
Spotify PostgreSQL Metrics

Service to extract and provide metrics on your PostgreSQL database.

This tool is a CLI (command line) tool that can be called to extract statistics and create metrics from your PostgreSQL database cluster. CLI is runnable in long running process mode, which will periodically send the gathered metrics forward.

The default metrics format is a Metrics 2.0 compatible JSON, which is created by a set of functions listed in the configuration file.

The extracted metrics can be printed out as direct output of the CLI tool, or sent out of the host the postgresql-metrics process is running in using FFWD.

The FFWD format is plain JSON sent over a UDP socket, so you can use whatever UDP socket endpoint, which understands JSON, to consume the metrics.

Prerequisites

The versions mentioned below are tested to work, but the code should work on many unlisted versions as well. Just add an issue or send a pull-request with missing prerequisites, if you test and confirm postgresql-metrics to work on other versions of the mentioned technologies.

- Python 3.5 or later
- PostgreSQL 9.3 or later
- Debian based distribution with systemd (packaging requirement only)

Building and Installing

You can build a Debian package by running the following in project root:

```
1 dpkg-buildpackage -us -uc -b
```

Notice that postgresql-metrics includes by default a systemd service to run as long running process, pushing metrics to FFWD as gathered. You need to stop the long running process after installing the package for configuration.

```
1 sudo systemctl stop postgresql-metrics
```

If you want to debug the process. For the systemd service, you can run *sudo journalctl -u postgresql-metrics* to see the service's log.

Edit Configuration

Edit the configuration in `/etc/postgresql-metrics/postgresql-metrics.yml` and in `/etc/postgresql-metrics/default/postgresql-metrics.yml`. Notice that the configuration in the default folder will be overwritten value by value from the configuration in the configuration root.

If you are not interested in using the default configuration overwriting functionality, just delete one of the configurations mentioned above, and keep using a single configuration file.

Edit at least the values under `postgres` section in the configuration to match your PostgreSQL cluster setup. Remember also to list the `databases` you want to gather metrics from. By database in this context we mean a database name you created within your PostgreSQL cluster.

Prepare Database

Before starting to extract the defined metrics, you need to setup your database cluster using the `prepare-db` CLI call. This will create the required extensions for your database, and a few functions that are used by the statistics gathering queries from the metrics process. The configured metrics user will be also granted access to the created statistics functions and views.

You need to provide administrator user to the `prepare-db` call, which the tool is kind enough to ask. You don't need to provide credentials if you are running the `prepare-db` with a local user that is configured to be trusted locally by the PostgreSQL cluster (in `pg_hba.conf`), and is a super user, like the default `postgres` user created by some distribution packages (e.g. Debian). You can do the `prepare-db` call e.g. as follows:

```
1 sudo su -c "postgresql-metrics prepare-db" postgres
```

It is safe to call the `prepare-db` multiple times for the same database (the call is idempotent).

Grant Access for Metrics User

In addition to granting access to the statistics gathering functions and views within your PostgreSQL cluster (previous step), you need to also add access to the metrics user into the host based access file (`pg_hba.conf`).

Add one line per database you are monitoring into the end of the `pg_hba.conf` file for your cluster:

```
1 host my_database_name postgresql_metrics_user 127.0.0.1/32 md5 #  
  metrics user access
```

Replace the `my_database_name` and `postgresql_metrics_user` with the values you configured into the `postgresql-metrics` configuration in **Edit Configuration** step above.

You need to reload (or restart) your server after editing *pg_hba.conf* for the changes to take effect.

Getting Metrics

After you have the postgresql-metrics configured, and the database prepared, you can print out all the metrics that will be extracted from your database by calling:

```
1 postgresql-metrics all
```

You need to call the command above as a user that has access to the WAL log directory under PostgreSQL, or the metric gathering WAL file amounts will fail. Single failed metric calls will not prevent the rest of gathering process.

You can also start the long running process again, if using systemd:

```
1 sudo systemctl start postgresql-metrics
```

Explaining the Gathered Metrics

This section explains the metrics we gather using this tool.

Notice that there are many system specific metrics that you should gather in addition to the Postgres specific metrics, for example:

- CPU usage
- Network usage, sent / received bytes per related network interface
- Memory usage
- Disk I/O operations
- I/O await times
- Disk usage and free space left

Database Specific Metrics

Called once per configured database inside your Postgres cluster.

- **get_stats_disk_usage_for_database:** This metric shows the size of each of your databases in bytes. Don't forget to measure the total disk usage of the disk your data directory resides in as well.
- **get_stats_tx_rate_for_database:** Shows the rate of transactions executed per second since the last call of this function. Shows the rate of executed rollbacks also separately.

-
- **get_stats_seconds_since_last_vacuum_per_table:** This metric shows the amount of time passed in seconds since the last vacuum was run per table in your database.
 - **get_stats_oldest_transaction_timestamp:** This metric shows the time the longest running transaction has been open in your database. This should be usually close to zero, but sometimes, for example when an administrator forgets to close a maintenance connection, you will see this value going up. Long running transactions are bad for your database, so fix the issue as soon as you see this metric increase.
 - **get_stats_index_hit_rates:** This metric shows you the amount of reads hitting the table indexes versus the amount of reads requiring sequential scan through the table. Depending on your table, the amount of data, and the created indexes, the index hit rate varies. You should understand your data well enough to know when high index usage is desirable to low index usage.
 - **get_stats_table_bloat:** This metric shows the amount of wasted space in the database table due to the MVCC process. Deletes and updates to the table just mark the obsolete data free, but does not really delete it. Vacuums do free some of this wasted data, but to get totally rid of table bloat you must re-create the table with vacuum full. The current implementation of table bloat metric is rather heavy, so you might want to disable it in case you see issues with it.

Database Cluster (Global) Metrics

Called once per your Postgres cluster.

- **get_stats_client_connections:** This metrics shows the amount of connections open to the database at the moment. The actual metrics gathering connections should be visible here as well. Notice that having more than a hundred connections open is usually a bad thing. Consider using a connection pooler, like pgbouncer.
- **get_stats_lock_statistics:** This metric shows locks being waited upon by queries, and the amount of locks granted. In general having any query waiting for locks for any extended period of time is a sign of problems, like heavy lock contention.
- **get_stats_heap_hit_statistics:** This metric shows the amount of reads hitting the memory buffers on your cluster, and also the amount of reads hitting the disk (or disk caches on the operating system). Also the heap hit ratio is calculated based on these values.

Notice that the read amounts are not actual read queries, but the amount of blocks read. You will get good idea of amount of reads hitting your database, when comparing these values with the transaction rate.

-
- **get_stats_replication_delays:** This metric shows the amount of bytes the replication delay is behind master per each slave. If the slave and the master are in synchronous state, the replication delay is zero.
 - **get_stats_wal_file_amount:** This graph shows the amount of files in your database clusters WAL log directory (pg_wal or pg_xlog). If the WAL file amount starts to suddenly increase, you probably have issues with your WAL archiving process, which might lead to the disk filling up, and you database cluster crashing.
 - **get_xid_remaining_ratio, get_multixact_remaining_ratio, get_multixact_members_remaining_ratio:** These metric shows the corresponding remaining % of transaction ids (“xid”), multixact ids (“mxid”), and multixact members that are available for postgres to use before exhaustion. Useful for ensuring that the vacuuming is working as intended for your postgres instance.
 - **get_multixact_members_per_mxid:** This metric emits the number of multixact members there are per multixact ID. A larger number means that it’ll be quicker for the multixact members exhaustion to happen (as can be seen in **get_multixact_members_usage_ratio**).

Short Overview of Python Modules

- **common.py:** Contains code common for the whole package, like logging and configuration parsing.
- **default_metrics.py:** Is responsible for turning statistics values into the default metrics JSON format.
- **localhost_postgres_stats.py:** Functions for statistics extraction from local Postgres data directory. These calls are automatically global from a database cluster perspective.
- **metrics_gatherer.py:** Functions for calling the statistics extraction functions and converting the results into correct metrics format. Called from the metrics_logic.py.
- **metrics_logic.py:** Contains the CLI tool, initialization, and the scheduling logic.
- **postgres_queries.py:** Functions for statistics extraction from the Postgres database.
- **prepare_db.py:** Code for preparing your databases for the metrics gathering process.

How to Add More Metrics

If you want to add more metrics into postgresql-metrics tool, you can do it by making the following changes to the source:

1. If you gather the metric using a Postgres SQL query, add the code into *postgres_queries.py*, and if you gather the metric by accessing the local Postgres data directory, add the code into *localhost_postgres_stats.py*.

-
2. Write a function for formatting your gathered metric values into wanted format, as is done in **default_metrics.py**. You can either expand the default metrics, or write your own format into another module.
 3. Write a function into **metrics_gatherer.py**, which will call the metric extraction functionality you wrote on the first step, and then the metric value formatting function you wrote on the previous step.
 4. Add the name of your metrics gatherer function, written in the previous step, into *postgresql-metrics* configuration file, with wanted time interval to call the metric gathering function. Notice that you need to add the function into the correct list of functions depending on whether you gather a metric that covers your whole database cluster, or a metric that targets a single database in your cluster. Data directory based metrics must be a 'global' metric.
 5. Update this README with explanation on what your new metric is about.