Open source database management systems – PostgreSQL & MySQL
Tech blog on PostgreSQL / MySQL for DBAs & Developers

APRIL 18, 2013 PRASHANTH GORIPARTHI 5

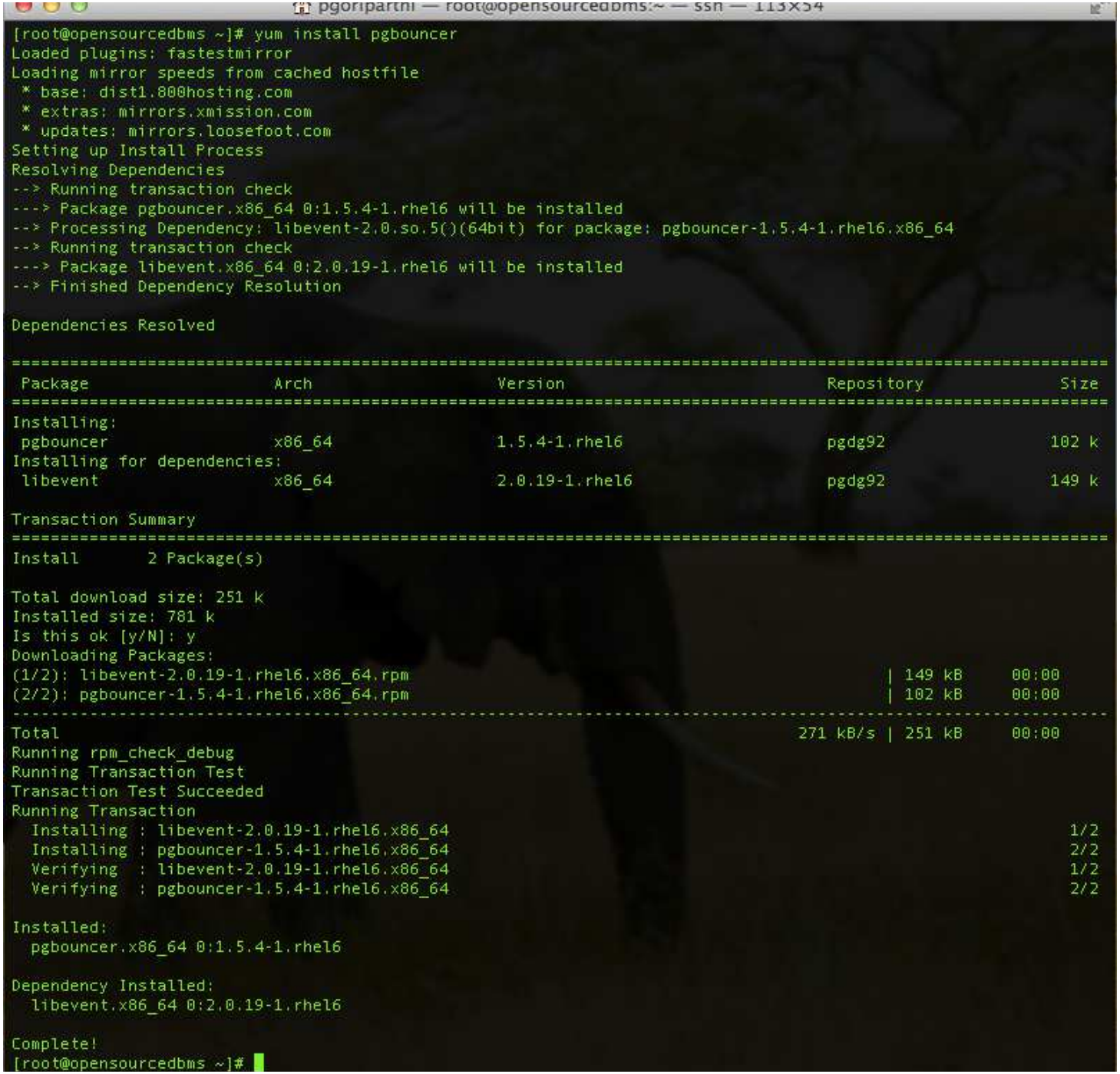
Setup pgbouncer connection pooling for PostgreSQL on CentOS/RedHat/Fedora

This post helps you install, setup and benchmark pgbouncer connection pooling for PostgreSQL 9.2 on CentOS 6/RedHat/Fedora based systems. pgbouncer is one of the PostgreSQL connection poolers. There are other connection poolers available for PostgreSQL. Any enduser applications 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 purpose of connection poolers in general is to lower the performance impact of opening new connections to PostgreSQL(or Other) databases. pgbouncer can connection pool in three ways, which are

- Session pooling
- Transaction pooling
- Statement pooling

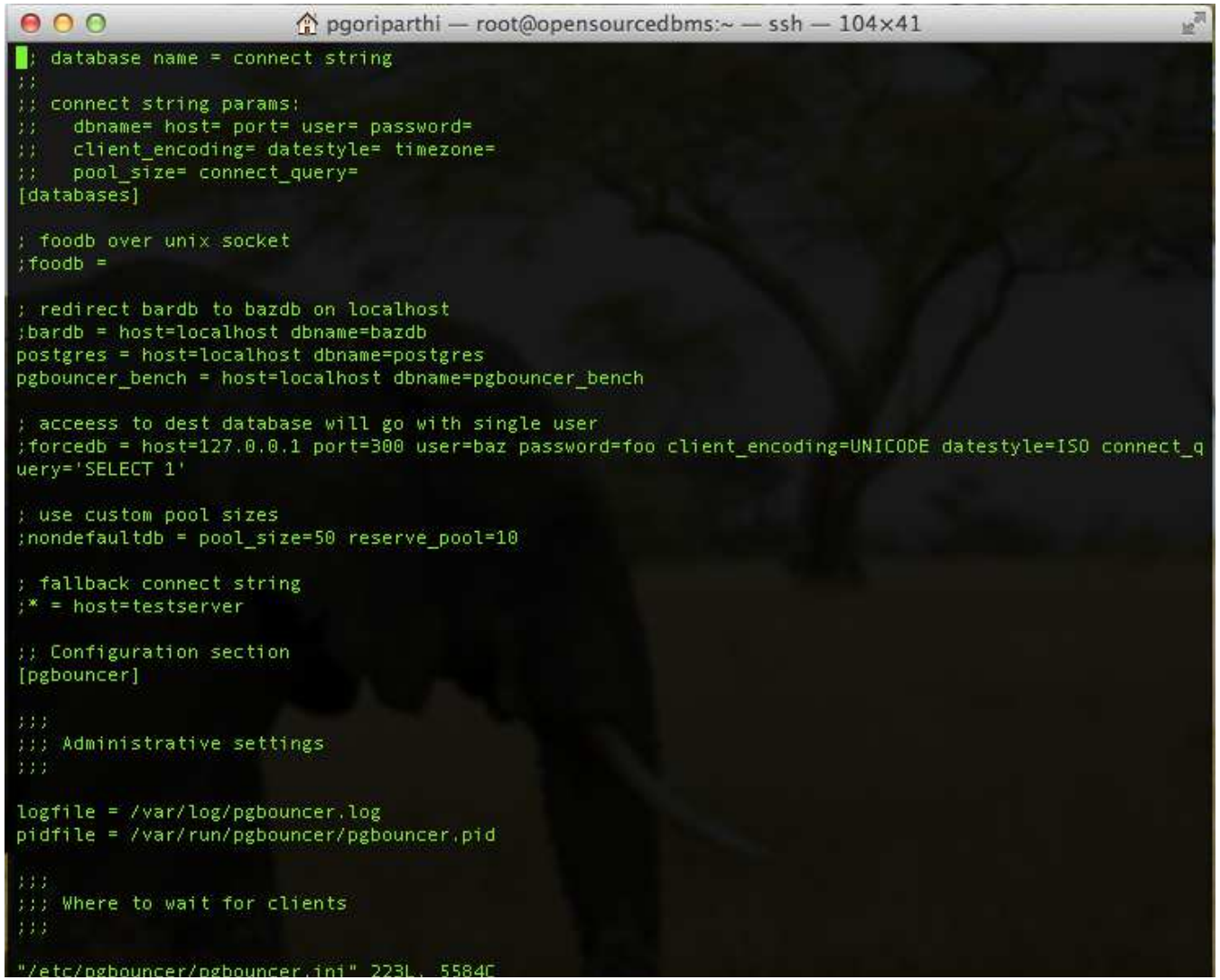
Second and Third options are more aggressive and not very easy to maintain/manage. In this post I will setup session pooling. I am assuming that you have a postgresql database server up and running before you start deploying pgbouncer. If haven't done installing postgresql before, please go through my other post on how to install Postgres database server. To install pgbouncer, simply yum install it as shown below.

```
yum install pgbouncer
```


pgoriparthi — root@opensourcedbms:~ — ssh — 113×54
[root@opensourcedbms ~]# yum install pgbouncer
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: dist1.800hosting.com
 * extras: mirrors.xmission.com
 * updates: mirrors.loosefoot.com
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package pgbouncer.x86_64 0:1.5.4-1.rhel6 will be installed
--> Processing Dependency: libevent-2.0.so.5()(64bit) for package: pgbouncer-1.5.4-1.rhel6.x86_64
--> Running transaction check
---> Package libevent.x86_64 0:2.0.19-1.rhel6 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
Package Arch Version Repository Size
Installing:
pgbouncer x86_64 1.5.4-1.rhel6 pgdg92 102 k
Installing for dependencies:
libevent x86_64 2.0.19-1.rhel6 pgdg92 149 k
Transaction Summary
Install 2 Package(s)
Total download size: 251 k
Installed size: 781 k
Is this ok [y/N]: y
Downloading Packages:
(1/2): libevent-2.0.19-1.rhel6.x86_64.rpm | 149 kB 00:00
(2/2): pgbouncer-1.5.4-1.rhel6.x86_64.rpm | 102 kB 00:00
Total 271 kB/s | 251 kB 00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing : libevent-2.0.19-1.rhel6.x86_64 1/2
Installing : pgbouncer-1.5.4-1.rhel6.x86_64 2/2
Verifying : libevent-2.0.19-1.rhel6.x86_64 1/2
Verifying : pgbouncer-1.5.4-1.rhel6.x86_64 2/2
Installed:
pgbouncer.x86_64 0:1.5.4-1.rhel6
Dependency Installed:
libevent.x86_64 0:2.0.19-1.rhel6
Complete!
[root@opensourcedbms ~]#

Once pgbouncer is installed we need to tweak its config and create some users. Let me walk through these with you. First lets edit the config file and tweak it.

```
vi /etc/pgbouncer/pgbouncer.ini
```



You will need to update/add the following lines to your config:

Add all the databases that you need to pass through from pgbouncer, I just added postgres/pgbouncer_bench db redirects here (I will be using pgbouner_bench database down for benchmarking). Also you can change the auth_type to your needs (I am using plain for illustration purpose).

```
postgres = host=localhost dbname=postgres
pgbouncer_bench = host=localhost dbname=pgbouncer_bench

listen_addr = *
auth_file = /etc/pgbouncer/userlist.txt
auth_type = plain
```

Your full config file should look as following:

```
;; database name = connect string
;;
;; connect string params:
;;   dbname= host= port= user= password=
;;   client_encoding= datestyle= timezone=
;;   pool_size= connect_query=
[databases]

; foodb over unix socket
;foodb =

; redirect bardb to bazdb on localhost
;bardb = host=localhost dbname=bazdb
postgres = host=localhost dbname=postgres
pgbouncer_bench = host=localhost dbname=pgbouncer_bench

; acceess to dest database will go with single user
;forcedb = host=127.0.0.1 port=300 user=baz password=foo client_encoding=UNICODE datestyle=ISO connect_query='SELE

; use custom pool sizes
;nondefaultdb = pool_size=50 reserve_pool=10
```

```
; fallback connect string
;* = host=testserver

;; Configuration section
[pgbouncer]

;;;
;;; Administrative settings
;;;

logfile = /var/log/pgbouncer.log
pidfile = /var/run/pgbouncer/pgbouncer.pid

;;;
;;; Where to wait for clients
;;;

; ip address or * which means all ip-s
listen_addr = *
listen_port = 6432

; unix socket is also used for -R.
; On debian it should be /var/run/postgresql
;unix_socket_dir = /tmp
;unix_socket_mode = 0777
;unix_socket_group =

;;;
;;; Authentication settings
;;;

; any, trust, plain, crypt, md5
auth_type = plain
;auth_file = /8.0/main/global/pg_auth
auth_file = /etc/pgbouncer/userlist.txt

;;;
;;; Users allowed into database 'pgbouncer'
;;;

; comma-separated list of users, who are allowed to change settings
admin_users = postgres

; comma-separated list of users who are just allowed to use SHOW command
stats_users = stats, postgres
```

```
;;;
;;; Pooler personality questions
;;;

; When server connection is released back to pool:
;   session      - after client disconnects
;   transaction  - after transaction finishes
;   statement    - after statement finishes
pool_mode = session

;
; Query for cleaning connection immediately after releasing from client.
; No need to put ROLLBACK here, pgbouncer does not reuse connections
; where transaction is left open.
;
; Query for 8.3+:
;   DISCARD ALL;
;
; Older versions:
;   RESET ALL; SET SESSION AUTHORIZATION DEFAULT
;
; Empty if transaction pooling is in use.
;
server_reset_query = DISCARD ALL

;
; Comma-separated list of parameters to ignore when given
; in startup packet.  Newer JDBC versions require the
; extra_float_digits here.
;
;ignore_startup_parameters = extra_float_digits

;
; When taking idle server into use, this query is ran first.
;   SELECT 1
;
;server_check_query = select 1

; If server was used more recently that this many seconds ago,
; skip the check query.  Value 0 may or may not run in immediately.
;server_check_delay = 30

;;;
;;; Connection limits
;;;
```

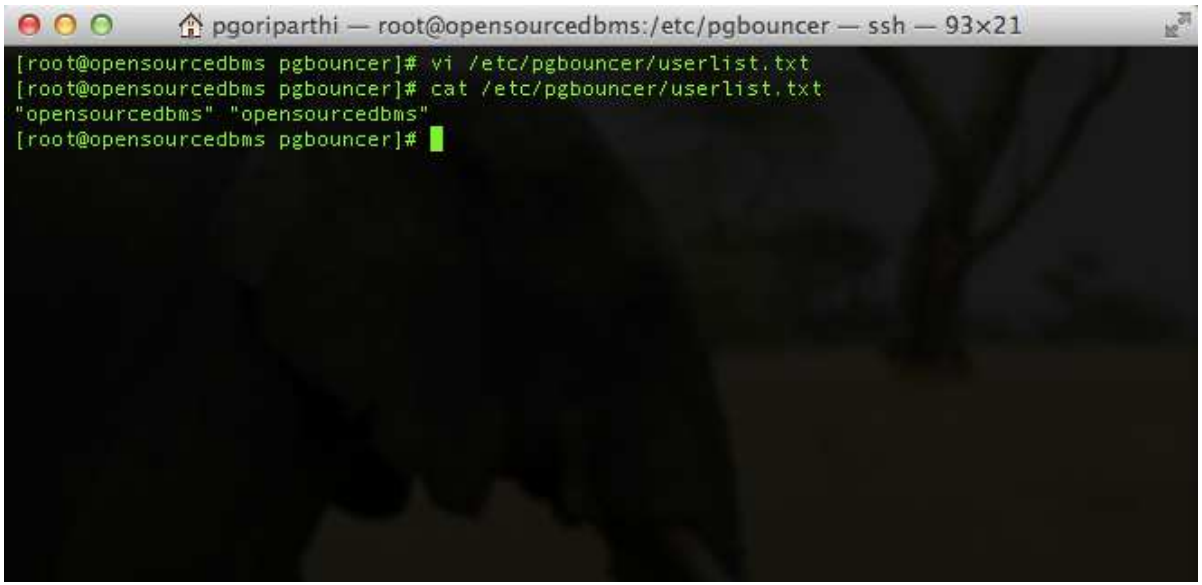
```
; total number of clients that can connect
max_client_conn = 100

; default pool size.  20 is good number when transaction pooling
; is in use, in session pooling it needs to be the number of
; max clients you want to handle at any moment
default_pool_size = 100

; how many additional connection to allow in case of trouble
;reserve_pool_size = 5

; if a clients needs to wait more than this many seconds, use reserve pool
;reserve_pool_timeout = 3

; log if client connects or server connection is made
;log_connections = 1

; log if and why connection was closed
;log_disconnections = 1

; log error messages pooler sends to clients
;log_pooler_errors = 1

; If off, then server connections are reused in LIFO manner
;server_round_robin = 0

;;;
;;; Timeouts
;;;

;; Close server connection if its been connected longer.
;server_lifetime = 1200

;; Close server connection if its not been used in this time.
;; Allows to clean unnecessary connections from pool after peak.
;server_idle_timeout = 60

;; Cancel connection attempt if server does not answer takes longer.
;server_connect_timeout = 15

;; If server login failed (server_connect_timeout or auth failure)
;; then wait this many second.
;server_login_retry = 15

;; Dangerous.  Server connection is closed if query does not return
;; in this time.  Should be used to survive network problems,
```

```
;; _not_ as statement_timeout. (default: 0)
;query_timeout = 0

;; Dangerous.  Client connection is closed if the query is not assigned
;; to a server in this time.  Should be used to limit the number of queued
;; queries in case of a database or network failure. (default: 0)
;query_wait_timeout = 0

;; Dangerous.  Client connection is closed if no activity in this time.
;; Should be used to survive network problems. (default: 0)
;client_idle_timeout = 0

;; Disconnect clients who have not managed to log in after connecting
;; in this many seconds.
;client_login_timeout = 60

;; Clean automatically created database entries (via "*") if they
;; stay unused in this many seconds.
; autodb_idle_timeout = 3600

;;;
;;; Low-level tuning options
;;;

;; buffer for streaming packets
;pkt_buf = 2048

;; man 2 listen
;listen_backlog = 128

;; networking options, for info: man 7 tcp

;; Linux: notify program about new connection only if there
;; is also data received.  (Seconds to wait.)
;; On Linux the default is 45, on other OS'es 0.
;tcp_defer_accept = 0

;; In-kernel buffer size (Linux default: 4096)
;tcp_socket_buffer = 0

;; whether tcp keepalive should be turned on (0/1)
;tcp_keepalive = 1

;; following options are Linux-specific.
;; they also require tcp_keepalive=1
```

```
;; count of keepaliva packets
;tcp_keepcnt = 0

;; how long the connection can be idle,
;; before sending keepalive packets
;tcp_keepidle = 0

;; The time between individual keepalive probes.
;tcp_keepintvl = 0

;; DNS lookup caching time
;dns_max_ttl = 15

;; DNS zone SOA lookup period
;dns_zone_check_period = 0
```

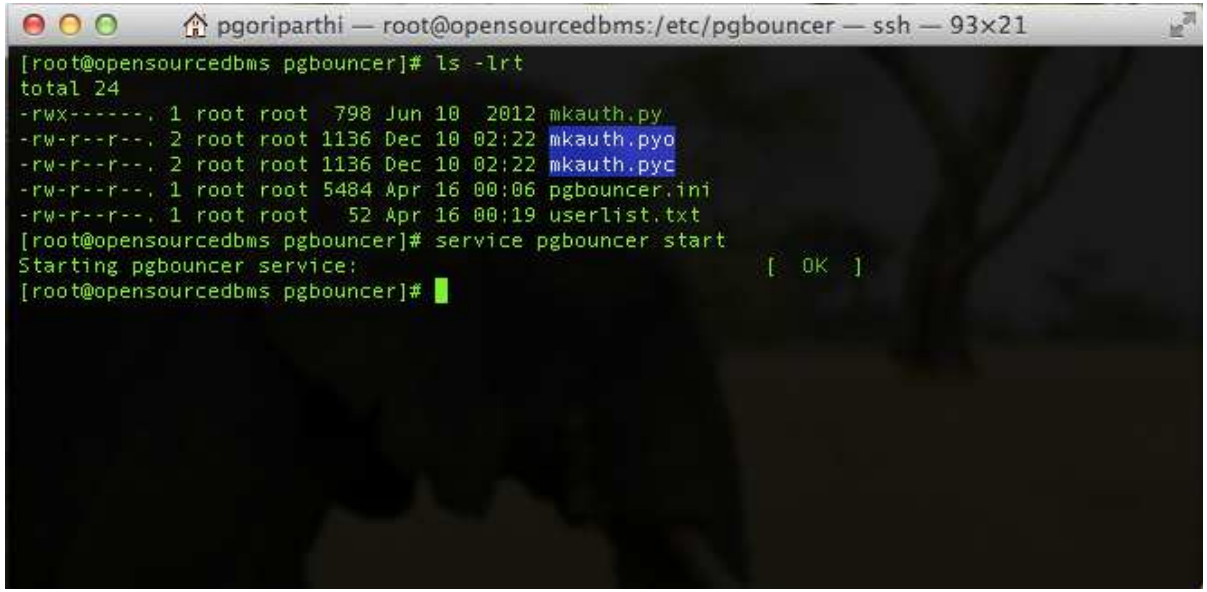
Now lets create users.txt file for accessing database through pgbouncer. These users are users that already exist in our existing database. Add all users you wish to give access to your databases through pgbouncer. Format of user/password should be like "user" "password".

```
vi /etc/pgbouncer/userlist.txt
```



Now that we created users we can start our connection pooler.

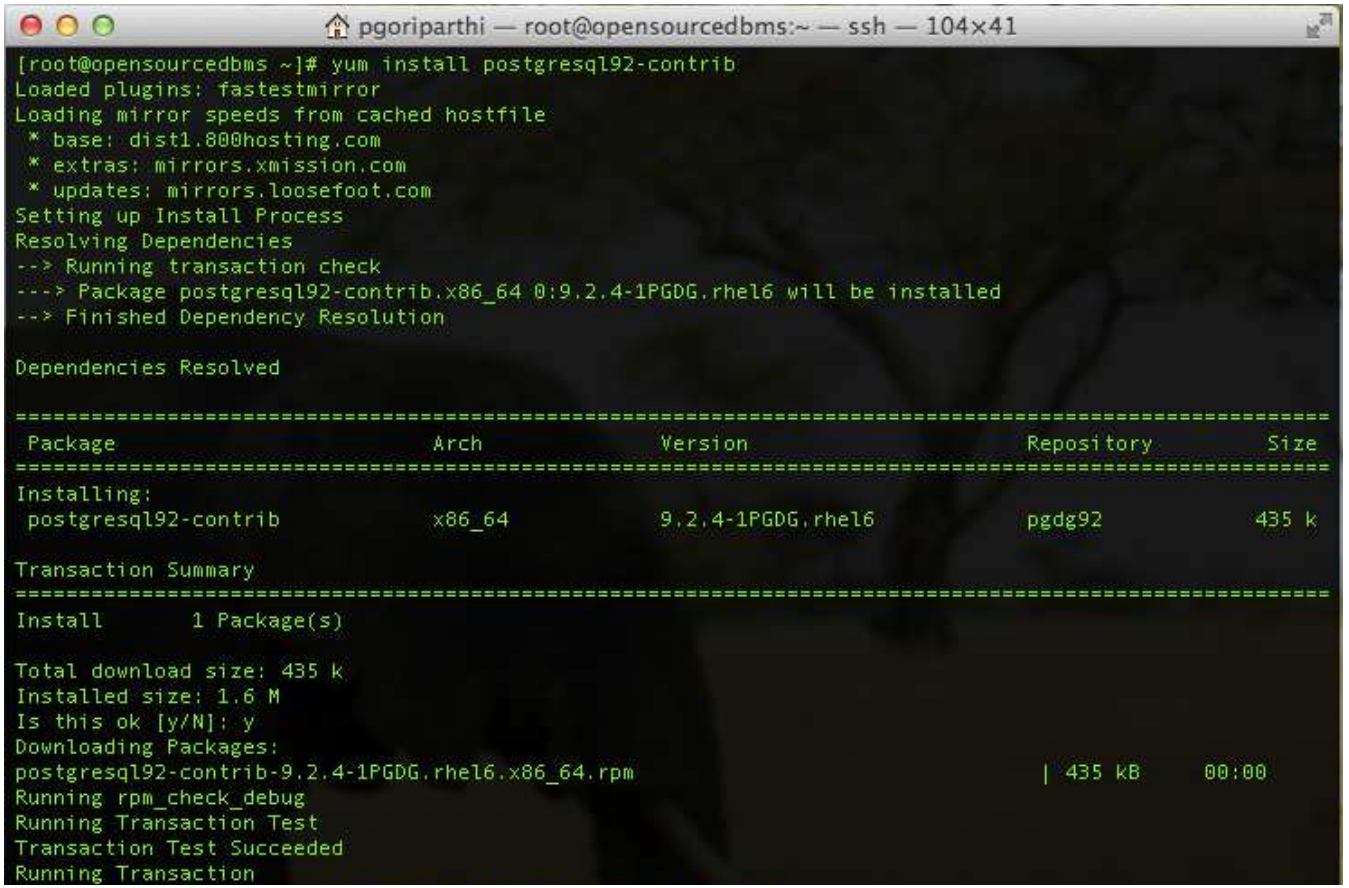
```
service pgbouncer start
```



Thats it folks you have pgbouncer up and running. Now lets connect to it at port 6432 [default, this can be updated in pgbouncer.ini file].

You have now successfully connected to pgbouncer. Lets benchmark and see if pgbouncer makes any difference or not. We need pgbench utility to benchmark postgres. To get that utility we need to install postgres contrib package.

```
yum install postgresql92-contrib
```



```
  Installing : postgresql92-contrib-9.2.4-1PGDG.rhel6.x86_64                1/1
  Verifying  : postgresql92-contrib-9.2.4-1PGDG.rhel6.x86_64                1/1

Installed:
  postgresql92-contrib.x86_64 0:9.2.4-1PGDG.rhel6

Complete!
[root@opensourcedbms ~]#
```

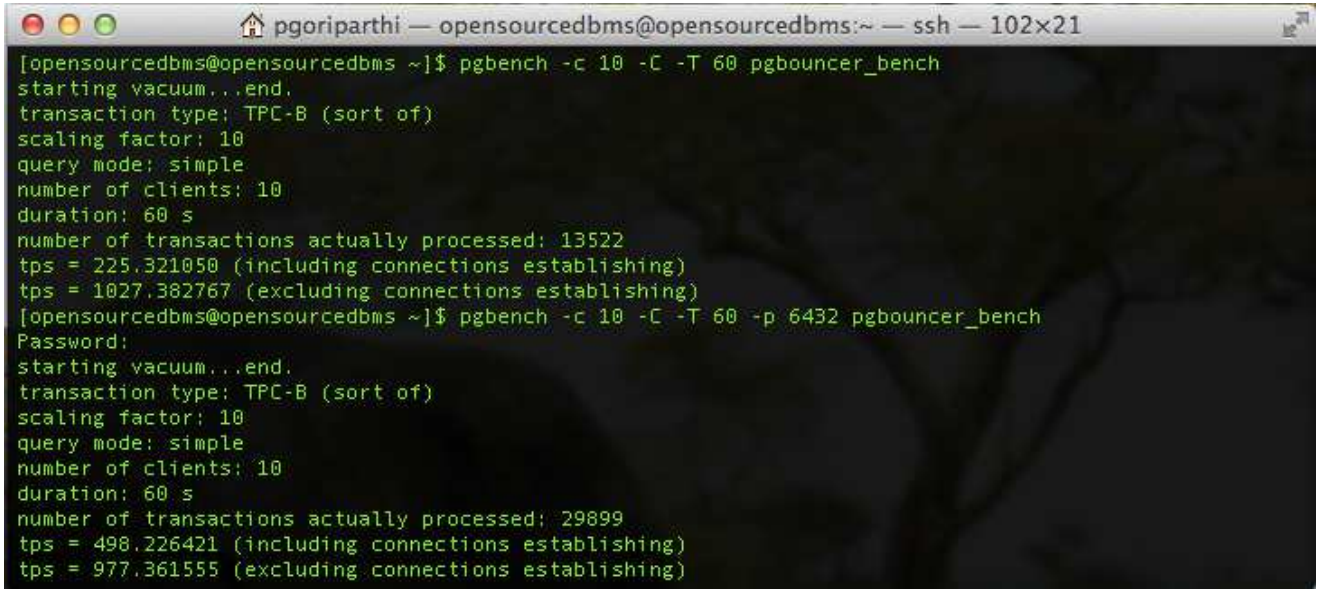
To benchmark we need to create pgbouncer_bench(temporary) database to run our connections against and populate this database with some data.

```
createdb pgbouncer_bench
pgbench -i -s 10 pgbouncer_bench
```

Now we can start benchmarking our database between port 5432 and 6432(pgbouncer) and see the performance difference.

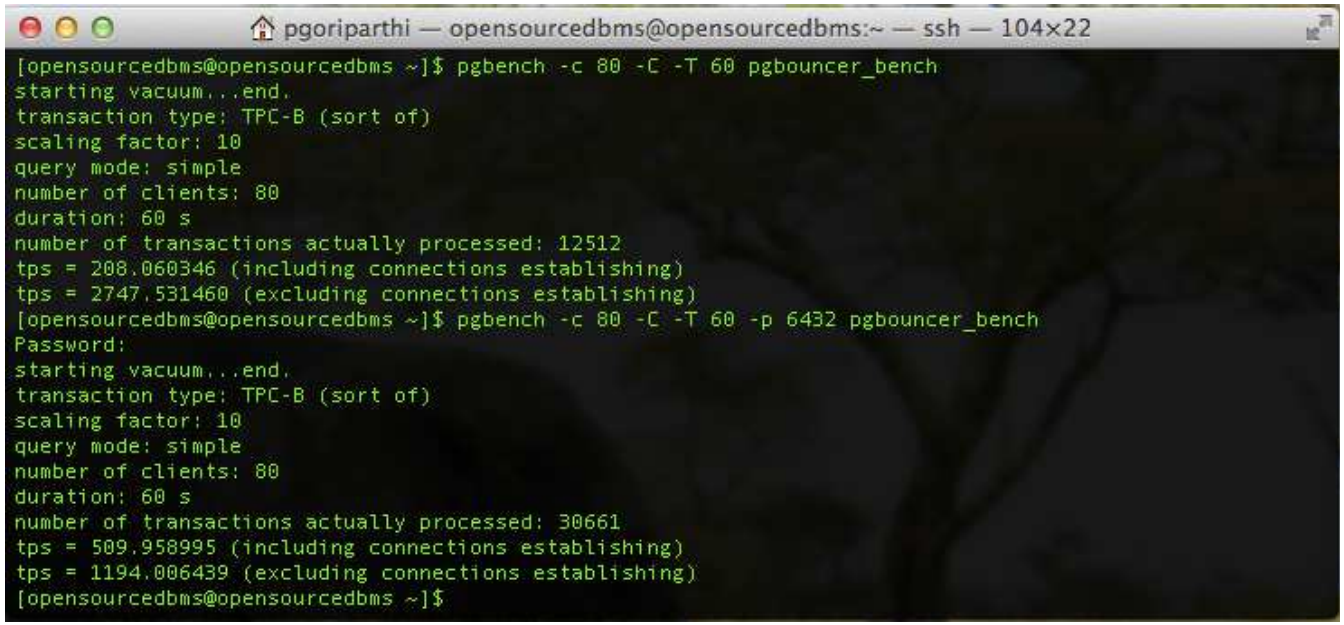
Performance at 10 client connections:

```
[opensourcedbms@opensourcedbms ~]$ pgbench -c 10 -C -T 60 pgbouncer_bench
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 10
query mode: simple
number of clients: 10
duration: 60 s
number of transactions actually processed: 13522
tps = 225.321050 (including connections establishing)
tps = 1027.382767 (excluding connections establishing)
[opensourcedbms@opensourcedbms ~]$ pgbench -c 10 -C -T 60 -p 6432 pgbouncer_bench
Password:
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 10
query mode: simple
number of clients: 10
duration: 60 s
number of transactions actually processed: 29899
tps = 498.226421 (including connections establishing)
tps = 977.361555 (excluding connections establishing)
```



Performance at 80 client connections:

```
[opensourcedbms@opensourcedbms ~]$ pgbench -c 80 -C -T 60 pgbouncer_bench
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 10
query mode: simple
number of clients: 80
duration: 60 s
number of transactions actually processed: 12512
tps = 208.060346 (including connections establishing)
tps = 2747.531460 (excluding connections establishing)
[opensourcedbms@opensourcedbms ~]$ pgbench -c 80 -C -T 60 -p 6432 pgbouncer_bench
Password:
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 10
query mode: simple
number of clients: 80
duration: 60 s
number of transactions actually processed: 30661
tps = 509.958995 (including connections establishing)
tps = 1194.006439 (excluding connections establishing)
```



You can see the significant improvement of transactions when we are connecting to database using pgbouncer. There are various tweaks that you can do to pgbouncer config file and tune according to your needs.

Please post questions in comments if you have any.

POSTGRESQL

About Prashanth Goriparthi

Sr. Database Administrator / Architect View all posts by Prashanth Goriparthi →

5 Comments

Tiah

August 8, 2013 at 9:40 am

Nice article, very useful 🙂

Reply

Shariman

March 19, 2014 at 6:13 am

Very detail and accurate guide. Thanks a lot

Reply

inetholic

March 31, 2014 at 5:28 pm

Thanks … i can increase TPS with pgbouncer … good job bro …

Reply

Akbar

February 4, 2015 at 3:50 am

Hai. I have some error.
I was successfully run this :
[postgres@edb ~]$ pgbench -c 10 -C -T 60 pgbouncer_bench

But when I run this :
[postgres@edb ~]$ pgbench -c 10 -C -T 60 -p 6432 pgbouncer_bench
Getting error :
Connection to database "pgbouncer_bench" failed:
ERROR: No such database: pgbouncer_bench

Do you have idea why this is happen?
Thanks

Reply

Jagdeep

December 22, 2015 at 4:19 pm

I am facing below mentioned error after following the above mentioned steps.

psql: ERROR: pgbouncer cannot connect to server

Could you please able to help me to resolve this issue?
Thanks in advance.

Reply

Leave a Reply

Name:

Email:

Website

* Captcha [Please fill/solve below] : *

Time limit is exhausted. Please reload the CAPTCHA.

5 × = twenty

Submit Comment