CRATE.IO

CrateDB for Time Series

How CrateDB compares to specialized time series data stores

July 2017

The Time Series Data Workload

IoT, digital business, cyber security, and other IT trends are increasing the amount of time series data we process. A time series database workload often looks like this:

1. Ingest many data points (millions) per second
2. Structured or unstructured data (JSON sensor readings, metrics, logs, GPS, etc.)
3. Indexed by timestamp
4. Queried in real-time

On top of those, the database might also have to meet high availability requirements, integrate with other systems, or support many concurrent users.

Time series is a challenging problem to solve. Traditional SQL databases are prohibitively difficult to scale to deliver this kind of mixed workload performance. It is also difficult to use them to process non-tabular data such as network logs and JSON data. For this reason, many developers turn towards NewSQL databases like CrateDB or specialized time series databases like InfluxDB.

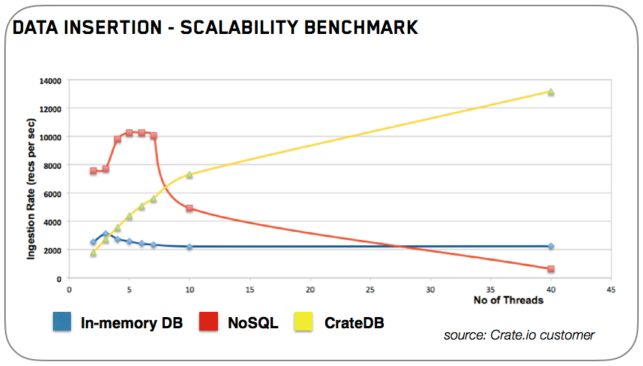
This paper explains how CrateDB supports time series data use cases and how it compares to specialized time series databases. The comparison points are summarized as follows:

CrateDB Advantage	CrateDB	InfluxDB
Versatility	Extensible, dynamic schemas allow you to easily add new data items and indexes. Full power of SQL, geospatial, and text search help to develop richer time series applications faster.	Single-purpose (time series) key-value store. Re-indexing & extending data are difficult, which slows development. No text search, so a different DB has to be used for log search/analytics.
No lock-in (ANSI SQL)	Standard ANSI SQL interface makes CrateDB easy to adopt and integrate.	Proprietary access language requires new learning, and makes integration harder and more expensive.
Scalability & Performance	CrateDB outperforms InfluxDB under concurrent user load and in queries that span data partitions. Invented to scale elastically to handle growing time series workloads, non-stop (24x7).	InfluxDB was developed as a single-server database (and performs very well if your time series workload can be managed on a single server). Distributed (scale-out) functionality was added recently, and is less mature.

CrateDB for Time Series: The ease of SQL with the power of NoSQL

CrateDB is a “NewSQL” database that overcomes time series limitations of traditional SQL databases. It provides a sophisticated distributed SQL engine built on a dynamic NoSQL foundation that is simple to scale.

High velocity data accumulation - the CrateDB architecture scales out horizontally on clusters of inexpensive servers to handle millions of data inserts per second.



Real-time time series query performance - Distributed processing, data partitioning and in-memory columnar indexes deliver time series query responses in milliseconds, even with many clients querying and inserting data concurrently.

Unmatched data model and query versatility - CrateDB employs NoSQL (JSON) storage and indexing technology beneath its SQL engine, which gives you the ability to evolve your data model without recoding or reloading. It also supports full-text search, geospatial, IP address, and user-defined functions for advanced, in-database analytics.

SQL for easy adoption and integration, and no lock-in - CrateDB supports the ANSI SQL standard so your team can put it to work with no retraining. It integrates with SQL code and tools you already use via JDBC, ODBC, Postgres wire protocol, or HTTP (REST). Plus, there are custom integrations for CrateDB that connect it to Grafana, Apache Kafka, and other popular time series ecosystem components.

CrateDB vs.InfluxDB

Time series application developers often compare CrateDB against single-purpose time series data stores like InfluxDB. The following functional and performance comparisons are meant to help you decide whether CrateDB is right for your time series project:

Functional comparison:

CrateDB and InfluxDB have many important functional similarities:

- Both excel at ingesting streams of time series data
- Both can partition, query, and aggregate timestamp-indexed data in real time
- Both handle structured and unstructured data
- Both are open source
- Both support data compression, and
- Both support policy-driven data aging automation (Influx is built-in; CrateDB via DevOps)

Unlike InfluxDB, CrateDB includes:

- ANSI SQL-standard access interface
- Dynamic schemas which speed up development as project requirements evolve.
  - In CrateDB, there is no limit to the number of columns in a table, and you can add columns and indexes on the fly without having to dump and reload data.
  - In InfluxDB, the number of columns per “series” is limited to 200, and the columns by which a series can be indexed and queried (“tag columns”) are fixed at table creation time. If you want to add a new tag column, you must redefine and reload the series with the new tag column.
- Broader query options, to analyze time series data in a wider variety of ways
  - To build richer time series applications, CrateDB supports geospatial queries, full-text search, table joins, sub-queries, and user-defined functions (e.g., for AI, anomaly detection, etc.), which saves you time and work.
  - For example, location and motion often provide context for the time series data (as in a vehicle tracking system). CrateDB has the built-in ability to map the location, speed, direction, and proximity of sensor readings.

Performance Comparison

To illustrate some of the performance differences between CrateDB and InfluxDB, we generated a year of sensor readings and stored them in CrateDB and also in InfluxDB. The sensor readings simulated a typical workload for a SaaS IoT platform...receiving readings from 3 different sensor types, from a variety of SaaS customers (“tenants”). Records contained:

- uuid - a universally unique identifier for the sensor reading
- ts - a timestamp of the sensor reading
- tenant_id - the id of the tenant from which the sensor is reading
- sensor_id - the id of the sensor
- sensor_type - the type of the sensor, represented in a 'root:type:subtype' tree-like topology
- v1 - a sensor reading in the form of an integer
- v2 - as above
- v3 - a sensor reading in the form of a floating point number
- v4 - as above
- v5 - a sensor reading in the form of a boolean

314 million sensor readings were stored in CrateDB and InfluxDB, respectively, each running on similar hardware (clusters with 24 CPU cores and 48GB of RAM), and with the following queries executed and benchmarked by a JMeter harness:

Query 1 - Data from a single tenant, aggregated over a single week
InfluxDB:

```
SELECT MIN(v1), MAX(v1), MEAN(v1), SUM(v1)
FROM ${measurement}
WHERE tenant_id='${tenant_id}' AND time >= '${week}' AND time <= '${week_next}'
```

CrateDB:

```
SELECT min(v1) as v1_min, max(v1) as v1_max, avg(v1) as v1_avg, sum(v1) as v1_sum
FROM t1
WHERE tenant_id = ? AND ts BETWEEN ? AND ?;
```

Query 2 - Data aggregated across multiple weeks, grouped & ordered by tenant

InfluxDB:

```
SELECT COUNT(uuid), MIN(*), MAX(*), MEAN(*)
FROM ${measurement}
WHERE time >= '${week}' AND time < '${week_next}'
GROUP BY tenant_id
```

CrateDB:

```
SELECT count(*) AS num_docs, tenant_id,
  min(v1) AS v1_min, max(v1) AS v1_max, avg(v1) AS v1_avg, sum(v1) AS v1_sum,
  min(v2) AS v2_min, max(v2) AS v2_max, avg(v2) AS v2_avg, sum(v2) AS v2_sum,
  min(v3) AS v3_min, max(v3) AS v3_max, avg(v3) AS v3_avg, sum(v3) AS v3_sum,
  min(v4) AS v4_min, max(v4) AS v4_max, avg(v4) AS v4_avg, sum(v4) AS v4_sum
FROM t1
WHERE ts BETWEEN ? AND ?
GROUP BY tenant_id
ORDER BY tenant_id;
```

Query 3 - Group by with string search

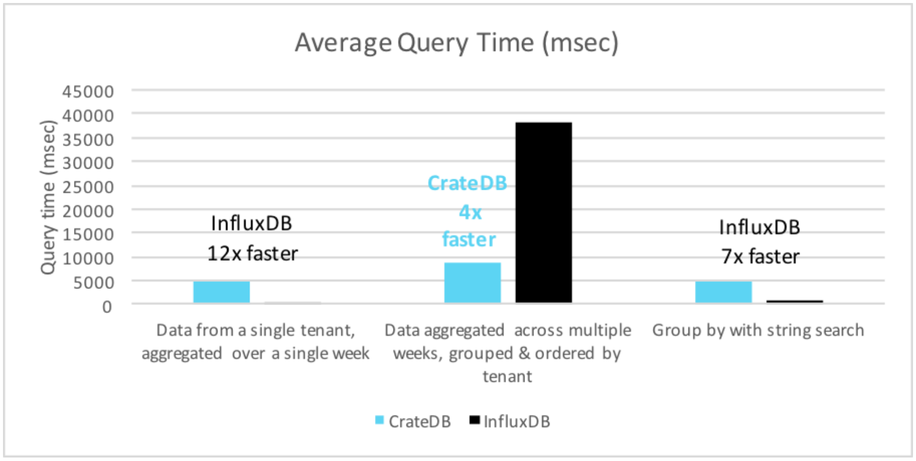
InfluxDB:

```
SELECT COUNT(v5)
FROM ${measurement}
WHERE (sensor_type = '${type[0]}' OR sensor_type = '${type[1]}' OR sensor_type = '${type[2]}')
AND tenant_id = '${tenant_id}'
GROUP BY sensor_type
```

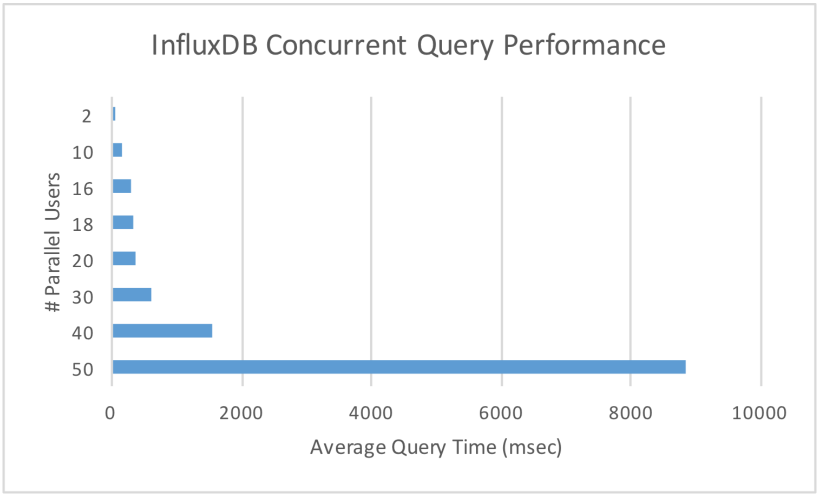
CrateDB:

```
SELECT sensor_type, COUNT(*) as sensor_count
FROM t1
WHERE taxonomy = ? AND tenant_id = ?
GROUP BY sensor_type;
```

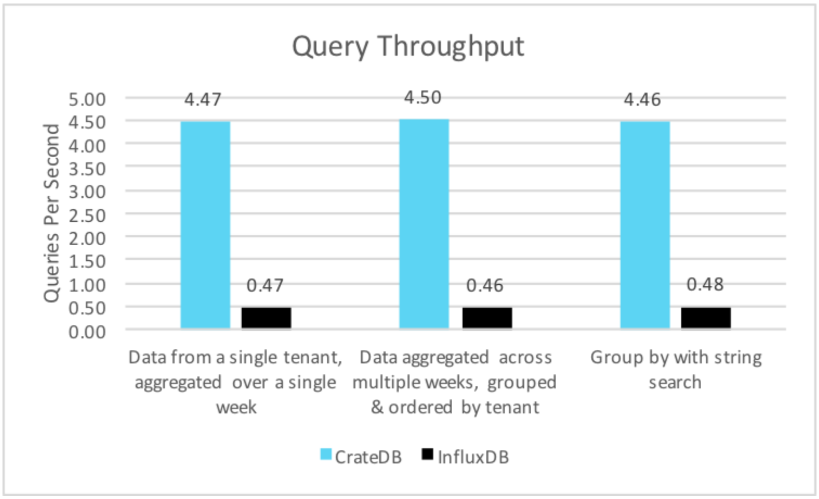
Queries that accessed data from within a single partition (queries 1 and 3, within the same week) worked fast in both databases, but faster in InfluxDB. A query that spanned multiple partitions (query 2), ran 4 times faster in CrateDB:



If you take those same queries, and run them under load, simulating multiple users querying the database at the same time, the performance of InfluxDB begins to degrade. The following graph shows the average query performance gradually increasing as users were added to the workload in InfluxDB. It finally began hitting a wall at the 40 concurrent user mark:



Reducing the number of concurrent connections to 20 for InfluxDB and CrateDB, a query throughput comparison shows CrateDB processing ~10x more queries per second than InfluxDB:



In conclusion, CrateDB provided better performance when answering queries for bigger user bases or when querying bigger data volumes (across multiple time partitions).

In Summary

If you compare time series databases for your project, we hope that this research between CrateDB and InfluxDB will help you. Take into consideration the following CrateDB advantages:

- Better performance
  - Querying larger time series (across time partitions)
  - More concurrent queries per second (in cases with many database users/clients)
- Functionality
  - Data model flexibility and the ease with which schema can be extended and new indexes added
  - Query flexibility - being able to perform search, SQL joins, user-defined queries, sub-selects, anomaly detection, geo-spatial, joins and other analyses