INFO-H415 – Advanced Databases
Project : Time Series Database with InfluxDB

Academic Year 2018-2019

Authors:

Shabana Salmaan (000471119)

Danish Amjad (000473952)

Table of Contents

List of Figures

List of Tables

1 Introduction

This report illustrates what is time series and its purpose in real-time applications. In order to describe in detail, this report depicts its usage by Time series database application. The time series database application is Influx DB. This paper will give a clear idea on time series data, its uses in Influx DB and its real-time scenarios.

2 Time Series Databases

Time Series Database (TSDB) is an out-of-the-box database when compared to RDBMS or NoSQL as its purely related to time-variant data. The main purpose of TSDB should be for storing user history where it can't change over time. In today's scenarios, TSDB is used with the growth of data monitoring and governance. The following topic describes on the topic in detail:

2.1 What is Time series?

This introduction is to understand what Time series all is about and helps to know the model and its usage in real time applications. It will also describe on how to act while analyzing time series data.

The definition of Time series is a series of data points that are sorted (especially indexed as in list or trees) with the help of time attributes (in chronological sequence). This type of time-related data is used everywhere and in every industry. Example of such time series data is the stock price in the stock market, daily weather report, insurance, accounting, etc. Every system that uses this time series data should always have a datatype timestamp. This is a mandatory attribute where the user can query on it. So, the result of such query is always is time range. Finally, Time series data can be analyzed to any attribute that can change over time and keeps the record of such processes.

2.2 Understanding the Time series components

The main components that are considered in time series data are **noise, seasonality, trend, cycle, and level**. This is to note that not all these components are always considered in an application. Depending on the application the user uses, the components are used. For instance, weather forecast uses seasonality instead they don't use noise for any analysis purposes. Below is the detailed definition of the mentioned components:

- **Noise**: The unrelated time data that get captured during the process but are ignored during the analysis as it's not relevant to predict anything based on it. This is identified using the relevant algorithm.
- **Seasonality:** When a time data is based on regular intervals and predictable fluctuations that can be categorized as daily, weekly, monthly, quarterly or yearly, then that data is called as Seasonality. This component is domain specific.
- **Trend**: Based on a long-term path of visualization that can either be on the positive or negative side.

- **Cycle**: Repeating on a default value, for an event to happen on that value.
- **Level**: The time data which refers to mean of the series. This can also be called as a level index.

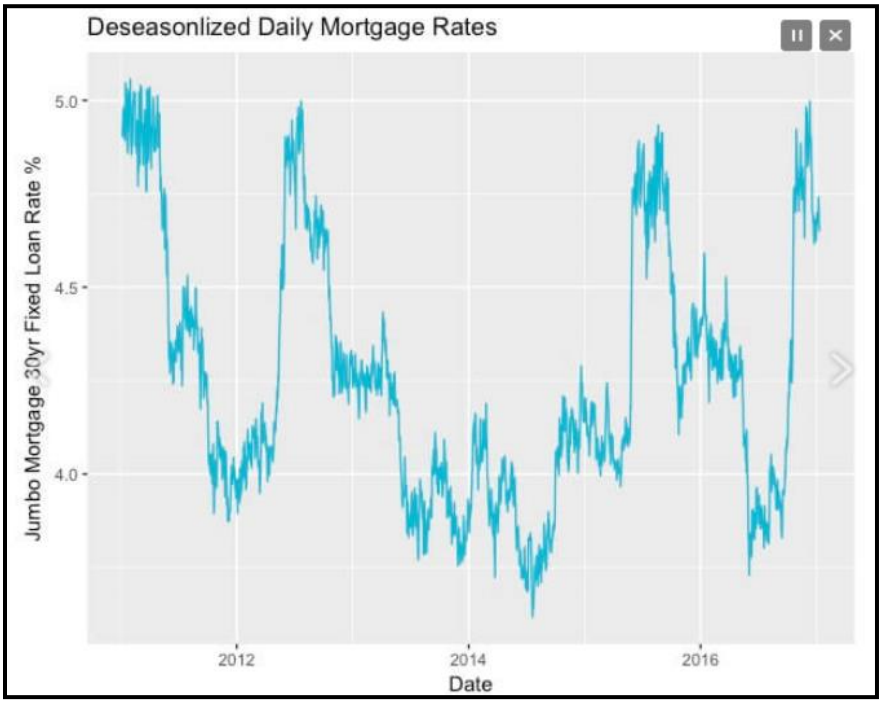


Figure 2-1 Zillow's jumbo 30-year fixed mortgage rates from the data world with the level displayed [1]

- A major advantage of time series is that it can be easily plotted in graphs but whereas it's tough to identify its components. Below graph depicts a loan service which uses a time series data over a range of time. As it is hard to see the components, we can use a technique to identify and that is called as **Decomposition**. Decomposition is to decompose or destruct the time series data into its components. The main purpose is to perform analysis based on the components and domain. Below is a sample figure showing various components of time series data based on the actual data.

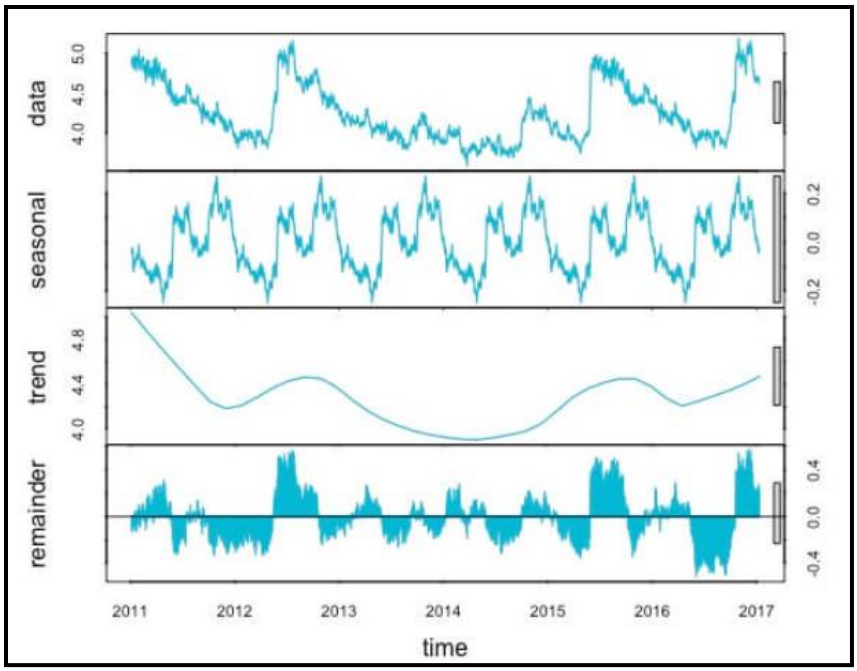


Figure 2-2 The components of the time series including the actual data and then seasonality, trend, and noise. [2]

2.3 Introduction to Time series database

The definition of Time series database is a type of database which is particularly built for time-stamped data. This can perform special operations like handling performance metrics, measurements, and time-stamped events. Like standard databases, the user will be able to perform DDL and DML to organize the time-stamped data in an effective way. The TSDB is specially designed to question the changes that took place over time. There are certain properties that differentiate TSDB from other standard databases, such as:

- Higher Write operation
- Data location based of regions
- Time-based easy range queries
- Lower granularity
- Usability
- Scalability

Due to these properties, the TSDB has gained popularity with the Internet of things (IoT is a set of physical devices that forms a network which can collect and exchange data to one another). Such emergence of IoT has been generating a large amount of data which has time-based attributes that keeps a track of all operations.

2.4 Why use a Timeseries database?

The question may arise “why can we use traditional or standard databases?” The answer is that we can use them to analyze time-related data. Then why it is necessary to use TSDB? Why

does the majority of users prefer TSDB (as per below graph comparison)? And why is that TSDB the fastest growing database?

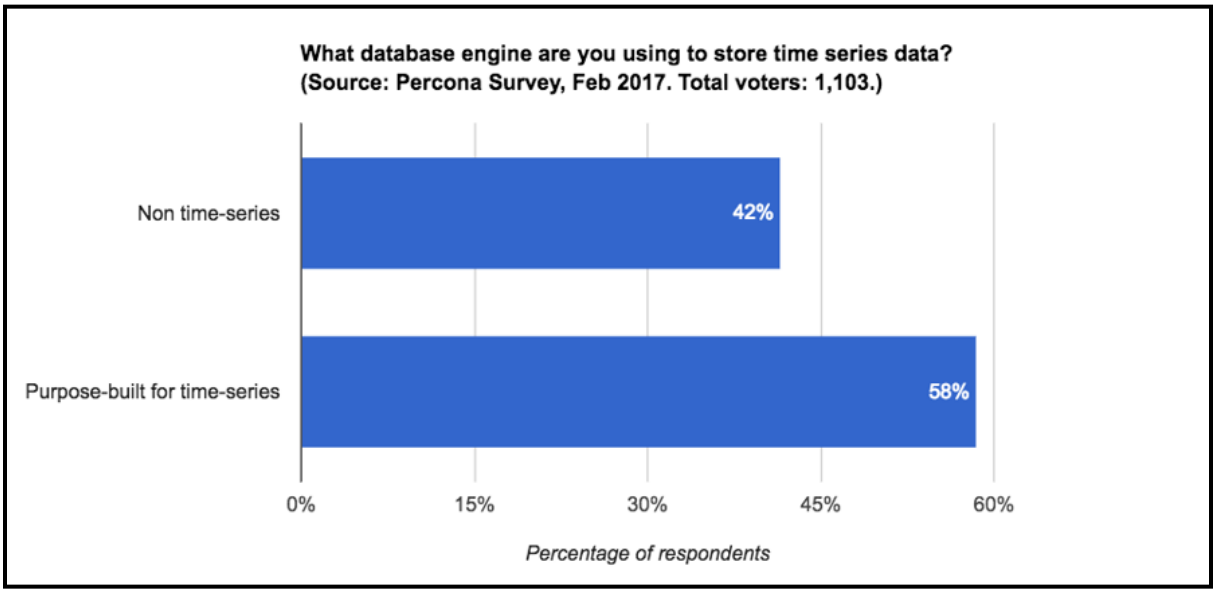


Figure 2-3 The percentage of respondents [3]

These questions can be answered due to two main reasons:

- **Scalability**: Time series which has large datasets gets collected faster whereas the standard databases are not designed to handle large datasets. Even NoSQL is better at scaling factor but can still perform well when fine-tuned. As in the case of TSDB (which is based on standard or NoSQL databases) can outperform by introducing effective scaling with time-related data. This result in improved performances, high ingest rates, better query execution at several scaling factors.
- **Usability**: TSDB has certain built-in functions and operations that can be used on data to analysis. Functions and operations such as continuous querying, time aggregation, retention policies, etc. These properties can still outperform a better experience to users and make the job easier.

These 2 factors are mainly considered among the developers to adopt a time series database and using them in the following scenarios:

- Monitoring software applications
- Tracking the physical system performance
- Asset applications
- Financial trading
- Business Intelligence

2.5 What are the requirements to use a Time Series Database?

The ability to scaling as the data grows is basically required for Time series uses. As discussed earlier, NoSQL can provide better scalability for the data growth than traditional standard databases but the TSDB provides additional aspects that are designed for time-related data that no NoSQL can be optimized to handle. The below are certain requirements to be considered when considering the TSDB:

- **Location of Data**: When the data related is not located on the same physical storage location. Queries can be slower than expected and even at times can get time-out errors. TSDB should be considered in such cases where a range of time data on the same location will be faster when accessing and provides effective analysis.
- **Fast range queries**: Analyzing such data requires to write queries with range values. Such voluminous data may cause the index to outbound the available memory and degrade I/O operations and can produce memory exceptions. NoSQL can be used to avoid such exceptions but must write complex queries. However, TSDB can have easier query writing to avoid these exceptions.
- **Higher performance**: TSDB provides priority on clusters which can be available for strong consistency, which ensures to provide high read and write performance during peak loads because they are designed to stay even in most situation.
- **Compactness**: As the time-related data gets old, the organization must be able to roll up the old data accordingly as the business requires and should be able to retrieve the data effectively.

2.6 Advantages of using Time-series Database

The following are the benefits of handling a time data considering its optimization:

- **Lower costs**: Easy and faster scale factor reduces the operational costs and hardware usage. This uses few resources and systems to manage outages.
- **The improved decision in business**: For analyzing data in real time, an organization can make faster and accurate decisions on system maintenance, changes in infrastructure, etc. which can impact the business decisions.
- **Infrastructure downtime**: When in case of power failure or network unavailability, TSDB is designed in such a way that can ensure that the data is available anytime even during such scenarios.
- **Higher performance**: TSDB can effectively support millions of data and can perform real-time analysis

3 InfluxDB in theory

3.1 Introduction

InfluxData is the company responsible for InfluxDB and they deliver open source platform to work on Time series data. InfluxDB is a modern engine created for metrics and events which is termed to be the leading time series platform and designed with the purpose to handle high performed loads. With the support of many key features, InfluxDB can be proved why the user prefers to work with time series data.

3.2 Example Use cases

This platform provides a variety of use cases. The following are topmost considered uses cases in Influx DB:

Internet of things and Sensor monitoring

As IoT data is real-time, time series and streaming data, InfluxDB provide a number of business services to encourage organizations to build solutions on IoT data. Services such as:

- Data Storage
- Aggregation
- Streaming analysis
- Data Visualization

Some of the customers from IoT platforms are Bevis, Citi Logics, Siemens, etc.

DevOps observability

InfluxDB is used among monitoring and controlling cloud infrastructure, database application and network components. During the critical business process, this helps the organization with services to identify and resolve issues. Services such as

- Precision monitoring
- Automation
- One place view
- Continuous delivery
- Distributed Environments

Some of the customers from the DevOps monitoring platform are Adobe, CISCO, eBay, etc.

The analysis in Real-time

To control and analysis, the volume of real-time is crucial with machine learning, InfluxDB helps in processing, analyzing and taking necessary steps on the data.

Some of the customers from real-time analysis platform are Xsellco, ToolsLib, ProcessOut, etc.

3.3 Holistic Picture

Here is a summary of some of the characteristics associated with InfluxDB.

Table 3-1 Properties of InfluxDB

Name of the property	Description
License	Open-source
Cloud-based	Yes
The language used to implement	Go
Server OS	Linux/OS X
Server-side scripting	No
Schema	Free
Datatypes	Numeric and Strings
Foreign keys	No
Secondary Indexes	No
Triggers	No
Partitioning	Horizontal partitioning
SQL	SQL
Languages supported	.Net, Go, Java, JavaScript, Lisp, Perl, PHP, Python, R, Ruby, Rust, and Scala
APIs	HTTP API

3.4 InfluxDB Rating

The DB-engine website monthly updates the rank of all databases currently used in real-time. Ranking the database is purely based on its popularity among the users. Additionally, the method of ranking calculation by using the following parameters:

- No of the quotation on websites
- Interest among the users
- Higher discussion rate
- Employment offers based on the system
- Higher the no of professional profiles

Considering the above-mentioned factors, in TSDB the top 10 ranked DB is as below figure from DB-engine website. From last few years, the InfluxDB is being ranked as #1 in TSDB and #36 in overall DBMS. The major reason for its popularity is that it's an Open source time series database.

26 systems in ranking, November 2018

Rank Nov 2018	Rank Oct 2018	Rank Nov 2017	DBMS	Database Model	Score Nov 2018	Score Oct 2018	Score Nov 2017
1.	1.	1.	InfluxDB	Time Series DBMS	13.64	+0.66	+4.30
2.	2.	↑ 4.	Kdb+	Multi-model	4.84	+0.47	+2.99
3.	3.	3.	Graphite	Time Series DBMS	2.85	+0.04	-0.01
4.	4.	↓ 2.	RRDtool	Time Series DBMS	2.73	+0.05	-0.47
5.	5.	5.	OpenTSDB	Time Series DBMS	2.02	+0.07	+0.32
6.	6.	↑ 7.	Prometheus	Time Series DBMS	1.95	+0.24	+1.14
7.	7.	↓ 6.	Druid	Time Series DBMS	1.36	+0.04	+0.38
8.	8.		TimescaleDB	Time Series DBMS	0.54	-0.03	
9.	9.	↓ 8.	KairosDB	Time Series DBMS	0.49	-0.05	+0.02
10.	10.	↓ 9.	eXtremeDB	Multi-model	0.30	+0.00	+0.00

Figure 3-1 Top 10 TSDB [4]

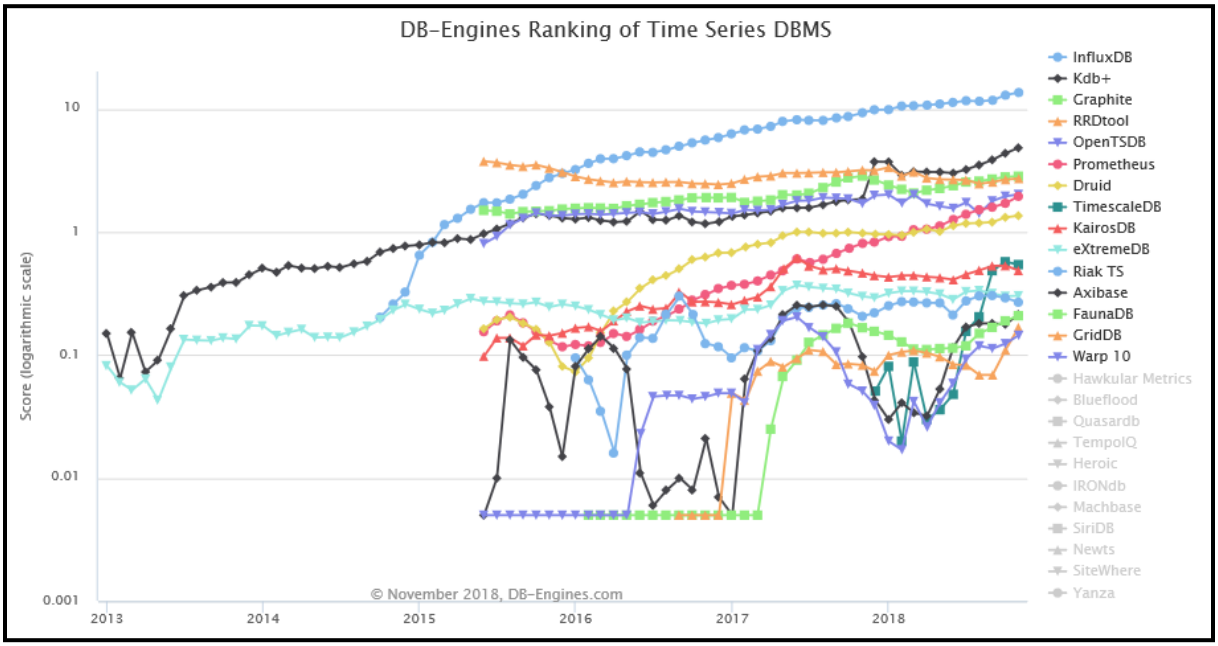


Figure 3-2 Growth of trend among TSDB [5]

3.5 InfluxDB modes

3.5.1 Opensource

Opensource TICK stack is the one among the 3 editions of InfluxData which is available through localhost for analyzing and monitoring the time series data. The following are the support that is available in the opensource TICK stack.

- Can support regular and irregular data
- Has Opensource core
- Extensible
- Support on the platform is optional
- Can be run on Cloud and Premises
- Can be used anytime by the user/developer

3.5.2 Premium (Cloud and Enterprise) + clusters

The premium edition of Influxdata platform is InfluxCloud and InfluxEnterprise. In addition to the opensource TICK stack support, the following are available support for the premium edition:

- Clustering has high availability
- Cluster Scalability
- Complete support over the platform
- Backup and restore options
- InfluxCloud runs on AWS whereas InfluxEnterprise can be run on any cloud
- Present or custom configuration

3.6 InfluxDB with friends

InfluxDB was created as a real-time analysis and monitoring application (like SaaS). Based on the user's interest, InfluxData created and open sourced other components. TICK together can do a lot of functions with Time-series data. The following are the TICK components:

T- Telegraph: It is a binary program that collects and reports Time series data across all servers

I- InfluxDB: Time series database to store and retrieve time-related data.

C- Chronograf: monitors and dashboard the Time series data for visualization purpose.

K- Kapacitor: Real-time ETL processing based on stream or batch mode.

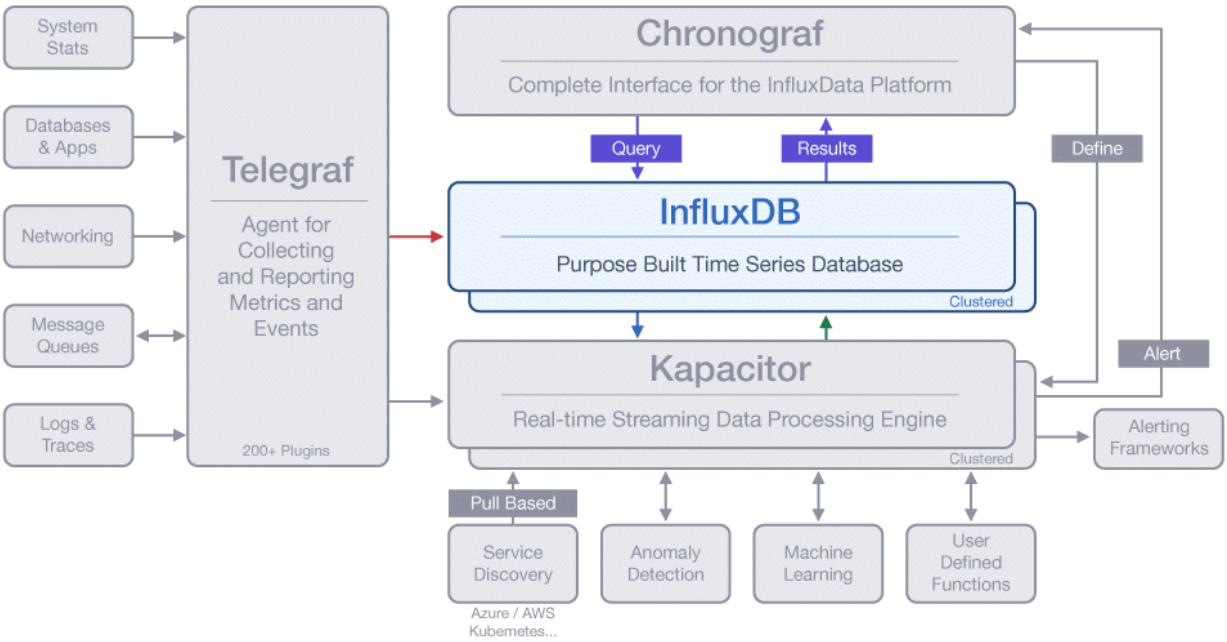


Figure 3-3 Overview of TICK Stack [6]

3.7 Data model

As mentioned in earlier chapters, A time series is a continuous stream of data points over time. So, the essential parts of the data model are measurement name, timestamp, and measurement value (field). For distinguishing different measurement values from different sources, we can add the use of *tags*. Moreover, if we need more than one measurement value, we can have a fieldset. InfluxDB supports all of these in its data model. [6] Here is the line protocol format used for writing data in InfluxDB [7]:

```
<measurement>[,<tag_key>=<tag_value>[,<tag_key>=<tag_value>]]
<field_key>=<field_value>[,<field_key>=<field_value>] [<timestamp>]
```

As we can see above, we need a measurement name, a list of key-value pairs for tags, a list of key-value pairs for fields and a timestamp.

A *retention policy* is also defined with each of the data points. A retention policy deals with the physical storage and purging of data. It defines the time-to-live for data. A time *series* can be defined as a combination of the *retention policy*, *measurement*, and *tags.*

In SQL jargon, a *measurement* can be regarded as a table. *Tags* and *values* can be regarded as attributes of the table. There is no need of explicit schema design. Whenever the data is inserted, the schema is inferred from the data.

3.8 Physical Implementation

InfluxDB looks like two-in-one database because of its management style of inverted indexes and actual data. [8]

3.8.1 Data storage

The storage engine for InfluxDB has different components. We'll try to explain the most important ones briefly.

3.8.1.1 Components

3.8.1.1.1 TSM Files

TSM files are physical incarnations of Time-Structured Merge-Tree (TSM Tree). A TSM file stores time series data in compressed form in a columnar fashion [9]. It's quite similar to LSM trees. The difference lies in the sharding of data based on some retention policies. If we had 4 days retention policy, the TSM files would be organized in shards containing data for 4 days.

The data in a TSM file is structured as shown in Figure 3-4. The file starts with a 5 bytes header. A sequence of blocks follows the header. Lastly, it contains Index and Footer entries.

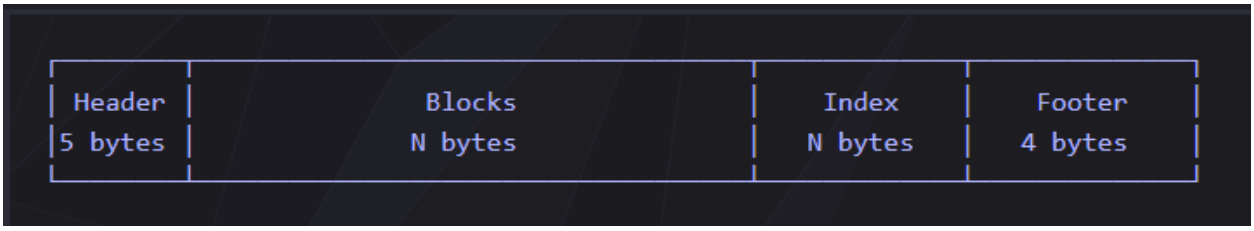


Figure 3-4 TSM File Structure. [9]

The footer contains the offset of the index in the TSM file. The Index contains the inverted index of time series. The index entries are ordered by *key* and time lexicographically. The index contains keys and all the blocks where those keys preside. Figure 3-5 shows the structure of the index. Type and Count represent the data type of the block and number of places in the data segment for a key. Block Metadata gives us information about the location and size of the block in the data segment. The data segment is represented as Blocks in Figure 3-4.

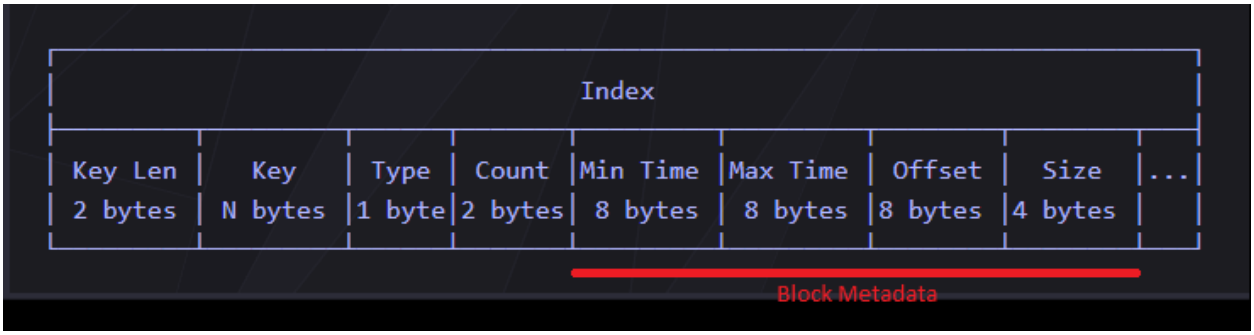


Figure 3-5 Index Structure [9]

For each data type, there are different **compression** strategies [9]:

- **Float:** XOR encodings as implemented in Facebook Gorilla.
- **Integers:** Zigzag encoding with simple8b encoding.
- **Booleans:** Simple bit packing.
- **Strings:** Snappy Compression

3.8.1.1.2 WAL

WAL files are high throughput write files. Every new entry or deletion is compressed and written first in WAL files. WAL files are organized in 10MB segments. Each of the segment relates to the data in Cache. All the new data is guaranteed to be written in WAL files using *fsync*.

3.8.1.1.3 Cache

The Cache is an uncompressed copy of all the data in WAL files. All the data in the cache is stored in memory. It is flushed to disk when either it has reached its memory limit or after a configurable amount of time. All the data is then flushed to TSM files.

3.8.1.2 Operational Scenarios

Here are some operational scenarios for understanding the underlying architecture in detail.

3.8.1.2.1 Inserts

Whenever there is a new entry in the database. It is inserted in a WAL file, added in Cache and inserted corresponding data in In-memory Index.

3.8.1.2.2 Updates

Whenever there is an update, a new record is added similar to Insert scenario. When the cache is flushed to disk the corresponding entry gets overwritten.

3.8.1.2.3 Deletes

Whenever there is a delete, an entry is added in a Tombstone. Whenever there is a query, the deleted data is removed from the result set after consulting with Tombstone. This is quite expensive, and it is resolved through compaction.

Normally, the deletes are a result of purging data due to the retention policy. In that case, the whole shard file is deleted, and memory indexes are refreshed.

3.8.1.2.4 Compaction

Compaction is taken care by the module Compactor. The compactor compacts the data when the WAL data needs to be written in disk files say L1. When the number of L1 files becomes large these are compacted until L4 level is reached. These files may contain different series packed together in a single file. An index optimization compaction separates different series in different files possible.

3.8.2 Indexing

There is no need of manual indexing of data, InfluxDB takes care of that. All the data which is inserted in InfluxDB is indexed automatically. There are two approaches to indexing looming around. First one is In-memory Index and the other one is Time Series Index (TSI).

In-memory Index is loaded at startup by reading all the index entries in all the TSM files. This index is quite useful and handy for smaller cardinality of *series*. For large cardinality of series say hundred million. This number will require a very large infrastructure possessing a large amount of main memory. For this reason, a newer approach is being introduced.

TSI moves the index files to disk as soon as it becomes cold. TSI achieves this functionality by using a memory map and using *Least Recently Used* (LRU) cache of the operating system. Since all the indexes are not in memory, it uses *HyperLogLog++* to keep sketches of cardinality estimates and *RobinHood* hashing for index lookups [8].

3.9 Querying

Ease of use is one of the goals of InfluxDB. That is why it supports a variety of different interfaces and query languages.

3.9.1 Interfaces

3.9.1.1 HTTP Interface

InfluxDB provides a REST interface with JSON to interact with it. It also provides basic security through HTTPS and JWT Tokens. It provides debug, ping, query and writes endpoints for debugging, checking the status, querying and writing data. [10]

3.9.1.2 Chronograf/Grafana

Chronograf and Grafana both use HTTP endpoint for interacting with InfluxDB. Both Support Visualizations, Alerting, and administration.

3.9.1.3 CLI

InfluxDB CLI also uses HTTP endpoint. It is an interactive shell for data and database management, querying the database and displaying the results in different formats.

3.9.1.4 Programming Language

Due to its flexible REST endpoint, InfluxDB provides a wide variety of client libraries. Only the Go library is maintained by Influx Data and all other libraries are maintained by open source communities. Some of the languages which have support for InfluxDB client library are:

- Go
- Java
- Python
- Javascript/Node.js
- MATLAB
- .NET

3.9.1.5 Kapacitor

Kapacitor is data processing engine for InfluxDB. It can process stream data and batch data as well. It also provides a domain specific language TICK scripts. It uses the functional approach of lamdas to write transformations on the data and perform complex *tasks* i.e. anomaly detection, alerting etc.

3.9.2 Querying Languages

Influx Data provides different styles of querying data in influxDB.

3.9.2.1 InfluxQL

Influx Query Language (called ad InfluxQL) is a SQL-like language developed to communicate with InfluxDB. It is like standard and traditional SQL built with improved featured which are necessary to store and perform analysis TSD. Here is the generic syntax:

SELECT <field_key> [, <field_key>, <tag_key>] FROM measurement_name> [,<measurement_name>] [WHERE <field_key> = <field_value>] [GROUP BY [time, tags]] [HAVING predicate]

Table 3-2 Operators

Notation operators	Order no(precendence)	Used in
|	1	Alternative
()	2	Group
[]	3	Option(0 - 1)
{}	4	Repetition(0 - n)

InfluxDB is implemented using GO language. Table 3-2 the notations using in InfluxQL which are like GO language. It provides a wide array of built-in functions for querying, filtering and writing data in InfluxQL. A demo application is also presented in a later section to explain the power of InfluxQL. The measurement names are treated as tables. Fields, tags and timestamp are treated as attributes. The difference in the results from SQL is that the timestamp field is always returned in the result set. Appendix A shows some more tables that give us more insight into InfluxQL keywords, functions, and operators.

3.9.2.2 TICK Scripts

It is a Domain Specific Language (DSL) used in Kapacitor for defining jobs for data processing and acting on that data. Kapacitor fetches the data and does the processing afterward. It uses function-chaining to define *pipeline*s of data processing. These pipelines are registered in Kapacitor as *tasks*. Functions can be considered as *nodes* in a Directed Acyclic Graph (DAG). Each node requires an input and produces an output. These nodes are chained with edges. Edges are defined by *pipe* operator. These nodes have certain properties which can also be modified using property functions which are accessed with a dot (.) operator. With TICK Scripts it's easy to do several tasks based on data in InfluxDB i.e. alerting, showing data at REST endpoint, invoking external service, joins histograms, getting input from a variety of sources and pushing data to a variety of data sinks.

3.9.2.3 Flux

Flux is a new programming language for querying time series data. It's being developed to be used as a standard which can be adopted by other databases. Until now, in v1.7, a lot of its functionality is developed, and a lot is under development. Influx Data has plans to release a production ready version in InfluxDB 2.0.

Flux combines three main features of TICK stack into one. It provides a querying engine/a querying language, a data processing framework and alerting mechanisms just like InfluxQL, TICKScripts, and Chronograf respectively. Having all of these functionalities built in one component really enhances the performance and productivity of its users.

Flux is designed in such a way that it prioritizes programmers' productivity upon language purity. It's Javascript-like syntax and LISP-like functional programming paradigm makes it highly usable and readable. InfluxData has designed this language to ensure that it is contributable and shareable. Oftentimes, it happens that a lot of users need code for a common scenario. Like other programming languages, we can just import the module written by someone from a public repository with just a link.

Flux treats each series as a table. And each series is defined by a measurement name, a tag set, and a field name. We can join different series with a join operator and we can also perform mathematical operations from different series afterward. Flux provides a range of data sources and data sinks i.e. InfluxDB, file, Kafka, S3, MySQL etc. and user-defined libraries. [11]

Figure 3-6 shows an example code of Flux. This simple code defines a scheduled task which runs every minute and checks for buffered disk usage alerts in *alerts* database, filter critical alerts and sends these alerts to Slack and *notifications* database. This shows us how a Flux code could look like after its release in v2.

```
option task = {name: "slack critical alerts", every: 1m}
import "slack"

lastNotificationTime = from(bucket: "notificatons")
  |> filter(fn: (r) => r.level == "critical" and r._field == "alert_time")
  |> group(none:true)
  |> last()
  |> recordValue(column:"_value")

from(bucket: "alerts")
  |> range(start: lastNotificationTime)
  |> filter(fn: (r) => r.level == "critical")
  // shape the alert data to what we care about in notifications
  |> renameColumn(from: "_time", to: "alert_time")
  |> renameColumn(from: "_value", to: "used_percent")
  // set the time the notification is being sent
  |> addColumn(key: "_time", value: now())
  // get rid of unneeded columns
  |> drop(columns: ["_start", "_stop"])
  // write the message
  |> map(fn: (r) => r._value = "{r.host} disk usage is at {r.used_percent}%")
  |> slack.to(config: loadSecret(name: "slack_alert_config"), message: "_value")
  |> to(bucket: "notifications")
```

Figure 3-6 An example of Flux code [11]

4 InfluxDB Application

4.1 Description of data

Influxdata has provided a publicly available time series data from **National Oceanic and Atmospheric Administration's (NOAA) Center for Operational Oceanographic Products and Services**. We have used this data for demonstrating the usage of InfluxDB.

This data contains around 15k water level readings from two sensors from (Santa Monica, CA (ID 9410840) and Coyote Creek, CA (ID 9414575)) over the period from August 18, 2015, through September 18, 2015. [12]

Please note that some of the measurements such as *average_temperature*, *h2o_pH*, *h2o_quality*, and *h2o_temperature* are fictitious.

4.2 Setup instructions

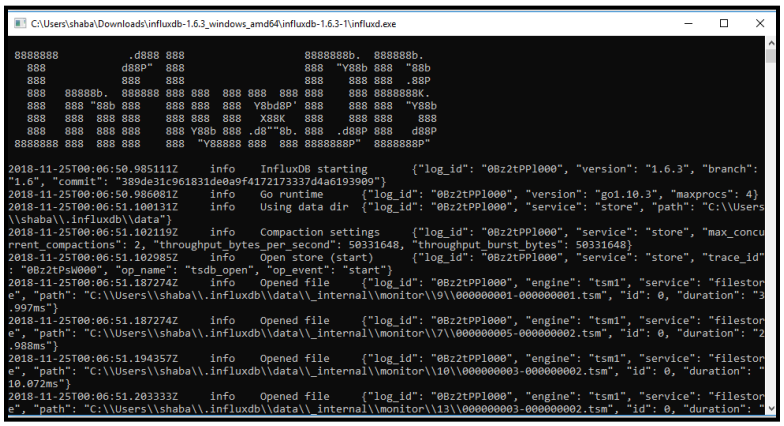
We will be needing InfluxDB and Chronograf for the demonstration. InfluxDB

4.2.1.1 Windows

- Download the latest version of InfluxDB, please visit https://portal.influxdata.com/downloads, register to download and choose InfluxDB v1.7.1. InfluxDB installation package consists of a dependent built-in set of binaries which are present in the download .zip folder.
- Check the **influxdb.conf** whether it has the following TCP port

```
bind-address = "127.0.0.1:8088"
```

- Unzip the downloaded folder and locate **influxdb** application and wait till InfluxDB server starts to launch influx DB



- Once the following informing is launched, double-click on influx application to launch the database service

.

4.2.1.2 Other OS

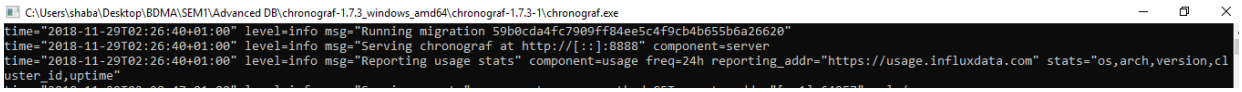
For other OS, please visit the original documentation of Influx Data here: https://docs.influxdata.com/influxdb/v1.7/introduction/installation/ and follow the instruction to install InfluxDB.

4.2.2 Chronograf

4.2.2.1 Windows

- Download Chronograf v1.7.3 from https://dl.influxdata.com/chronograf/releases/chronograf-1.7.3_windows_amd64.zip.

- Unzip the downloaded file and run the application **chronograf.**



4.2.2.2 Other OS

For other OS please follow the following link:

https://docs.influxdata.com/chronograf/v1.7/introduction/installation/#download-and-install

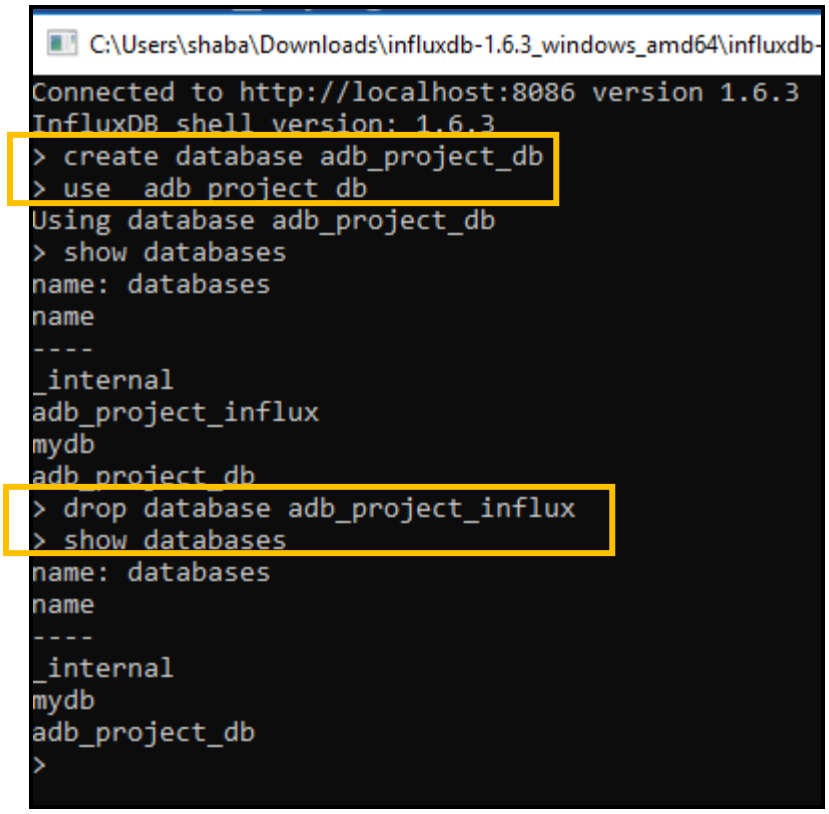
4.3 Schema design

Like all databases, the schema in InfluxDB also refers to the organization of data. The schema in InfluxDB refers to database, retention policies, series; measurements, tags, and fields. Although InfluxDB infers the data types and makes the schema for series on its own, we still have to define the name of database and the retention policies. When we talk about auto-inferring of schema we always talk about the Line Protocol mentioned in Section 3.7 above.

Let's create the schema required for data loading while learning some of the DDL for InfluxDB.

Database Management (CREATE, SHOW, USE & DROP)

```
--Database management queries

CREATE DATABASE adb_project_db
USE  adb_project_db
DROP DATABASE adb_project_influx
SHOW DATABASES
```



Retention Policy Management(CREATE, ALTER & DROP)

InfluxDB automatically creates a retention policy while creating a database. This helps to delete the expired and unnecessary data and prevents from database heavy loads. It is recommended to create the best retention policies. The following are the parameters to be considered when deciding the retention policy:

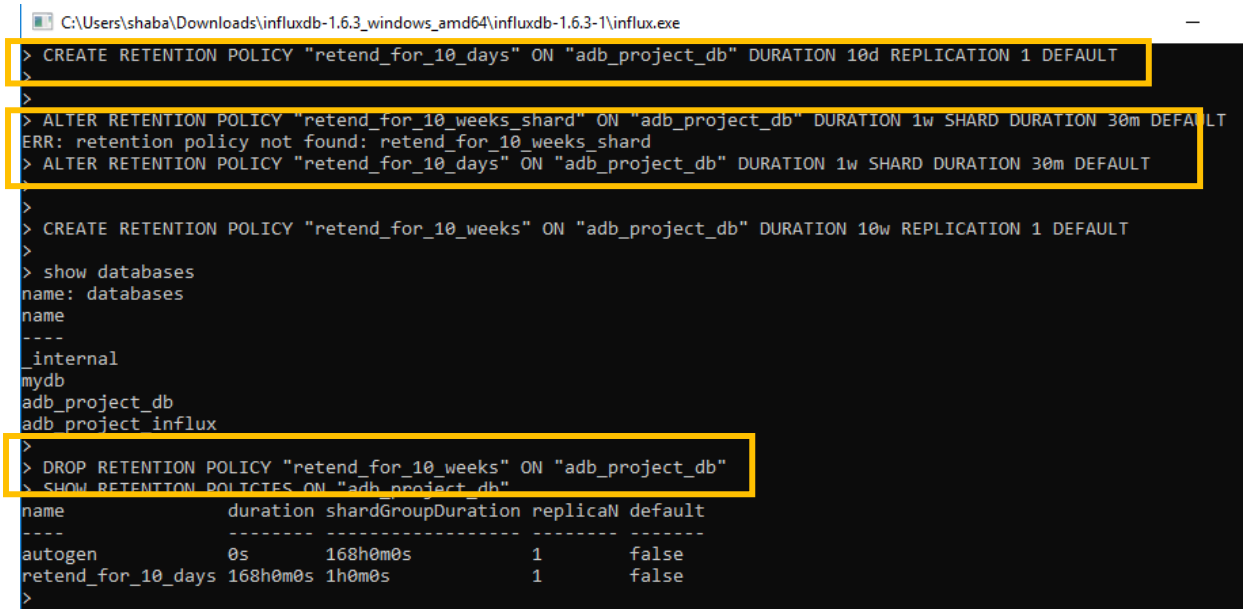
Duration decides the time frame to have the data stored.

Replication: decides the number of independent copies in a cluster for each stage.

Shard duration(optional): decides the time frame for the shard group.

Default: newly created retention policy will be set to default for the defined database. The default retention policy is infinity. However, the data is divided in 7 days shards.

```
--Retention policy queries

CREATE RETENTION POLICY "retend_for_10_days" ON "adb_project_db" DURATION 10d REPLICATION 1 DEFAULT

ALTER RETENTION POLICY "retend_for_10_days" ON "adb_project_db" DURATION 1w SHARD DURATION 30m DEFAULT

DROP RETENTION POLICY "retend_for_10_weeks" ON "adb_project_db"
```



4.4 Data loading

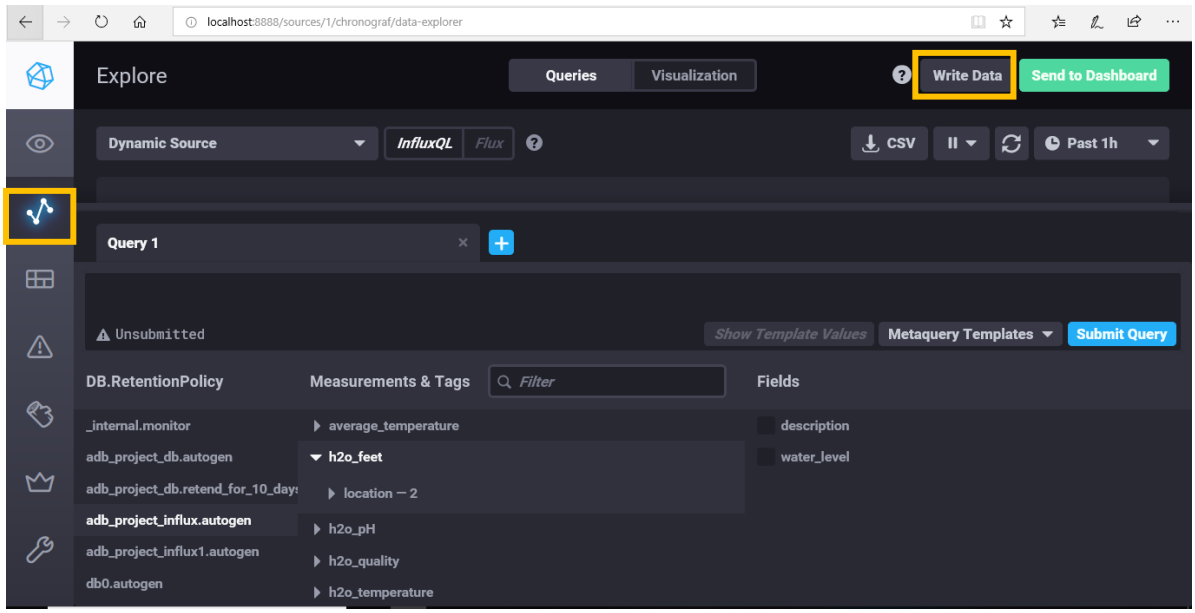
Data can be loaded in InfluxDB in following ways:

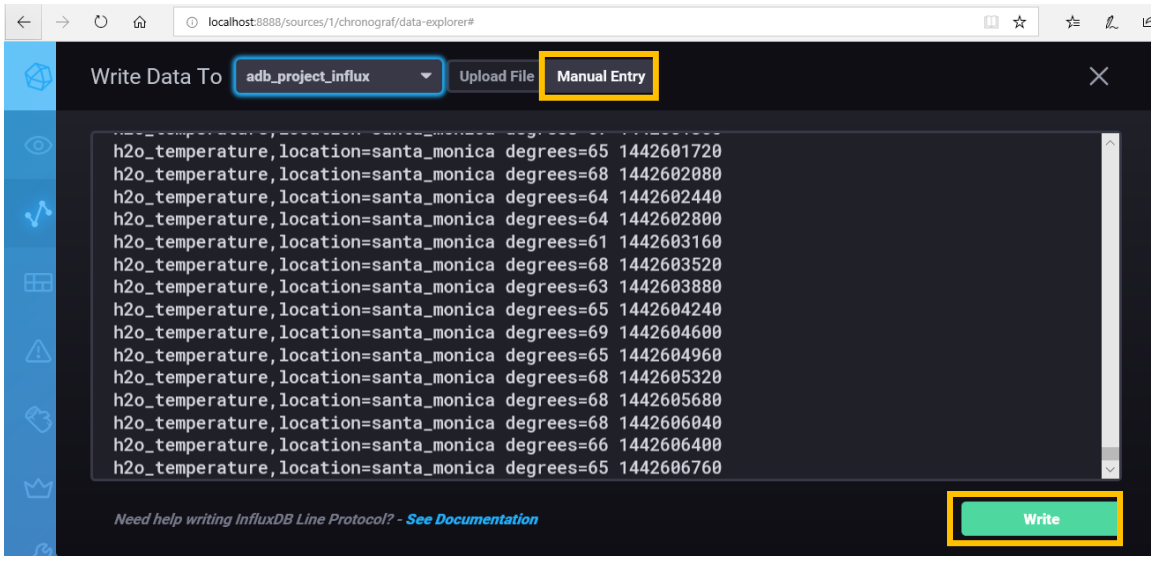
- CLI
- Chronograf
- Telegraf
- Kapacitor

We have used **Chronograf** just for the sake of showing its integration with InfluxDB.

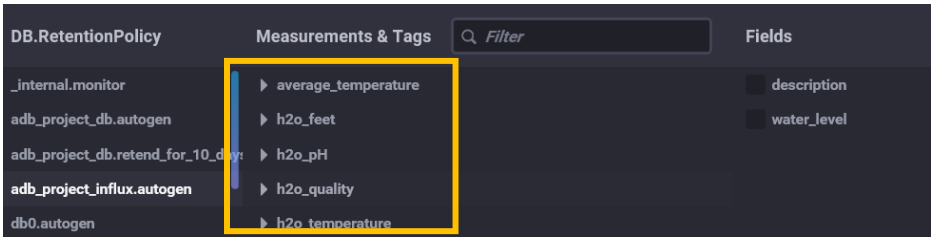
Please follow the following steps:

- Go to the URL to launch Chronograf, **http://localhost:8888/**
- Click on Data Explorer tab and choose Write Data to manually write the data under Manual Entry (copy and paste the time series data) and click on Write





- Data can be explored as represented below in Explore tab



4.5 Querying

4.5.1 InfluxQL

4.5.2 Simple queries

4.5.2.1 Querying all data

With the limit of 10, Select all fields and tags from the measurement h2o_feet

SELECT * FROM "h2o_feet" LIMIT 5

```
> SELECT * FROM "h2o_feet" LIMIT 10
name: h2o_feet
time                           description          location     water_level
----                           -----------          --------     -----------
1970-01-01T00:00:01.439856Z    below 3 feet         santa_monica 2.064
1970-01-01T00:00:01.439856Z    between 6 and 9 feet coyote_creek 8.12
1970-01-01T00:00:01.43985636Z  below 3 feet         santa_monica 2.116
1970-01-01T00:00:01.43985636Z  between 6 and 9 feet coyote_creek 8.005
1970-01-01T00:00:01.43985672Z  below 3 feet         santa_monica 2.028
```

4.5.2.2 Projection

Select specific tags and fields

SELECT "level description","location","water_level" FROM "h2o_feet" LIMIT 10

```
>  SELECT "level description","location","water_level" FROM "h2o_feet" LIMIT 10
name: h2o_feet
time                           level description location     water_level
----                           ----------------- --------     -----------
1970-01-01T00:00:01.439856Z                      coyote_creek 8.12
1970-01-01T00:00:01.439856Z                      santa_monica 2.064
1970-01-01T00:00:01.43985636Z                    coyote_creek 8.005
1970-01-01T00:00:01.43985636Z                    santa_monica 2.116
1970-01-01T00:00:01.43985672Z                    coyote_creek 7.887
```

4.5.2.3 Projection with identifiers

Select specific with identifiers

SELECT "level description"::field,"location"::tag,"water_level"::field FROM "h2o_feet" LIMIT 10

```
> SELECT "level description"::field,"location"::tag,"water_level"::field FROM "h2o_feet" LIMIT 10
name: h2o_feet
time                           level description location     water_level
----                           ----------------- --------     -----------
1970-01-01T00:00:01.439856Z                      coyote_creek 8.12
1970-01-01T00:00:01.439856Z                      santa_monica 2.064
1970-01-01T00:00:01.43985636Z                    coyote_creek 8.005
1970-01-01T00:00:01.43985636Z                    santa_monica 2.116
1970-01-01T00:00:01.43985672Z                    coyote_creek 7.887
```

4.5.2.4 Projection of fields only

Select only fields or tags

SELECT *::field FROM "h2o_quality" LIMIT 5

```
> SELECT *::field FROM "h2o_quality" LIMIT 10
name: h2o_quality
time                           index
----                           -----
1970-01-01T00:00:01.439856Z    41
1970-01-01T00:00:01.439856Z    99
1970-01-01T00:00:01.43985636Z  11
1970-01-01T00:00:01.43985636Z  56
1970-01-01T00:00:01.43985672Z  65
```

4.5.2.5 Basic Arithmetic

select with arithmetic

SELECT ("water_level" * 8) + 10 from "h2o_feet" limit 5

```
> SELECT ("water_level" * 8) + 10 from "h2o_feet" limit 5
name: h2o_feet
time                           water_level
----                           -----------
1970-01-01T00:00:01.439856Z    74.96
1970-01-01T00:00:01.439856Z    26.512
1970-01-01T00:00:01.43985636Z  74.04
1970-01-01T00:00:01.43985636Z  26.928
1970-01-01T00:00:01.43985672Z  73.096
>
```

SELECT ("water_level" * 8) + 10 as "wc", "water_level" from "h2o_feet" limit 5

```
> SELECT ("water_level" * 8) + 10 as "wc", "water_level" from "h2o_feet" limit 5


name: h2o_feet
time        wc      water_level
----        --      -----------
1439856000  74.96   8.12
1439856000  26.512  2.064
1439856360  74.04   8.005
1439856360  26.928  2.116
1439856720  73.096  7.887
```

4.5.2.6 Querying from more than one measurement

select from one or more measurements

SELECT * FROM "h2o_feet","h2o_pH","h2o_quality" limit 5

```
> SELECT * FROM "h2o_feet","h2o_pH","h2o_quality" limit 5
name: h2o_feet
time                           description          index location     pH randtag water_level
----                           -----------          ----- --------     -- ------- -----------
1970-01-01T00:00:01.439856Z    below 3 feet               santa_monica           2.064
1970-01-01T00:00:01.439856Z    between 6 and 9 feet       coyote_creek           8.12
1970-01-01T00:00:01.43985636Z  below 3 feet               santa_monica           2.116
1970-01-01T00:00:01.43985636Z  between 6 and 9 feet       coyote_creek           8.005
1970-01-01T00:00:01.43985672Z  below 3 feet               santa_monica           2.028

name: h2o_pH
time                           description index location     pH randtag water_level
----                           ----------- ----- --------     -- ------- -----------
1970-01-01T00:00:01.439856Z                      coyote_creek 7
1970-01-01T00:00:01.439856Z                      santa_monica 6
1970-01-01T00:00:01.43985636Z                    coyote_creek 8
1970-01-01T00:00:01.43985636Z                    santa_monica 6
1970-01-01T00:00:01.43985672Z                    coyote_creek 8

name: h2o_quality
time                           description index location     pH randtag water_level
----                           ----------- ----- --------     -- ------- -----------
1970-01-01T00:00:01.439856Z                41    coyote_creek    1
1970-01-01T00:00:01.439856Z                99    santa_monica    2
1970-01-01T00:00:01.43985636Z              11    coyote_creek    3
1970-01-01T00:00:01.43985636Z              56    santa_monica    2
1970-01-01T00:00:01.43985672Z              65    santa_monica    3
```

4.5.2.7 Querying from the default retention policy

select all from a measurement using a default retention policy.

```
SELECT * FROM "adb_project_influx".."h2o_feet" limit 5
```

```
> SELECT * FROM "adb_project_influx".."h2o_feet" limit 5
name: h2o_feet
time                           description          location     water_level
----                           -----------          --------     -----------
1970-01-01T00:00:01.439856Z    below 3 feet         santa_monica 2.064
1970-01-01T00:00:01.439856Z    between 6 and 9 feet coyote_creek 8.12
1970-01-01T00:00:01.43985636Z  below 3 feet         santa_monica 2.116
1970-01-01T00:00:01.43985636Z  between 6 and 9 feet coyote_creek 8.005
1970-01-01T00:00:01.43985672Z  below 3 feet         santa_monica 2.028
```

4.5.2.8 Selection based on field value

```
SELECT * FROM "h2o_feet" WHERE "water_level" > 5 LIMIT 5
```

```
> SELECT * FROM "h2o_feet" WHERE "water_level" > 5 LIMIT 5
name: h2o_feet
time                           description          location     water_level
----                           -----------          --------     -----------
1970-01-01T00:00:01.439856Z    between 6 and 9 feet coyote_creek 8.12
1970-01-01T00:00:01.43985636Z  between 6 and 9 feet coyote_creek 8.005
1970-01-01T00:00:01.43985672Z  between 6 and 9 feet coyote_creek 7.887
1970-01-01T00:00:01.43985708Z  between 6 and 9 feet coyote_creek 7.762
1970-01-01T00:00:01.43985744Z  between 6 and 9 feet coyote_creek 7.635
```

4.5.2.9 Selection based on tag values

SELECT * FROM "h2o_feet" WHERE "description" = 'below 3 feet' LIMIT 2

```
> SELECT * FROM "h2o_feet" WHERE "description" = 'below 3 feet' LIMIT 2
name: h2o_feet
time                           description  location     water_level
----                           -----------  --------     -----------
1970-01-01T00:00:01.439856Z    below 3 feet santa_monica 2.064
1970-01-01T00:00:01.43985636Z  below 3 feet santa_monica 2.116
```

4.5.2.10 Selection based on tag value and some arithmetic

SELECT * FROM "h2o_feet" WHERE "water_level" + 50 > 25.98 LIMIT 2

```
> SELECT * FROM "h2o_feet" WHERE "water_level" + 50 > 25.98 LIMIT 2
name: h2o_feet
time                           description          location     water_level
----                           -----------          --------     -----------
1970-01-01T00:00:01.439856Z    below 3 feet         santa_monica 2.064
1970-01-01T00:00:01.439856Z    between 6 and 9 feet coyote_creek 8.12
```

4.5.2.11 Selection based on fields and tags both

with tag and field key_value

SELECT "water_level" FROM "h2o_feet" WHERE "location" <> 'santa_monica' AND (water_level < -0.59 OR water_level > 9.95) LIMIT 2

```
> SELECT "water_level" FROM "h2o_feet" WHERE "location" <> 'santa_monica' AND (water_level < -0.59 OR water_level > 9.95
) LIMIT 2
name: h2o_feet
time                           water_level
----                           -----------
1970-01-01T00:00:01.44083268Z 9.957
1970-01-01T00:00:01.44083304Z 9.964
```

4.5.2.12 Selection based on fields and tags both

with group by

SELECT MEAN("index") FROM "h2o_quality" GROUP BY location,randtag LIMIT 2

```
> SELECT MEAN("index") FROM "h2o_quality" GROUP BY location,randtag LIMIT 2
name: h2o_quality
tags: location=coyote_creek, randtag=1
time                  mean
----                  ----
1970-01-01T00:00:00Z 50.69033760186263

name: h2o_quality
tags: location=coyote_creek, randtag=2
time                  mean
----                  ----
1970-01-01T00:00:00Z 49.661867544220485
```

SELECT MEAN("water_level"), MAX("water_level"), MIN("water_level") FROM "h2o_feet" GROUP BY location

```
> SELECT MEAN("water_level"), MAX("water_level"), MIN("water_level") FROM "h2o_feet" GROUP BY location
name: h2o_feet
tags: location=coyote_creek
time mean              max   min
---- ----              ---   ---
0    5.359342451341385 9.964 -0.61

name: h2o_feet
tags: location=santa_monica
time mean           max   min
---- ----           ---   ---
0    3.530863470081 7.205 -0.243
```

4.5.2.13 Selection based on fields and tags both

with time intervals

SELECT COUNT("water_level") FROM "h2o_feet" WHERE time >= 1439856000 AND time <= 1439870040 GROUP BY time(12m),"location"

```
> SELECT COUNT("water_level") FROM "h2o_feet" WHERE time >= 1439856000 AND time <= 1439870040 GROUP BY time(12m),"locati
on"
name: h2o_feet
tags: location=coyote_creek
time                 count
----                 -----
1970-01-01T00:00:00Z 40

name: h2o_feet
tags: location=santa_monica
time                 count
----                 -----
1970-01-01T00:00:00Z 40
>
```

4.5.2.14 Selection based on fields and tags both

SELECT COUNT("water_level") FROM "h2o_feet" WHERE "location"='coyote_creek' AND time >= 1439856000 AND time < 1439870040 GROUP BY time(12m,6m)

--the query performs for 12 minutes window and 6 minutes offset

```
> SELECT COUNT("water_level") FROM "h2o_feet" WHERE "location"='coyote_creek' AND time >= 1439856000 AND time < 14398700
40 GROUP BY time(12m,6m)
name: h2o_feet
time                 count
----                 -----
1969-12-31T23:54:00Z 39
>
```

4.5.2.15 Selection based on fields and tags both

within to clause

SELECT * INTO "h2o_feet_copy_1" FROM "h2o_feet" WHERE "location" = 'coyote_creek'

```
> SELECT * INTO "h2o_feet_copy_1" FROM "h2o_feet" WHERE "location" = 'coyote_creek'
name: result
time                 written
----                 -------
1970-01-01T00:00:00Z 7604
```

4.5.2.16 Selection based on fields and tags both

Sub-query

```
SELECT "all_the_means" FROM (SELECT MEAN("water_level") AS "all_the_means" FROM
"h2o_feet" WHERE time >= 1439856000 AND time <= 1439856000 GROUP BY time(6m) )
WHERE "all_the_means" > 1
```

```
>  SELECT "all_the_means" FROM (SELECT MEAN("water_level") AS "all_the_means" FROM "h2o_feet" WHERE time >= 1439856000 A
ND time <= 1439856000 GROUP BY time(6m) ) WHERE "all_the_means" > 1
name: h2o_feet
time                   all_the_means
----                   -------------
1970-01-01T00:00:00Z 5.092
```

4.5.3 Continuous query

InfluxQL can create a continuous query to run automatically with a defined period to get result data set. This is an added InfluxDB's advantage that can be created to store the real-time data.

The following are the main use cases for continuous query:

- Data retention
- Downsampling
- Calculating high-cost queries
- Instead of using HAVING clause
- Instead of complex NESTED queries

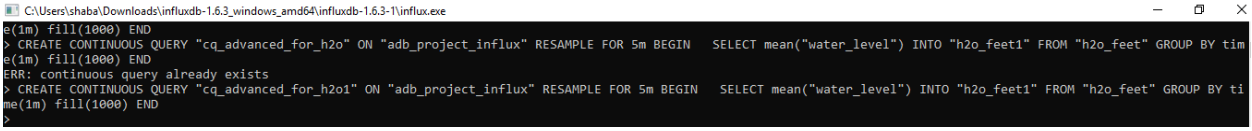
Continuous queries are used on real-time data and they use the present server's timestamp **GROUP BY time().** With this time() interval, the InfluxDB's time check to execute the range the query can cover. If the GROUP BY time() is set to one hour, the Continuous query starts at every one hour. This is how the Continuous query is used for scheduling the queries in real time.

The basic syntax of Continuous query is

```
CREATE CONTINUOUS QUERY <cq_name> ON <database_name> RESAMPLE EVERY <interval>
FOR <interval> BEGIN   <cq_query> END
```

```
C:\Users\shaba\Downloads\influxdb-1.6.3_windows_amd64\influxdb-1.6.3-1\influx.exe
> SHOW CONTINUOUS QUERIES
name: _internal
name query
---- -----

name: mydb
name query
---- -----

name: adb_project_db
name query
---- -----

name: adb_project_influx
name                 query
----                 -----
cq_advanced_for_h2o  CREATE CONTINUOUS QUERY cq_advanced_for_h2o ON adb_project_influx RESAMPLE F
o_feet1 FROM adb_project_influx.autogen.h2o_feet GROUP BY time(1m) fill(1000) END
cq_advanced_for_h2o1 CREATE CONTINUOUS QUERY cq_advanced_for_h2o1 ON adb_project_influx RESAMPLE
2o_feet1 FROM adb_project_influx.autogen.h2o_feet GROUP BY time(1m) fill(1000) END
```

```
CREATE CONTINUOUS QUERY "cq_advanced_for_h2o" ON "adb_project_influx" RESAMPLE FOR 2h BEGIN  SELECT mean("water_level") INTO "h2o_feet1" FROM "h2o_feet" GROUP BY time(1m) fill(1000) END
```



5 InfluxQL VS SQL

InfluxQL provides out of the box functionality for dealing with time series related queries. Such as moving averages, percentiles, derivatives, seasonal predictors etc.

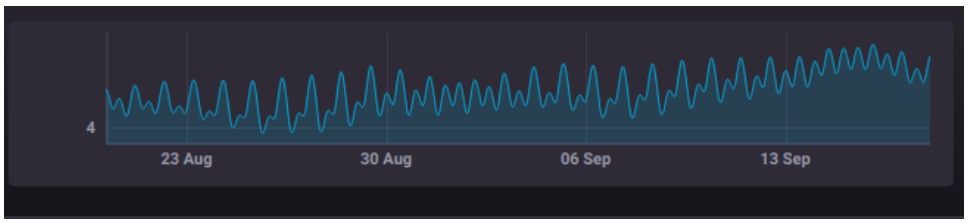
InfluxQL does not support the joins which are ubiquitous in an SQL world. But it provides ease of use in other ways. Let's consider an example of exponential moving averages. In SQL the code will look something like this:

```
select id, water_level, avg(water_level) over (partition by group_nr order by time_read) as rolling_avg  from (

select id, water_level, time_read, interval_group, id - row_number() over (partition by interval_group order by time_read) as group_nr

from ( select id, time_read,

'epoch'::timestamp + '900 seconds'::interval * (extract(epoch from time_read)::int4 / 900) as interval_group, water_level  from h2o_feet

) t1 ) t2

order by time_read;
```

But if we code this in influxQL. It will look something like the following:

```
SELECT exponential_moving_average("water_level", 1000) AS "mean_water_level" FROM "NOAA_water_database"."autogen"."h2o_feet"
```

And it looks like the following in chronograf:



Screenshot from Chronograf

6 Benchmarks

All of the following benchmarks are performed by Influx Data themselves [14] [15] [16] [17]. They chose a common DevOps monitoring use case. They deployed same components over 100 hosts and collected 100 values per host with a measurement interval of 10s. The data comprises of 24 hours of readings. So, the total values in the dataset are about 87M.

The test setup is two AWS c4.4xlarge instances running Ubuntu 16.04 LTS. One machine served as the data load and query client, and the other ran only the database server. These machines were connected through a very fast 1.29 Gbits/sec network.

The performance is measured across three different dimensions:

- Write Throughput
- Disk Space consumption
- Query throughput

We have collected the stats from the website of Influx Data and we have made three graphs for holistic picture and comparison.

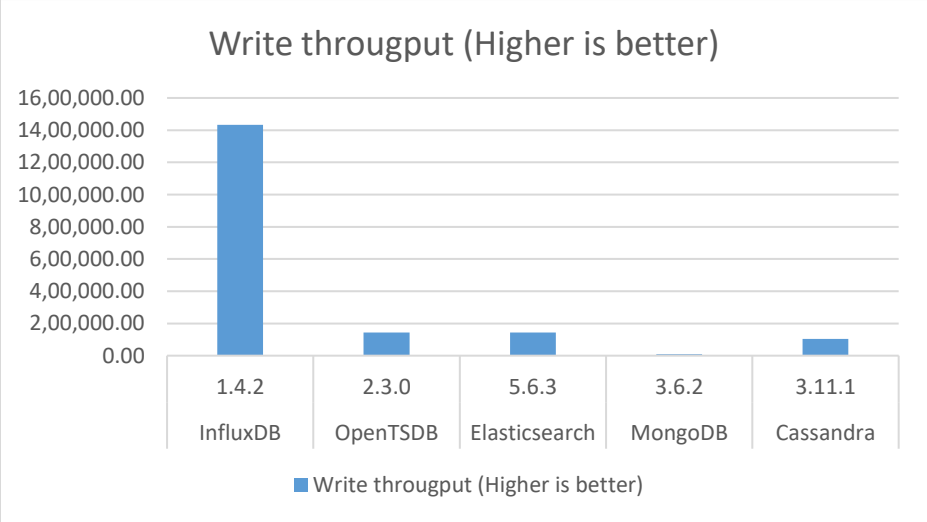


Figure 6-1 Write Throughput

For writing scenario, they used 4 concurrent writers. All the data was pre-generated and was available as a whole. As we can see in Figure 6-1, InfluxDB outperforms all the other databases.

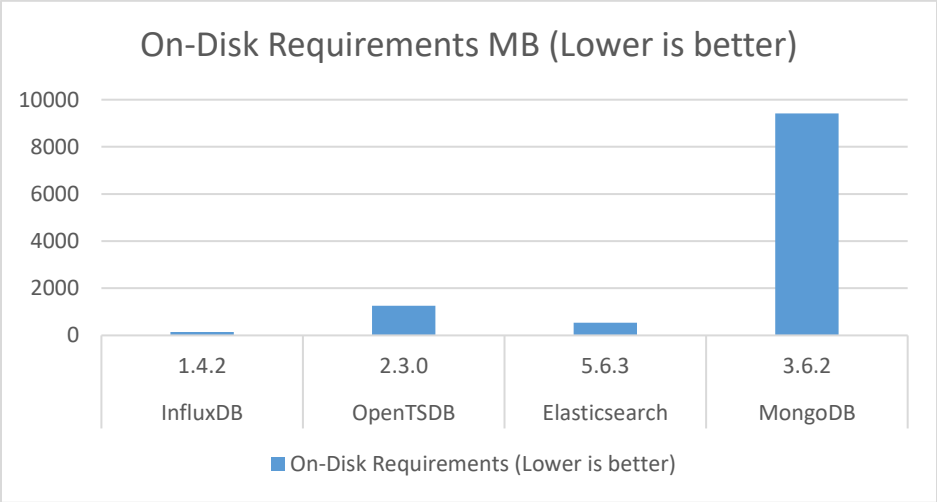


Figure 6-2 Disk Usage

For disk usage, we can see in Figure 6-2 that InfluxDB has far better compression rates than other databases in the benchmark. It took only 145 MB for InfluxDB to store 87M values. On the other hand, MongoDB required almost 9GB to store the same values.

For query throughput, we can see that InfluxDB outperforms OpenTSDB and MongoDB by 7 times and 3 times respectively. Elasticsearch performed 20% better than InfluxDB because Elasticsearch maintains a query cache and it is returning results from cache. But overall InfluxDB seems quite promising in this regard too.

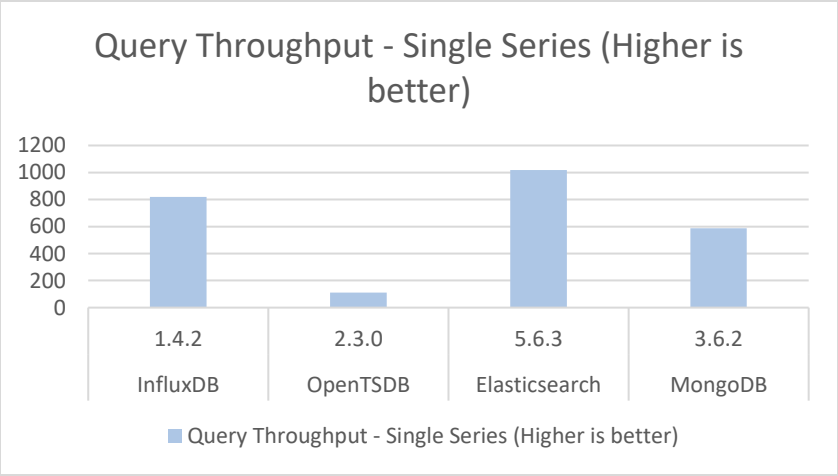


Figure 6-3 Query throughput

7 Conclusion

Standard SQL databases can be used for analyzing time series data, but they weren't solemnly created for time analysis. InfluxDB ensures that the time series database could store a huge volume of data and could help with real-time analysis in a timely manner. The present scenario in the industry believes that timing is everything and to process them to identify as a single point of time series data. So, to conclude, InfluxDB is SQL like a database but with time related pre-set primary key.

Additionally, InfluxDB is developed in such a way that the schema can be recognized, and their preferences can be changed during the course of time. That means the user does not have to pre-define the schemas. And also, there are no external dependencies required for developing Time series tables.

To Conclude, InfluxDB is an open source TSDB that is developed for faster read and write, extreme high availability of storage space and optimized querying for analyzing, monitoring and measuring the performance of time-related data.

8 References

[1] "algorithmia," [Online]. Available: https://blog.algorithmia.com/introduction-to-time-series/.

[2] Algorithmia. [Online]. Available: https://blog.algorithmia.com/introduction-to-time-series/.

[3] "Percona," February 2017. [Online]. Available: https://www.percona.com/blog/2017/02/10/percona-blog-poll-database-engine-using-store-time-series-data/.

[4] "DB-Engines," [Online]. Available: https://db-engines.com/en/ranking/time+series+dbms.

[5] "DB-Engines," [Online]. Available: https://db-engines.com/en/ranking_trend/time+series+dbms.

[6] R. Betts, "Data model and write path | InfluxDB Internals," 27 October 2017. [Online]. Available: https://www.influxdata.com/blog/influxdb-internals-101-part-one/. [Accessed 11 December 2018].

[7] Influx Data, "InfluxDB Line Protocol reference | InfluxDB Documentation," [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_reference/. [Accessed 12 December 2018].

[8] Influx Data, "Time Series Index (TSI) Overview | InfluxDB Documentation," Influx Data, 2018. [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/time-series-index/. [Accessed 12 December 2018].

[9] Influx Data, "In-memory indexing and the Time-Structured Merge Tree (TSM) | InfluxData Documentation," Influx Data, 2018. [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/storage_engine/. [Accessed 11 12 2018].

[10] Influx Data, "InfluxDB HTTP API reference | InfluxData Documentation," [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/tools/api/. [Accessed 12 December 2018].

[11] P. Dix, 07 November 2018. [Online]. Available: https://speakerdeck.com/pauldix/flux-and-influxdb-2-dot-0?slide=107. [Accessed 12 December 2018].

[12] , "Sample Data," , [Online]. Available: https://dzone.com/articles/introduction-to-time-series.

[13] , "What the heck is time-series data (and why do I need a time-series database)?," , [Online]. Available: https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563.

[14] , "Time Series Databases Explained | Database for Time Series Data | Basho," , [Online]. Available: http://basho.com/resources/time-series-databases/.

[15] , "Time Series | In Search of Agile Time Series Database," , [Online]. Available: https://taowen.gitbooks.io/tsdb/content/time-series/time-series.html.

[16] , "Station Selection - NOAA Tides Currents," , [Online]. Available: https://tidesandcurrents.noaa.gov/stations.html?type=Water+Levels.

[17] Influx Data, "Real Time Analytics | Processing, Analyzing & Acting on Time Series Data," Influx Data, [Online]. Available: https://www.influxdata.com/customers/real-time-analytics/.

[18] Influx Data, "Modern IoT Data Platform | Delivering Time to Value for IoT Projects," Influx Data, [Online]. Available: https://www.influxdata.com/customers/iot-data-platform/.

[19] , "Introduction to Time Series Database – at15 – Medium," , [Online]. Available: https://medium.com/@at15/introduction-to-time-series-database-df5e064e4ac3.

[20] , "Introduction to Time Series | Algorithmia Blog," , [Online]. Available: https://blog.algorithmia.com/introduction-to-time-series/.

[21] Influx Data, "Infrastructure and Application Monitoring | Real Value DevOps Monitoring," Influx Data, [Online]. Available: https://www.influxdata.com/customers/infrastructure-and-application-monitoring/.

[22] Influx Data, "InfluxQL Continuous Queries | InfluxData Documentation," Influx Data, [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/query_language/continuous_queries/#examples-of-advanced-syntax.

[23] , "InfluxDB System Properties," , [Online]. Available: https://db-engines.com/en/system/InfluxDB.

[24] , "InfluxDB Moves to Cloud-Native Architecture - The New Stack," , [Online]. Available: https://thenewstack.io/influxdb-moves-to-cloud-native-architecture/.

[25] Influx Data, " InfluxQL Reference | InfluxData Documentation," Influx Data, [Online]. Available: https://docs.influxdata.com/influxdb/v1.0/query_language/spec/.

[26] Influx Data, " Database Management | InfluxData Documentation," Influx Data, [Online]. Available: https://docs.influxdata.com/influxdb/v1.0/query_language/database_management/.

[27] Influx Data, " Data exploration using InfluxQL | InfluxData Documentation," Influx Data, [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/query_language/data_exploration/.

[28] "NOAA_data," [Online]. Available: https://s3.amazonaws.com/noaa.water-database/NOAA_data.txt. [Accessed 12 December 2018].

[29] Influx Data, "Sample data | InfluxData Documentation," Influx Data, [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/query_language/data_download/#data-sources-and-things-to-note. [Accessed 12 December 2018].

9 Appendix

9.1 Appendix A

This appendix gives us some tables regarding InfluxDB functions and mathematical operators.

Table 9-1 Syntactical notations from GO language

Notations	Syntax in InfluxQL
Alternative	Term { Term_arg }
Expr	Alternative { "|" Alternative_term }
Group	"(" Expr ")"
Option	"[" Expr "]"
Production	production "=" [Expr] "."
Repetition	"{" Expr "}"
Term	production | token ["..." token] | Group | Option | Repetition

Table 9-2 InfluxQL Identifiers

Type of identifier	Rules to use the identifier
identifier	unquoted_identifier | quoted_identifier
unquoted_identifier	(letter) { letter | digit }
quoted_identifier	`"` unicode_char { unicode_char } `"`

Table 9-3 InfluxQL keywords

ALL	ALTER	ANY	AS	ASC	BEGIN
BY	CREATE	CONTINUOUS	DATABASE(S)	DEFAULT	DELETE
DESC	DESTINATIONS	DIAGNOSTICS	DISTINCT	DROP	DURATION
END	EVERY	EXPLAIN	FIELD	FOR	FROM
GRANT(S)	GROUP(S)	IN	INF	INSERT	INTO
KEY(S)	KILL	LIMIT	SHOW	MEASUREMENT(S)	
NAME	OFFSET	ON	ORDER	PASSWORD	POLICY(-IES)
PRIVILEGES	QUERY(-IES)	READ	REPLICATION	RESAMPLE	RETENTION
REVOKE	SELECT	SERIES	SET	SHARD(S)	SLIMIT
SOFFSET	STATS	SUBSCRIPTION(S)		SUBSCRIPTIONS	TAG
TO	USER(S)	VALUES	WHERE	WITH	WRITE

Table 9-4 InfluxQL datatypes

Literal names	InfluxQL Datatypes	Literal format
Integers	int_lit	("1" ... "9") { digit }
Floats	float_lit	int_lit "." int_lit
Strings	string_lit	`'` { unicode_char } `'`
Durations(length of time)	duration_lit	int_lit duration_unit
	duration_unit	"u" | "µ" *--microseconds (1 millionth of a second)* | "ms"--milliseconds (1 thousandth of a second) | "s"*--second* | "m" *--minute* | "h"*--hour* | "d"*--day* | "w"*--week*
Dates & Times	time_lit	e.g."2006-01-02 15:04:05.999999" | "2006-01-02"
Booleans	bool_lit	TRUE | FALSE
Regular Expressions	regex_lit	"/" { unicode_char } "/"

Table 9-5 InfluxQL Functions

Functions		
Aggregations	**Selectors**	**Technical Analysis**
COUNT()	*BOTTOM()*	*CHANDE_MOMENTUM_OSCILLATOR()*
DISTINCT()	*FIRST()*	*EXPONENTIAL_MOVING_AVERAGE()*
INTEGRAL()	*LAST()*	*DOUBLE_EXPONENTIAL_MOVING_AVERAGE()*
MEAN()	*MAX()*	*KAUFMANS_EFFICIENCY_RATIO()*
MEDIAN()	*MIN()*	*KAUFMANS_ADAPTIVE_MOVING_AVERAGE()*
MODE()	*PERCENTILE()*	*TRIPLE_EXPONENTIAL_MOVING_AVERAGE()*
SPREAD()	*SAMPLE()*	*TRIPLE_EXPONENTIAL_DERIVATIVE()*
STDDEV()	*TOP()*	*RELATIVE_STRENGTH_INDEX()*
SUM()		
Predictors		
HOLT_WINTERS()		
Transformations		
ABS()	*COS()*	*HISTOGRAM()*
ACOS()	*CUMULATIVE_SUM()*	*LN()*
ASIN()	*DERIVATIVE()*	*LOG()*
ATAN()	*DIFFERENCE()*	*LOG2()*
ATAN2()	*ELAPSED()*	*LOG10()*
CEIL()	*EXP()*	*MOVING_AVERAGE()*
TAN()	*NON_NEGATIVE_DERIVATIVE()*	*NON_NEGATIVE_DIFFERENCE()*
POW()	*ROUND()*	*SQRT()*
SIN()	*FLOOR()*	

Table 9-6 InfluxQL Operators

Mathematical Operators
Addition
Subtraction
Multiplication
Division
Modulo
Bitwise AND
Bitwise OR
Bitwise Exclusive-OR