A Comparison of Time Series Databases for Storage and Usage of Data from Fish Farms

Adrian Langseth

September 15, 2020

1 Introduction

In choosing a time series database for the project, we needed to start at defining the necessary qualities of the database we would need to use. The main objective of this project was assessing whether such a fish farm data system is feasible to do using open source components. Therefore, a central demand for the database is that it must be open-source. Furthermore, we would like to setup the database on a server of our choosing. This is a demand which falls under the demand for open-source, as a database being open source overlaps with it being available to set up on own servers.
In the requirement from the customer, it was specifically laid out that the project must be horizontally scalable and allow for integration of new types of sensors. In the context of the database, this necessitates a push-based system. A push-based system is a system in which the application must push the data to the database any time new data is to be added to the database, whereas with the pull-based system, the data must be published at an endpoint from which the database pulls data at a set frequency. The pull-based system is not fit for our project as it does not allow for irregular, event-driven data collection, such as sensor detection of salmon louse. A pull-based system may still be viable if it integrates a push-based gateway.
This project is chiefly about data storage and a main requirement for this component is the long-term storage and persistence of the data collected. The customer required a 10-year unaltered storage period. This would disqualify any DBMS which automatically does any downsampling without the possibility of disabling this feature.
Lastly, we must be able to search in data utilizing metadata, such as searching for all sensor data coming from sensors in a specific location. This typically means that we must either setup a separate relational database, or integrating the metadata with the data.
In the next section we will consider four of the most-prevalent and suitable Time Series databases: InfluxDB, Prometheus, Graphite and TimescaleDB. We will assess these in light of the aforementioned requirements.

2 InfluxDB

InfluxDB is an open-source time series database developed for commercial integration on InfluxDB Cloud. It is free to download and use on our own servers.

Influx is originally a push-based system, but recently added support to its Kronograf for implementing pulling of data through the Prometheus scrapers.influxdata

InfluxDB allows definition of retention policies. These retention policies control how data is downsampled and removed. Through retention policies we can instruct InfluxDB on the required data persistence and resolution.

InfluxDB has no integrated solution for storing and coupling metadata, which would require the setup and connection of multiple external relational databases to allow for search in data on metadata. This is a drawback to InfluxDB as a viable candidate, but not a death blow. Although InfluxDB requires more work to implement, it would still be functional and fulfill requirements.

3 Prometheus

Prometheus is a free open-source database optimized for time series. However, Prometheus uses a pull-based system. An application must publish metrics at its endpoints and Prometheus periodically fetches this data. Prometheus provides an intermediary gateway called the *pushgateway*. This gateway does allow for pushing data to the database, but only allows a single timestamp which must be the timestamp of the data upload, not necessarily the sensor timestamp at the time of measurement. Hence, it is unsuitable for our project which requires support for both regular frequential time series data and irregular event-triggered data. Prometheus is not further considered for selection.

4 Graphite

Graphite is an open-source monitoring system with a three-component system: Graphite-Web: an analysis-providing web application, Carbon: a data-processing daemon, and a time series database called Whisper. Whisper uses a fixed size file based system. This means it has a defined resolution for data retention, meaning that it keeps a higher resolution for recent data and lowers resolution for the data as the data gets older. This is incredibly useful in many use-cases based on the temporal locality principle. On the other hand, this makes Graphite unsuitable for our project because of the requirement to keep data unaltered for 10 years.

5 TimescaleDB

Open-Source TimescaleDB is a free open-source relational database optimized for time series. Similar to InfluxDB it offers hosting in its cloud infrastructure,

however we will be hosting ourselves so this is of no use to us.

As a relational database supporting full SQL, TimescaleDB supports event-based data insertion. This means we can set our own regular handling of data, as well as setting up insertion-queries for event-driven, irregular data.

TimescaleDB enables data retention policies through automated data management tasks and a scheduling framework. We can define these ourselves to meet the requirements of the customer.

TimescaleDB implements relational databases with support for full SQL, including JOIN operations, allowing to either store the full metadata for each time series, as well as storing metadata in a separate relational database and utilizing it at query time. This results in easy metadata-based search, saving our team large amounts of overhead to implement metadata-based querying.

6 Conclusion

Of the four candidates, Timescale is the time series database best suited for our needs. It is a open-source time series database which allows for customization of retention policies. It allows both regular and irregular data, while being relational and allowing metadata storage and querying. InfluxDB would be a decent choice as it fulfills all our needs. However, with the time constraint on our project and considering the project main goal is to assess the viability of such a solution, TimescaleDB will be chosen due to its advantage of metadata integration.