Architecture of Enterprise Applications 18 Timeseries Database

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - 时序数据库
  - InfluxDB Key Concepts
  - InfluxDB Internals
  - InfluxDB Getstarted

https://www.influxdata.com

- Objectives
  - 能够根据数据特性和数据访问模式，识别适合时序数据库存储的数据，设计并实现其在时序数据库中的存储和访问方案

What is a time series database?

- A time series database (TSDB) is a database optimized for time-stamped or time series data.
  - Time series data are simply measurements or events that are tracked, monitored, downsampled, and aggregated over time.
  - This could be server metrics, application performance monitoring, network data, sensor data, events, clicks, trades in a market, and many other types of analytics data.

- A time series database is built specifically for handling metrics and events or measurements that are time-stamped.
  - A TSDB is optimized for measuring change over time.
  - Properties that make time series data very different than other data workloads are data lifecycle management, summarization, and large range scans of many records.

Why is a time series database important now?

- Today, everything that can be a component is a component.
  - In addition, we are witnessing the instrumentation of every available surface in the material world — streets, cars, factories, power grids, ice caps, satellites, clothing, phones, microwaves, milk containers, planets, human bodies.
  - Everything has, or will have, a sensor.
  - So now, everything inside and outside the company is emitting a relentless stream of metrics and events or time series data.

- The underlying platforms need to evolve to support these new workloads
  - more data points, more data sources, more monitoring, more controls.
  - What we need is a performant, scalable, purpose-built time series database.

DB-Engines Ranking of Time Series DBMS

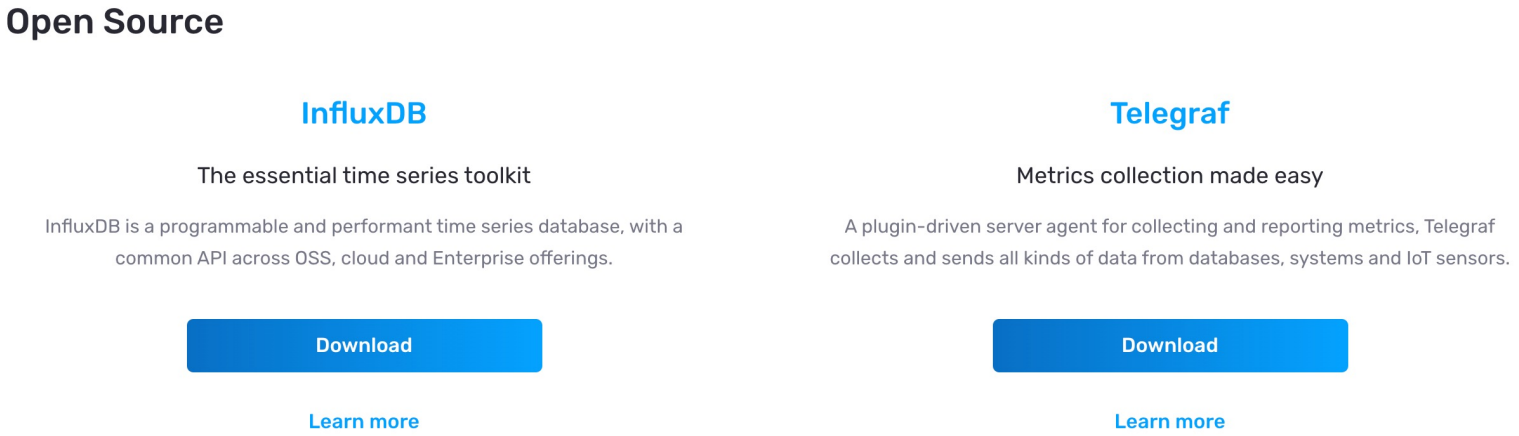
- **DB-Engines Ranking of Time Series DBMS**
  - https://db-engines.com/en/ranking/time+series+dbms

☐ include secondary database models — 38 systems in ranking, November 2021

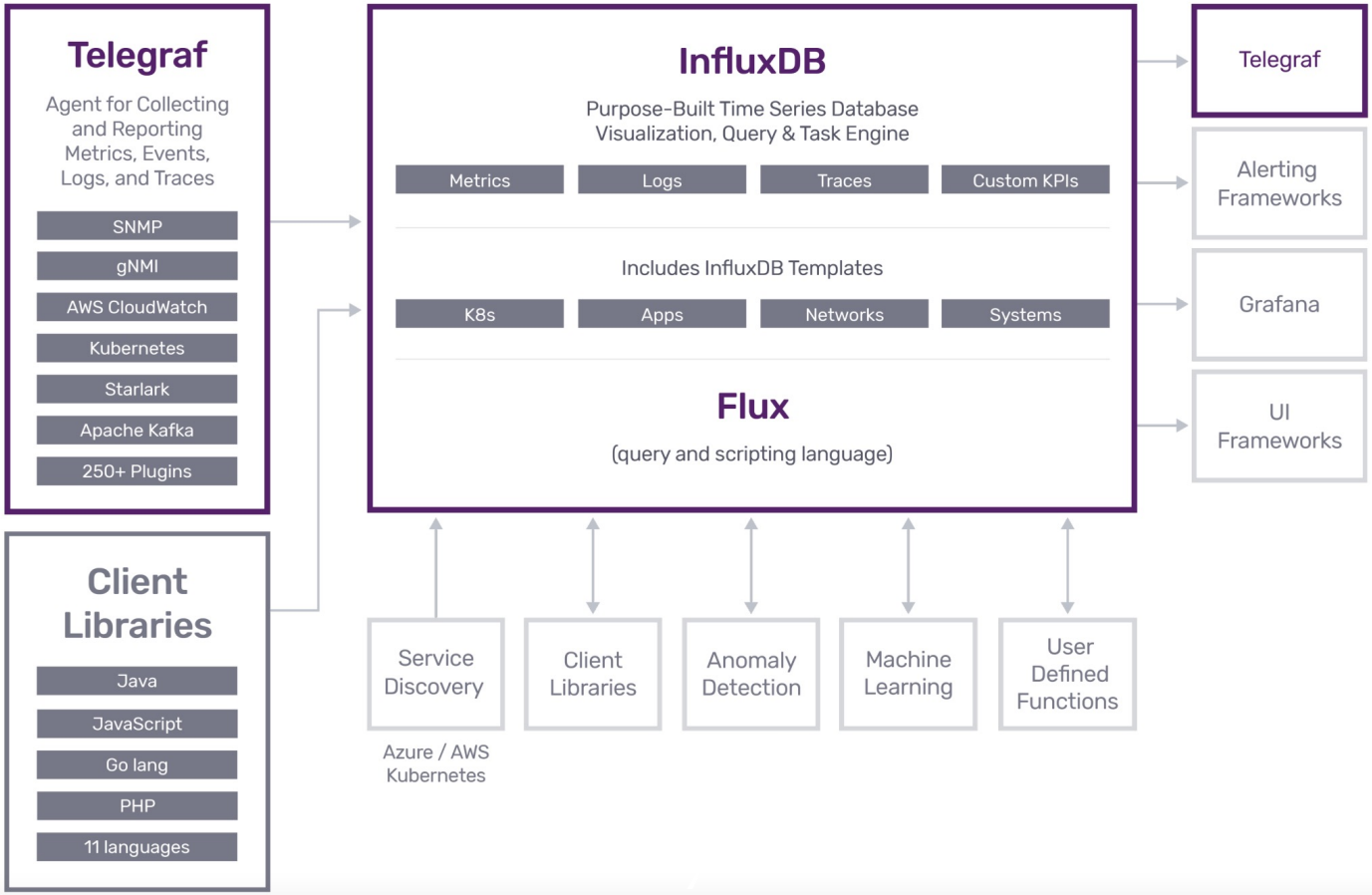
Rank			DBMS	Database Model	Score		
Nov 2021	Oct 2021	Nov 2020			Nov 2021	Oct 2021	Nov 2020
1.	1.	1.	InfluxDB	Time Series, Multi-model	28.54	+0.03	+3.59
2.	2.	2.	Kdb+	Time Series, Multi-model	7.95	-0.05	+0.41
3.	3.	3.	Prometheus	Time Series	6.52	-0.12	+0.83
4.	4.	4.	Graphite	Time Series	5.63	+0.13	+0.99
5.	5.	↑ 6.	TimescaleDB	Time Series, Multi-model	4.33	+0.37	+1.37
6.	6.	↑ 7.	Apache Druid	Multi-model	3.81	+0.36	+1.31
7.	7.	↓ 5.	RRDtool	Time Series	2.52	+0.24	-0.54
8.	8.	8.	OpenTSDB	Time Series	1.95	+0.09	-0.32
9.	9.	9.	Fauna	Multi-model	1.63	+0.06	-0.16
10.	10.	10.	GridDB	Time Series, Multi-model	1.28	-0.02	+0.46

InfluxDB

- InfluxDB is the time series platform
  - Build real-time applications for analytics, IoT and cloud-native services in less time with less code using InfluxDB.
  - At its heart is a database purpose-built to handle the epic volumes and countless sources of time-stamped data produced by sensors, applications and infrastructure.
  - If time is relevant to your data, you need a time series database.



InfluxDB & Telegraf

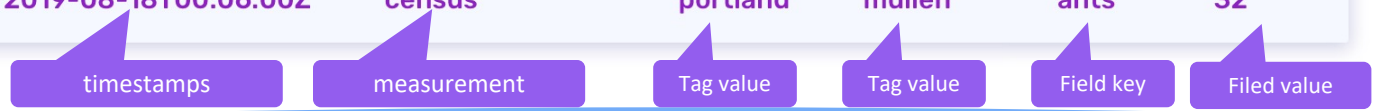


InfluxDB key concepts

- InfluxDB data elements
  - InfluxDB structures data using elements such as timestamps, field keys, field values, tags, etc.
  - The sample data below is used to illustrate data elements concepts

bucket: `my_bucket`

_time	_measurement	location	scientist	_field	_value
2019-08-18T00:00:00Z	census	klamath	anderson	bees	23
2019-08-18T00:00:00Z	census	portland	mullen	ants	30
2019-08-18T00:06:00Z	census	klamath	anderson	bees	28
2019-08-18T00:06:00Z	**census**	**portland**	**mullen**	**ants**	**32**



InfluxDB key concepts

- Timestamp
  - All data stored in InfluxDB has a _time column that stores timestamps.
  - On disk, timestamps are stored in epoch nanosecond format. InfluxDB formats timestamps show the date and time in RFC3339 UTC associated with data.
  - Timestamp precision is important when you write data.
- Measurement
  - The _measurement column shows the name of the measurement census.
  - A measurement acts as a container for tags, fields, and timestamps.
  - Use a measurement name that describes your data.
  - The name census tells us that the field values record the number of bees and ants.

InfluxDB key concepts

REliable, INtelligent & Scalable Systems

- Fields
  - A field includes a field key stored in the _field column and a field value stored in the _value column.
- Field key
  - A field key is a string that represents the name of the field.
- Field value
  - A field value represents the value of an associated field.
  - Field values can be strings, floats, integers, or booleans.
- Field set
  - A field set is a collection of field key-value pairs associated with a timestamp. The sample data includes the following field sets:

```
census bees=23i,ants=30i 1566086400000000000
census bees=28i,ants=32i 1566086760000000000
       -----------------
           Field set
```

InfluxDB key concepts

- Tags
  - Tags include tag keys and tag values that are stored as strings and metadata.
- Tag key
  - The tag keys in the sample data are location and scientist.
- Tag value
  - The tag key location has two tag values: klamath and portland. The tag key scientist also has two tag values: anderson and mullen.
- Tag set
  - The collection of tag key-value pairs make up a tag set. The sample data includes the following four tag sets:

```
location = klamath, scientist = anderson
location = portland, scientist = anderson
location = klamath, scientist = mullen
location = portland, scientist = mullen
```

InfluxDB key concepts

- **Fields aren't indexed:**
  - Fields are required in InfluxDB data and are not indexed.
  - Queries that filter field values must scan all field values to match query conditions.
  - As a result, queries on tags > are more performant than queries on fields.
  - **Store commonly queried metadata in tags.**

- **Tags are indexed:**
  - Tags are optional.
  - You don't need tags in your data structure, but it's typically a good idea to include tags.
  - Because tags are indexed, queries on tags are faster than queries on fields.
  - This makes tags ideal for storing commonly-queried metadata.

InfluxDB key concepts

- *Why your schema matters*
  - If most of your queries focus on values in the fields, for example, a query to find when 23 bees were counted:

```
from(bucket: "bucket-name")
  |> range(start: 2019-08-17T00:00:00Z, stop: 2019-08-19T00:00:00Z)
  |> filter(fn: (r) => r._field == "bees" and r._value == 23)
```

  - InfluxDB scans every field value in the dataset for bees before the query returns a response.

InfluxDB key concepts

- *Why your schema matters*
  - If our sample census data grew to millions of rows, to optimize your query, you could rearrange your schema so the fields (bees and ants) becomes tags and the tags (location and scientist) become fields:

_time	_measurement	bees	_field	_value
2019-08-18T00:00:00Z	census	23	location	klamath
2019-08-18T00:00:00Z	census	23	scientist	anderson
2019-08-18T00:06:00Z	census	**28**	**location**	**klamath**
2019-08-18T00:06:00Z	census	28	scientist	anderson

_time	_measurement	ants	_field	_value
2019-08-18T00:00:00Z	census	30	location	portland
2019-08-18T00:00:00Z	census	30	scientist	mullen
2019-08-18T00:06:00Z	census	**32**	**location**	**portland**
2019-08-18T00:06:00Z	census	32	scientist	mullen

InfluxDB key concepts

- Bucket schema
  - In InfluxDB Cloud, a bucket with the explicit schema-type requires an explicit schema for each measurement. Measurements contain tags, fields, and timestamps. An explicit schema constrains the shape of data that can be written to that measurement.
  - The following schema constrains census data:

name	type	data_type
time	timestamp	
location	tag	string
scientist	tag	string
ants	field	integer
bees	field	integer

InfluxDB key concepts

- Series
  - A **series key** is a collection of points that share a measurement, tag set, and field key. For example, the sample data includes two unique series keys:

_measurement	tag set	_field
census	location=klamath,scientist=anderson	bees
census	location=portland,scientist=mullen	ants

  - A **series** includes timestamps and field values for a given series key. From the sample data, here's a **series key**and the corresponding **series**:

```
# series key
census,location=klamath,scientist=anderson bees

# series
2019-08-18T00:00:00Z 23
2019-08-18T00:06:00Z 28
```

InfluxDB key concepts

- Point
  - A **point** includes the series key, a field value, and a timestamp. For example, a single point from the sample data looks like this:

```
2019-08-18T00:00:00Z census ants 30 portland mullen
```

- Bucket
  - All InfluxDB data is stored in a bucket. A **bucket** combines the concept of a database and a retention period (the duration of time that each data point persists). A bucket belongs to an organization.

- Organization
  - An InfluxDB **organization** is a workspace for a group of users. All dashboards, tasks, buckets, and users belong to an organization.

InfluxDB design principle

- InfluxDB implements optimal design principles for time series data.
  - Some of these design principles may have associated tradeoffs in performance.

- Time-ordered data
  - To improve performance, data is written in time-ascending order.
- Strict update and delete permissions
  - To increase query and write performance, InfluxDB tightly restricts **update** and **delete** permissions.
  - Time series data is predominantly new data that is never updated.
  - Deletes generally only affect data that isn't being written to, and contentious updates never occur.

InfluxDB design principle

- InfluxDB implements optimal design principles for time series data.
  - Some of these design principles may have associated tradeoffs in performance.

- Handle read and write queries first
  - InfluxDB prioritizes read and write requests over strong consistency.
  - InfluxDB returns results when a query is executed.
  - Any transactions that affect the queried data are processed subsequently to ensure that data is eventually consistent.
  - Therefore, if the ingest rate is high (multiple writes per ms), query results may not include the most recent data.
- Schemaless design
  - InfluxDB uses a schemaless design to better manage discontinuous data.
  - Time series data are often ephemeral, meaning the data appears for a few hours and then goes away. For example, a new host that gets started and reports for a while and then gets shut down.

InfluxDB design principle

- InfluxDB implements optimal design principles for time series data.
  - Some of these design principles may have associated tradeoffs in performance.

- Datasets over individual points
  - Because the data set is more important than an individual point, InfluxDB implements powerful tools to aggregate data and handle large data sets.
  - Points are differentiated by timestamp and series, so don't have IDs in the traditional sense.
- Duplicate data
  - To simplify conflict resolution and increase write performance, InfluxDB assumes data sent multiple times is duplicate data. Identical points aren't stored twice.
  - If a new field value is submitted for a point, InfluxDB updates the point with the most recent field value. In rare circumstances, data may be overwritten.

InfluxDB storage engine

- The InfluxDB storage engine ensures that:
  - Data is safely written to disk
  - Queried data is returned complete and correct
  - Data is accurate (first) and performant (second)

- The storage engine includes the following components:
  - Write Ahead Log (WAL)
  - Cache
  - Time-Structed Merge Tree (TSM)
  - Time Series Index (TSI)

InfluxDB storage engine

- Writing data from API to disk
  - The storage engine handles data from the point an API write request is received through writing data to the physical disk.
  - Data is written to InfluxDB using line protocol sent via HTTP POST request to the /writeendpoint.
  - Batches of points are sent to InfluxDB, compressed, and written to a WAL for immediate durability.
  - Points are also written to an in-memory cache and become immediately queryable.
  - The in-memory cache is periodically written to disk in the form of TSM files.
  - As TSM files accumulate, the storage engine combines and compacts accumulated them into higher level TSM files.
- While points can be sent individually, for efficiency, most applications send points in batches.
  - Points in a POST body can be from an arbitrary number of series, measurements, and tag sets.
  - Points in a batch do not have to be from the same measurement or tagset.

InfluxDB storage engine

- Write Ahead Log (WAL)
  - The **Write Ahead Log** (WAL) retains InfluxDB data when the storage engine restarts. The WAL ensures data is durable in case of an unexpected failure.

- When the storage engine receives a write request, the following steps occur:
  - The write request is appended to the end of the WAL file.
  - Data is written to disk using fsync().
  - The in-memory cache is updated.
  - When data is successfully written to disk, a response confirms the write request was successful.

InfluxDB storage engine

- Cache
  - The **cache** is an in-memory copy of data points currently stored in the WAL.

- The cache:
  - Organizes points by key (measurement, tag set, and unique field) Each field is stored in its own time-ordered range.
  - Stores uncompressed data.
  - Gets updates from the WAL each time the storage engine restarts. The cache is queried at runtime and merged with the data stored in TSM files.

InfluxDB storage engine

- Time-Structured Merge Tree (TSM)
  - To efficiently compact and store data, the storage engine groups field values by series key, and then orders those field values by time.
  - The storage engine uses a **Time-Structured Merge Tree** (TSM) data format.

- TSM files store compressed series data in a columnar format.
  - To improve efficiency, the storage engine only stores differences (or *deltas*) between values in a series.
  - Column-oriented storage lets the engine read by series key and omit extraneous data.

InfluxDB storage engine

- Time Series Index (TSI)
  - As data cardinality (the number of series) grows, queries read more series keys and become slower.
  - The **Time Series Index** ensures queries remain fast as data cardinality grows.
  - The TSI stores series keys grouped by measurement, tag, and field. This allows the database to answer two questions well:
    - What measurements, tags, fields exist? (This happens in meta queries.)
    - Given a measurement, tags, and fields, what series keys exist?

InfluxDB file system layout

- InfluxDB file structure
  - *Engine path*
    - Directory path to the storage engine, where InfluxDB stores time series data, includes the following directories:
    - **data**: stores Time-Structured Merge Tree (TSM) files
    - **wal**: stores Write Ahead Log (WAL) files.
  - *Bolt path*
    - File path to the Boltdb database, a file-based key-value store for non-time series data, such as InfluxDB users, dashboards, tasks, etc.
  - *Configs path*
    - File path to influx CLI connection configurations (configs).
  - *InfluxDB configuration files*
    - Some operating systems and package managers store a default InfluxDB (influxd) configuration file on disk.

InfluxDB file system layout

- File System Layout

macOS default paths

Path	Default
Engine path	`~/.influxdbv2/engine/`
Bolt path	`~/.influxdbv2/influxd.bolt`
SQLite path	`~/.influxdbv2/influxd.sqlite`
Configs path	`~/.influxdbv2/configs`

macOS file system overview

```
~/.influxdbv2/
├── engine/
│   ├── data/
│   │   └── TSM directories and files
│   └── wal/
│       └── WAL directories and files
├── configs
├── influxd.bolt
└── influxd.sqlite
```

InfluxDB shards and shard groups

- Shards
  - A shard contains encoded and compressed time series data for a given time range defined by the shard group duration.
  - All points in a series within the specified shard group duration are stored in the same shard.
  - A single shard contains multiple series, one or more TSM files on disk, and belongs to a shard group.

- Shard groups
  - A shard group belongs to an InfluxDB bucket and contains time series data for a specific time range defined by the shard group duration.
  - In **InfluxDB OSS**, a shard group typically contains only a single shard.
  - In an **InfluxDB Enterprise 1.x cluster**, shard groups contain multiple shards distributed across multiple data nodes.

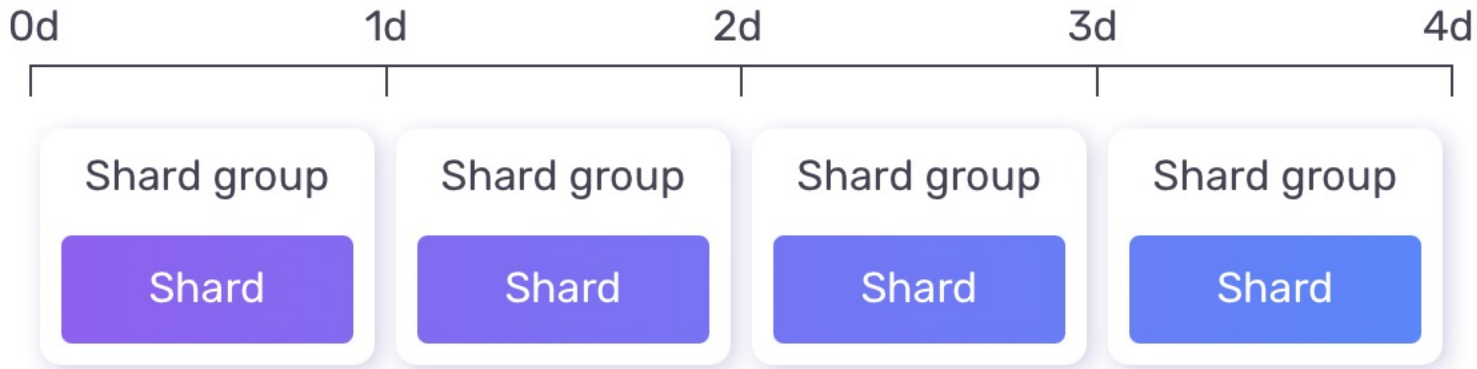
InfluxDB shards and shard groups

- Shard group duration
  - The **shard group duration** specifies the time range for each shard group and determines how often to create a new shard group. By default, InfluxDB sets the shard group duration according to the retention period of the bucket:

Bucket retention period	Default shard group duration
less than 2 days	1h
between 2 days and 6 months	1d
greater than 6 months	7d

InfluxDB shards and shard groups

- Shard group diagram
  - The following diagram represents a **bucket** with a **4d retention period** and a **1d shard group duration**:



Shard life-cycle

- Shard precreation
  - The InfluxDB **shard precreation service** pre-creates shards with future start and end times for each shard group based on the shard group duration.
  - The precreator service does not pre-create shards for past time ranges.
  - When backfilling historical data, InfluxDB creates shards for past time ranges as needed, resulting in temporarily lower write throughput.

- Shard writes
  - InfluxDB writes time series data to un-compacted or "hot" shards. When a shard is no longer actively written to, InfluxDB compacts shard data, resulting in a "cold" shard.
  - Typically, InfluxDB writes data to the most recent shard group, but when backfilling historical data, InfluxDB writes to older shards that must first be un-compacted. When the backfill is complete, InfluxDB re-compacts the older shards.

Shard life-cycle

- Shard compaction
  - InfluxDB compacts shards at regular intervals to compress time series data and optimize disk usage.
  - InfluxDB uses the following four compaction levels:
  - **Level 1 (L1):** InfluxDB flushes all newly written data held in an in-memory cache to disk.
  - **Level 2 (L2):** InfluxDB compacts up to eight L1-compacted files into one or more L2 files by combining multiple blocks containing the same series into fewer blocks in one or more new files.
  - **Level 3 (L3):** InfluxDB iterates over L2-compacted file blocks (over a certain size) and combines multiple blocks containing the same series into one block in a new file.
  - **Level 4 (L4): Full compaction**—InfluxDB iterates over L3-compacted file blocks and combines multiple blocks containing the same series into one block in a new file.

Shard life-cycle

- Shard deletion
  - The InfluxDB **retention enforcement service** routinely checks for shard groups older than their bucket's retention period.
  - Once the start time of a shard group is beyond the bucket's retention period, InfluxDB deletes the shard group and associated shards and TSM files.
  - In buckets with an infinite retention period, shards remain on disk indefinitely.

Run InfluxDB

- Install InfluxDB v2.1
  - Use Homebrew
    - We recommend using Homebrew to install InfluxDB v2.1 on macOS:
    - $ brew update
    - $ brew install influxdb

- Start InfluxDB
  - $influxd

Set up InfluxDB through the UI

1. With InfluxDB running, visit localhost:8086.
2. Click **Get Started**

Set up your initial user

1. Enter a **Username** for your initial user.
2. Enter a **Password** and **Confirm Password** for your user.
3. Enter your initial **Organization Name**.
4. Enter your initial **Bucket Name**.
5. Click **Continue**.

Run InfluxDB

- Configure your token as an environment variable
  - $ export INFLUX_TOKEN=YourAuthenticationToken

- Start the Telegraf service
  - $ telegraf -config http://localhost:8086/api/v2/telegrafs/0xoX00oOx0xoX00o

InfluxDB Data Explorer

REliable, INtelligent & Scalable Systems

Access InfluxDB via Java

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>com.influxdb</groupId>
    <artifactId>influxdb-client-java</artifactId>
    <version>2.0.0</version>
  </dependency>
</dependencies>
```

Access InfluxDB via Java

- Mem.java

```
@Measurement(name = "mem")
public class Mem {
    @Column(tag = true)
    String host;
    @Column
    Double used_percent;
    @Column(timestamp = true)
    Instant time;
}
```

Access InfluxDB via Java

- Mem.java

```
public class InfluxDB2Example {
    public static void main(final String[] args) {

        // You can generate a Token from the "Tokens Tab" in the UI
        String token = "p1kh6_ThGWtqQkd2K3iUIznAqIam6zMWZ8K2kVeI0wHSWB3t-
meWktjjHcD9N6RfUnFmGv_pdELUfkpuq6w2YQ==";
        String bucket = ”My Notebook";
        String org = "SJTU";

        InfluxDBClient client = InfluxDBClientFactory.create("http://localhost:8086", token.toCharArray());

        String data = "mem,host=host1 used_percent=23.43234543";
        try (WriteApi writeApi = client.getWriteApi()) {
            writeApi.writeRecord(bucket, org, WritePrecision.NS, data);
        }
```

Access InfluxDB via Java

- Mem.java

```
Point point = Point
        .measurement("mem")
        .addTag("host", "host1")
        .addField("used_percent", 23.43234543)
        .time(Instant.now(), WritePrecision.NS);

try (WriteApi writeApi = client.getWriteApi()) {
    writeApi.writePoint(bucket, org, point);
}

Mem mem = new Mem();
mem.host = "host1";
mem.used_percent = 23.43234543;
mem.time = Instant.now();

try (WriteApi writeApi = client.getWriteApi()) {
    writeApi.writeMeasurement(bucket, org, WritePrecision.NS, mem);
}
```

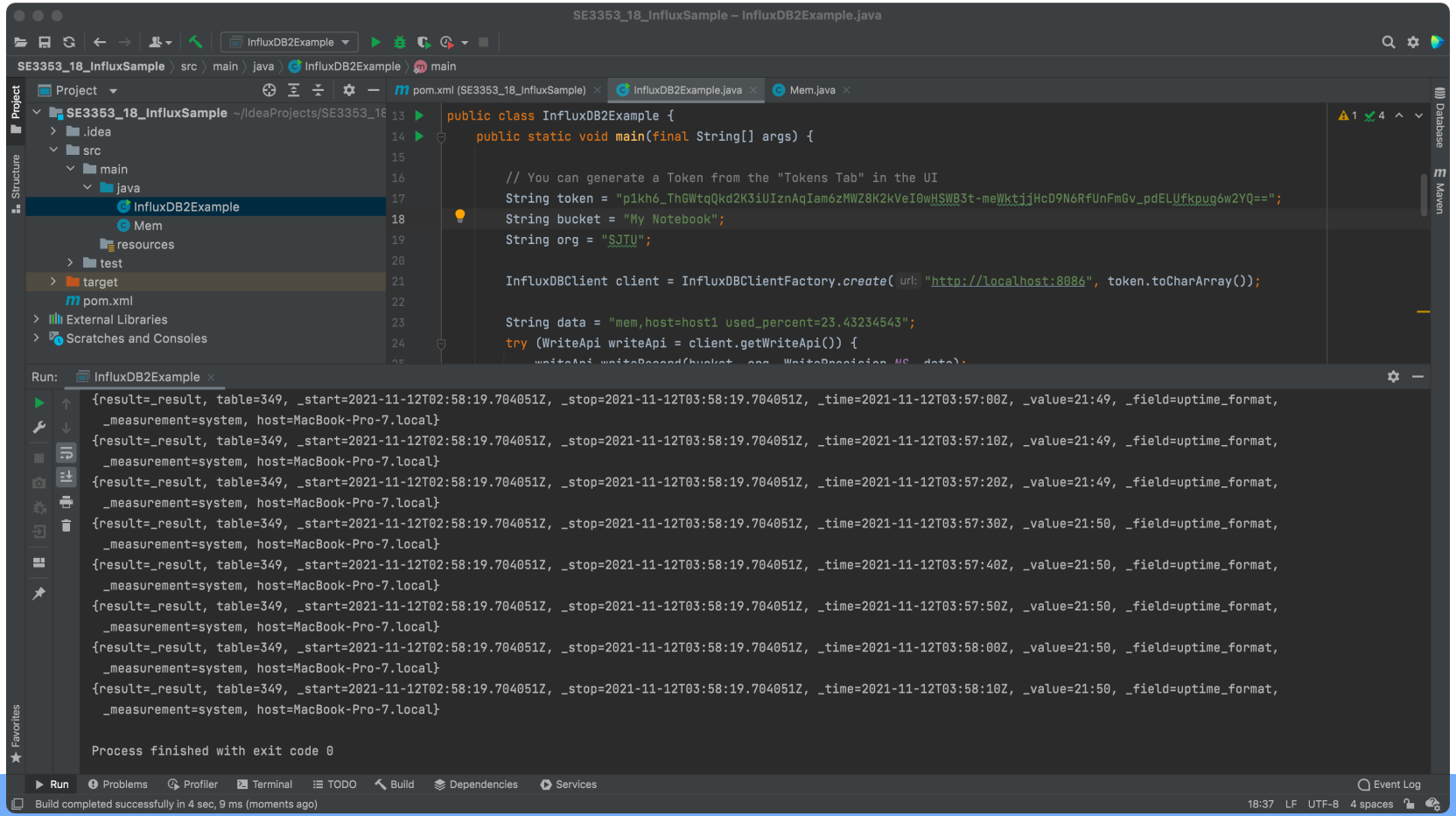
Access InfluxDB via Java

- Mem.java

```
        String query = String.format("from(bucket: \"%s\") |> range(start: -1h)", bucket);
        List<FluxTable> tables = client.getQueryApi().query(query, org);
        Iterator<FluxTable> it = tables.iterator();
        while (it.hasNext()) {
            FluxTable ft = it.next();
            List<FluxRecord> records = ft.getRecords();
            for ( FluxRecord record: records) {
                System.out.println(record.getValues());
            }
        }
    }
}
```

Access InfluxDB via Java

REIn
REliable, INtelligent & Scalable Systems



作业九

- 请你按照你的理解，用Word文档答复下列问题：
  1. 请阐述日志结构数据库适合什么样的应用场景？(1分)
  2. 请阐述日志结构数据库中的读放大和写放大分别是什么意思？(1分)
  3. 日志结构合并树中，WAL的作用是什么？ (1分)
  4. 请你在自己的机器上安装 InfluxDB，并像课程上所演示的一样监控你的笔记本电脑的状态，在Web界面的Explore中截图贴在Word文档中，并根据截图简要说明一下你的笔记本电脑的运行状态。 (2分)

  – 请提交包含上述问题答案的文档

- 评分标准：
  – 上述问题答案不唯一，只要你的说理合理即可视为正确。

References

- 时序数据库简介
  - https://zhuanlan.zhihu.com/p/115567891
- Time series database (TSDB) explained
  - https://www.influxdata.com/time-series-database/
- DB-Engines Ranking of Time Series DBMS
  - https://db-engines.com/en/ranking/time+series+dbms
- Why Time Series Matters for Metrics, Real-Time Analytics and Sensor Data
  - http://get.influxdata.com/rs/972-GDU-533/images/why%20time%20series.pdf

Thank You!