

# Querying Metrics in TimescaleDB

Published  
2021-07-20

RELEASE

# Table of Contents

**Introduction**

**Preparations**

**Querying the Metrics Database**

# Introduction

TimescaleDB is an open-source time-series database optimized for fast ingest and complex queries and supporting full SQL. It is based on and functions as an extension of PostgreSQL.

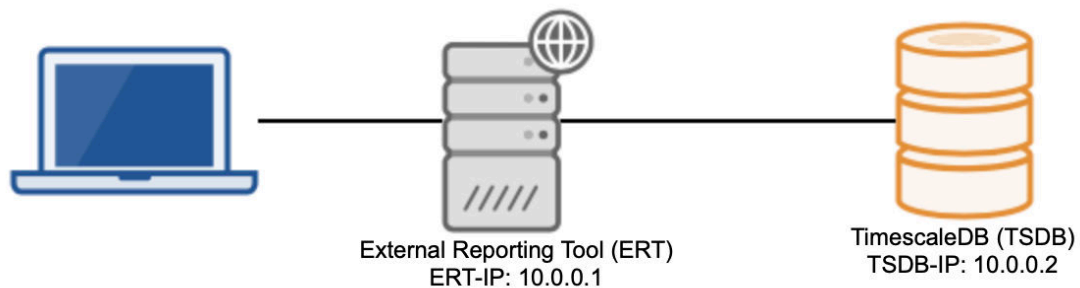
Paragon Active Assurance stores time-series data (metrics) in a TimescaleDB database (hereafter referred to as the "metrics database"). You can use external reporting tools such as Grafana to query metrics from this database.

The external reporting tool must support PostgreSQL databases as data sources.

To get started, you need to:

- specify the parameters for connecting to the metrics database;
- update the metrics database configuration to allow connections from the node where the external tool is installed.

In the following, we assume the scenario shown in the diagram below:



## Preparations

### IN THIS SECTION

- [Enabling Ingestion into TimescaleDB | 2](#)
- [Setting Up Support for PostgreSQL Databases | 2](#)
- [Allowing Incoming Connections from an External Tool TODO no longer present in the Confluence doc -- no longer needed? | 3](#)

- Parameters for Connecting to the Metrics Database | 3
- Additional Settings When Defining the PostgreSQL Data Source | 4

## Enabling Ingestion into TimescaleDB

To enable ingestion of Paragon Active Assurance data into TimescaleDB, do as follows:

- Enable the services `paa-metrics-service` and `paa-timescaledb`:

```
sudo systemctl enable paa-metrics-service paa-timescaledb
```

- In `/etc/netrounds/netrounds.conf` make the following settings:

```
KAFKA_METRICS_ENABLED=True
AVAILABLE_TIME_SERIES=['rrd', 'kafka']
```

- Restart all `netrounds-*` and `paa-*` services:

```
sudo systemctl restart netrounds-* paa-*
```

## Setting Up Support for PostgreSQL Databases

External tools use different approaches to supporting PostgreSQL databases:

- If the tool requires a JDBC driver, download the latest driver for PostgreSQL from <https://jdbc.postgresql.org/download.html> before proceeding with the configuration.
- Other reporting tools provide support for PostgreSQL databases through a data source plugin: this can either be installed separately or be included in their standard installation.

## Allowing Incoming Connections from an External Tool TODO no longer present in the Confluence doc -- no longer needed?

1. Take note of the IP address ERT-IP where the external tool is running (for example, 10.0.0.1).
2. Connect via SSH to the database node TSDB.
3. Switch to the postgres user: `su - postgres`
4. Open the file `$PGDATA/pg_hba.conf`.
5. Append the following line to that file (consult <https://www.postgresql.org/docs/12/auth-pg-hba-conf.html> on how to allow connections from subnets):
  - for IPv4: `host paa paaread <ERT-IP>/32 md5`; in our example, `host paa paaread 10.0.0.1/32 md5`
  - for IPv6: `host paa paaread <ERT-IP>/128 md5`
6. Run `pg_ctl reload`.
7. Now it is possible to connect from ERT to the database instance paa defined in the TSDB database server as user paaread.

## Parameters for Connecting to the Metrics Database

At this point, it is possible to connect to the database using the following connection parameters:

- Hostname: either TSDB or TSDB-IP
- Port: 7432
- Database: paa-metrics
- Username: <Paragon Active Assurance username>
- Password: <Paragon Active Assurance password>
- TLS/SSL Mode: disable

<Paragon Active Assurance username> and <Paragon Active Assurance password> might differ from one external tool to another, as they depend on the system module responsible for access management.

For example, within Grafana it is possible to configure one data source with credentials of the paaread read-only user, and all users will have their own credentials to access to the Grafana portal and the permission to use the same data source.

On the other hand, if there is a direct connection from the user to the database, each user that is allowed to access the database should have its own database credentials.

An external tool may request a JDBC connection to connect to the metrics database. If so, go to <https://jdbc.postgresql.org/documentation/head/connect.html> to find out how to compose the JDBC connection string given the parameters above.

## Additional Settings When Defining the PostgreSQL Data Source

- If the external tool has a TimescaleDB option, enable it so that the tool can suggest and use the features provided by this PostgreSQL extension.
- Set the following connection limits in the external tool:
  - Maximum number of open connections: 2
  - Maximum number of idle connections: 1
  - Maximum lifetime of a connection: 14,400 seconds (= 4 hours)

# Querying the Metrics Database

### IN THIS SECTION

- [Introduction | 4](#)
- [Basic TimescaleDB Functions | 5](#)
- [Database Objects to Query | 6](#)

## Introduction

The metrics database uses the TimescaleDB extension to efficiently manage time series, providing additional functions for data aggregation and rollup.

In this chapter we will introduce some of the TimescaleDB functions commonly used to retrieve time series data, as well as the database objects and fields of the metrics database and their usage within an SQL query.

## Basic TimescaleDB Functions

The `time_bucket` function in TimescaleDB allows you to customize the granularity of an underlying dataset. This function accepts arbitrary time intervals as well as optional offsets and returns the bucket start time.

As an example, suppose we have a table `temperature_measurement`:

```
CREATE TABLE temperature_measurement(  
  id BIGINT PRIMARY KEY NOT NULL,  
  "time" timestamp with time zone, -- time when the temperature has been read  
  device text, -- identifier of the device that read the temperature  
  temperature double precision -- metric value  
);
```

Now if measurements are taken every 10 seconds, using `time_bucket` we can run aggregation functions on a coarser granularity. The query below returns the median value (50th percentile) of the temperature within 30-second buckets:

```
SELECT time_bucket('30 seconds',"time") as 30s_bucket,  
  percentile_cont(0.5)  
  WITHIN GROUP (ORDER BY temperature) as "Temperature (Celsius)"  
FROM temperature_measurement  
GROUP BY 30s_bucket  
ORDER BY 30s_bucket;
```

**NOTE:** We recommend that you provide an alias for the field generated by `time_bucket`, different from the time column. In the above example, the alias `30s_bucket` is used.

The same alias must also be used in the `GROUP BY` and `ORDER BY` clauses.

Further functions that can be used for advanced analytics queries are found under the heading [Advanced analytic-queries](#) in the TimescaleDB documentation.

## Database Objects to Query

Each plugin in Paragon Active Assurance has:

- one view for tests
- five views for monitors with metrics aggregated at different granularities: 10 seconds (raw), 1 minute, 5 minutes, 30 minutes, and 60 minutes.

The naming convention for the view objects is the following:

- `vw_(monitor|test)_metrics_<plugin_name>` for raw data
- `vw_monitor_metrics_minute<minutes>_<plugin_name>` for predefined time buckets on raw data for monitoring.

As an example, the HTTP plugin has the following database objects that can be queried:

```
vw_test_metrics_http
vw_monitor_metrics_http
vw_monitor_metrics_minute1_http
vw_monitor_metrics_minute5_http
vw_monitor_metrics_minute30_http
vw_monitor_metrics_minute60_http
```

Each view contains two categories of information:

- measurement
- metric

The measurement category is needed to retrieve the information related to a task in a test or monitor, and it must be included in the WHERE clause of an SQL statement. Examples of fields in this category are:

```
monitor_name, task_name, measurement_name, test_name
```

The metric category consists of values of metrics retrieved at a given point in time.

The time column is common to all views and must be included in each query.

The columns holding specific measurements differ from one task to another, and they are usually part of the metrics table in the view definition.



For example, the `vw_monitor_metrics_http` view contains:

```
metrics.connect_time_avg,
metrics.first_byte_time_avg,
metrics.response_time_min,
metrics.response_time_avg,
metrics.response_time_max,
metrics.size_avg,
metrics.speed_avg,
metrics.es_timeout,
metrics.es_response,
metrics.es
```

Example: Here is how to write a query for the median of the response time average for an HTTP task associated with "Measurement 1" (test or monitor name) using a 30-second time bucket between 2021-06-21 09:55:30+00 and 2021-06-21 09:57:00+00.

```
SELECT
  -- time field
  time_bucket('30s',"time") as "time",
  -- metric
  percentile_cont(0.5) WITHIN GROUP (ORDER BY response_time_avg) as "avg_response_time"
FROM
  -- data source
  vw_monitor_metrics_http
WHERE
  -- time interval to analyze
  "time" between '2021-06-21 09:55:30+00' AND '2021-06-21 09:57:00+00'
  AND
  -- measurement identifier
  measurement_name = 'Measurement 1'
  AND
  -- account identifier
  account_short_name = 'account_1'
GROUP BY "time"
ORDER BY "time"
```

---

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. Copyright © 2021 Juniper Networks, Inc. All rights reserved.