Time-series, PostgreSQL, and You

Mathis Van Eetvelde

About me

Mathis Van Eetvelde

Developer Advocate
@Timescale

@meetvelde

@mathisve

Mathis Van Eetvelde

mathisvaneetvelde.com

Agenda

Topics we will cover today

01

What is time-series data?

Time-series DDL

Financial time-series data

```
CREATE TABLE stocks (
   time TIMESTAMPTZ NOT NULL,
   price INT NULL,
   symbol TEXT NULL
);
```

IoT time-series data

```
CREATE TABLE measurements (
   time TIMESTAMPTZ NOT NULL,
   deviceID INT NULL,
   sensor_value REAL NULL,
   location geometry NULL
);
```

...

Kinds of time-series data

- Financial data
  - Stocks
  - OLTP (Online Transaction Processing)
- IoT data
  - Smartphones
  - Smart fridges, TVs, etc.
  - Industrial sensors
- Observability data
  - Traces, spans
  - Logs
  - Metrics

Time-series data means scale

NYSE transactions per **day**

4.2B

Consolidated Tape A
trading volume
https://www.nyse.com/markets/us-equity-volumes

NYSE transactions per **second**

162K

$$\frac{\text{4.2B Transactions}}{\text{23400 Trading Seconds}}$$

NYSE transactions in **5 years**

5.4T

CFTC requires that financial records be kept for 5 years and “readily available” for 2 years

02

Time-series data on PostgreSQL

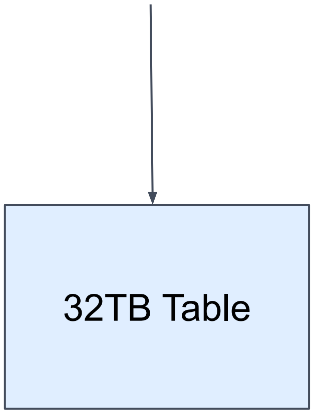
Why PostgreSQL?

- Flourishing ecosystem
- Tried and tested
- Not a DSL (InfluxQL, FQL, etc.)
- You don’t have to write your own JOINs on the application level
- You already know PostgreSQL

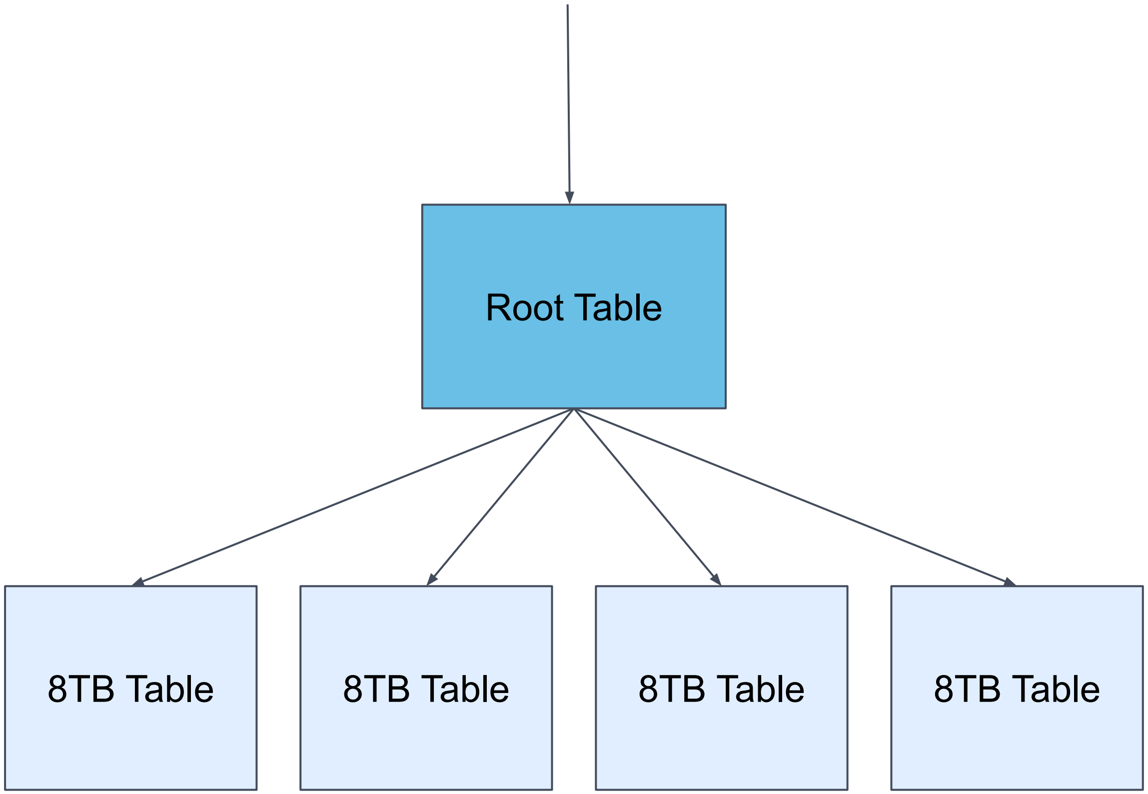
Hard limitations of PostgreSQL

- 4.2 Billion databases
- 1.4 Billion tables per database
- 32 TB per table (8192 bytes block size)
- 4.2 Billion pages per table

Partitioning



Partitioning



Partitioning

- Smaller tables
- Smaller indexes
- Partition Pruning

Partitioning

- Inheritance based partitioning (pre PG 10)
    - Harder
    - Versatile
- Declarative partitioning since (PG 10)
    - Easier

Inheritance based partitioning

Create the root table

```
CREATE TABLE measurement (
   time TIMESTAMPTZ NOT NULL,
   sensorID INT NULL,
   sensor_value REAL NULL
);
```

Create inheritance tables

```
CREATE TABLE measurement_y2022m01 () INHERITS (measurement);
CREATE TABLE measurement_y2022m02 () INHERITS (measurement);
CREATE TABLE measurement_y2022m03 () INHERITS (measurement);
```

Create indexes

```
CREATE INDEX ON measurement_y2022m01 (time);
CREATE INDEX ON measurement_y2022m02 (time);
CREATE INDEX ON measurement_y2022m03 (time);
```

Trigger function

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
   IF ( NEW.time >= DATE '2022-01-01' AND
        NEW.time < DATE '2022-02-01' ) THEN
       INSERT INTO measurement_y2022m01 VALUES (NEW.*);
                   …
   ELSIF ( NEW.time >= DATE '2022-03-01' AND
           NEW.time < DATE '2022-04-01' ) THEN
       INSERT INTO measurement_y2022m03 VALUES (NEW.*);
   ELSE
       RAISE EXCEPTION 'Date out of range.';
   END IF;
   RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER insert_measurement_trigger

   BEFORE INSERT ON measurement

   FOR EACH ROW EXECUTE FUNCTION measurement_insert_trigger();
```

Inserts

```
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-3', 2, 2.9);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-6', 1, 3.5);
…
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-27', 3, 2.5);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-29', 2, 5.3);
```

```
SELECT tableoid, * from measurement;

 tableoid |          time          | sensorid |  sensor_value
----------+------------------------+----------+----------------
    19103 | 2022-01-03 00:00:00+00 |    2     |      2.9
    19103 | 2022-01-06 00:00:00+00 |    1     |      3.5
     …    |           …            |    …     |       …
    19109 | 2022-03-27 00:00:00+00 |    3     |      2.5
    19109 | 2022-03-29 00:00:00+00 |    2     |      5.3
```

Partition Pruning - on

```
SET enable_partition_pruning = on; -- default
```

```
EXPLAIN ANALYZE  SELECT * FROM measurement WHERE time > '2022-03-01';

Index Scan using measurementy2022m03_time_idx on measurementy2022m03 measurement
(cost=0.15..23.05 rows=617 width=16) (actual time=0.005..0.006 rows=5 loops=1)
  Index Cond: ("time" > '2022-03-01 00:00:00+00'::timestamp with time zone)
Planning Time: 0.090 ms
Execution Time: 0.019 ms
(4 rows)
```

Partition Pruning - off

```
SET enable_partition_pruning = off;
```

```
EXPLAIN ANALYZE SELECT * FROM measurement WHERE time > '2022-03-01';

Append  (cost=0.15..78.41 rows=1851 width=16) (actual time=0.031..0.033 rows=5 loops=1)
  -> Index Scan using measurementy2022m01_time_idx on measurementy2022m01 measurement_1
        Index Cond: ("time" > '2022-03-01 00:00:00+00'::timestamp with time zone)
  -> Index Scan using measurementy2022m02_time_idx on measurementy2022m02 measurement_2
        Index Cond: ("time" > '2022-03-01 00:00:00+00'::timestamp with time zone)
  -> Index Scan using measurementy2022m03_time_idx on measurementy2022m03 measurement_3
        Index Cond: ("time" > '2022-03-01 00:00:00+00'::timestamp with time zone)
Planning Time: 0.123 ms
Execution Time: 0.053 ms
(9 rows)
```

Declarative Partitioning

Create the root table

```
CREATE TABLE measurement (
   time TIMESTAMPTZ NOT NULL,
   sensorID INT NULL,
   sensor_value REAL NULL
) PARTITION BY RANGE (time);
```

Create partitions

```
CREATE TABLE measurementy2022m01 PARTITION OF measurement
   FOR VALUES FROM ('2022-01-01') TO ('2022-02-01');
CREATE TABLE measurementy2022m02 PARTITION OF measurement
   FOR VALUES FROM ('2022-02-01') TO ('2022-03-01');
CREATE TABLE measurementy2022m03 PARTITION OF measurement
   FOR VALUES FROM ('2022-03-01') TO ('2022-04-01');
```

Inserts

```
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-3', 2, 2.9);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-6', 1, 3.5);
…
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-27', 3, 2.5);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-29', 2, 5.3);
```

```
SELECT tableoid, * from measurement;

 tableoid |          time          | sensorid |  sensor_value
----------+------------------------+----------+---------------
    18918 | 2022-01-03 00:00:00+00 |    2     |      2.9
    18918 | 2022-01-06 00:00:00+00 |    1     |      3.5
     …    |           …            |    …     |      …
    18924 | 2022-03-27 00:00:00+00 |    3     |      2.5
    18924 | 2022-03-29 00:00:00+00 |    2     |      5.3
```

Indexes

```
CREATE INDEX ON measurement (time);


SELECT
   tablename, indexname
FROM
   pg_indexes
WHERE
   tablename LIKE 'measurement%';

    tablename          |          indexname
-----------------------+------------------------------
 measurement           | measurement_time_idx
 measurementy2022m01   | measurementy2022m01_time_idx
 measurementy2022m02   | measurementy2022m02_time_idx
 measurementy2022m03   | measurementy2022m03_time_idx
```

Declarative partitioning

- Range partitioning
- List partitioning
- Hash partitioning
- Composite partitioning

03

Time-series data and You

Solutions

- Manual labor
- Diy bash script
- pg_partman & pg_cron

TimescaleDB

Postgres for time-series

Quick install

Install using your favorite package manager

```
apt install timescaledb-2-postgresql-14

yum install timescaledb-2-postgresql-14

sudo pacman -Syu timescaledb timescaledb-tune
```

Use TimescaleDB in a container

```
docker run -d --name timescaledb -p 5432:5432
```

Or install from source

```
git clone https://github.com/timescale/timescaledb.git
cd timescaledb
git checkout 2.5.1
./bootstrap
cd build && make
make install
```

https://docs.timescale.com/install/latest/

TimescaleDB

Create a table

```
CREATE TABLE measurement (
   time TIMESTAMPTZ NOT NULL,
   sensorID INT NULL,
   sensor_value REAL NULL
);
```

Create a hypertable

```
SELECT create_hypertable('measurement','time');
```

Create a distributed hypertable

```
SELECT create_distributed_hypertable('measurement', 'time', 'sensorID');
```

Inserts

```
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-3', 2, 2.9);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-01-6', 1, 3.5);
                                          …
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-27', 3, 2.5);
INSERT INTO measurement (time, sensorID, sensor_value) VALUES ('2022-03-29', 2, 5.3);
```

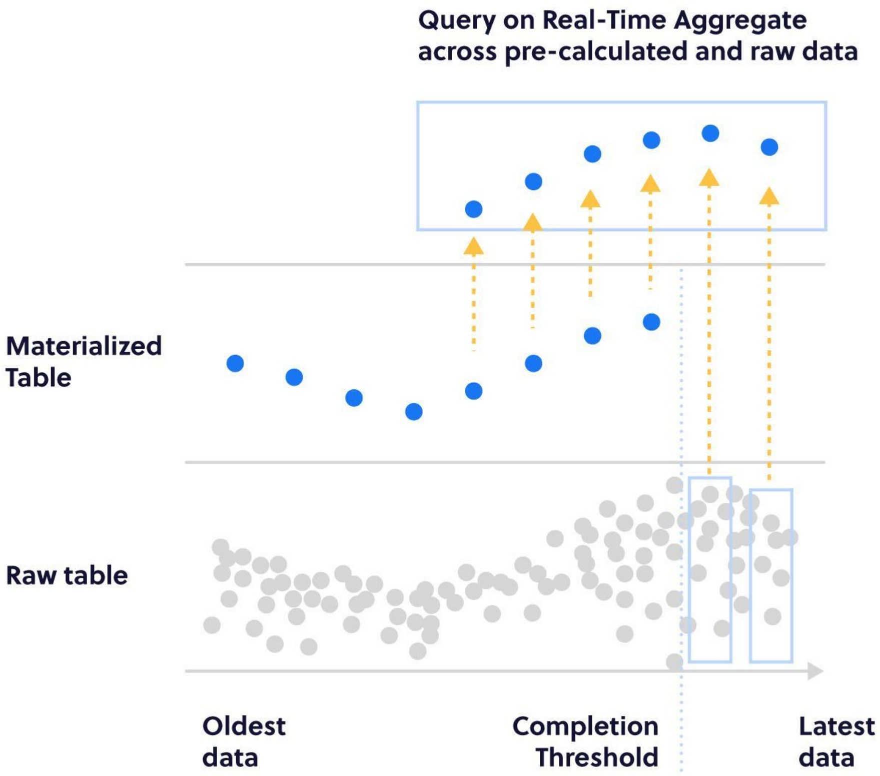
```
SELECT tableoid, * from measurement;

tableoid |          time          | sensorid | sensor_value
----------+------------------------+----------+--------------
   19204 | 2022-01-03 00:00:00+00 |        2 |          2.9
   19210 | 2022-01-06 00:00:00+00 |        1 |          3.5
   19216 | 2022-01-15 00:00:00+00 |        3 |          6.2
   19222 | 2022-01-23 00:00:00+00 |        1 |          3.2
    …    |           …            |    …     |           …
   19240 | 2022-02-13 00:00:00+00 |        2 |          2.6
   19246 | 2022-02-24 00:00:00+00 |        2 |          1.7
   19246 | 2022-02-28 00:00:00+00 |        1 |          6.1
   19252 | 2022-03-06 00:00:00+00 |        3 |          5.2
   19252 | 2022-03-08 00:00:00+00 |        2 |          2.3
```

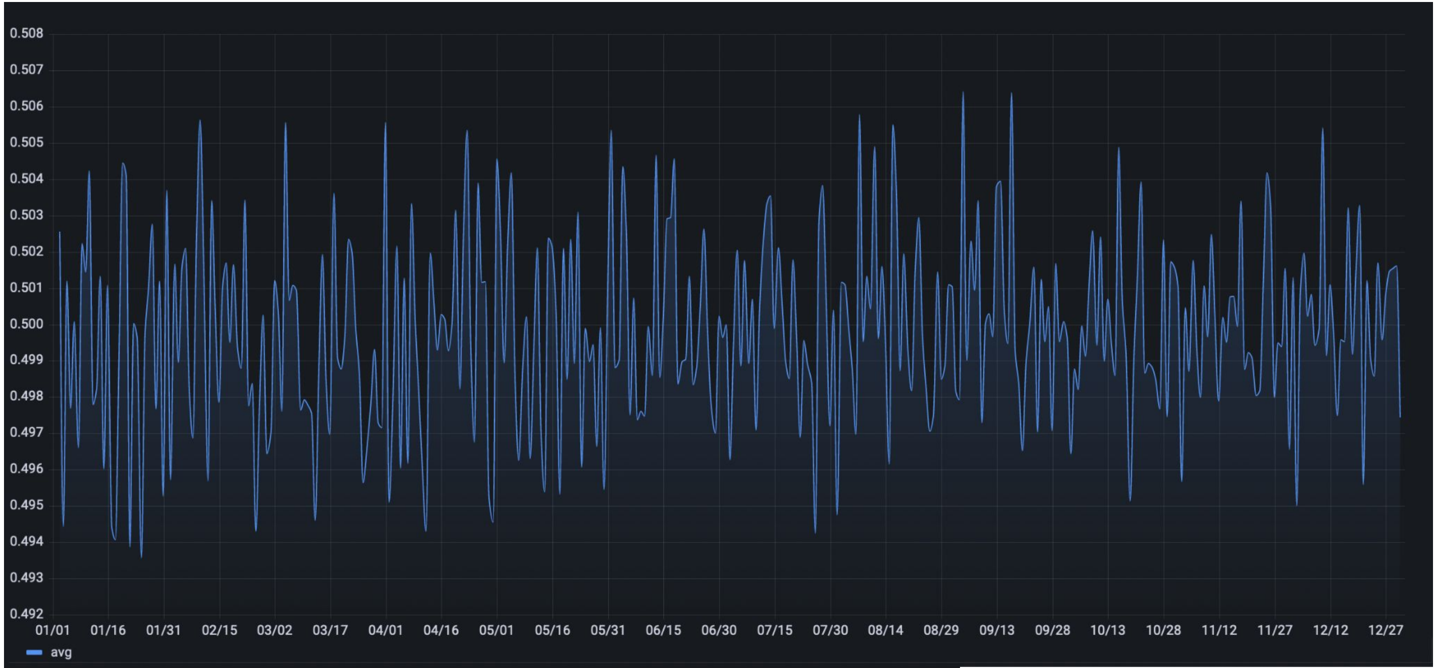
Auxiliary functions

```sql
SELECT show_chunks('measurement');

SELECT drop_chunks('measurement', INTERVAL '5 months');

SELECT reorder_chunk('_timescaledb_internal._hyper_4_78_chunk', 'measurement_time_idx');

SELECT attach_tablespace('disk2', 'measurement');

SELECT move_chunk(
 chunk => '_timescaledb_internal._hyper_4_78_chunk',
 destination_tablespace => 'disk2',
 index_destination_tablespace => 'disk2',
 reorder_index => 'measurement_time_idx',
);
```

https://docs.timescale.com/api/latest/hypertable/

Continuous aggregates





B (PostgreSQL)

```
select day as time, avg from measurement_daily;
```

Format as | Time series | Query Builder | Show Help ›

Continuous aggregates

```sql
CREATE MATERIALIZED VIEW measurement_daily
WITH (timescaledb.continuous) AS
SELECT
   time_bucket('1 day', "time") AS day,
   avg(sensor_value) AS avg,
   count(sensor_value) AS count
FROM measurement
GROUP BY day;
```

```sql
SELECT day AS time, avg FROM measurement_daily;
```

Compression

```sql
ALTER TABLE measurement SET (
 timescaledb.compress,
 timescaledb.compress_orderby = 'time DESC',
 timescaledb.compress_segmentby = 'sensorID'
);

SELECT add_compression_policy('measurement', INTERVAL '1 month');
```

Up to 94% compression!

Compression

As a simplified example, you might have a table that looks like this to start with:

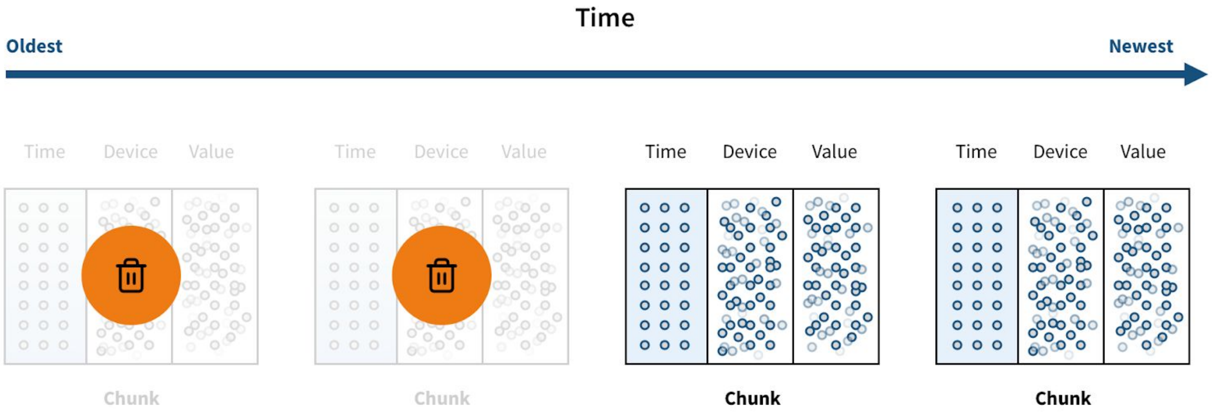
time	device_id	cpu	energy_consumption
12:00:02	1	88.2	0.8
12:00:02	2	300.5	0.9
12:00:01	1	88.6	0.85
12:00:01	2	299.1	0.95

When compression is applied, the data is converted to a single row containing an array, like this:

time	device_id	cpu	energy_consumption
[12:00:02, 12:00:02, 12:00:01, 12:00:1]	[1, 2, 1, 2]	[88.2, 300.5, 88.6, 299.1]	[0.8, 0.9, 0.85, 0.95]

Data Retention

```sql
SELECT add_retention_policy('measurement', INTERVAL '5 years');
```



Timescale Cloud

TimescaleDB in the cloud

Timescale Cloud

Create a service

Service summary

Advanced configuration

You can reconfigure your service anytime later on!

Region

us-east-1

High-availability

Enable replicas for your mission critical workloads.

Enable replica

Configuration

Scale later on

0.5 CPU / 2 GB RAM / 10 GB Storage

Pricing

30-day trial

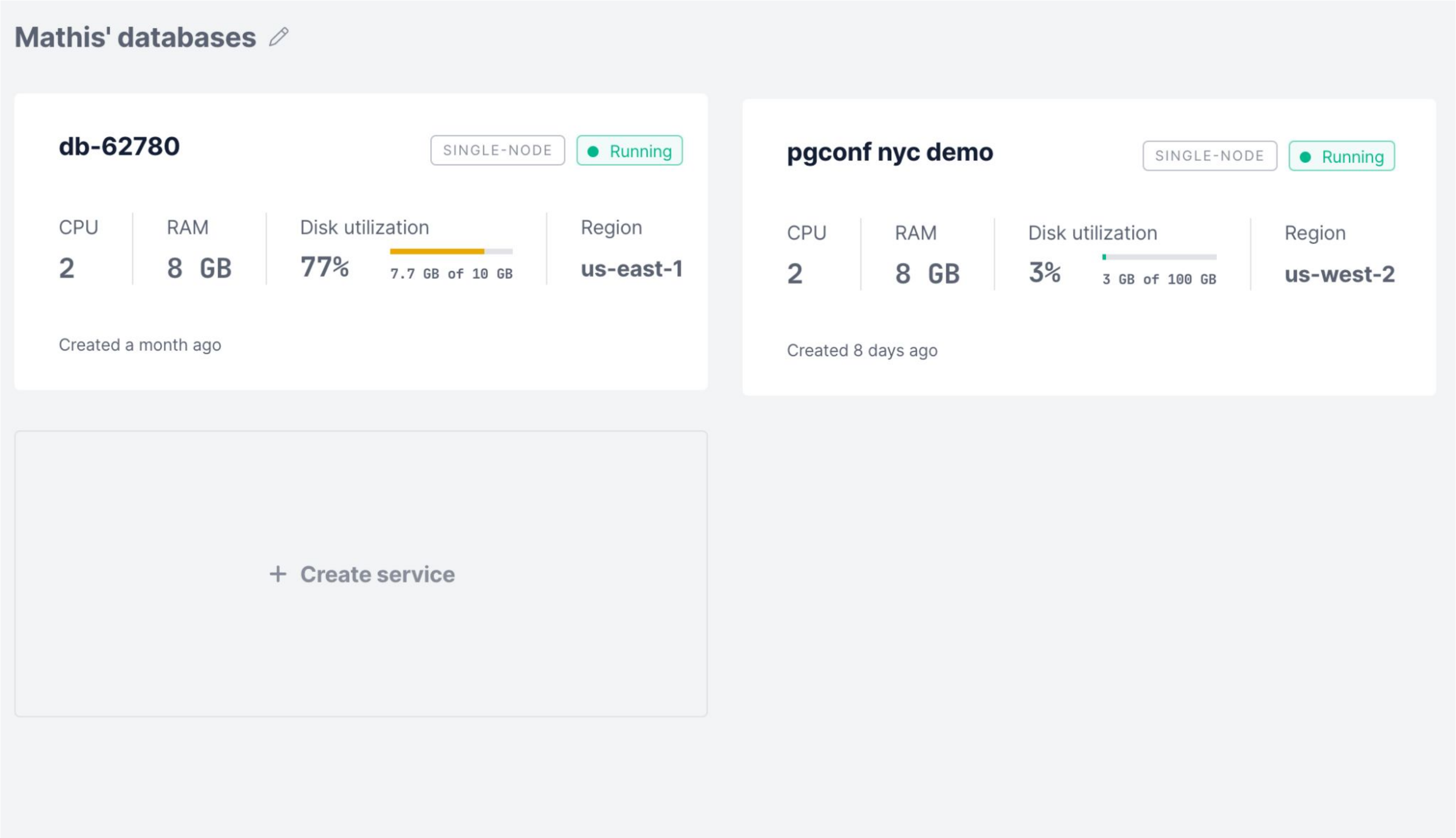
~~$0.053~~ $0 / hour

Compute	~~$0.041~~ $0 / hour
Storage	~~$0.012~~ $0 / hour
Monthly est.	~~$39~~ $0 / mo

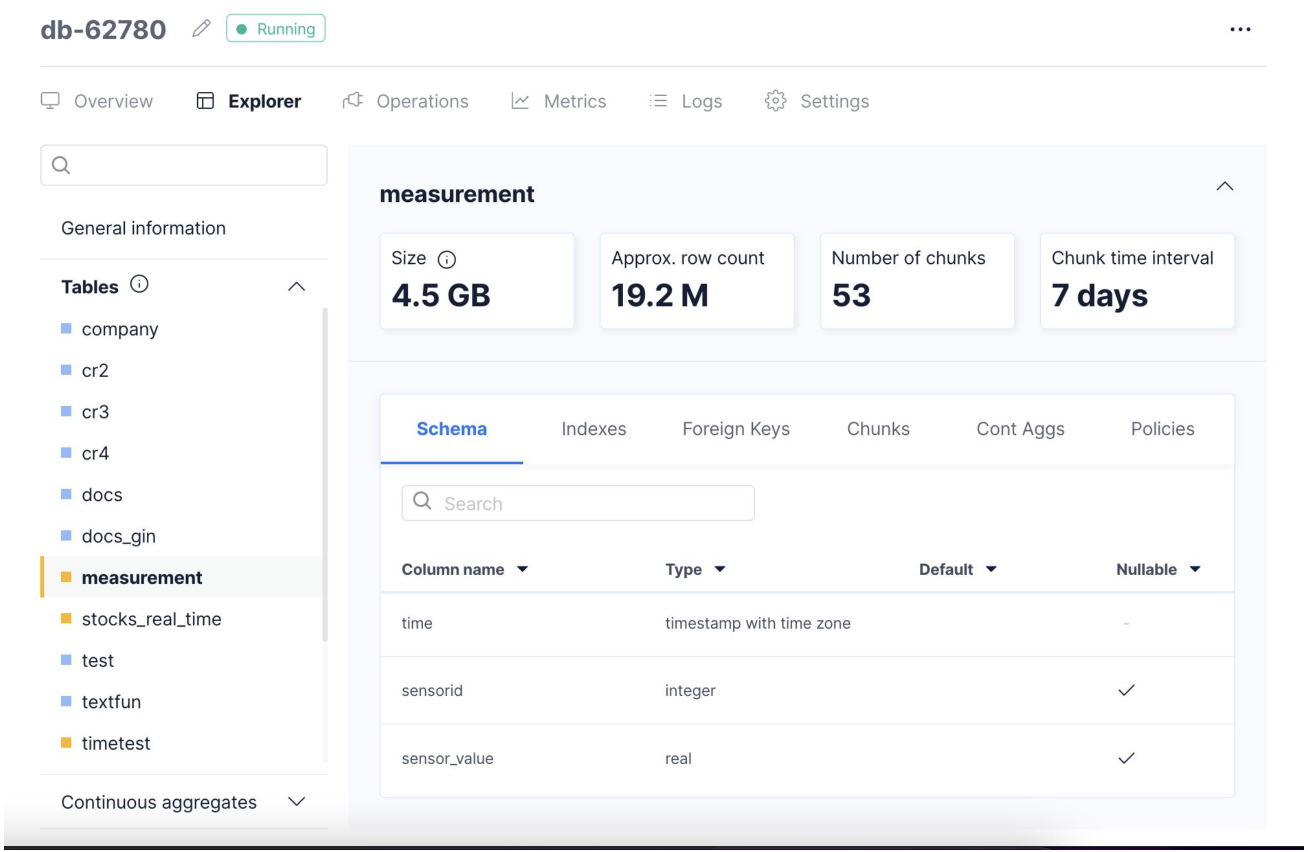
Interactive demo. Deploy a service with a demo dataset to learn more about TimescaleDB.

Create service

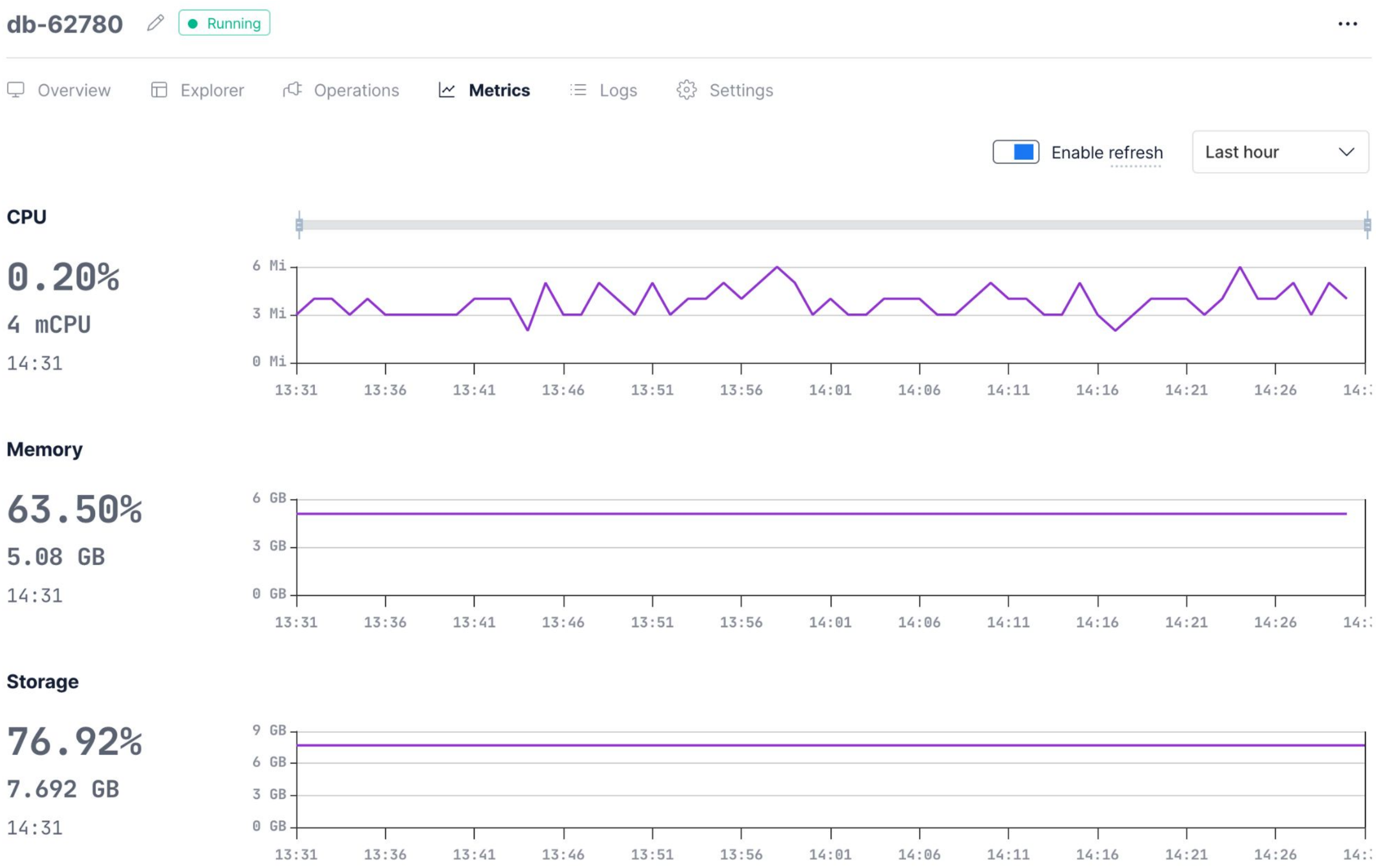
Timescale Cloud



Timescale Cloud



Timescale Cloud



Timescale Cloud

multinode-test

MULTI-NODE Running

Nodes		CPU/node	RAM/node	Region
Access nodes	1	8	32 GB	us-east-1
Data nodes	3	2	8 GB	

Highest disk utilizations

bafww0amr8	1%	420 MB of 50 GB	nkl5jlm6nm	0%	340 MB of 200 GB
fyod6yv530	0%	340 MB of 200 GB	m0scp73hko	0%	340 MB of 200 GB

Questions?

Scan the QR code to visit the Timescale home page!

Thank you!

Scan the QR code to visit the Timescale home page!