# InfluxDB vs. Elasticsearch for Time Series Analysis


When starting a new project that will eventually contain time series data (logging user activity, system logs, etc.), one needs to make a decision on which storage to use for this information. Elasticsearch, sometimes within the ELK Stack (Elasticsearch, Logstash, Kibana), is a popular solution for both the storage and visualization of such data. However, Elasticsearch's primary function is as a document indexing and search engine, raising doubts as to how well it performs in handling time series data. To address these doubts, we're going to compare Elasticsearch with another database popularly used for this purpose—InfluxDB.

## What Is Time Series Data?

Time series data can be defined as data points indexed by their temporal order, where the distance between two data points may or may not be equal. If the frequency at which data points are taken is constant (e.g., sampling the data every 10 ms) then the series is called a discrete time data series.

In computer systems, all user data can potentially be represented as time series data, as all stored information has a time component that can provide different metrics in different scenarios. For example, Twitter, Facebook, and LinkedIn have data on the user's registration date, as well as the dates and times at which various actions performed (tweet or article posted, activity liked, etc.).

Even though the data's time component is important in such scenarios, there are some other use cases in which it is crucial, as metrics calculated from this data are based largely on time

intervals. Some relevant examples include the tracking of user activities by Google Analytics and Netflix, or metrics tracking the function of running systems (e.g., JMX, operating system or network statistics).

Obviously, having data arriving at a higher frequency can create challenges, including having to handle a greater number of write requests per second and needing to store all the data. One sensor, with a sampling frequency of 30 requests per second and a payload of 1KB, can generate 86MB of information each day, meaning 100 sensors would create a data load of 8GB per day. Querying and aggregating such a large amount of data to extract useful information is another issue to be considered. Deciding on the right storage engine to use for time series data is one of the first challenges to overcome when designing a temporal data-generating system.

# What Databases Are Used to Store and Analyze Time Series Data?



One of the first steps towards storing and analyzing your time series data is choosing the right database engine for its persistence. In principle, you can use almost any database engine for storage, but issues can arise later, when you want to perform analysis on the information collected. Here, we are going to discuss two popular databases that are used to store and analyze time series data, mostly in combination with other tools.

## InfluxDB

InfluxDB is a database whose general purpose is to store time series data. The storage and querying of data are optimized for data points with a time component.

When it comes to storing data, the InfluxDB team has developed a storage engine that follows the LSM tree paradigm. The engine organizes its data in shards for each block time interval depending on the retention policy. If the retention policy is unlimited, then shards will be stored for seven days. Each shard is related to the database for which it is created. Besides the data shards, the storage engine consists of multiple other components like the in-memory index, WAL, cache for data stored in WAL, TSM files where the data is compressed and stored, FileStore, and more.

Time series can be organized in databases where you can logically organize the kinds of data you are planning to store. Measurements such as `disk_space`, `cpu_load`, and others can, in a way, be treated as tables in SQL, where the primary key is always the time component. These “tables” have to contain at least one key component that describes which data is stored (e.g.,

`core_1, server_12`) and its numeric value. When queried, the measurement table will return the timestamp, together with the keys and value for the stored data.

Writing data to InfluxDB can be done in many ways. You can use the command line interface provided, the client libraries for your language, or the REST API made available for both reading and writing data to the database. This API also allows you to create or drop databases and tables.

```
curl -G 'http://localhost:8086/query?pretty=true' --data-urlencode
"db=mydb" --data-urlencode "q=SELECT \"value\" FROM
\"cpu_load_short\" WHERE \"region\"='us-west'"
```

InfluxDB example that returns values from the *cpu_load_short* table

To facilitate the easy extraction of data from the database, InfluxDB provides a SQL-ish interface, so called because it does not make all SQL commands available. Data aggregation can also be performed on the database level without any need for external processing. Support for aggregation queries is built into InfluxDB and can be accessed through the SQL and REST interfaces. Some of the functions available for aggregation are `COUNT()`, `DISTINCT()`, `INTEGRAL()`, `MEAN()`, `MEDIAN()`, `MODE()`, `SPREAD()`, `STDDEV()`, and `SUM()`.

```
SELECT COUNT("water_level") FROM "h2o_feet"
```

Sample aggregation query

Also, InfluxDB has support for data transformation, selector, and even prediction queries. By calling functions, you can easily transform data before returning it to the client that is going to consume it (see https://docs.influxdata.com/influxdb/v1.3/query_language/functions/).

```
SELECT DERIVATIVE("water_level") FROM "h2o_feet" WHERE "location" =
'santa_monica' AND time >= '2015-08-18T00:00:00Z' AND time <=
'2015-08-18T00:30:00Z'
```

Sample data transformation query—returns derivative of water levels between each data point over some specified date interval

InfluxDB provides a simple web admin interface where you can run queries on the database, view data in tabular form, and also execute some DDL operations. In order to visualize your data, you can use other tools like Grafana (https://www.grafana.com/), where you can configure access to your InfluxDB data and visualize it with just a couple of clicks.

Installing InfluxDB is quite simple. Depending on the operating system, you can either use package manager or download the binaries and install it manually. Even though InfluxDB is an open-source database, released under an MIT license, it has some features that are not

open-sourced, such as those related to clustering. However, they are available in the InfluxEnterprise product offered by the maintainer, InfluxData.

## Elasticsearch

On the other hand, Elasticsearch is defined as a search and analysis engine, based on a popular indexing engine called Lucene. Since its initial release in 2010, Elasticsearch has gained popularity as a fast and scalable document indexing and search engine with millions of users worldwide.

Elasticsearch (ES) stores data in indexes, similar to relational databases, where data is logically separated. A single index can contain data for users (personal information, hobbies, etc.), companies (e.g., name, addresses, phone numbers), or other entities. The ability to split indexes into one or many shards is a crucial feature that allows ES to greatly outperform InfluxDB in the horizontal scaling, distribution, and parallelization of data. ES is designed as a distributed system in which it is easy to add more instances to the cluster—it will move shards and replicates to new instances automatically in order to maximize the cluster's availability.

Internally, ES relies on Lucene's implementation of inverted indexes, which can be viewed as a map of terms and the documents in which these terms can be found. This is useful in that it can return a subset of documents containing terms specified in a search query. An example of an inverted index is shown below:

| Term | Document |
|---|---|
| brown | 1, 2 |
| dog | 3 |
| fox | 1, 4 |
| jumps | 2, 3, 5 |
| lazy | 1, 2, 3 |
| over | 3 |
| quick | 1 |
| the | 1, 2, 3, 4, 5 |

Another data structure that significantly boosts performance, especially during aggregation queries, is the doc values structure. While inverted indexes map terms to documents, doc values maps the other way around, mapping documents to terms. Basically, a map consists of a list of documents and the terms that are contained in each document.

| Document | Words |
|---|---|
| 1 | Brown, fox, lazy, the |
| 2 | brown, jumps, lazy, the |
| 3 | Dog, jumps, lazy, over, the |
| 4 | Fox, the |
| 5 | Jumps, the |



Similar to InfluxDB, Elasticsearch provides a HTTP REST API and Java API for communication and data manipulation. There are also language-specific libraries available, which are usually wrappers around the aforementioned APIs.

The storage of time series data begins with defining mappings. In Elasticsearch, this means telling the engine how it should store the data, and also the fields that we are going to send for indexing. Defining this in advance will improve the subsequent performance when querying the data you need.

The timestamp added to your mappings should be one of the properties that you index. In previous versions, a `_timestamp` field was added to every record inserted by default, but this practice was deprecated in version ES v2.

Elasticsearch provides an aggregation framework that can be really useful during analysis, as it gives developers the option to perform aggregation over an entire set of documents, or time series data in our case. In contrast to InfluxDB, where you can only perform aggregation over numerical data, Elasticsearch can also handle textual data, which becomes very helpful when you have logs that contain messages, exceptions, and other text-based information.

In comparison with InfluxDB, Elasticsearch needs a little more configuration to run on large data sets. You need to define your mappings, which fields are indexed, what kind of data they contain (full text, numerical, etc), number of primary shards, and so on. Some of these settings—and others besides—cannot be changed once documents are added to the index, so they need to be considered carefully before you start using it. The only way to remedy problems later is to

actually create a new index with new settings, reindex all the documents, and then switch to the new index. Carrying this out on a live system is, of course, not something you want to do.

Elasticsearch is also used in combination with Logstash and Kibana (together called the ELK Stack) for log monitoring. Logstash is an open-source processing pipeline that gives you ability to import logs from different sources (files, message queues, databases, etc.), transform them, and output them to any store, including Elasticsearch. Kibana, on the other hand, gives a visual representation of the stored log files, and allows you to easily configure customized dashboards, tables, and other aids to visualization.

# Summary

Storing a large amount of time series data can be a tough task, one that requires considerable effort and research on which storage engine to use. Both InfluxDB and Elasticsearch have their pros and cons, and there are no hard and fast rules on which is the right one for a particular use case.

Even though InfluxDB is a relatively young database, it is more specialized in time series data. Benchmarking[1] has shown that it can handle a higher number of writes than Elasticsearch, and further development should allow it to easily become the leader in time series data storage. One downside of InfluxDB is that its clustering option is not open-sourced—this feature is only available in InfluxEnterprise, which is a proprietary package offered by the main project supporter, InfluxData. On the other hand, Elasticsearch is designed as a distributed system, facilitating scaling. Nodes can easily be added to the cluster without incurring any licensing costs through simple configuration changes, with Elasticsearch doing the rest of the work for you.

Also, if you plan on logging textual data in your database—log messages, exceptions, requests, or responses, for example—and later querying them by content, Elasticsearch is a better solution since it specializes in textual data searching. Using InfluxDB for this kind of logging and later querying might require the use of an additional search engine, which can cause further problems in terms of synchronizing the data between the two systems. As a final consideration, it is worth pointing out that the aggregation framework in Elasticsearch is more extendable, and is not limited to just numerical and textual data.

1