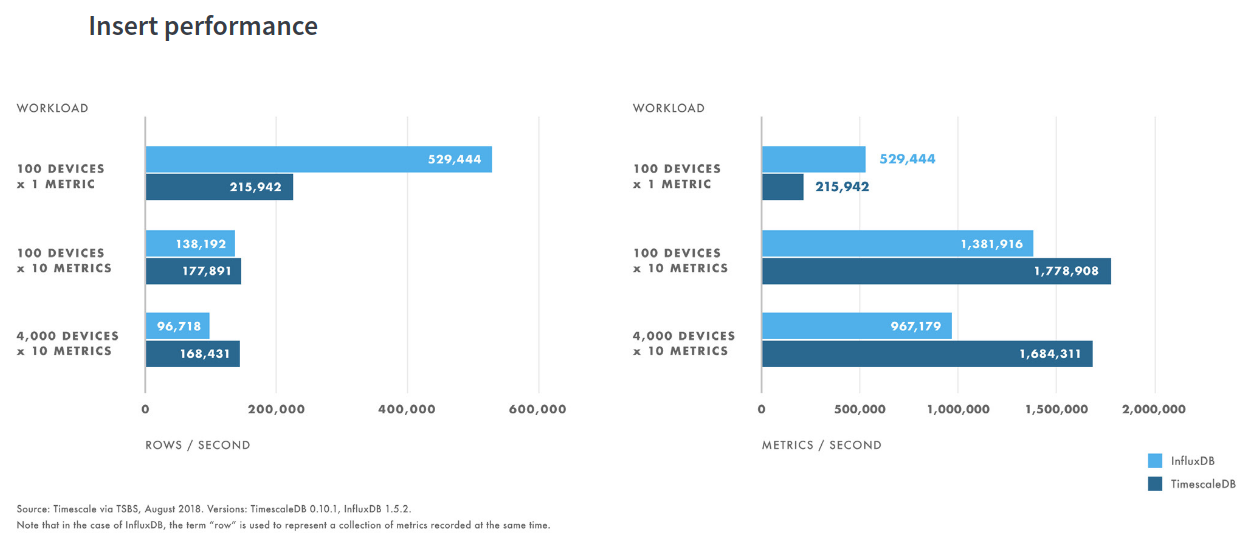
|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| Timestamp Resolution | [Nanoseconds](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) | [Microsecond](https://www.inovex.de/blog/timescaledb-vs-influxdb-zeitreihen-iiot/) |

InfluxDB supports a much higher precision of resolution.

# Performance

There isn’t much documentation on performance of InfluxDB vs TimescaleDB, but one source of information comes directly from a white paper produced by TimescaleDB. The results are informative, but should not be fully trusted considering they are comparing a competitor product. The full report can be found at

<https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf>

Fig. 3 insert performance

Here they show that influx performs better for workloads of low-cardinality, while TimescaleDb performs better when working with higher levels of cardinality

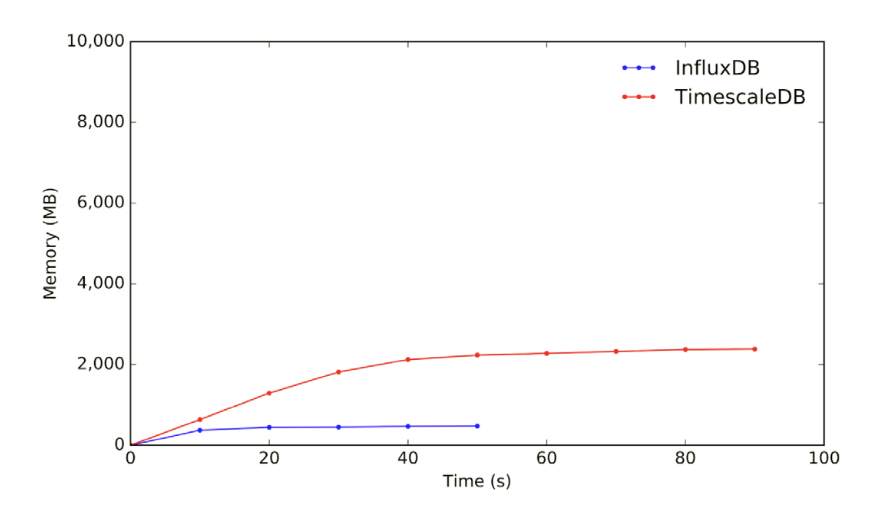


Fig. 4 - 100 devices sending one metric. Both lines represent sending the same amount of data, this indicates why the lines are not the same length

In figure 4 shows that in a low-cardinality workload, InfluxDB is faster and uses less memory than TimescaleDB. In this figure we can see some unfair bias being introduced. The axis scale goes up to 10,000 MB which has the effect of making it appear that the two lines are closer together then they actually are.

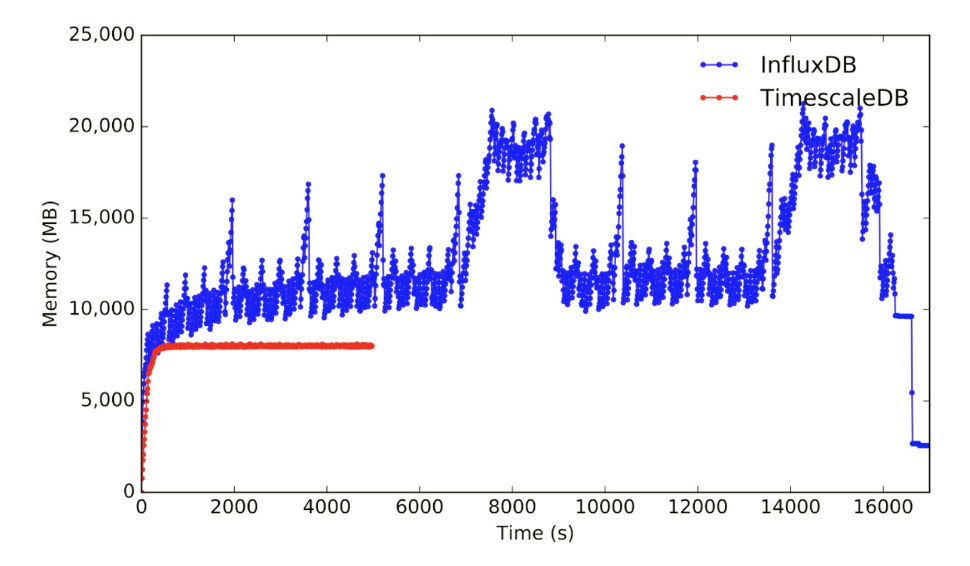
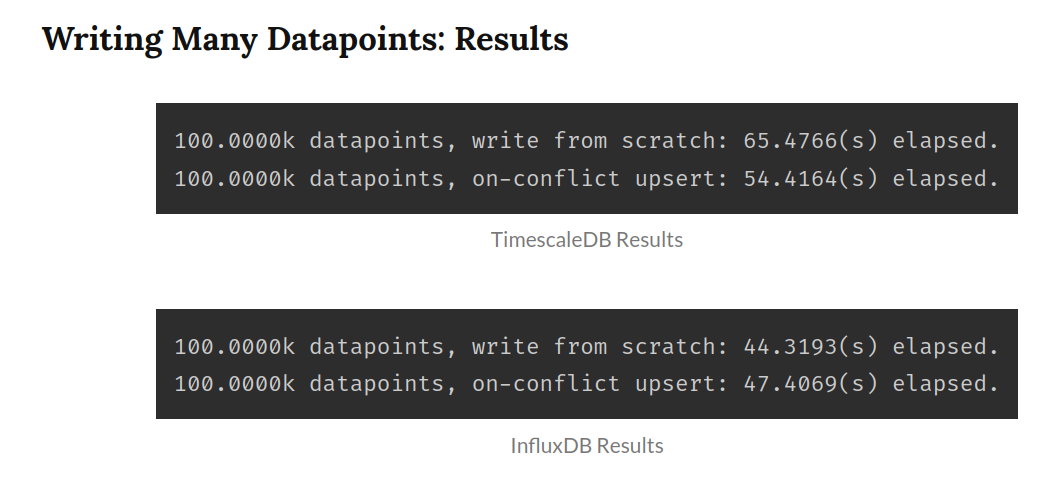


Fig. 5 - 100,000 devices sending 10 metrics

In Fig. 5 we are shown the network performance in a high-cardinality workload. They show in this case TimescaleDB takes less time and less memory.

|  |  |  |
| --- | --- | --- |
|  | InfluxDB | TimescaleDB |
| 100 devices x 1 metrics x 30 days | [12MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) | [700MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) |
| 100 devices x 10 metrics x 30 days | [113MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) | [1400MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) |
| 1000 devices x 10 metrics x 3 days | [769MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) | [5900MB](https://www.outfluxdata.com/assets/Timescale_WhitePaper_Benchmarking_Influx.pdf) |

Above we can see that Influx performs much better when it comes to storing the same data in a smaller space.



Img. 1 (https://bausk.dev/a-practical-comparison-of-timescaledb-and-influxdb)

Moving to an independent source with Img. 1, who tests write performance between the two. We can see that InfluxDB is slightly faster when writing 100k datapoints form a single device.

# Conclusion

Benefits for selecting TimescaleDB

* High Availability support
* Could have better performance in a network with high-cardinality
* Supports more data types
* Supports a distributed environment
* Transaction concepts with ACID
* Offers XML support
* Has a SQL stanard User Concepts support
* Developers familiar with SQL don’t need to learn a new language

Benefits for selecting InfluxDB

* Requires less storage space on smaller networks
* Stores data faster in low-cardinality network loads
* Offers In-memory capabilities
* Schemeless design for easier development
* Nano timestamp resolution

Both InfluxDB and TimescaleDB are viable options for the BRAINE project. Due to the schemeless design of InfluxDB, high timestamp resolution, and easability of implementation, we are going with InfluxDB. If High Availability and/or Distributed environment is required then we can switch to using TimescaleDB.

1. https://github.com/influxdata/influxdb [↑](#footnote-ref-2)
2. https://docs.influxdata.com/influxdb/v1.8/guides/hardware\_sizing/ [↑](#footnote-ref-3)
3. https://docs.timescale.com/latest/introduction [↑](#footnote-ref-26346)
4. https://www.metricfire.com/blog/prometheus-or-graphite/#Letrsquos-take-a-look-at-Graphite [↑](#footnote-ref-4)
5. https://www.mongodb.com/unstructured-data/schemaless [↑](#footnote-ref-5)