--- title: "PgBouncer" draft: false ---

pgBouncer - A Lightweight Connection Pooler for PostgreSQL

PgBouncer

Lightweight connection pooler for PostgreSQL.

Homepage
    https://pgbouncer.github.io
Sources, bugtracking
    https://github.com/pgbouncer/pgbouncer

Building

PgBouncer depends on few things to get compiled:

- GNU Make 3.81+
- libevent 2.0
- (optional) OpenSSL 1.0.1 for TLS support.
- (optional) c-ares as alternative to libevent's evdns.

When dependencies are installed just run:

```
$ ./configure --prefix=/usr/local --with-libevent=libevent-prefix
$ make
$ make install
```

If you are building from git, or are building for Windows, please see separate build instructions below.

DNS lookup support

Starting from PgBouncer 1.4, it does hostname lookups at connect time instead just once at config load time. This requires proper async DNS implementation. Following list shows supported backends and their probing order:

backend	parallel	EDNS0 (1)	/etc/hosts	SOA lookup (2)	note
c-ares	yes	yes	yes	yes	ipv6+CNAME buggy in <=1.10
udns	yes	yes	no	yes	ipv4-only
evdns, libevent 2.x	yes	no	yes	no	does not check /etc/hosts updates
getaddrinfo_a, glibc 2.9+	yes	yes (3)	yes	no	N/A on non-linux
getaddrinfo, libc	no	yes (3)	yes	no	N/A on win32, requires pthreads
evdns, libevent 1.x	yes	no	no	no	buggy

1. EDNS0 is required to have more than 8 addresses behind one hostname.
2. SOA lookup is needed to re-check hostnames on zone serial change
3. To enable EDNS0, add *options edns0* to /etc/resolv.conf

./configure also has flags *--enable-evdns* and *--disable-evdns* which turn off automatic probing and force use of either *evdns* or *getaddrinfo_a()* .

PAM authorization

To enable PAM authorization *./configure* has a flag *--with-pam* (default value is no). When compiled with PAM support new global authorization type *pam* appears which can be used to validate users through PAM.

Building from GIT

Building PgBouncer from GIT requires that you fetch libusual submodule and generate the header and config files before you can run configure:

```
$ git clone https://github.com/pgbouncer/pgbouncer.git
$ cd pgbouncer
$ git submodule init
$ git submodule update
$ ./autogen.sh
$ ./configure ...
$ make
$ make install
```

Additional packages required: autoconf, automake, libevent-dev, libtool, autoconf-archive, python-docutils, and pkg-config.

Building for WIN32

At the moment only build env tested is MINGW32 / MSYS. Cygwin and Visual $ANYTHING are untested. Libevent 2.x is required for DNS hostname lookup.

Then do the usual:

```
$ ./configure ...
$ make
```

If cross-compiling from Unix:

```
$ ./configure --host=i586-mingw32msvc ...
```

Running on WIN32

Running from command-line goes as usual, except -d (daemonize), -R (reboot) and -u (switch user) switches will not work.

To run pgbouncer as a Windows service, you need to configure *service_name* parameter to set name for service. Then:

```
$ pgbouncer -regservice config.ini
```

To uninstall service:

```
$ pgbouncer -unregservice config.ini
```

To use Windows Event Log, set "syslog = 1" in config file. But before you need to register pgbevent.dll:

```
$ regsvr32 pgbevent.dll
```

To unregister it, do:

```
$ regsvr32 /u pgbevent.dll
```

--- title: "Usage" draft: false ---

Synopsis

```
pgbouncer [-d][-R][-v][-u user]
pgbouncer -V|-h
```

On Windows computers, the options are:

```
pgbouncer.exe [-v][-u user]
pgbouncer.exe -V|-h
```

Additional options for setting up a Windows service:

```
pgbouncer.exe --regservice
pgbouncer.exe --unregservice
```

DESCRIPTION

pgbouncer is a PostgreSQL connection pooler. Any target application can be connected to **pgbouncer** as if it were a PostgreSQL server, and **pgbouncer** will create a connection to the actual server, or it will reuse one of its existing connections.

The aim of **pgbouncer** is to lower the performance impact of opening new connections to PostgreSQL.

In order not to compromise transaction semantics for connection pooling, **pgbouncer** supports several types of pooling when rotating connections:

Session pooling
: Most polite method. When client connects, a server connection will be assigned to it for the whole duration the client stays connected. When the client disconnects, the server connection will be put back into the pool. This is the default method.

Transaction pooling
: A server connection is assigned to client only during a transaction. When PgBouncer notices that transaction is over, the server connection will be put back into the pool.

Statement pooling
: Most aggressive method. The server connection will be put back into pool immediately after a query completes. Multi-statement transactions are disallowed in this mode as they would break.

The administration interface of **pgbouncer** consists of some new `SHOW` commands available when connected to a special 'virtual' database **pgbouncer** .

Quick-start

Basic setup and usage as following.

1. Create a pgbouncer.ini file. Details in **pgbouncer(5)** . Simple example:

   ```
   [databases]
   template1 = host=127.0.0.1 port=5432 dbname=template1

   [pgbouncer]
   listen_port = 6543
   listen_addr = 127.0.0.1
   auth_type = md5
   auth_file = users.txt
   logfile = pgbouncer.log
   pidfile = pgbouncer.pid
   admin_users = someuser
   ```

2. Create `users.txt` file that contains users allowed in:

   ```
   "someuser" "same_password_as_in_server"
   ```

3. Launch **pgbouncer** :

   ```
   $ pgbouncer -d pgbouncer.ini
   ```

4. Have your application (or the **psql** client) connect to **pgbouncer** instead of directly to PostgreSQL server:

```
$ psql -p 6543 -U someuser template1
```

5. Manage **pgbouncer** by connecting to the special administration database **pgbouncer** and issuing `show help;` to begin:

```
$ psql -p 6543 -U someuser pgbouncer
pgbouncer=# show help;
NOTICE:  Console usage
DETAIL:
  SHOW [HELP|CONFIG|DATABASES|FDS|POOLS|CLIENTS|SERVERS|SOCKETS|LISTS|VERSION]
  SET key = arg
  RELOAD
  PAUSE
  SUSPEND
  RESUME
  SHUTDOWN
```

6. If you made changes to the pgbouncer.ini file, you can reload it with:

```
pgbouncer=# RELOAD;
```

Command line switches

-d	Run in background. Without it the process will run in foreground. Note: Does not work on Windows, **pgbouncer** need to run as service there.
-R	Do an online restart. That means connecting to the running process, loading the open sockets from it, and then using them. If there is no active process, boot normally. Note: Works only if OS supports Unix sockets and the *unix_socket_dir* is not disabled in config. Does not work on Windows machines. Does not work with TLS connections, they are dropped.
-u *user*	Switch to the given user on startup.
-v	Increase verbosity. Can be used multiple times.
-q	Be quiet - do not log to stdout. Note this does not affect logging verbosity, only that stdout is not to be used. For use in init.d scripts.
-V	Show version.
-h	Show short help.
--regservice	Win32: Register pgbouncer to run as Windows service. The **service_name** config parameter value is used as name to register under.
--unregservice	Win32: Unregister Windows service.

Admin console

The console is available by connecting as normal to the database **pgbouncer** :

```
$ psql -p 6543 pgbouncer
```

Only users listed in configuration parameters **admin_users** or **stats_users** are allowed to login to the console. (Except when *auth_type=any* , then any user is allowed in as a stats_user.)

Additionally, the username **pgbouncer** is allowed to log in without password, if the login comes via Unix socket and the client has same Unix user UID as the running process.

Show commands

The **SHOW** commands output information. Each command is described below.

SHOW STATS;

Shows statistics.

database
: Statistics are presented per database.

total_xact_count
: Total number of SQL transactions pooled by **pgbouncer** .

total_query_count
: Total number of SQL queries pooled by **pgbouncer** .

total_received
: Total volume in bytes of network traffic received by **pgbouncer** .

total_sent
: Total volume in bytes of network traffic sent by **pgbouncer** .

total_xact_time
: Total number of microseconds spent by **pgbouncer** when connected to PostgreSQL in a transaction, either idle in transaction or executing queries.

total_query_time
: Total number of microseconds spent by **pgbouncer** when actively connected to PostgreSQL, executing queries.

total_wait_time
: Time spent by clients waiting for a server in microseconds.

avg_xact_count
: Average transactions per second in last stat period.

avg_query_count
: Average queries per second in last stat period.

avg_recv
: Average received (from clients) bytes per second.

avg_sent
: Average sent (to clients) bytes per second.

avg_xact_time
: Average transaction duration in microseconds.

avg_query_time
: Average query duration in microseconds.

avg_wait_time
: Time spent by clients waiting for a server in microseconds (average per second).

SHOW STATS_TOTALS;

Subset of **SHOW STATS** showing the total values (**total_**).

SHOW STATS_AVERAGES;

Subset of **SHOW STATS** showing the average values (**avg_**).

SHOW SERVERS;

type
: S, for server.

user
: User name **pgbouncer** uses to connect to server.

database
: Database name.

state
: State of the pgbouncer server connection, one of **active** , **used** or **idle** .

addr
: IP address of PostgreSQL server.

port
: Port of PostgreSQL server.

local_addr
: Connection start address on local machine.

local_port
: Connection start port on local machine.

connect_time
: When the connection was made.

request_time
: When last request was issued.

ptr
: Address of internal object for this connection. Used as unique ID.

link
: Address of client connection the server is paired with.

remote_pid
: PID of backend server process. In case connection is made over Unix socket and OS supports getting process ID info, its OS PID. Otherwise it's extracted from cancel packet server sent, which should be PID in case server is PostgreSQL, but it's a random number in case server it is another PgBouncer.

SHOW CLIENTS;

type
: C, for client.

user
: Client connected user.

database
: Database name.

state
: State of the client connection, one of **active** , **used** , **waiting** or **idle** .

addr
: IP address of client.

port
: Port client is connected to.

local_addr
: Connection end address on local machine.

local_port
: Connection end port on local machine.

connect_time
: Timestamp of connect time.

request_time
: Timestamp of latest client request.

ptr
: Address of internal object for this connection. Used as unique ID.

link
: Address of server connection the client is paired with.

remote_pid
: Process ID, in case client connects over Unix socket and OS supports getting it.

SHOW POOLS;

A new pool entry is made for each couple of (database, user).

database
: Database name.

user
: User name.

cl_active
: Client connections that are linked to server connection and can process queries.

cl_waiting
: Client connections have sent queries but have not yet got a server connection.

sv_active
: Server connections that linked to client.

sv_idle
: Server connections that unused and immediately usable for client queries.

sv_used
: Server connections that have been idle more than *server_check_delay* , so they needs *server_check_query* to run on it before it can be used.

sv_tested
: Server connections that are currently running either *server_reset_query* or *server_check_query* .

sv_login
: Server connections currently in logging in process.

maxwait
: How long the first (oldest) client in queue has waited, in seconds. If this starts increasing, then the current pool of servers does not handle requests quick enough. Reason may be either overloaded server or just too small of a **pool_size** setting.

pool_mode
: The pooling mode in use.

SHOW LISTS;

Show following internal information, in columns (not rows):

databases
: Count of databases.

users
: Count of users.

pools
: Count of pools.

free_clients
: Count of free clients.

used_clients

Count of used clients.

login_clients
: Count of clients in **login** state.

free_servers
: Count of free servers.

used_servers
: Count of used servers.

SHOW USERS;

name
: The user name

pool_mode
: The user's override pool_mode, or NULL if the default will be used instead.

SHOW DATABASES;

name
: Name of configured database entry.

host
: Host pgbouncer connects to.

port
: Port pgbouncer connects to.

database
: Actual database name pgbouncer connects to.

force_user
: When user is part of the connection string, the connection between pgbouncer and PostgreSQL is forced to the given user, whatever the client user.

pool_size
: Maximum number of server connections.

pool_mode
: The database's override pool_mode, or NULL if the default will be used instead.

SHOW FDS;

Internal command - shows list of file descriptors in use with internal state attached to them.

When the connected user has user name "pgbouncer", connects through Unix socket and has same UID as the running process, the actual FDs are passed over the connection. This mechanism is used to do an online restart. Note: This does not work on Windows machines.

This command also blocks the internal event loop, so it should not be used while PgBouncer is in use.

fd
: File descriptor numeric value.

task
: One of **pooler** , **client** or **server** .

user
: User of the connection using the FD.

database
: Database of the connection using the FD.

addr
: IP address of the connection using the FD, **unix** if a Unix socket is used.

port
: Port used by the connection using the FD.

cancel
: Cancel key for this connection.

link
: fd for corresponding server/client. NULL if idle.

SHOW CONFIG;

Show the current configuration settings, one per row, with following columns:

key
: Configuration variable name

value
: Configuration value

changeable

Either **yes** or **no** , shows if the variable can be changed while running. If **no** , the variable can be changed only boot-time.

SHOW DNS_HOSTS;

Show host names in DNS cache.

hostname
: Host name.

ttl
: How many seconds until next lookup.

addrs
: Comma separated list of addresses.

SHOW DNS_ZONES

Show DNS zones in cache.

zonename
: Zone name.

serial
: Current serial.

count
: Host names belonging to this zone.

Process controlling commands

PAUSE [db];

PgBouncer tries to disconnect from all servers, first waiting for all queries to complete. The command will not return before all queries are finished. To be used at the time of database restart.

If database name is given, only that database will be paused.

DISABLE db;

Reject all new client connections on the given database.

ENABLE db;

Allow new client connections after a previous **DISABLE** command.

KILL db;

Immediately drop all client and server connections on given database.

SUSPEND;

All socket buffers are flushed and PgBouncer stops listening for data on them. The command will not return before all buffers are empty. To be used at the time of PgBouncer online reboot.

RESUME [db];

Resume work from previous **PAUSE** or **SUSPEND** command.

SHUTDOWN;

The PgBouncer process will exit.

RELOAD;

The PgBouncer process will reload its configuration file and update changeable settings.

Signals

SIGHUP
: Reload config. Same as issuing command **RELOAD;** on console.

SIGINT

Safe shutdown. Same as issuing **PAUSE;** and **SHUTDOWN;** on console.

SIGTERM

Immediate shutdown. Same as issuing **SHUTDOWN;** on console.

Libevent settings

From libevent docs:

```
It is possible to disable support for epoll, kqueue, devpoll, poll
or select by setting the environment variable EVENT_NOEPOLL,
EVENT_NOKQUEUE, EVENT_NODEVPOLL, EVENT_NOPOLL or EVENT_NOSELECT,
respectively.

By setting the environment variable EVENT_SHOW_METHOD, libevent
displays the kernel notification method that it uses.
```

See also

pgbouncer(5) - man page of configuration settings descriptions.

https://pgbouncer.github.io/

https://wiki.postgresql.org/wiki/PgBouncer

--- title: "Configuration" draft: false ---

Description

The configuration file is in "ini" format. Section names are between "[" and "]". Lines starting with ";" or "#" are taken as comments and ignored. The characters ";" and "#" are not recognized when they appear later in the line.

Generic settings

logfile

Specifies log file. Log file is kept open so after rotation `kill -HUP` or on console `RELOAD;` should be done. Note: On Windows machines, the service must be stopped and started.

Default: not set.

pidfile

Specifies the pid file. Without a pidfile, daemonization is not allowed.

Default: not set.

listen_addr

Specifies list of addresses, where to listen for TCP connections. You may also use `*` meaning "listen on all addresses". When not set, only Unix socket connections are allowed.

Addresses can be specified numerically (IPv4/IPv6) or by name.

Default: not set

listen_port

Which port to listen on. Applies to both TCP and Unix sockets.

Default: 6432

unix_socket_dir

Specifies location for Unix sockets. Applies to both listening socket and server connections. If set to an empty string, Unix sockets are disabled. Required for online reboot (-R) to work. Note: Not supported on Windows machines.

Default: /tmp

unix_socket_mode

File system mode for Unix socket.

Default: 0777

unix_socket_group

Group name to use for Unix socket.

Default: not set

user

If set, specifies the Unix user to change to after startup. Works only if PgBouncer is started as root or if it's already running as given user.

Note: Not supported on Windows machines.

Default: not set

auth_file

The name of the file to load user names and passwords from. The file format is the same as the PostgreSQL 8.x pg_auth/pg_pwd file, so this setting can be pointed directly to one of those backend files. Since version 9.0, PostgreSQL does not use such text file, so it must be generated manually. See section Authentication file format below about details.

Default: not set.

auth_hba_file

HBA configuration file to use when auth_type is `hba`. Supported from version 1.7 onwards.

Default: not set

auth_type

How to authenticate users.

pam
: PAM is used to authenticate users, auth_file is ignored. This method is not compatible with databases using auth_user option. Service name reported to PAM is "pgbouncer". Also, *pam* is still not supported in HBA configuration file.

hba
: Actual auth type is loaded from auth_hba_file. This allows different authentication methods different access paths. Example: connection over Unix socket use `peer` auth method, connection over TCP must use TLS. Supported from version 1.7 onwards.

cert
: Client must connect over TLS connection with valid client cert. Username is then taken from CommonName field from certificate.

md5
: Use MD5-based password check. auth_file may contain both MD5-encrypted or plain-text passwords. This is the default authentication method.

plain
: Clear-text password is sent over wire. Deprecated.

trust
: No authentication is done. Username must still exist in auth_file.

any
: Like the `trust` method, but the username given is ignored. Requires that all databases are configured to log in as specific user. Additionally, the console database allows any user to log in as admin.

auth_query

Query to load user's password from database.

Direct access to pg_shadow requires admin rights. It's preferable to use non-admin user that calls SECURITY DEFINER function instead.

Note that the query is run inside target database, so if a function is used it needs to be installed into each database.

Default: `SELECT usename, passwd FROM pg_shadow WHERE usename=$1`

auth_user

If `auth_user` is set, any user not specified in auth_file will be queried through the `auth_query` query from pg_shadow in the database using `auth_user`. Auth_user's password will be taken from `auth_file`.

Direct access to pg_shadow requires admin rights. It's preferable to use non-admin user that calls SECURITY DEFINER function instead.

Default: not set.

pool_mode

Specifies when a server connection can be reused by other clients.

session
: Server is released back to pool after client disconnects. Default.

transaction
: Server is released back to pool after transaction finishes.

statement
: Server is released back to pool after query finishes. Long transactions spanning multiple statements are disallowed in this mode.

max_client_conn

Maximum number of client connections allowed. When increased then the file descriptor limits should also be increased. Note that actual number of file descriptors used is more than max_client_conn. Theoretical maximum used is:

```
max_client_conn + (max_pool_size * total_databases * total_users)
```

if each user connects under its own username to server. If a database user is specified in connect string (all users connect under same username), the theoretical maximum is:

```
max_client_conn + (max_pool_size * total_databases)
```

The theoretical maximum should be never reached, unless somebody deliberately crafts special load for it. Still, it means you should set the number of file descriptors to a safely high number.

Search for `ulimit` in your favorite shell man page. Note: `ulimit` does not apply in a Windows environment.

Default: 100

default_pool_size

How many server connections to allow per user/database pair. Can be overridden in the per-database configuration.

Default: 20

min_pool_size

Add more server connections to pool if below this number. Improves behavior when usual load comes suddenly back after period of total inactivity.

Default: 0 (disabled)

reserve_pool_size

How many additional connections to allow to a pool. 0 disables.

Default: 0 (disabled)

reserve_pool_timeout

If a client has not been serviced in this many seconds, pgbouncer enables use of additional connections from reserve pool. 0 disables.

Default: 5.0

max_db_connections

Do not allow more than this many connections per-database (regardless of pool - i.e. user). It should be noted that when you hit the limit, closing a client connection to one pool will not immediately allow a server connection to be established for another pool, because the server connection for the first pool is still open. Once the server connection closes (due to idle timeout), a new server connection will immediately be opened for the waiting pool.

Default: unlimited

max_user_connections

Do not allow more than this many connections per-user (regardless of pool - i.e. user). It should be noted that when you hit the limit, closing a client connection to one pool will not immediately allow a server connection to be established for another pool, because the server connection for the first pool is still open. Once the server connection closes (due to idle timeout), a new server connection will immediately be opened for the waiting pool.

server_round_robin

By default, pgbouncer reuses server connections in LIFO (last-in, first-out) manner, so that few connections get the most load. This gives best performance if you have a single server serving a database. But if there is TCP round-robin behind a database IP, then it is better if pgbouncer also uses connections in that manner, thus achieving uniform load.

Default: 0

ignore_startup_parameters

By default, PgBouncer allows only parameters it can keep track of in startup packets - `client_encoding` , `datestyle` , `timezone` and `standard_conforming_strings` .

All others parameters will raise an error. To allow others parameters, they can be specified here, so that pgbouncer knows that they are handled by admin and it can ignore them.

Default: empty

disable_pqexec

Disable Simple Query protocol (PQexec). Unlike Extended Query protocol, Simple Query allows multiple queries in one packet, which allows some classes of SQL-injection attacks. Disabling it can improve security. Obviously this means only clients that exclusively use Extended Query protocol will stay working.

Default: 0

application_name_add_host

Add the client host address and port to the application name setting set on connection start. This helps in identifying the source of bad queries etc. This logic applies only on start of connection, if application_name is later changed with SET, pgbouncer does not change it again.

Default: 0

conffile

Show location of current config file. Changing it will make PgBouncer use another config file for next `RELOAD` / `SIGHUP` .

Default: file from command line.

service_name

Used on win32 service registration.

Default: pgbouncer

job_name

Alias for service_name .

Log settings

syslog

Toggles syslog on/off As for windows environment, eventlog is used instead.

Default: 0

syslog_ident

Under what name to send logs to syslog.

Default: pgbouncer (program name)

syslog_facility

Under what facility to send logs to syslog. Possibilities: `auth` , `authpriv` , `daemon` , `user` , `local0-7` .

Default: daemon

log_connections

Log successful logins.

Default: 1

log_disconnections

Log disconnections with reasons.

Default: 1

log_pooler_errors

Log error messages pooler sends to clients.

Default: 1

stats_period

Period for writing aggregated stats into log.

Default: 60

verbose

Increase verbosity. Mirrors "-v" switch on command line. Using "-v -v" on command line is same as *verbose=2* in config.

Default: 0

Console access control

admin_users

Comma-separated list of database users that are allowed to connect and run all commands on console. Ignored when auth_type is `any` , in which case any username is allowed in as admin.

Default: empty

stats_users

Comma-separated list of database users that are allowed to connect and run read-only queries on console. That means all SHOW commands except SHOW FDS.

Default: empty.

Connection sanity checks, timeouts

server_reset_query

Query sent to server on connection release, before making it available to other clients. At that moment no transaction is in progress so it should not include ABORT or ROLLBACK .

The query is supposed to clean any changes made to database session so that next client gets connection in well-defined state. Default is DISCARD ALL which cleans everything, but that leaves next client no pre-cached state. It can be made lighter, e.g. DEALLOCATE ALL to just drop prepared statements, if application does not break when some state is kept around.

When transaction pooling is used, the server_reset_query is not used, as clients must not use any session-based features as each transaction ends up in different connection and thus gets different session state.

Default: DISCARD ALL

server_reset_query_always

Whether server_reset_query should be run in all pooling modes. When this setting is off (default), the server_reset_query will be run only in pools that are in sessions-pooling mode. Connections in transaction-pooling mode should not have any need for reset query.

It is workaround for broken setups that run apps that use session features over transaction-pooled pgbouncer. Is changes non-deterministic breakage to deterministic breakage - client always lose their state after each transaction.

Default: 0

server_check_delay

How long to keep released connections available for immediate re-use, without running sanity-check queries on it. If 0 then the query is ran always.

Default: 30.0

server_check_query

Simple do-nothing query to check if the server connection is alive.

If an empty string, then sanity checking is disabled.

Default: SELECT 1;

server_lifetime

The pooler will try to close server connections that have been connected longer than this. Setting it to 0 means the connection is to be used only once, then closed. [seconds]

Default: 3600.0

server_idle_timeout

If a server connection has been idle more than this many seconds it will be dropped. If 0 then timeout is disabled. [seconds]

Default: 600.0

server_connect_timeout

If connection and login won't finish in this amount of time, the connection will be closed. [seconds]

Default: 15.0

server_login_retry

If login failed, because of failure from connect() or authentication that pooler waits this much before retrying to connect. [seconds]

Default: 15.0

client_login_timeout

If a client connects but does not manage to login in this amount of time, it will be disconnected. Mainly needed to avoid dead connections stalling SUSPEND and thus online restart. [seconds]

Default: 60.0

autodb_idle_timeout

If the automatically created (via "*") database pools have been unused this many seconds, they are freed. The negative aspect of that is that their statistics are also forgotten. [seconds]

Default: 3600.0

dns_max_ttl

How long the DNS lookups can be cached. If a DNS lookup returns several answers, pgbouncer will robin-between them in the meantime. Actual DNS TTL is ignored. [seconds]

Default: 15.0

dns_nxdomain_ttl

How long error and NXDOMAIN DNS lookups can be cached. [seconds]

Default: 15.0

dns_zone_check_period

Period to check if zone serial has changed.

PgBouncer can collect DNS zones from host names (everything after first dot) and then periodically check if zone serial changes. If it notices changes, all host names under that zone are looked up again. If any host IP changes, its connections are invalidated.

Works only with UDNS and c-ares backends (`--with-udns` or `--with-cares` to configure).

Default: 0.0 (disabled)

TLS settings

client_tls_sslmode

TLS mode to use for connections from clients. TLS connections are disabled by default. When enabled, client_tls_key_file and client_tls_cert_file must be also configured to set up key and cert PgBouncer uses to accept client connections.

disable
: Plain TCP. If client requests TLS, it's ignored. Default.

allow
: If client requests TLS, it is used. If not, plain TCP is used. If client uses client-certificate, it is not validated.

prefer
: Same as `allow` .

require
: Client must use TLS. If not, client connection is rejected. If client uses client-certificate, it is not validated.

verify-ca
: Client must use TLS with valid client certificate.

verify-full
: Same as `verify-ca` .

client_tls_key_file

Private key for PgBouncer to accept client connections.

Default: not set.

client_tls_cert_file

Certificate for private key. Clients can validate it.

Default: not set.

client_tls_ca_file

Root certificate file to validate client certificates.

Default: unset.

client_tls_protocols

Which TLS protocol versions are allowed. Allowed values: `tlsv1.0` , `tlsv1.1` , `tlsv1.2` . Shortcuts: `all` (tlsv1.0,tlsv1.1,tlsv1.2), `secure` (tlsv1.2), `legacy` (all).

Default: `all`

client_tls_ciphers

Default: fast

client_tls_ecdhcurve

Elliptic Curve name to use for ECDH key exchanges.

Allowed values: none (DH is disabled), auto (256-bit ECDH), curve name.

Default: auto

client_tls_dheparams

DHE key exchange type.

Allowed values: none (DH is disabled), auto (2048-bit DH), legacy (1024-bit DH).

Default: auto

server_tls_sslmode

TLS mode to use for connections to PostgreSQL servers. TLS connections are disabled by default.

disable
: Plain TCP. TCP is not event requested from server. Default.

allow
: FIXME: if server rejects plain, try TLS?

prefer
: TLS connection is always requested first from PostgreSQL, when refused connection will be established over plain TCP. Server certificate is not validated.

require
: Connection must go over TLS. If server rejects it, plain TCP is not attempted. Server certificate is not validated.

verify-ca
: Connection must go over TLS and server certificate must be valid according to server_tls_ca_file . Server host name is not checked against certificate.

verify-full
: Connection must go over TLS and server certificate must be valid according to server_tls_ca_file . Server host name must match certificate info.

server_tls_ca_file

Root certificate file to validate PostgreSQL server certificates.

Default: unset.

server_tls_key_file

Private key for PgBouncer to authenticate against PostgreSQL server.

Default: not set.

server_tls_cert_file

Certificate for private key. PostgreSQL server can validate it.

Default: not set.

server_tls_protocols

Which TLS protocol versions are allowed. Allowed values: tlsv1.0 , tlsv1.1 , tlsv1.2 . Shortcuts: all (tlsv1.0,tlsv1.1,tlsv1.2), secure (tlsv1.2), legacy (all).

Default: all

server_tls_ciphers

Default: fast

Dangerous timeouts

Setting following timeouts cause unexpected errors.

query_timeout

Queries running longer than that are canceled. This should be used only with slightly smaller server-side statement_timeout, to apply only for network problems. [seconds]

Default: 0.0 (disabled)

query_wait_timeout

Maximum time queries are allowed to spend waiting for execution. If the query is not assigned to a server during that time, the client is disconnected. This is used to prevent unresponsive servers from grabbing up connections. [seconds]

It also helps when server is down or database rejects connections for any reason. If this is disabled, clients will be queued infinitely.

Default: 120

client_idle_timeout

Client connections idling longer than this many seconds are closed. This should be larger than the client-side connection lifetime settings, and only used for network problems. [seconds]

Default: 0.0 (disabled)

idle_transaction_timeout

If client has been in "idle in transaction" state longer, it will be disconnected. [seconds]

Default: 0.0 (disabled)

Low-level network settings

pkt_buf

Internal buffer size for packets. Affects size of TCP packets sent and general memory usage. Actual libpq packets can be larger than this so, no need to set it large.

Default: 4096

max_packet_size

Maximum size for PostgreSQL packets that PgBouncer allows through. One packet is either one query or one result set row. Full result set can be larger.

Default: 2147483647

listen_backlog

Backlog argument for listen(2). Determines how many new unanswered connection attempts are kept in queue. When queue is full, further new connections are dropped.

Default: 128

sbuf_loopcnt

How many times to process data on one connection, before proceeding. Without this limit, one connection with a big result set can stall PgBouncer for a long time. One loop processes one pkt_buf amount of data. 0 means no limit.

Default: 5

suspend_timeout

How many seconds to wait for buffer flush during SUSPEND or reboot (-R). Connection is dropped if flush does not succeed.

Default: 10

tcp_defer_accept

For details on this and other tcp options, please see `man 7 tcp` .

Default: 45 on Linux, otherwise 0

tcp_socket_buffer

Default: not set

tcp_keepalive

Turns on basic keepalive with OS defaults.

On Linux, the system defaults are **tcp_keepidle=7200** , **tcp_keepintvl=75** , **tcp_keepcnt=9** . They are probably similar on other OS-es.

Default: 1

tcp_keepcnt

Default: not set

tcp_keepidle

Default: not set

tcp_keepintvl

Default: not set

Section [databases]

This contains key=value pairs where key will be taken as a database name and value as a libpq connect-string style list of key=value pairs. As actual libpq is not used, so not all features from libpq can be used (service=, .pgpass).

Database name can contain characters `_0-9A-Za-z` without quoting. Names that contain other chars need to be quoted with standard SQL ident quoting: double quotes where "" is taken as single quote.

"*" acts as fallback database: if the exact name does not exist, its value is taken as connect string for requested database. Such automatically created database entries are cleaned up if they stay idle longer then the time specified in autodb_idle_timeout parameter.

dbname

Destination database name.

Default: same as client-side database name.

host

Host name or IP address to connect to. Host names are resolved on connect time, the result is cached per `dns_max_ttl` parameter. If DNS returns several results, they are used in round-robin manner.

Default: not set, meaning to use a Unix socket.

port

Default: 5432

user, password

If `user=` is set, all connections to the destination database will be done with the specified user, meaning that there will be only one pool for this database.

Otherwise PgBouncer tries to log into the destination database with client username, meaning that there will be one pool per user.

The length for `password` is limited to 128 characters maximum.

auth_user

Override of the global `auth_user` setting, if specified.

pool_size

Set maximum size of pools for this database. If not set, the default_pool_size is used.

connect_query

Query to be executed after a connection is established, but before allowing the connection to be used by any clients. If the query raises errors, they are logged but ignored otherwise.

pool_mode

Set the pool mode specific to this database. If not set, the default pool_mode is used.

max_db_connections

Configure a database-wide maximum (i.e. all pools within the database will not have more than this many server connections).

client_encoding

Ask specific `client_encoding` from server.

datestyle

Ask specific `datestyle` from server.

timezone

Ask specific **timezone** from server.

Section [users]

This contains key=value pairs where key will be taken as a user name and value as a libpq connect-string style list of key=value pairs. As actual libpq is not used, so not all features from libpq can be used.

pool_mode

Set the pool mode to be used for all connections from this user. If not set, the database or default pool_mode is used.

Include directive

The PgBouncer config file can contain include directives, which specify another config file to read and process. This allows for splitting the configuration file into physically separate parts. The include directives look like this:

```
%include filename
```

If the file name is not absolute path it is taken as relative to current working directory.

Authentication file format

PgBouncer needs its own user database. The users are loaded from a text file in following format:

```
"username1" "password" ...
```

```
"username2" "md5abcdef012342345" ...
```

There should be at least 2 fields, surrounded by double quotes. The first field is the username and the second is either a plain-text or a MD5-hidden password. PgBouncer ignores the rest of the line.

This file format is equivalent to text files used by PostgreSQL 8.x for authentication info, thus allowing PgBouncer to work directly on PostgreSQL authentication files in data directory.

Since PostgreSQL 9.0, the text files are not used anymore. Thus the auth file needs to be generated. See *./etc/mkauth.py* for sample script to generate auth file from *pg_shadow* table.

PostgreSQL MD5-hidden password format:

```
"md5" + md5(password + username)
```

So user *admin* with password *1234* will have MD5-hidden password *md545f2603610af569b6155c45067268c6b* .

HBA file format

It follows the format of PostgreSQL pg_hba.conf file - http://www.postgresql.org/docs/9.4/static/auth-pg-hba-conf.html

There are following differences:

- Supported record types: *local* , *host* , *hostssl* , *hostnossl* .
- Database field: Supports *all* , *sameuser* , *@file* , multiple names. Not supported: *replication* , *samerole* , *samegroup* .
- Username field: Supports *all* , *@file* , multiple names. Not supported: +*groupname* .
- Address field: Supported IPv4, IPv6. Not supported: DNS names, domain prefixes.
- Auth-method field: Supported methods: *trust* , *reject* , *md5* , *password* , *peer* , *cert* . Not supported: *gss* , *sspi* , *ident* , *ldap* , *radius* , *pam* . Also username map (*map=*) parameter is not supported.

Example

Minimal config:

```
[databases]
template1 = host=127.0.0.1 dbname=template1 auth_user=someuser

[pgbouncer]
pool_mode = session
listen_port = 6543
listen_addr = 127.0.0.1
auth_type = md5
auth_file = users.txt
logfile = pgbouncer.log
pidfile = pgbouncer.pid
admin_users = someuser
stats_users = stat_collector
```

Database defaults:

```
[databases]

; foodb over Unix socket
foodb =

; redirect bardb to bazdb on localhost
bardb = host=127.0.0.1 dbname=bazdb

; access to destination database will go with single user
forcedb = host=127.0.0.1 port=300 user=baz password=foo client_encoding=UNICODE datestyle=ISO
```

Example of secure function for auth_query:

```
CREATE OR REPLACE FUNCTION pgbouncer.user_lookup(in i_username text, out uname text, out phash text)
RETURNS record AS $$
BEGIN
    SELECT usename, passwd FROM pg_catalog.pg_shadow
    WHERE usename = i_username INTO uname, phash;
    RETURN;
END;
```

```
$$ LANGUAGE plpgsql SECURITY DEFINER;
REVOKE ALL ON FUNCTION pgbouncer.user_lookup(text) FROM public, pgbouncer;
GRANT EXECUTE ON FUNCTION pgbouncer.user_lookup(text) TO pgbouncer;
```

See also

pgbouncer(1) - man page for general usage, console commands.

https://pgbouncer.github.io/

https://wiki.postgresql.org/wiki/PgBouncer

--- title: "PgBouncer TODO list" draft: false ---

Highly visible missing features

Significant amount of users feel the need for those.

- Protocol-level plan cache.
- LISTEN/NOTIFY. Requires strict SQL format.

Waiting for contributors...

Problems / cleanups

- Bad naming in data strctures: * PgSocket->auth_user [vs. PgDatabase->auth_user] * PgSocket->db [vs. PgPool->db]
- other per-user settings
- Maintenance order vs. lifetime_kill_gap: http://lists.pgfoundry.org/pipermail/pgbouncer-general/2011-February/000679.html
- per_loop_maint/per_loop_activate take too much time in case of moderate load and lots of pools. Perhaps active_pool_list would help, which contains only pools touched in current loop.
- new states for clients: idle and in-query. That allows to apply client_idle_timeout and query_timeout without walking all clients on maintenance time.
- check if SQL error codes are correct
- removing user should work - kill connections
- keep stats about error counts
- cleanup of logging levels, to make log more useful
- to test: - signal flood - no mem / no fds handling
- fix high-freq maintenance timer - it's only needed when PAUSE/RESUME/shutdown is issued.
- Get rid of SBUF_SMALL_PKT logic - it makes processing code complex. Needs a new sbuf_prepare_*() to notify sbuf about short data. [Plain 'false' from handler postpones processing to next event loop.]
- units for config parameters.

Dubious/complicated features

- Load-balancing / failover. Both are already solved via DNS. Adding load-balancing config in pgbouncer might be good idea. Adding failover decision-making is not...
- User-based route. Simplest would be to move db info to pool and fill username into dns.
- some preliminary notification that fd limit is full
- Move all "look-at-full-packet" situations to SBUF_EV_PKT_CALLBACK
- *pool_mode = plproxy* - use postgres in full-duplex mode for autocommit queries, multiplexing several queries into one connection. Should result in more efficient CPU usage of server.
- SMP: spread sockets over per-cpu threads. Needs confirmation that single-threadedness can be problem. It can also be that only accept() + login handling of short connection is problem that could be solved by just having threads for login handling, which would be lot simpler or just deciding that it is not worth fixing.