JUNIPER
NETWORKS

Engineering
Simplicity

Querying Metrics in TimescaleDB

Published

2023-11-28

Table of Contents

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. Full information on TimescaleDB is found in the official documentation: https://docs.timescale.com/timescaledb/latest/overview/

TimescaleDB was introduced in Control Center version 3.1.0.

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 a data source.

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.

> ⚠ **WARNING**: When using TimescaleDB you must be sure to allocate sufficient disk space to this database. See the recommendations in the Installation Guide, chapter Installing Required OS and Software.

In the following, we assume a scenario where an external client connects to Control Center in order to extract data from its TimescaleDB database.

Preparations

IN THIS SECTION

Enabling Ingestion into TimescaleDB and Publishing of Metrics to Kafka

To enable ingestion of Paragon Active Assurance data into TimescaleDB, enable and start the services `netrounds-metrics` and `netrounds-timescaledb`:

```
sudo ncc services enable metrics timescaledb
sudo ncc services start metrics timescaledb
```

To be able to use TimescaleDB in conjunction with the Streaming API (see the *Streaming API Guide*), you must enable publishing of database metrics to the Kafka event-streaming platform. In the file `/etc/netrounds/netrounds.conf`, set:

```
KAFKA_METRICS_ENABLED = True
```

NOTE: Don't forget to uncomment this line.

Then restart all services:

```
sudo ncc services restart
```

Connecting to TimescaleDB

Connecting an external client to the TimescaleDB database is done in two steps:

- Setting up SSH tunneling between the database client node and the TimescaleDB node

- Defining a connection in the database client using the parameters for TimescaleDB

The subsequent sections describe these operations in more detail.

> **NOTE**: The definition of the SSH tunnel depends on the database client in use and the underlying operating system: in some circumstances (for example, with PgAdmin), both the SSH tunnel and the connection parameters can be defined within the database client itself.

Please refer to the documentation of the database client in use for more information.

SSH Tunneling configuration

To establish an SSH tunnel between the database client and TimescaleDB, the following information is required:

- SSH port (default value 22)

- OS user in the Control Center node

- FQDN/Public IP of the Control Center node

- Location of the Control Center public key and its password

- The database port number to which TimescaleDB listens (7432)

- The port number local to the database client that will forward the traffic to TimescaleDB. Its value depends on the port number available in the database client node and might differ from the listening port for TimescaleDB.

The section below shows how to set up an SSH tunnel on Linux.

Example: SSH Tunneling on Linux

> **NOTE**: The instructions below refer to a database client running in a Linux environment outside Control Center.

This example describes how to define an SSH tunnel between the database client and the TimescaleDB server.

Definitions:

- *Client:* The computer running the database client which you want to connect to the TimescaleDB database server.

- *Server:* The Control Center instance running the TimescaleDB database server.

Below, `<Control Center IP/FQDN>` is the IP address or Fully Qualified Domain Name of the node where Control Center is running. It is assumed that TimescaleDB on the server listens to the default port 7432.

Prerequisites:

- You have a private SSH key `paa_cc.key` created on the client, and you have copied the public key to the server.

- `openssh-server` is installed on the server.

Do the following:

- Open a terminal window on the client.

- Run this command to set up background SSH port forwarding:

```
ssh -i <path_to_private_key>/paa_cc.key \
-o ExitOnForwardFailure=yes -N -L <local_port>:127.0.0.1:7432 \
<os_user>@<Control Center IP/FQDN> &

echo $?
```

where `<os_user>` is the OS user on the TimescaleDB server and `<local_port>` is the port in the client node that forwards the traffic to the TimescaleDB listening port.

Database Configuration

Once there is an active SSH tunnel between the database client and the TimescaleDB database, you can establish a connection from the database client towards the TimescaleDB database using the following connection parameters:

- Hostname: 127.0.0.1

- Port: `<local_port>` (port number in the database client node that forwards the traffic to the TimescaleDB listening port)

- Database: `paa-metrics`

- Username: (`paaread` or database user with `paaread` role assigned)

- Password: (password of the database user given as Username)

- TLS/SSL Mode: `prefer`

Please refer to the documentation of the database client in use for the exact names of the above options.

For example, if `psql` is the database client in use and there is an SSH tunnel defined as in the section , the command

```
psql -h 127.0.0.1 -p <local_port> -U paaread paa-metrics
```

will successfully create a connection towards TimescaleDB. You will be prompted for a password, which is `paaread`.

Managing Database Users

Default User

By default, TimescaleDB in Control Center has a user `paaread`, with password `paaread`, which is allowed to access all the database objects related to metrics in *read-only* mode.

For security reasons, it is strongly recommended that you alter the default password of this user after logging in for the first time as user `paaread`. How to do this is described in the Operations Guide, chapter Updating Databases with New User Passwords.

Additional Users

If multiple database users need to access the TimescaleDB database to query metrics data, log in as superuser `paa`, with password `paa`. You can then create new users and grant them the same privileges as the `paaread` user.

Configuring Data Retention Periods

You can change the data retention periods for raw data and rollups in the file `/etc/netrounds/metrics.yaml`. The relevant properties are laid out in the table below:

Property name	Retention period for	Default retention period
retain-duration-raw	10-second interval	72.0h (3 days)
retain-duration-1min	1-minute rollups	168.0h (7 days)

`retain-duration-5min`	5-minute rollups	720.0h (30 days)
`retain-duration-30min`	30-minute rollups	4320.0h (180 days)
`retain-duration-hour`	60-minute rollups	8760.0h (365 days)

In addition, there is a property `cleanup-poll-duration` which governs how soon any changes to the retention periods will take effect. If you set a retention period lower than `cleanup-poll-duration`, the change will come into effect only after `cleanup-poll-duration`. In such a situation, it is best to lower `cleanup-poll-duration` as well.

Durations must be specified as decimal numbers with one of the following unit suffixes: ns, us, ms, s, m, h, as in the above table. (There is no "day" suffix.)

Troubleshooting

Problem: No data in the TimescaleDB tables when running queries.

- *Solution:*

  Make sure you have followed all the steps in the guide for turning on TimescaleDB ingestion (see the section "Enabling Ingestion into TimescaleDB" on page 2).

  You can also make sure that the metrics are being published to Kafka by using the `kafkacat` tool in Control Center:

  ```
  sudo apt-get install kafkacat
  kafkacat -b localhost:9092 -t netrounds.callexecuter.metrics -C -e
  ```

  If nothing is published, make sure that `KAFKA_METRICS_ENABLED=True` in `/etc/netrounds/netrounds.conf`, and after updating that file restart all services with

  ```
  sudo ncc services restart
  ```

Querying the Metrics Database

Introduction

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

In this chapter we will describe the database objects and fields of the metrics database and their usage within an SQL query.

For full coverage of TimescaleDB features, go to docs.timescale.com.

Database Objects to Query

Each plugin in Paragon Active Assurance has a number of database views which the user can query:

- 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 metrics data

- `vw_monitor_metrics_minute<minutes>_<plugin_name>` for metrics aggregated over time intervals of size `<minutes>` for the purpose of monitoring.

For 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:* This is metadata such as the name of the test or monitor and the Test Agent(s) used. It is needed to retrieve the information related to a task in a test or monitor, and it should be included in the WHERE clause of an SQL statement to narrow down the search. Examples of fields in this category are:

    - Common to monitoring and testing: `account_short_name`, `task_name`, `stream_name`, `plugin_name`

    - Specific to testing: `test_name`, `test_step_name`, `periodic_test_name`

    - Specific to monitoring: `monitor_name`, `task_started`, `monitor_tags`

    ```
    monitor_name, task_name, stream_name, test_name, test_agent_name
    ```

- *metric:* This is data collected in the network at a given point in time, such as throughput, delay, and jitter.

The `_time` column is common to all views and must be included in each query. However, the set of metrics returned depends on the task type: for example, an HTTP measurement will have a different set of metrics from a Ping measurement.

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:

```
connect_time_avg,
first_byte_time_avg,
response_time_min,
response_time_avg,
response_time_max,
size_avg,
speed_avg,
```

```
es_timeout,

es_response,

es
```

Running Queries in TimescaleDB

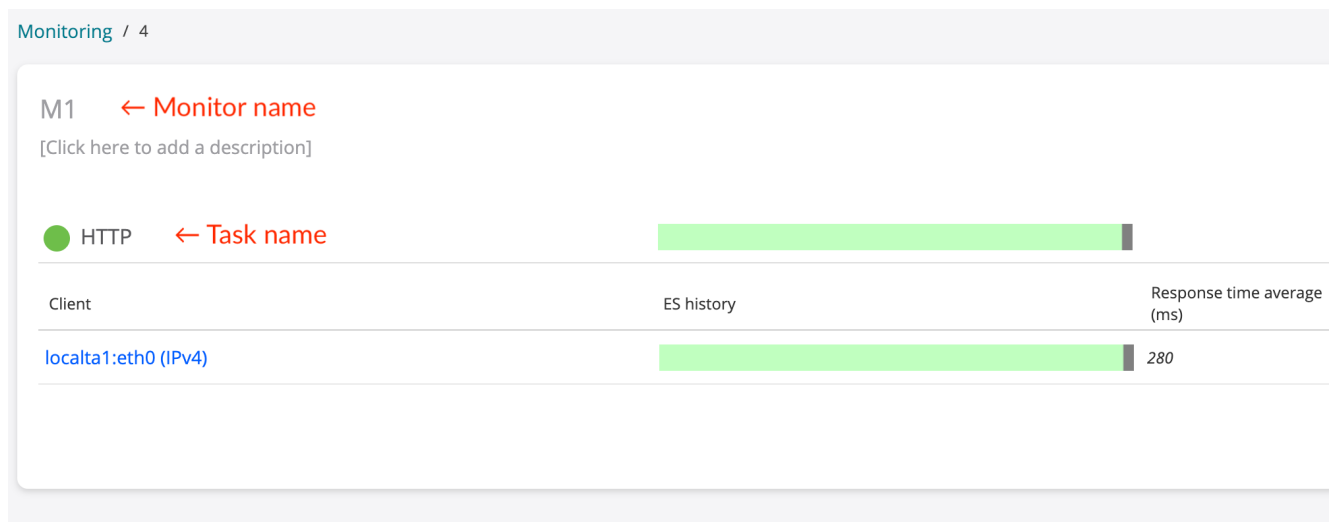To run queries in TimescaleDB it is important to:

- identify the measurement task that collects the metrics

- identify and set the time interval of interest

- set the data granularity for the metrics.

Identifying the Task of Interest

Metrics collected during execution of a test or monitor are produced by all the measurement tasks defined in the test or monitor. To uniquely identify such a task in the metrics database, it is essential to retrieve the right information from Control Center.

Identifying a Monitor Task

- In Control Center, click the **Monitoring** button on the left-side bar.

- Click **List**.

- Click the monitor name in the list and note the name of the task of interest. An example is shown below:

Monitoring / 4

M1   ← Monitor name
[Click here to add a description]

● HTTP   ← Task name

Client	ES history	Response time average (ms)
localta1:eth0 (IPv4)		280

- Click the client name and note the name of the stream, which is displayed as shown in the following image:


```
M1 - HTTP
http://www.juniper.net - localta1:eth0 (IPv4)    ← Stream name

2022-03-23 09:16:07 - 2022-03-23 09:31:07 (10 second resolution)
   GRAPH        TABLE
```

- Run this query, where `monitor_name`, `task_name`, and `stream_name` are taken from the Control Center GUI as just shown:

```
SELECT DISTINCT monitor_id,monitor_name,
                task_id,task_name,
                id as stream_id,stream_name
FROM
streams

WHERE monitor_name =
'<monitor_name>'

AND task_name = '<task_name>'
AND stream_name = '<stream_name>';
```

Take note of `monitor_id`, `task_id` and `stream_id` in the output. For example:

```
-[ RECORD 1 ]+----------------------------------------------
monitor_id   | 6
monitor_name | M1
task_id      | 8
task_name    | HTTP
stream_id    | 10
stream_name  | http://www.juniper.net - localta1:eth0 (IPv4)
```

Identifying a Test Task

This is similar to monitor task identification (see above) except that in the Control Center GUI, you click the **Tests** button on the left-side bar. The query is also modified accordingly:

```
SELECT DISTINCT test_id,test_name,
                task_id,task_name,
                id as stream_id,stream_name
FROM
streams

WHERE test_name =
'<test_name>'
AND task_name = '<task_name>'
AND stream_name='<stream_name>';
```

Setting the Time Interval of Interest

To ensure that we are querying data in the right time interval, it is important to know the time zone of reference in the metrics database and whether it differs from the time representation in Control Center.

To get the current time zone of the database, run this query:

```
SELECT now(),current_setting('TIMEZONE');
```

Example of output:

```
            now             | current_setting
----------------------------+-----------------
 2022-03-22 18:06:01.247507+00 | Etc/UTC
(1 row)
```

In this case, the current time zone of the database is UTC, so all time intervals are represented in UTC.

To convert a local time zone to UTC, perform the following steps:

1. Define the local time in the format `YYYY-MM-dd HH:mm` (`<local_time>`).

2. Select the appropriate local time zone:

- Run this query (`<continent>` is the name of the continent):

```
SELECT name
FROM pg_timezone_names
WHERE name ilike '<continent>/%'
ORDER BY name;
```

Select the time zone name corresponding to the city closest to you and assign it to `local_time_zone` (for example, `America/New_York`)

.

3. Run the query below to get the time in UTC:

```
SELECT TIMESTAMP '<local_time>' AT TIME ZONE '<local_time_zone>';
```

Example

Here is how to convert the time `2021-12-16 15:00` from Pacific Time to UTC:

1. `<local_time>` is `2021-12-16 15:00`.

2. Run this query to retrieve time zones in the Americas:

```
SELECT name
FROM pg_timezone_names
WHERE name ilike 'America/%'
ORDER BY name;
```

3. Select `America/Los_Angeles` from the list.

4. Run this query to have the time zone converted:

```
SELECT TIMESTAMP '2021-12-16 15:00' AT TIME ZONE 'America/Los_Angeles';
        timezone
------------------------
 2021-12-16 23:00:00+00
(1 row)
```

Setting Data Granularity for Metrics

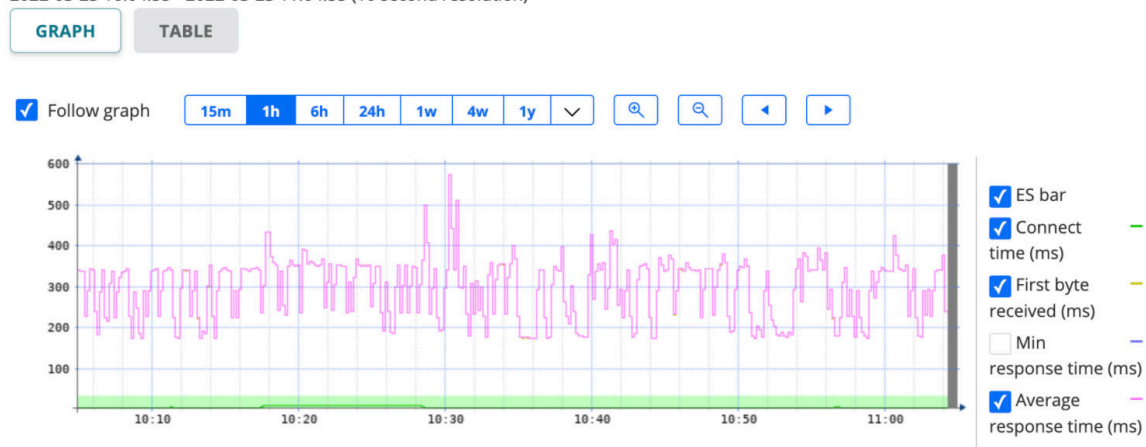Depending on the time interval of interest, it is possible to run queries on metrics using either

- the predefined views available in TimescaleDB directly, or

- the predefined views as a data source for metrics aggregation on custom time intervals.

Below is an example. Suppose we are analyzing the stream below:

Here we have:

- `monitor_id = 6` for the `M1` monitor

- `task_id = 8` for the `HTTP` task in `M1`

- `stream_id = 10` for the stream named `http://www.juniper.net - localta1:eth0 (IPv4)`

- Time interval delimited by `<time_start>` and `<time_end>`:

  - `<time_start> = 2022-03-23 10:04:55+00`

  - `<time_end> = 2022-03-23 11:04:55+00`

Queries on Predefined Views

These queries are recommended if the data granularity of interest is already present in TimescaleDB.

The format of the query is:

```sql
SELECT
  -- time field
  _time AS "time",
  -- metric
  response_time_avg as "avg_response_time"
FROM
  -- data source
  <data_source>
WHERE
  -- time interval to analyze
  _time between '<time_start>' AND '<time_end>'
  AND
  -- constraints on metadata
  monitor_id = <monitor_id>
  AND task_id = <task_id>
  AND _stream_id = <stream_id>;
```

The value of <data_source> is one of the objects listed in the "Database Objects to Query" on page 7 section.

Example

Given the context described in the previous paragraphs, the query

```sql
SELECT
  -- time field
  _time AS "time",
  -- metric
  response_time_avg as "avg_response_time"
FROM
  -- data source
  vw_monitor_metrics_minute5_http
WHERE
  -- time interval to analyze
  _time between '2022-03-23 10:04:55+00' AND '2022-03-23 11:04:55+00'
  AND
  -- constraints on metadata
  monitor_id = 6
```

```
    AND task_id = 8
    AND _stream_id = 10;
```

returns the value of the `response_time_avg` metric for a 5-minute interval detected between `2022-03-23` `10:04:55+00` and `2022-03-23 11:04:55+00` for the stream `http://www.juniper.net - localta1:eth0 (IPv4)` related to the task `HTTP` defined in the monitor `M1`.

Below is an example of output that may be returned in this case:

```
         time          | avg_response_time
------------------------+--------------------
 2022-03-23 10:05:00+00 | 276.53887939453125
 2022-03-23 10:10:00+00 |  281.2976379394531
 2022-03-23 10:15:00+00 |  324.9761657714844
 2022-03-23 10:20:00+00 |  331.6311340332031
 2022-03-23 10:25:00+00 |  289.0729064941406
 2022-03-23 10:30:00+00 |  320.9573974609375
 2022-03-23 10:35:00+00 | 239.66653442382812
 2022-03-23 10:40:00+00 | 305.64556884765625
 2022-03-23 10:45:00+00 |  314.2927551269531
 2022-03-23 10:50:00+00 |  261.7337646484375
 2022-03-23 10:55:00+00 | 278.86273193359375
 2022-03-23 11:00:00+00 |  302.1415100097656
```

Queries with Custom Time Interval

To run aggregate functions on metrics based on time intervals, TimescaleDB introduced the concept of *time buckets* to customize the granularity of the measurements given an underlying data set.

The function `time_bucket` accepts arbitrary time intervals as well as optional offsets and returns the bucket start time.

A user can apply aggregate functions on metrics grouped by the output from `time_bucket` to define metrics granularity dynamically, if the time interval of interest is not available in the TimescaleDB database.

The format of the query is:

```
SELECT
  -- time field
  time_bucket('<bucket_interval>',_time) as "time",
  -- metrics
  <metrics>
```

```
FROM
  -- data source
  <data_source>
WHERE
  -- time interval to analyze
  _time between '<time_start>' AND '<time_end>'
  AND
  -- constraints on metadata
  monitor_id = <monitor_id>
  AND task_id = <task_id>
  AND _stream_id = <stream_id>
GROUP BY time_bucket('<bucket_interval>',_time)
ORDER BY "time";
```

where `<data_source>` is one of the objects listed in the "Database Objects to Query" on page 7 section and `<metrics>` is a comma-separated list of metrics or an expression involving metrics.

Metrics for HTTP tasks are likewise listed in the "Database Objects to Query" on page 7 section.

`<bucket_interval>` denotes the interval width in use for rolling up the metric values. A value for `<bucket_interval>` consists of a number followed by a time unit such as:

- `s` (or `second`) for seconds

- `m` (or `minute`) for minutes

- `h` (or `hour`) for hours

- `d` (or `day`) for days

Example

Given the context described in the previous paragraphs and the change

- `<bucket_interval>` = `10m` (10-minute data granularity)

the query

```
SELECT
  -- time field
  time_bucket('10m',_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 '2022-03-23 10:04:55+00' AND '2022-03-23 11:04:55+00'
    AND
    -- constraints on metadata
    monitor_id = 6
    AND task_id = 8
    AND _stream_id = 10im
GROUP BY time_bucket('10m',_time)
ORDER BY "time";
```

returns the continuous 50% percentile value of the `response_time_avg` metric within 10-minute intervals detected between `2022-03-23 10:04:55+00` and `2022-03-23 11:04:55+00` for the stream `http://www.juniper.net - localta1:eth0 (IPv4)` related to the task `HTTP` defined in the monitor `M1`.

The output returned in this case looks like this:

```
         time          | avg_response_time
-----------------------+--------------------
 2022-03-23 10:00:00+00 | 288.12400817871094
 2022-03-23 10:10:00+00 |    338.41650390625
 2022-03-23 10:20:00+00 | 345.27549743652344
 2022-03-23 10:30:00+00 | 295.31700134277344
 2022-03-23 10:40:00+00 |   338.8695068359375
 2022-03-23 10:50:00+00 | 280.78651428222656
 2022-03-23 11:00:00+00 | 337.31800842285156
```

NOTE: For better query performance, it is recommended that you use as data source the view with the coarsest granularity that guarantees acceptable data accuracy.

Further information on TimescaleDB features for data analysis can be found in the TimescaleDB documentation: docs.timescale.com/timescaledb/latest/getting-started/query-data/

Example: Configuring Grafana as Reporting Tool

Introduction

This section describes how to query metrics in Control Center using Grafana.

Grafana is an open-source visualization and analytics software tool that lets you query, visualize, alert on, and explore metrics stored in a time-series database through tables and graphs.

> **NOTE**: This chapter requires a running instance of Grafana. You can download it from grafana.com/grafana/download and install it by following the instructions on that page.

Grafana allows you to create custom dashboards with multiple panels. Each panel uses a *data source* and runs a query against the data source. The query is defined using a query editor specific to the data source selected; in addition, the query editor manages the visualization options for the panel.

Setting Up the Data Source

Once Grafana is running, the next step is to define a data source for TimescaleDB so that Grafana can perform queries on the metrics. In the Preparations chapter, the section "Connecting to TimescaleDB" on page 2 instructed you to configure the SSH tunneling in addition to the database connection settings. In Grafana, configuring the SSH tunneling and the connecting to the database happen in two distinct phases, as described in the following sections.

Configuring the SSH Tunnel

Given the following:

- `<Control Center IP/FQDN>`, the IP or Fully Qualified Domain Name of the node where Control Center is running, and

- 7432, the database port which TimescaleDB listens to,

the SSH tunnel between the Grafana and the TimescaleDB servers can be defined as follows:

- Open a terminal window on the server where Grafana is installed.

- Install the `openssh-server` package.

- Copy the Control Center public key, `key.pub`, into `<path_to_key_pub>`on the Grafana server.

- Run this command to set up background SSH port forwarding:

```
ssh -i <path_to_key_pub>/key.pub \
-o ExitOnForwardFailure=yes -f -L <local_port>:127.0.0.1:7432 \
<os_user>@<Control Center IP/FQDN> sleep 15

echo $?
```

where `<os_user>` is the OS user in the TimescaleDB server and `<local_port>` is the port on the Grafana server that forwards the traffic to the TimescaleDB listening port.

The SSH tunnel is now set, and it is possible to define TimescaleDB as a data source in Grafana.

Defining the Data Source in Grafana

Once the SSH tunnel is active, the next step is to define a data source for TimescaleDB so that Grafana can perform queries on the metrics.

1. Open your web browser and go to `http://<Grafana host IP/FQDN>:3000/`. Here, 3000 is the HTTP port that Grafana listens to by default, and `<Grafana host IP/FQDN>` refers to the IP or Fully Qualified Domain Name of the node where Grafana is installed.

2. On the login page, enter the credentials for the admin user.

3. Click the configuration symbol (cogwheel) on the left. On the **Data sources** tab, click the **Add data source** button. Search for "Postgres" and click **Select**.

4. Fill out the PostgreSQL data source form as shown in the image below. In this case, the data source name used in the next steps is "Metrics Database", and the value of `<local_port>` is 9432.

Data Sources / PostgreSQL

Type: PostgreSQL

ⵟⵟⵟ Settings

| Name | ⓘ | Metrics Database | | Default | 🔵 |

PostgreSQL Connection

Host	127.0.0.1:9432
Database	paa-metrics
User	paaread	Password	••••••
TLS/SSL Mode	disable

Connection limits

Max open	2	ⓘ
Max idle	1	ⓘ
Max lifetime	14400	ⓘ

PostgreSQL details

Version	ⓘ	12 ⇕
TimescaleDB	🔵	Help ›
Min time interval	10s	ⓘ

5. Once you are done, click **Save & test**.

Creating a Dashboard

1. On the sidebar, click the plus sign, then **Dashboard** and finally **Add an empty panel**.

2. In the **New dashboard/Edit panel** view, go to the **Query** tab.

3. Configure your query by selecting "Metrics Database" from the data source selector.

4. Click **Edit SQL** to provide a query in raw SQL.

5. In the text field that appears, enter a query formatted like the one below for an HTTP monitor named "Stream 1". Note how the interval delimiters are replaced with the Grafana macros `$__timeFrom()` and `$__timeTo()`. (More Grafana macros for PostgreSQL are available here: grafana.com/grafana/download)

```
SELECT
    -- time field (alias "time" is required)
    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 $__timeFrom() AND $__timeTo()
    AND
    -- stream identifier
    stream_name = 'Stream 1'
    AND
    -- account identifier
    account_short_name = 'account_1'
GROUP BY "time"
ORDER BY "time";
```

6. Click the **Query Inspector** button to validate. Then, on the panel to the right, on the **Query** tab, click the **Refresh** button. Data will be visualized in the panel, and above the chart you can adjust the time interval to analyze.

7. Click the **Save** icon in the top right corner of the screen to save the dashboard.

8. Add a descriptive name, and then click **Save**.