Open PostgreSQL Monitoring Documentation

Release 2.4

OPMDG

Apr 13, 2022

Contents

CHAPTER 1

Important

Warning: This documentation is a work in progress.

If you have any question that is not yet answered here, feel free to create an issue, so that we can improve it.

CHAPTER 2

Contents:

2.1 General

2.1.1 The UI is up, but I don't see anything

Here are the things to check and fix:

Are there perfdata generated?

On the Nagios server, check if perfdata are generated. The location is defined in the `proces-service-perfdata-file` and `proces-host-perfdata-file` commands on Nagios, as seen in *Nagios & nagios_dispatcher* section.

Assuming default configuration (`/var/lib/nagios3/spool/perfdata/`), you can check like this:

```
# ls -al /var/lib/nagios3/spool/perfdata/
```

Warning: Obviously, you need to have configured Nagios so that checks are actually performed to expect perfdata generated.

If files are getting created, you can skip to the next item. Otherwise, you need to configure Nagios so perfdata are processed, as documented in *Nagios & nagios_dispatcher*. Also be careful, if the `nagios_dispatcher` is running, perfdata files won't stay long in the perfdata directory.

Is nagios_dispatcher running?

If perfdata are being generated but are accumulating, you have an issue with the *nagios dispatcher*:

- check and double check the configuration file
  - perfdata directory

  - connection credentials
  - host and port
- check the connection with the opm database
  - PostgreSQL logs
  - pg_hba and/or credentials
- check that the daemon is running:

```
root:~# ps aux | grep nagios_dispatcher
```

- check the nagios dispatcher logs
- beware: `debug=1` in the dispatcher configuration file will make it stop

at the first problem in the data.

When the perfdata are being removed from the perfdata directory, you can move to the next item.

Are the perfdata accumulating in the hub table?

If the nagios_dispatcher is running and perfdata files being removed, lines should be added in the `wh_nagios.hub` table of the `opm` database:

```
opm=# SELECT COUNT(*) FROM wh_nagios.hub;
```

Warning: If you don't see any line appearing in this table, it's very likely that you're looking at the wrong server and/or the wrong database.

According to the *wh_nagios* documentation, you should have setup a cron to call the `wh_nagios.dispatch_record()` stored function every minute.

If after some minutes the number of records doesn't fall, there's a problem with this cron. Make sure to double check:

```
* output of this cron if you redirected it to a file or a mail
* credential access (you can use a `.pgpass file
```

<http://www.postgresql.org/docs/current/static/libpq-pgpass.html>`_ if needed)

- host, port, user, database...
- check the connection with the opm database
  - PostgreSQL logs
  - pg_hba and/or credentials

Once the `wh_nagios.dispatch_record()` is successfully called, the data will appear in the user tables. You can check for instance the number of servers the UI knows about:

```
opm=# SELECT COUNT(*) FROM public.servers;
```

I still can't see anything in the UI

- check the credentials for the *dedicated UI user*
- check the *UI configuration file*
- check the connection with the opm database
  - PostgreSQL logs
  - pg_hba and/or credentials
- check the UI logs. For instance, if you used an *Apache server*, the `opm.log`. If you tried with the *morbo tool*, then the standard output.
- check that you connect to the good OPM UI server.

2.1.2 Getting started

Overview

OPM is a set of tools working above Nagios (or similar) to store into a database the perfdata retrieved by any Nagios agent and graph them in a web dashboard.

check_pgactivity (https://github.com/OPMDG/check_pgactivity) is a Nagios agent dedicated to PostgreSQL. It is technically separated from OPM.

Requirements & Installation

We suppose that you have in place and configured to your need:

- a working Nagios (or similar) installation
- agents displaying perfdata (e.g. check_pgactivity or other Nagios probes)
- an empty PostgreSQL database (9.3 or later) on the instance of your choice to store the data.

See the *Installation* page to install:

- the *opm_core* and *wh_nagios* extensions
- the *nagios dispatcher script* to retrieve the perfdata
- the *web user interface*.

2.1.3 Support

Community Support

You can find help, news and security alerts on the opm-users mailing list :

https://groups.google.com/forum/?hl=fr#!forum/opm-users

You can also join directly the developer team on the #opm channel of the freenode IRC network

To report an issue, please use the bug tracking system in the github project page: https://github.com/opmdg/

Commercial Support

DALIBO, as the main sponsor of the project, can provide enterprise-grade support services for both PostgreSQL and OPM. See http://www.dalibo.com for more details.

2.2 OPM Core

2.2.1 Installation

Requirements

To install OPM, you need a 9.3 or more PostgreSQL cluster, standard compiling tools and Nagios. The PostgreSQL cluster and Nagios can be installed on the servers you want, and can be installed on the same server.

System

Install the PostgreSQL development packages of your Linux distribution if necessary (e.g. `postgresql96-devel` on Red Hat, `postgresql-server-dev-9.6` on Debian).

The tool `pg_config` is required. Distributions packages put it usually in `/usr/pgsql-9.6/bin/pg_config` (Red Hat) or `/usr/lib/postgresql/9.6/bin/pg_config` (Debian). It may be necessary to put it in your `$PATH` at least temporarily (e.g. `export PATH=/usr/lib/postgresql/9.6/bin:$PATH`), especially if you have more than one version of PostgreSQL on the server.

We suppose that the repositories **opm-core** (https://github.com/OPMDG/opm-core) and **opm-wh_nagios** (https://github.com/OPMDG/opm-wh_nagios) are stored into **/usr/local/src/opm/** (your OPM directory).

OPM core

We need to install the core of OPM first. From your opm directory, as user `root`:

```
root:/usr/local/src/opm# cd opm-core/pg
root:/usr/local/src/opm/opm-core/pg# make install
```

It will copy some files into the extensions directory of PostgreSQL.

Then, using a superuser role:

```
postgres@postgres=# CREATE DATABASE opm;
postgres@postgres=# \c opm
postgres@opm=# CREATE EXTENSION opm_core;
```

You'll need to create a first OPM administrator account:

```
postgres@opm=# SELECT create_admin('admin1', 'agoodpassword');
```

This is the user you'll need to log on the UI.

wh_nagios

To install the module "wh_nagios", from your OPM directory as user "root":

```
root:/usr/local/src/opm# cd opm-wh_nagios/pg
root:/usr/local/src/opm/wh_nagios/pg# make install
```

Then, using a superuser role, in your **opm** database:

```
postgres@opm=# CREATE EXTENSION hstore;
CREATE EXTENSION

postgres@opm=# CREATE EXTENSION wh_nagios;
CREATE EXTENSION
```

Then, you need to create a crontab that will process incoming data and dispatch them. For instance, to trigger it every minute:

```
* * * * * psql -c 'SELECT wh_nagios.dispatch_record()' opm
```

This crontab can belong to any user, as long as it can connect to the PostgreSQL `opm` database with any PostgreSQL role.

To import data into a warehouse, you need a PostgreSQL role. We recommand to create a dedicated role, for instance:

```
postgres@opm=# CREATE ROLE opm_dispatcher LOGIN PASSWORD 'anothergoodpassword';
```

You must then allow this role to import data in a warehouse, by calling `public.grant_dispatch`. For instance, if the PostgreSQL role is `opm` and the warehouse is `wh_nagios`:

```
postgres@opm=# SELECT grant_dispatcher('wh_nagios', 'opm_dispatcher');
```

Nagios & nagios_dispatcher

The dispatcher `nagios_dispatcher.pl` aims to dispatch perfdata from Nagios files to the `wh_nagios` warehouse.

`nagios_dispatcher` requires the DBD::Pg perl module. It is packaged in `perl-DBD-Pg` (Red Hat), `libdbd-pg-perl` (Debian) or in the CPAN.

We'll need first to setup Nagios to create its perfdata files that "nagios_dispatcher" will poll and consume. As `root`, create the command file and destination folder:

```
root:~# mkdir -p /var/lib/nagios3/spool/perfdata/
root:~# chown nagios: /var/lib/nagios3/spool/perfdata/
root:~# cat <<'EOF' >> /etc/nagios3/commands.cfg
define command{
    command_name    process-service-perfdata-file
    command_line    /bin/mv /var/lib/nagios3/service-perfdata /var/lib/nagios3/spool/
↪perfdata/service-perfdata.$TIMET$
}
define command{
    command_name    process-host-perfdata-file
    command_line    /bin/mv /var/lib/nagios3/host-perfdata /var/lib/nagios3/spool/
↪perfdata/host-perfdata.$TIMET$
}
EOF
```

Then, in your Nagios main configuration file, make sure the following parameter are set accordingly:

```
process_performance_data=1
host_perfdata_file=/var/lib/nagios3/host-perfdata
service_perfdata_file=/var/lib/nagios3/service-perfdata
host_perfdata_file_processing_command=process-host-perfdata-file
service_perfdata_file_processing_command=process-service-perfdata-file
host_perfdata_file_template=DATATYPE::HOSTPERFDATA\tTIMET::$TIMET$\tHOSTNAME::
↪$HOSTNAME$\tHOSTPERFDATA::$HOSTPERFDATA$\tHOSTCHECKCOMMAND::$HOSTCHECKCOMMAND
↪$\tHOSTSTATE::$HOSTSTATE$\tHOSTSTATETYPE::$HOSTSTATETYPE$\tHOSTOUTPUT::$HOSTOUTPUT$
service_perfdata_file_template=DATATYPE::SERVICEPERFDATA\tTIMET::$TIMET$\tHOSTNAME::
↪$HOSTNAME$\tSERVICEDESC::$SERVICEDESC$\tSERVICEPERFDATA::$SERVICEPERFDATA
↪$\tSERVICECHECKCOMMAND::$SERVICECHECKCOMMAND$\tHOSTSTATE::$HOSTSTATE
↪$\tHOSTSTATETYPE::$HOSTSTATETYPE$\tSERVICESTATE::$SERVICESTATE$\tSERVICESTATETYPE::
↪$SERVICESTATETYPE$\tSERVICEOUTPUT::$SERVICEOUTPUT$
host_perfdata_file_mode=a
service_perfdata_file_mode=a
host_perfdata_file_processing_interval=15
service_perfdata_file_processing_interval=15
```

Note: If you're using Icinga2 instead of Nagios, you need instead to:

- enable perfdata:

```
$ icinga2 feature enable perfdata
```

- configure data format in `/etc/icinga2/features-enabled/perfdata.conf`:

```
library "perfdata"
object PerfdataWriter "perfdata" {
    host_perfdata_path = "/var/spool/icinga2/perfdata/host-perfdata"
    service_perfdata_path = "/var/spool/icinga2/perfdata/service-perfdata"
    rotation_interval = 15s
    host_format_template = "DATATYPE::HOSTPERFDATA\tTIMET::$icinga.timet
↪$\tHOSTNAME::$host.name$\tHOSTPERFDATA::$host.perfdata$\tHOSTCHECKCOMMAND::
↪$host.check_command$\tHOSTSTATE::$host.state$\tHOSTSTATETYPE::$host.state_type$
    service_format_template = "DATATYPE::SERVICEPERFDATA\tTIMET::$icinga.timet
↪$\tHOSTNAME::$host.name$\tSERVICEDESC::$service.name$\tSERVICEPERFDATA::
↪$service.perfdata$\tSERVICECHECKCOMMAND::$service.check_command$\tHOSTSTATE::
↪$host.state$\tHOSTSTATETYPE::$host.state_type$\tSERVICESTATE::$service.state
↪$\tSERVICESTATETYPE::$service.state_type$
}
```

Note that compared to Nagios format templates, we removed the `$HOSTOUTPUT$` and `$SERVICEOUTPUT$`. Icinga has a slightly different service output handling and its `SERVICEOUTPUT` actually mix both the equivalent of `SERVICEOUTPUT` and `SERVICELONGOUTPUT` from Nagios format, which may break the parsing.

Icinga2 has different macros names from Nagios. For a complete list see documentation.

Note: Beware: the perfdata files can accumulate very quickly if not consumed by the `nagios_dispatcher` script.

Create the dispatcher configuration file:

```
root:~# mkdir -p /usr/local/etc/
root:~# cat <<EOF > /usr/local/etc/nagios_dispatcher.conf
daemon=1
```

(continues on next page)

(continued from previous page)

```
directory=/var/lib/nagios3/spool/perfdata/
frequency=5
db_connection_string=dbi:Pg:dbname=opm host=127.0.0.1
db_user=YOUR_USER
db_password=YOUR_PASS
debug=0
syslog=1
hostname_filter = /^$/ # Empty hostname. Never happens
service_filter = /^$/ # Empty service
label_filter = /^$/ # Empty label
EOF

root:~# chown nagios /usr/local/etc/nagios_dispatcher.conf
```

Note: With our previous examples, **db_user** would've been set to `opm_dispatcher` and **db_password** should be set to `anothergoodpassword`. If you use Icinga2, `directory` must be set to `/var/spool/icinga2/perfdata/` (Icinga2 default

directory for perfdata).

Install the `nagios_dispatcher.pl` file into `/usr/local/bin/`

```
root:~# cp /usr/local/src/opm/wh_nagios/bin/nagios_dispatcher.pl /usr/local/bin
```

You can test that it works using the command line (you may have to set `daemon=0`):

```
/usr/local/bin/nagios_dispatcher.pl --verbose -c /usr/local/etc/nagios_dispatcher.conf
```

The files in the Nagios `perfdata` directory must disappear one by one.

If your operating system uses systemd

In `nagios_dispatcher.conf` you must set `daemon` to 0 and modify the connection string. The full file becomes:

```
daemon=0
directory=/var/lib/nagios3/spool/perfdata/
frequency=5
db_connection_string=dbi:Pg:dbname=opm;host=127.0.0.1
db_user=YOUR_USER
db_password=YOUR_PASS
debug=0
syslog=1
hostname_filter = /^$/ # Empty hostname. Never happens
service_filter = /^$/ # Empty service
label_filter = /^$/ # Empty label
```

Create the file `/etc/systemd/system/nagios_dispatcher.service` with the following content:

```
[Unit]
Description=Nagios Dispatcher Service
After=network.target

[Service]
Type=simple
```

(continues on next page)

(continued from previous page)

```
User=nagios
ExecStart=/usr/local/bin/nagios_dispatcher.pl -c /usr/local/etc/nagios_dispatcher.conf
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

Now enable and start the service:

```
systemctl enable nagios_dispatcher
systemctl start nagios_dispatcher
```

Check that the `nagios_dispatcher` process shows up in the process list.

If your operating system uses inittab

Add the following line at the end of the `/etc/inittab` file:

```
d1:23:respawn:/usr/bin/perl -w /usr/local/bin/nagios_dispatcher.pl --daemon --config /
↪usr/local/etc/nagios_dispatcher.conf
```

and reload it:

```
root:~# init q
```

If your operating system uses upstart

Create the file `/etc/init/nagios_dispatcher.conf`, with the following content:

```
# This service maintains nagios_dispatcher

start on stopped rc RUNLEVEL=[2345]
stop on starting runlevel [016]

respawn
exec /usr/local/bin/nagios_dispatcher.pl -c /usr/local/etc/nagios_dispatcher.conf
```

and start the job:

```
root:~# initctl start nagios_dispatcher
```

User interface

The default user interface is based on the web framework Mojolicious. You need to install:

- Perl (5.10 or above)
- Mojolicious (4.63 or above, **less than 5.0 !**)
- Mojolicious::Plugin::I18N (version 0.9)
- DBD::Pg perl module
- PostgreSQL (9.3 or above)
- A CGI/Perl webserver

You can install “Mojolicious” with your Linux distribution package system if old enough packages of Mojolicious are available.

Another option is from CPAN:

```
curl -L cpanmin.us | perl - Mojolicious@4.99
curl -L cpanmin.us | perl - Mojolicious::Plugin::I18N@0.9
curl -L cpanmin.us | perl - DBI
curl -L cpanmin.us | perl - DBD::Pg
```

Alternatively, you can download the required archives and install them manually:

```
wget http://backpan.perl.org/authors/id/S/SR/SRI/Mojolicious-4.99.tar.gz
tar xzf Mojolicious-4.99.tar.gz
cd Mojolicious-4.99
perl Makefile.PL
make
sudo make install
cd ..
wget http://backpan.perl.org/authors/id/S/SH/SHARIFULN/Mojolicious-Plugin-I18N-0.9.
↪tar.gz
tar xzf Mojolicious-Plugin-I18N-0.9.tar.gz
cd Mojolicious-Plugin-I18N-0.9
make
sudo make install
```

To install the UI plugin `wh_nagios` (or any other UI plugin), from your `opm` directory as user `root`:

```
root:/usr/local/src/opm# cd opm-core/ui/modules
root:/usr/local/src/opm/opm-core/ui/modules# ln -s /usr/local/src/opm/opm-wh_nagios/
↪ui wh_nagios
```

Then, on your OPM database side, you need to create another user for the UI:

```
postgres@opm=# CREATE USER opmui WITH ENCRYPTED PASSWORD 'yetanothergoodpassword';
postgres@opm=# SELECT * from grant_appli('opmui');
```

Finally, in `/usr/local/src/opm/opm-core/ui`, copy the **opm.conf-dist** file to **opm.conf**, and edit it to suit you needs, for instance:

```
{
    ...
    "database" : {
        "dbname"   : "opm",
        "host"     : "127.0.0.1",
        "port"     : "5432",
        "user"     : "opmui",
        "password" : "opmui"
    },
    ...
    "plugins" : [ "wh_nagios" ]
}
```

This user is only needed for the connection between the UI and the database. You only have to use it in the opm.conf file

To test the web user interface quickly, you can use either `morbo` or `hypnotoad`, both installed with Mojolicious. Example with Morbo:

```
user:/usr/local/src/opm/opm-core/ui/opm$ morbo ./script/opm
[Fri Nov 29 12:12:52 2013] [debug] Helper "url_for" already exists, replacing.
```

(continues on next page)

(continued from previous page)

```
[Fri Nov 29 12:12:52 2013] [debug] Reading config file "/home/ioguix/git/opm/ui/opm/
↪opm.conf".
[Fri Nov 29 12:12:53 2013] [info] Listening at "http://*:3000".
Server available at http://127.0.0.1:3000.
```

- Alternativeley, this example uses `hypnotoad`, which suits production better:

  ```
  user:/usr/local/src/opm-core/ui/opm$ hypnotoad -f ./script/opm
  ```

Note: Removing "-f" makes it daemonize.

Note: If you want to change default listen port when using hypnotoad, you can edit the *opm.conf* file to specify listen port or any hypnotoad related parameter, like this:

```
"hypnotoad" : {
    "listen" : ["http://*:6666"],
    "worker" : 10
},
...,
```

- To configure nginx to forward requests to a `hypnotoad` application server:

  ```
  upstream hypnotoad {
    server 127.0.0.1:8080;
  }

  server {
    listen 80;

    location / {
          proxy_pass http://hypnotoad;
          proxy_set_header Host $host;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto "http";
    }
  }
  ```

Note: You should ensure that hypnotoad starts on boot, e.g. in **/etc/rc.local**:

```
su - www-data -c 'hypnotoad /var/www/opm-core/ui/script/opm'
```

If you want to use Apache, here is a quick configuration sample using CGI:

```
<VirtualHost *:80>
        ServerAdmin webmaster@example.com
        ServerName opm.example.com
        DocumentRoot /var/www/opm/public/

        <Directory /var/www/opm/public/>
                AllowOverride None
```

(continues on next page)

(continued from previous page)

```
                Order allow,deny
                allow from all
                IndexIgnore *

                RewriteEngine On
                RewriteBase /
                RewriteRule ^$ opm.cgi [L]
                RewriteCond %{REQUEST_FILENAME} !-f
                RewriteCond %{REQUEST_FILENAME} !-d
                RewriteRule ^(.*)$ opm.cgi/$1 [L]
        </Directory>

        ScriptAlias /opm.cgi /var/www/opm/script/opm
        <Directory /var/www/opm/script/>
                AddHandler cgi-script .cgi
                Options +ExecCGI
                AllowOverride None
                Order allow,deny
                allow from all
                SetEnv MOJO_MODE production
                SetEnv MOJO_MAX_MESSAGE_SIZE 4294967296
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/opm.log
        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog ${APACHE_LOG_DIR}/opm.log combined
</VirtualHost>
```

(assuming that the directory **/usr/local/src/opm/opm-core/ui** has been symlinked to **/var/www/opm**).

For a complete list and specifications on supported http servers, please check the Mojolicious official documentation.

2.3 Probes

2.3.1 check_pgactivity

check_pgactivity - PostgreSQL plugin for Nagios

Contents

SYNOPSIS

```
check_pgactivity {-w|--warning THRESHOLD} {-c|--critical THRESHOLD} [-s|--service␣
↪SERVICE ] [-h|--host HOST] [-U|--username ROLE] [-p|--port PORT] [-d|--dbname␣
↪DATABASE] [-S|--dbservice SERVICE_NAME] [-P|--psql PATH] [--debug] [--status-file␣
↪FILE] [--path PATH] [-t|--timemout TIMEOUT]
check_pgactivity [-l|--list]
check_pgactivity [--help]
```

DESCRIPTION

check_pgactivity is designed to monitor PostgreSQL clusters from Nagios. It offers many options to measure and monitor useful performance metrics.

-s, **–service** SERVICE

The Nagios service to run. See section SERVICES for a description of available services or use `--list` for a short service and description list.

-h, **–host** HOST

Database server host or socket directory (default: $PGHOST or "localhost")

See section `CONNECTIONS` for more informations.

-U, **–username** ROLE

Database user name (default: $PGUSER or "postgres").

See section `CONNECTIONS` for more informations.

-p, **–port** PORT

Database server port (default: $PGPORT or "5432").

See section `CONNECTIONS` for more informations.

-d, **–dbname** DATABASE

Database name to connect to (default: $PGDATABASE or "template1").

WARNING! This is not necessarily one of the database that will be checked. See `--dbinclude` and `--dbexclude`.

See section `CONNECTIONS` for more informations.

-S, **–dbservice** SERVICE_NAME

The connection service name from pg_service.conf to use.

See section `CONNECTIONS` for more informations.

–dbexclude REGEXP

Some services automatically check all the databases of your cluster (note: that does not mean they always need to connect on all of them to check them though). `--dbexclude` excludes any database whose name matches the given Perl regular expression. Repeat this option as many time as needed.

See `--dbinclude` as well. If a database match both dbexclude and dbinclude arguments, it is excluded.

–dbinclude REGEXP

Some services automatically check all the databases of your cluster (note: that does not imply that they always need to connect to all of them though). Some always exclude the 'postgres' database and templates. `--dbinclude` checks **ONLY**databases whose names match the given Perl regular expression. Repeat this option as many time as needed.

See `--dbexclude` as well. If a database match both dbexclude and dbinclude arguments, it is excluded.

-w, **–warning** THRESHOLD

The Warning threshold.

-c, **–critical** THRESHOLD

The Critical threshold.

-F, **–format** OUTPUT_FORMAT

The output format. Supported output are: `binary`, `debug`, `human`, `nagios` and `nagios_strict`.

Using the `binary` format, the results are written in a binary file (using perl module `Storable`) given in argument `--output`. If no output is given, defaults to file `check_pgactivity.out` in the same directory as the script.

The `nagios_strict` format is equivalent to the `nagios` format. The only difference is that it enforces the unit follow the strict Nagios specs: B, c, s or %. Any unit absent from this list is dropped (Bps, Tps, etc).

–tmpdir DIRECTORY

Path to a directory where the script can create temporary files. The script relies on the system default temporary directory if possible.

-P, **–psql** FILE

Path to the `psql` executable (default: "psql").

–status-file PATH

Path to the file where service status information is kept between successive calls. Default is to save check_pgactivity.data in the same directory as the script.

–dump-status-file

Dump the content of the status file and exit. This is useful for debugging purpose.

–dump-bin-file [PATH]

Dump the content of the given binary file previously created using `--format binary`. If no path is given, defaults to file `check_pgactivity.out` in the same directory as the script.

-t, **–timeout** TIMEOUT

Timeout (default: "30s"), as raw (in seconds) or as an interval. This timeout will be used as `statement_timeout` for psql and URL timeout for `minor_version` service.

-l, **–list**

List available services.

-V, –version

Print version and exit.

–debug

Print some debug messages.

-?, –help

Show this help page.

THRESHOLDS

THRESHOLDS provided as warning and critical values can be raw numbers, percentages, intervals or sizes. Each available service supports one or more formats (eg. a size and a percentage).

Percentage

If THRESHOLD is a percentage, the value should end with a '%' (no space). For instance: 95%.

Interval

If THRESHOLD is an interval, the following units are accepted (not case sensitive): s (second), m (minute), h (hour), d (day). You can use more than one unit per given value. If not set, the last unit is in seconds. For instance: "1h 55m 6" = "1h55m6s".

Size

If THRESHOLD is a size, the following units are accepted (not case sensitive): b (Byte), k (KB), m (MB), g (GB), t (TB), p (PB), e (EB) or Z (ZB). Only integers are accepted. Eg. `1.5MB` will be refused, use `1500kB`.

The factor between units is 1024 bytes. Eg. `1g = 1G = 1024\*1024\*1024.`

CONNECTIONS

check_pgactivity allows two different connection specifications: by service or by specifying values for host, user, port, and database. Some services can run on multiple hosts, or needs to connect to multiple hosts.

You might specify one of the parameters below to connect to your PostgreSQL instance. If you don't, no connection parameters are given to psql: connection relies on binary defaults and environment.

The format for connection parameters is:

Parameter `--dbservice SERVICE_NAME`

Define a new host using the given service. Multiple hosts can be defined by listing multiple services separated by a comma. Eg.

```
--dbservice service1,service2
```

Parameters `--host HOST`, `--port PORT`, `--user ROLE` or `--dbname DATABASE`

One parameter is enough to define a new host. Usual environment variables (PGHOST, PGPORT, PGDATABASE, PGUSER, PGSERVICE) or default values are used for missing parameters.

If multiple values are given, define as many host as maximum given values.

Values are associated by position. Eg.:

```
--host h1,h2 --port 5432,5433
```

Means "host=h1 port=5432" and "host=h2 port=5433".

If the number of values is different between parameters, any host missing a parameter will use the first given value for this parameter. Eg.:

```
--host h1,h2 --port 5433
```

Means: "host=h1 port=5433" and "host=h2 port=5433".

Services are defined first

For instance:

```
--dbservice s1 --host h1 --port 5433
```

means: use "service=s1" and "host=h1 port=5433" in this order. If the service supports only one host, the second host is ignored.

Mutual exclusion between both methods

You can not overwrite services connections variables with parameters `--host HOST`, `--port PORT`, `--user ROLE` or `--dbname DATABASE`

SERVICES

Descriptions and parameters of available services.

archive_folder

Check if all archived WALs exist between the oldest and the latest WAL in the archive folder and make sure they are 16MB. The given folder must have archived files from ONE cluster. The version of PostgreSQL that created the archives is only checked on the last one, for performance consideration.

This service requires the argument `--path` on the command line to specify the archive folder path to check. Obviously, it must have access to this folder at the filesystem level: you may have to execute it on the archiving server rather than on the PostgreSQL instance.

The optional argument `--suffix` defines the suffix of your archived WALs; this is useful for compressed WALs (eg. .gz, .bz2, . . .). Default is no suffix.

This service needs to read the header of one of the archives to define how many segments a WAL owns. Check_pgactivity automatically handles files with extensions .gz, .bz2, .xz, .zip or .7z using the following commands:

```
gzip -dc
bzip2 -dc
xz -dc
unzip -qqp
7z x -so
```

If needed, provide your own command that writes the uncompressed file to standard output with the `--unarchiver` argument.

Optional argument `--ignore-wal-size` skips the WAL size check. This is useful if your archived WALs are compressed and check_pgactivity is unable to guess the original size. Here are the commands check_pgactivity uses to guess the original size of .gz, .xz or .zip files:

```
gzip -ql
xz -ql
unzip -qql
```

Default behaviour is to check the WALs size.

Perfdata contains the number of archived WALs and the age of the most recent one.

Critical and Warning define the max age of the latest archived WAL as an interval (eg. 5m or 300s).

Required privileges: unprivileged role; the system user needs read access to archived WAL files.

Sample commands:

```
check_pgactivity -s archive_folder --path /path/to/archives -w 15m -c 30m
check_pgactivity -s archive_folder --path /path/to/archives --suffix .gz -w␣
↪15m -c 30m
check_pgactivity -s archive_folder --path /path/to/archives --ignore-wal-
↪size --suffix .bz2 -w 15m -c 30m
check_pgactivity -s archive_folder --path /path/to/archives --unarchiver
↪"unrar p" --ignore-wal-size --suffix .rar -w 15m -c 30m
```

archiver (8.1+)

Check if the archiver is working properly and the number of WAL files ready to archive.

Perfdata returns the number of WAL files waiting to be archived.

Critical and Warning thresholds are optional. They apply on the number of files waiting to be archived. They only accept a raw number of files.

Whatever the given threshold, a critical alert is raised if the archiver process did not archive the oldest waiting WAL to be archived since last call.

Required privileges: unprivileged role (10+); superuser (<10).

autovacuum (8.1+)

Check the autovacuum activity on the cluster.

Perfdata contains the age of oldest running autovacuum and the number of workers by type (VACUUM, VACUUM ANALYZE, ANALYZE, VACUUM FREEZE).

Thresholds, if any, are ignored.

Required privileges: unprivileged role.

backends (all)

Check the total number of connections in the PostgreSQL cluster.

Perfdata contains the number of connections per database.

Critical and Warning thresholds accept either a raw number or a percentage (eg. 80%). When a threshold is a percentage, it is compared to the difference between the cluster parameters `max_connections` and `superuser_reserved_connections`.

Required privileges: an unprivileged user only sees its own queries; a pg_monitor (10+) or superuser (<10) role is required to see all queries.

backends_status (8.2+)

Check the status of all backends. Depending on your PostgreSQL version, statuses are: `idle`, `idle in transaction`, `idle in transaction (aborted)` (>=9.0 only), `fastpath function call`, `active`, `waiting for lock`, `undefined`, `disabled` and `insufficient` `privilege`. **insufficient privilege** appears when you are not allowed to see the statuses of other connections.

This service supports the argument `--exclude REGEX` to exclude queries matching the given regular expression.

You can use multiple `--exclude REGEX` arguments.

Critical and Warning thresholds are optional. They accept a list of 'status_label=value' separated by a comma. Available labels are `idle`, `idle_xact`, `aborted_xact`, `fastpath`, `active` and `waiting`. Values are raw numbers or time units and empty lists are forbidden. Here is an example:

```
-w 'waiting=5,idle_xact=10' -c 'waiting=20,idle_xact=30,active=1d'
```

Perfdata contains the number of backends for each status and the oldest one for each of them, for 8.2+.

Note that the number of backends reported in Nagios message **includes**excluded backends.

Required privileges: an unprivileged user only sees its own queries; a pg_monitor (10+) or superuser (<10) role is required to see all queries.

backup_label_age (8.1+)

Check the age of the backup label file.

Perfdata returns the age of the backup_label file, -1 if not present.

Critical and Warning thresholds only accept an interval (eg. 1h30m25s).

Required privileges: unprivileged role (9.3+); superuser (<9.3)

bgwriter (8.3+)

Check the percentage of pages written by backends since last check.

This service uses the status file (see `--status-file` parameter).

Perfdata contains the ratio per second for each `pg_stat_bgwriter` counter since last execution. Units Nps for checkpoints, max written clean and fsyncs are the number of "events" per second.

Critical and Warning thresholds are optional. If set, they *only* accept a percentage.

Required privileges: unprivileged role.

btree_bloat

Estimate bloat on B-tree indexes.

Warning and critical thresholds accept a comma-separated list of either raw number(for a size), size (eg. 125M) or percentage. The thresholds apply to **bloat** size, not object size. If a percentage is given, the threshold will apply to the bloat size compared to the total index size. If multiple threshold values are passed, check_pgactivity will choose the largest (bloat size) value.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

It also supports a `--exclude REGEX` parameter to exclude relations matching a regular expression. The regular expression applies to "database.schema_name.relation_name". This enables you to filter either on a relation name for all schemas and databases, on a qualified named relation (schema + relation) for all databases or on a qualified named relation in only one database.

You can use multiple `--exclude REGEX` parameters.

Perfdata will return the number of indexes of concern, by warning and critical threshold per database.

A list of the bloated indexes will be returned after the perfdata. This list contains the fully qualified bloated index name, the estimated bloat size, the index size and the bloat percentage.

Required privileges: superuser (<10) able to log in all databases, or at least those in `--dbinclude`; superuser (<10); on PostgreSQL 10+, a user with the role pg_monitor suffices, provided that you grant SELECT on the system table pg_statistic to the pg_monitor role, in each database of the cluster: `GRANT SELECT ON pg_statistic TO pg_monitor;`

commit_ratio (all)

Check the commit and rollback rate per second since last call.

This service uses the status file (see –status-file parameter).

Perfdata contains the commit rate, rollback rate, transaction rate and rollback ratio for each database since last call.

Critical and Warning thresholds are optional. They accept a list of comma separated 'label=value'. Available labels are **rollbacks**, **rollback_rate**and **rollback_ratio**, which will be compared to the number of rollbacks, the rollback rate and the rollback ratio of each database. Warning or critical will be raised if the reported value is greater than **rollbacks**, **rollback_rate**or **rollback_ratio**.

Required privileges: unprivileged role.

configuration (8.0+)

Check the most important settings.

Warning and Critical thresholds are ignored.

Specific parameters are : `--work_mem`, `--maintenance_work_mem`, `--shared_buffers`,`--wal_buffers`, `--checkpoint_segments`, `--effective_cache_size`, `--no_check_autovacuum`, `--no_check_fsync`, `--no_check_enable`, `--no_check_track_counts`.

Required privileges: unprivileged role.

connection (all)

Perform a simple connection test.

No perfdata is returned.

This service ignores critical and warning arguments.

Required privileges: unprivileged role.

custom_query (all)

Perform the given user query.

Specify the query with `--query`. The first column will be used to perform the test for the status if warning and critical are provided.

The warning and critical arguments are optional. They can be of format integer (default), size or time depending on the `--type` argument. Warning and Critical will be raised if they are greater than the first column, or less if the `--reverse` option is used.

All other columns will be used to generate the perfdata. Each field name is used as the name of the perfdata. The field value must contain your perfdata value and its unit appended to it. You can add as many fields as needed. Eg.:

```
SELECT pg_database_size('postgres'),
       pg_database_size('postgres')||'B' AS db_size
```

Required privileges: unprivileged role (depends on the query).

database_size (8.1+)

Check the variation of database sizes, and **return the size** of every databases.

This service uses the status file (see `--status-file` parameter).

Perfdata contains the size of each database.

Critical and Warning thresholds accept either a raw number, a percentage, or a size (eg. 2.5G). They are applied on the size difference for each database since the last execution. The aim is to detect unexpected database size variation.

This service supports both `--dbexclude` and `--dbinclude` parameters.

Required privileges: unprivileged role.

hit_ratio (all)

Check the cache hit ratio on the cluster.

This service uses the status file (see `--status-file` parameter).

Perfdata returns the cache hit ratio per database. Template databases and databases that do not allow connections will not be checked, nor will the databases which have never been accessed.

Critical and Warning thresholds are optional. They only accept a percentage.

This service supports both `--dbexclude` and `--dbinclude` parameters.

Required privileges: unprivileged role.

hot_standby_delta (9.0)

Check the data delta between a cluster and its hot standbys.

You must give the connection parameters for two or more clusters.

Perfdata returns the data delta in bytes between the master and each hot standby cluster listed.

Critical and Warning thresholds are optional. They can take one or two values separated by a comma. If only one value given, it applies to both received and replayed data. If two values are given, the first one applies to received data, the second one to replayed ones. These thresholds only accept a size (eg. 2.5G).

This service raises a Critical if it doesn't find exactly ONE valid master cluster (ie. critical when 0 or 2 and more masters).

Required privileges: unprivileged role.

is_hot_standby (9.0+)

Checks if the cluster is in recovery and accepts read only queries.

This service ignores critical and warning arguments.

No perfdata is returned.

Required privileges: unprivileged role.

is_master (all)

Checks if the cluster accepts read and/or write queries. This state is reported as "in production" by pg_controldata.

This service ignores critical and warning arguments.

No perfdata is returned.

Required privileges: unprivileged role.

invalid_indexes

Check if there is there are invalid indexes in a database.

A critical alert is raised if an invalid index is detected.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

This service supports a `--exclude REGEX` parameter to exclude indexes matching a regular expression. The regular expression applies to "database.schema_name.index_name". This enables you to filter either on a relation name for all schemas and databases, on a qualified named index (schema + index) for all databases or on a qualified named index in only one database.

You can use multiple `--exclude REGEX` parameters.

Perfdata will return the number of invalid indexes per database.

A list of invalid indexes will be returned after the perfdata. This list contains the fully qualified index name. If excluded index is set, the number of exclude indexes is returned.

Required privileges: unprivileged role able to log in all databases.

is_replay_paused (9.1+)

Checks if the replication is paused. The service will return UNKNOWN if executed on a master server.

Thresholds are optional. They must be specified as interval. OK will always be returned if the standby is not paused, even if replication delta time hits the thresholds.

Critical or warning are raised if last reported replayed timestamp is greater than given threshold AND some data received from the master are not applied yet. OK will always be returned if the standby is paused, or if the standby has already replayed everything from master and until some write activity happens on the master.

Perfdata returned: * paused status (0 no, 1 yes, NaN if master) * lag time (in second) * data delta with master (0 no, 1 yes)

Required privileges: unprivileged role.

last_analyze (8.2+)

Check on each databases that the oldest `analyze` (from autovacuum or not) is not older than the given threshold.

This service uses the status file (see `--status-file` parameter) with PostgreSQL 9.1+.

Perfdata returns oldest `analyze` per database in seconds. With PostgreSQL 9.1+, the number of [auto]analyses per database since last call is also returned.

Critical and Warning thresholds only accept an interval (eg. 1h30m25s) and apply to the oldest execution of analyse.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

Required privileges: unprivileged role able to log in all databases.

last_vacuum (8.2+)

Check that the oldest vacuum (from autovacuum or otherwise) in each database in the cluster is not older than the given threshold.

This service uses the status file (see `--status-file` parameter) with PostgreSQL 9.1+.

Perfdata returns oldest vacuum per database in seconds. With PostgreSQL 9.1+, it also returns the number of [auto]vacuums per database since last execution.

Critical and Warning thresholds only accept an interval (eg. 1h30m25s) and apply to the oldest vacuum.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

Required privileges: unprivileged role able to log in all databases.

locks (all)

Check the number of locks on the hosts.

Perfdata returns the number of locks, by type.

Critical and Warning thresholds accept either a raw number of locks or a percentage. For percentage, it is computed using the following limits for 7.4 to 8.1:

```
max_locks_per_transaction * max_connections
```

for 8.2+:

```
max_locks_per_transaction * (max_connections + max_prepared_transactions)
```

for 9.1+, regarding lockmode :

```
  max_locks_per_transaction * (max_connections + max_prepared_transactions)
or max_pred_locks_per_transaction * (max_connections + max_prepared_
↪transactions)
```

Required privileges: unprivileged role.

longest_query (all)

Check the longest running query in the cluster.

Perfdata contains the max/avg/min running time and the number of queries per database.

Critical and Warning thresholds only accept an interval.

This service supports both `--dbexclude` and `--dbinclude` parameters.

It also supports argument `--exclude REGEX` to exclude queries matching the given regular expression from the check.

Above 9.0, it also supports `--exclude REGEX` to filter out application_name.

You can use multiple `--exclude REGEX` parameters.

Required privileges: an unprivileged role only checks its own queries; a pg_monitor (10+) or superuser (<10) role is required to check all queries.

max_freeze_age (all)

Checks oldest database by transaction age.

Critical and Warning thresholds are optional. They accept either a raw number or percentage for PostgreSQL 8.2 and more. If percentage is given, the thresholds are computed based on the "autovacuum_freeze_max_age" parameter. 100% means that some table(s) reached the maximum age and will trigger an autovacuum freeze. Percentage thresholds should therefore be greater than 100%.

Even with no threshold, this service will raise a critical alert if a database has a negative age.

Perfdata returns the age of each database.

This service supports both `--dbexclude` and `--dbinclude` parameters.

Required privileges: unprivileged role.

minor_version (all)

Check if the cluster is running the most recent minor version of PostgreSQL.

Latest versions of PostgreSQL can be fetched from PostgreSQL official website if check_pgactivity has access to it, or must be given as a parameter.

Without `--critical` or `--warning` parameters, this service attempts to fetch the latest version numbers online. A critical alert is raised if the minor version is not the most recent.

You can optionally set the path to your prefered retrieval tool using the `--path` parameter (eg. `--path '/usr/bin/wget'`). Supported programs are: GET, wget, curl, fetch, lynx, links, links2.

If you do not want to (or cannot) query the PostgreSQL website, provide the expected versions using either `--warning` OR `--critical`, depending on which return value you want to raise.

The given string must contain one or more MINOR versions separated by anything but a '.'. For instance, the following parameters are all equivalent:

```
--critical "10.1 9.6.6 9.5.10 9.4.15 9.3.20 9.2.24 9.1.24 9.0.23 8.4.22"
--critical "10.1, 9.6.6, 9.5.10, 9.4.15, 9.3.20, 9.2.24, 9.1.24, 9.0.23, 8.4.
↪22"
--critical "10.1,9.6.6,9.5.10,9.4.15,9.3.20,9.2.24,9.1.24,9.0.23,8.4.22"
--critical "10.1/9.6.6/9.5.10/9.4.15/9.3.20/9.2.24/9.1.24/9.0.23/8.4.22"
```

Any other value than 3 numbers separated by dots (before version 10.x) or 2 numbers separated by dots (version 10 and above) will be ignored. If the running PostgreSQL major version is not found, the service raises an unknown status.

Perfdata returns the numerical version of PostgreSQL.

Required privileges: unprivileged role; access to http://www.postgresql.org required to download version numbers.

oldest_2pc (8.1+)

Check the oldest *two-phase commit transaction* (aka. prepared transaction) in the cluster.

Perfdata contains the max/avg age time and the number of prepared transactions per databases.

Critical and Warning thresholds only accept an interval.

Required privileges: unprivileged role.

oldest_idlexact (8.3+)

Check the oldest *idle* transaction.

Perfdata contains the max/avg age and the number of idle transactions per databases.

Critical and Warning thresholds only accept an interval.

This service supports both `--dbexclude` and `--dbinclude` parameters.

Above 9.2, it supports `--exclude` to filter out connections. Eg., to filter out pg_dump and pg_dumpall, set this to 'pg_dump,pg_dumpall'.

Required privileges: an unprivileged role checks only its own queries; a pg_monitor (10+) or superuser (<10) role is required to check all queries.

pg_dump_backup

Check the age and size of backups.

This service uses the status file (see `--status-file` parameter).

The `--path` argument contains the location to the backup folder. The supported format is a glob pattern matching every folder or file that you need to check. If appropriate, the probe should be run as a user with sufficient privileges to check for the existence of files.

The `--pattern` is required, and must contain a regular expression matching the backup file name, extracting the database name from the first matching group. For example, the pattern "(w+)-d+.dump" can be used to match dumps of the form:

```
mydb-20150803.dump
otherdb-20150803.dump
mydb-20150806.dump
otherdb-20150806.dump
mydb-20150807.dump
```

Optionally, a `--global-pattern` option can be supplied to check for an additional global file.

Tip [For compatibility with pg_back, you should use] `--path` '/path/*{dump,sql}' `--pattern` '(w+)_[0-9-_]+.dump' `--global-pattern` 'pg_global_[0-9-_]+.sql'

The `--critical` and `--warning` thresholds are optional. They accept a list of 'metric=value' separated by a comma. Available metrics are `oldest` and `newest`, respectively the age of the oldest and newest backups, and `size`, which must be the maximum variation of size since the last check, expressed as a size or a percentage. `mindeltasize`, expressed in B, is the minimum variation of size needed to raise an alert.

This service supports the `--dbinclude` and `--dbexclude` arguments, to respectively test for the presence of include or exclude files.

The argument `--exclude` enables you to exclude files younger than an interval. This is useful to ignore files from a backup in progress. Eg., if your backup process takes 2h, set this to '125m'.

Perfdata returns the age of the oldest and newest backups, as well as the size of the newest backups.

Required privileges: unprivileged role; the system user needs read access on the directory containing the dumps (but not on the dumps themselves).

pga_version

Check if this script is running the given version of check_pgactivity. You must provide the expected version using either `--warning` OR `--critical`.

No perfdata is returned.

Required privileges: none.

pgdata_permission (8.2+)

Check that the instance data directory rights are 700, and belongs to the system user currently running postgresql.

The check on rights works on all Unix systems.

Checking the user only works on Linux systems (it uses /proc to avoid dependencies). Before 9.3, you need to provide the expected owner using the `--uid` argument, or the owner will not be checked.

Required privileges: <11:superuser v11: user with pg_monitor or pg_read_all_setting

The system user must also be able to read the folder containing PGDATA: **the service has to be executed locally on the monitored server.**

replication_slots (9.4+)

Check the number of WAL files and pg_replslot files retained by each replication slots.

Perfdata returns the number of WAL and pg_replslot files that each replication slot has to keep. This service needs superuser privileges since v10 to obtain pg_replslot files. Unless replslot_files will be at 0.

Critical and Warning thresholds are optional. They accept either a raw number (for backward compatibility, only wal threshold will be used) or a list 'wal=value' and 'replslot=value'. Respectively number of kept wal files or number of files in pg_replslot for each slot.

Required privileges: <10: unprivileged role v10: unprivileged role, or superuser to monitor logical replication v11: unpriviledged user with GRANT EXECUTE on function pg_ls_dir(text)

Here is an example:

```
-w 'wal=50,replslot=20' -c 'wal=100,replslot=40'
```

settings (9.0+)

Check if the current settings have changed since they were stored in the service file.

The "known" settings are recorded during the very first call of the service. To update the known settings after a configuration change, call this service again with the argument `--save`.

No perfdata.

Critical and Warning thresholds are ignored.

A Critical is raised if at least one parameter changed.

Required privileges: unprivileged role.

sequences_exhausted (7.4+)

Check all sequences assigned to a column (the smallserial, serial and bigserial types), and raise an alarm if the column or sequences gets too close to the maximum value.

Perfdata returns the sequences that trigger the alert.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

Critical and Warning thresholds accept a percentage of the sequence filled.

Required privileges: unprivileged role able to log in all databases

stat_snapshot_age (9.5+)

Check the age of the statistics snapshot (statistics collector's statistics). This probe helps to detect a frozen stats collector process.

Perfdata returns the statistics snapshot age.

Critical and Warning thresholds accept a raw number of seconds.

Required privileges: unprivileged role.

streaming_delta (9.1+)

Check the data delta between a cluster and its standbys in streaming replication.

Optional argument `--slave` allows you to specify some slaves that MUST be connected. This argument can be used as many times as desired to check multiple slave connections, or you can specify multiple slaves connections at one time, using comma separated values. Both methods can be used in a single call. The provided values must be of the form "APPLICATION_NAME IP". Both following examples will check for the presence of two slaves:

```
--slave 'slave1 192.168.1.11' --slave 'slave2 192.168.1.12'
--slave 'slave1 192.168.1.11','slave2 192.168.1.12'
```

This service supports a `--exclude REGEX` parameter to exclude every result matching a regular expression on application_name or IP address fields.

You can use multiple `--exclude REGEX` parameters.

Perfdata returns the data delta in bytes between the master and every standbies found, the number of standbies connected and the number of excluded standbies.

Critical and Warning thresholds are optional. They can take one or two values separated by a comma. If only one value is supplied, it applies to both flushed and replayed data. If two values are supplied, the first one applies to flushed data, the second one to replayed data. These thresholds only accept a size (eg. 2.5G).

Required privileges: unprivileged role.

table_unlogged (9.5+)

Check if tables are changed to unlogged. In 9.5, you can switch between logged and unlogged.

Without `--critical` or `--warning` parameters, this service attempts to fetch all unlogged tables.

A critical alert is raised if an unlogged table is detected.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

This service supports a `--exclude REGEX` parameter to exclude relations matching a regular expression. The regular expression applies to "database.schema_name.relation_name". This enables you to filter either on a relation name for all schemas and databases, on a qualified named relation (schema + relation) for all databases or on a qualified named relation in only one database.

You can use multiple `--exclude REGEX` parameters.

Perfdata will return the number of unlogged tables per database.

A list of the unlogged tables will be returned after the perfdata. This list contains the fully qualified table name. If `--exclude REGEX` is set, the number of excluded tables is returned.

Required privileges: unprivileged role able to log in all databases, or at least those in `--dbinclude`.

table_bloat

Estimate bloat on tables.

Warning and critical thresholds accept a comma-separated list of either raw number(for a size), size (eg. 125M) or percentage. The thresholds apply to **bloat** size, not object size. If a percentage is given, the threshold will apply to the bloat size compared to the table + TOAST size. If multiple threshold values are passed, check_pgactivity will choose the largest (bloat size) value.

This service supports both `--dbexclude` and `--dbinclude` parameters. The 'postgres' database and templates are always excluded.

This service supports a `--exclude REGEX` parameter to exclude relations matching the given regular expression. The regular expression applies to "database.schema_name.relation_name". This enables you to filter either on a relation name for all schemas and databases, on a qualified named relation (schema + relation) for all databases or on a qualified named relation in only one database.

You can use multiple `--exclude REGEX` parameters.

Warning: With a non-superuser role, this service can only check the tables that the given role is granted to read!

Perfdata will return the number of tables matching the warning and critical thresholds, per database.

A list of the bloated tables will be returned after the perfdata. This list contains the fully qualified bloated table name, the estimated bloat size, the table size and the bloat percentage.

Required privileges: superuser (<10) able to log in all databases, or at least those in `--dbinclude`; superuser (<10); on PostgreSQL 10+, a user with the role pg_monitor suffices, provided that you grant SELECT on the system table pg_statistic to the pg_monitor role, in each database of the cluster: `GRANT SELECT ON pg_statistic TO pg_monitor;`

temp_files (8.1+)

Check the number and size of temp files.

This service uses the status file (see `--status-file` parameter) for 9.2+.

Perfdata returns the number and total size of temp files found in `pgsql_tmp` folders. They are aggregated by database until 8.2, then by tablespace (see GUC temp_tablespaces).

Starting with 9.2, perfdata returns as well the number of temp files per database since last run, the total size of temp files per database since last run and the rate at which temp files were generated.

Critical and Warning thresholds are optional. They accept either a number of file (raw value), a size (unit is **mandatory** to define a size) or both values separated by a comma.

Thresholds are applied on current temp files being created AND the number/size of temp files created since last execution.

Required privileges: <10: superuser v10: an unprivileged role is possible but it will not monitor databases

that it cannot access, nor live temp files v11: an unprivileged role is possible but must be granted EXECUTE

on functions pg_ls_dir(text), pg_read_file(text), pg_stat_file(text); the same restrictions than on v10 will still apply

uptime (8.1+)

Returns time since postmaster start (“uptime”, from 8.1), since configuration reload (from 8.4), and since shared memory initialization (from 10).

Please note that the uptime is unaffected when the postmaster resets all its children (for example after a kill -9 on a process or a failure).

From 10+, the ‘time since shared memory init’ aims at detecting this situation: in fact we use the age of the oldest non-client child process (usually checkpointer, writer or startup). This needs pg_monitor access to read pg_stat_activity.

Critical and Warning thresholds are optional. If both are set, Critical is raised when the postmaster uptime or the time since shared memory initialization is less than the critical threshold.

Warning is raised when the time since configuration reload is less than the warning threshold. If only a warning or critical threshold is given, it will be used for both cases. Obviously these alerts will disappear from themselves once enough time has passed.

Perfdata contain the three values (when available).

Required privileges: pg_monitor on PG10+; otherwise unprivileged role.

wal_files (8.1+)

Check the number of WAL files.

Perfdata returns the total number of WAL files, current number of written WAL, the current number of recycled WAL, the rate of WAL written to disk since the last execution on the master cluster and the current timeline.

Critical and Warning thresholds accept either a raw number of files or a percentage. In case of percentage, the limit is computed based on:

```
100% = 1 + checkpoint_segments * (2 + checkpoint_completion_target)
```

For PostgreSQL 8.1 and 8.2:

```
100% = 1 + checkpoint_segments * 2
```

If `wal_keep_segments` is set for 9.0 to 9.4, the limit is the greatest of the following formulas:

```
100% = 1 + checkpoint_segments * (2 + checkpoint_completion_target)
100% = 1 + wal_keep_segments + 2 * checkpoint_segments
```

For 9.5 and above, the limit is:

```
100% =  max_wal_size      (as a number of WAL)
      + wal_keep_segments (if set)
```

Required privileges: <10:superuser (<10) v10:unprivileged user with pg_monitor v11:unprivileged user with pg_monitor, or with grant EXECUTE on function

pg_ls_waldir

EXAMPLES

Execute service "last_vacuum" on host "host=localhost port=5432":

```
check_pgactivity -h localhost -p 5432 -s last_vacuum -w 30m -c 1h30m
```

Execute service "hot_standby_delta" between hosts "service=pg92" and "service=pg92s":

```
check_pgactivity --dbservice pg92,pg92s --service hot_standby_delta -w 32MB -
↪c 160MB
```

Execute service "streaming_delta" on host "service=pg92" to check its slave "stby1" with the IP address "192.168.1.11":

```
check_pgactivity --dbservice pg92 --slave "stby1 192.168.1.11" --service␣
↪streaming_delta -w 32MB -c 160MB
```

Execute service "hit_ratio" on host "slave" port "5433, excluding database matching the regexps "idelone" and "(?i:sleep)":

```
check_pgactivity -p 5433 -h slave --service hit_ratio --dbexclude idelone --
↪dbexclude "(?i:sleep)" -w 90% -c 80%
```

Execute service "hit_ratio" on host "slave" port "5433, only for databases matching the regexp "importantone":

```
check_pgactivity -p 5433 -h slave --service hit_ratio --dbinclude␣
↪importantone -w 90% -c 80%
```

VERSION

check_pgactivity version 2.4, released on Wed Jan 30 2019

LICENSING

This program is open source, licensed under the PostgreSQL license. For license terms, see the LICENSE provided with the sources.

AUTHORS

Author: Open PostgreSQL Monitoring Development Group Copyright: (C) 2012-2019 Open PostgreSQL Monitoring Development Group

2.4 Developers

2.4.1 OPM Development Group (OPMDG)

The Open PostgreSQL Monitoring Development Group (OPMDG) is an international, unincorporated association of individuals and companies who have contributed to the OPM project.

The right to modify the official code base and accept contributions ('pull requests') is hold by a group called the "OPM Committers". The current team of committers is listed below:

- Julien Rouhaud: https://github.com/rjuju
- Jehan-Guillaume De Rorthais: https://github.com/ioguix
- Thomas Reiss: https://github.com/frost242

The OPM Committers generally act as spokespeople for the OPMDG.

Contributors to OPM are selected to be committers based on the following loose criteria:

- several substantial contributions to the project
- responsibility for maintenance of one or more areas of the codebase
- track record of reviewing and helping other contributors with their patches
- high quality code contributions which require very little revision or correction for commit
- demonstrated understanding of the process and criteria for patch acceptance

Committers who have become inactive and have not contributed significantly to the OPM project in several months can be removed as committers.

2.4.2 Development Information

The OPM project is open to any productive contribution.

Here's a few links, if you want to help us build a better tool:

- for anything related to the core, graphs, UI. . . :

https://github.com/OPMDG/opm-core/issues

- for anything related to the Nagios perfdata handling in the UI:

https://github.com/OPMDG/opm-wh_nagios/issues

- for anything related to the check_pgactivity probe:

https://github.com/OPMDG/check_pgactivity/issues

- The code is hosted on github. Feel free to clone our repos and send Pull Resquests. The github organization is:

https://github.com/OPMDG

2.4.3 Warehouses

Overview

A warehouse is used to store a kind of data. In version 1 and 2, the only available warehouse is wh_nagios, which stores Nagios' perfdata.

Each warehouse must have a unique name, lowercase, with a leading **wh_**, and it's own schema, named as the warehouse (with the leading **wh_**). All objects have to be in this schema, and should probably be configured to be dumped.

Content

A warehouse should at least provide a **pg** subdirectory containing a PostgreSQL extension depending on **opm-core** extension. It can also provide an **ui** subdirectory if the warehouse wants to provide some ui content. Then, it can also provides various subdirectories for its need. For instance, *wh_nagios* warehouse provides a **bin** subdirectory containing the *nagios_dispatcher* tool.

Therefore, a typical warehouse structure would be:

```
wh_my_warehouse
 \_ pg
 \_ ui
```

Implementing the PostgreSQL extension

ACL don't have to be handled by the warehouse, as the only regular database access should be done by the ui. The ACL are handled by the opm_core extension. Only a few tables and stored functions have to be implemented (see below).

In order to integrate with the opm_core module, the warehouse extension has to implement at least some objects.

Tables

- **services**:

A table that inherits public.services and its constraints, which will store every needed information for a service within the warehouse. A typical declaration will look like:

```
CREATE TABLE wh_name.services (
  useful_col     datatype,
  ...
  PRIMARY KEY (id),
  FOREIGN KEY (id_server) REFERENCES public.servers (id) ON UPDATE CASCADE ON DELETE␣
↪CASCADE),
```

(continues on next page)

(continued from previous page)

```
  UNIQUE (id_server, service)
) INHERITS (public.services);
SELECT pg_catalog.pg_extension_config_dump('wh_name.services', '') ;
```

- **metrics**:

A table that inherits public.metrics and its constraints, which will store every metrics (label information on all graphs) for every service within the warehouse. A typical declaration will look like:

```
CREATE TABLE wh_name.metrics (
    useful_col         datatype,
    ...
    PRIMARY KEY (id),
    FOREIGN KEY (id_service) REFERENCES wh_name.services (id) MATCH FULL ON DELETE␣
↪CASCADE ON UPDATE CASCADE
)
INHERITS (public.metrics);
SELECT pg_catalog.pg_extension_config_dump('wh_name.metrics', '') ;
```

- **series**:

A table that inherits public.series and its constraints, which will store association between metrics and graphs for every service within the warehouse. A typical declaration will look like:

```
CREATE TABLE wh_nagios.series (
    FOREIGN KEY (id_graph)  REFERENCES public.graphs (id) MATCH FULL ON DELETE␣
↪CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (id_metric) REFERENCES wh_name.metrics (id) MATCH FULL ON DELETE␣
↪CASCADE ON UPDATE CASCADE
)
INHERITS (public.series);
CREATE UNIQUE INDEX ON wh_name.series (id_metric, id_graph);
CREATE INDEX ON wh_name.series (id_graph);
SELECT pg_catalog.pg_extension_config_dump('wh_name.series', '') ;
```

Stored functions

- **get_metric_data((id_metric bigint, timet_begin timestamp with time zone, timet_end timestamp with time zone) RETURNS TABLE (timet timestamp with time zone, value numeric)**:

Function that will be called by opm_core when displaying a graph. It should return all timestamped stored value for a specific metric within the specified interval. The data don't need to be ordered by the timestamp.

- **grant_dispatcher(p_rolname text) RETURNS TABLE (operat text, approle name, appright text, objtype text, objname text)**:

Function that will be called by opm_core, when granting the right a role to dispatch data. Is must return the list of all objects granted. This is meant to grant CONNECT, USAGE, INSERT… permission on the warehouse's objects that store data.

- **revoke_dispatcher(p_rolname text) RETURNS TABLE (operat text, approle name, appright text, objtype text, objname text)**:

Function that will be called by opm_core, when revoking from a role to dispatch data. This function is the exact opposite of **grant_dispatcher**, GRANT being replaced with REVOKE.

- **purge_service(VARIADIC bigint[]) RETURNS bigint**:

Function that will purge data according to the related **servalid** interval. It must return the number of services actually purged.

And optionally:

- **cleanup_service(id_service bigint)**:

This function won't be called by the core module. Each warehouse has to handle his way of cleaning data (if needed). It has to update the warehouse's **services.last_cleanup** column when executed.

2.5 LICENSE

CHAPTER 3

Indices and tables

- genindex
- modindex
- search