Insight for DBAs PostgreSQL

Subscribe to RSS Feed

Configuring PgBouncer for Multi-Port Access

June 5, 2023

Robert Bernier

From time to time, situations occur where unusual circumstances dictate out-of-the-box thinking.

For example, suppose you have a system where you've installed multiple data clusters onto a single host. What you end up with is a host breaking up and sharing valuable system resources, i.e., CPU, RAM, disk, etc., between multiple instances of PostgreSQL data clusters. While easy to do in a development environment, it does represent technical debt when it reaches production. Sooner or later, one must address this issue; otherwise, one can suffer the consequences of handicapping your entire database infrastructure.

Let's now move forward in time: your system has scaled, of course, and this shortcut of using multiple data clusters on a single host has now become a performance bottleneck. The problem is you either don't or can't refactor your application servers; something, maybe, about not having enough time in the day. And, as you may already know, while Postgres can sit on both a UNIX DOMAIN socket and IPv4, IPv6 port, etc., one is nevertheless constrained to listen to just the one port.

So what do you do?

For the experienced sysadmin, there are actually quite a number of "magical" techniques. However, in this case, with a little help from systemd, which manages all service processes, we will solve this little puzzle using PgBouncer with a concoction of configuration files.

Scenario

Configure the system such that Postgres resides on its default port of 5432 and PgBouncer sits on three ports, i.e., 6432, 6433, and 6433, accessing the resident Postgres server.

The PgBouncer connection pooler will use an administrative account, the ROLE PgBouncer, for the purpose of user authentication. Authentication is to be achieved by accessing the Postgres server's pg_shadow table and comparing the resultant hash to all incoming connections (this won't work for cloud setups such as, for example, Amazon RDS).

A set of Systemd configuration files will be created and edited in order to manage the PgBouncer service.

About the files

Below is a summary of the files and how they will be edited. Remember, these configuration files are of a hypothetical nature using minimal settings, which, of course, will need to be updated to match a realistic production environment.

```Shell
├── pgbouncer
│   ├── pgbouncer.ini
│   └── userlist.txt
├── postgres
│   ├── add_function.sql
│   ├── add_user.sql
└── systemd
    ├── pgbouncer_override
    └── pgbouncer.socket
```

pgbouncer.ini

This configuration file defines all behavior and is installed in its default location, "/etc/pgbouncer".

Only one domain socket is used. For our purposes, the *listen_port* runtime parameter is just noise and is superseded by the other runtime parameters as declared in file pgbouncer.socket.

TIP: *Backup the original pgbouncer.ini as it references ALL runtime parameters.*

```INI
[databases]
* = host=localhost

```

```
[users]
# left blank

[pgbouncer]
logfile = /var/log/postgresql/pgbouncer.log
pidfile = /var/run/postgresql/pgbouncer.pid

;; these parameters are implicitly disabled
listen_addr = 0.0.0.0
listen_port = 6432

unix_socket_dir = /var/run/postgresql
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
auth_user = pgbouncer
auth_query = SELECT p_user, p_password FROM public.lookup($1)

pool_mode = session

;; Use <appname - host> as application_name on server.
application_name_add_host = 1
```

userlist.txt

Contains a single user account and its password for authentication. Note that the md5 hashed password is *pgbouncer*.

```INI
"pgbouncer" "md5be5544d3807b54dd0637f2439ecb03b9"
```

add_user.sql

Adds the ROLE "pgbouncer" to the Postgres data cluster. While under normal circumstances this is not required, PgBouncer uses this ROLE in order to validate all logins.

For our purposes, the password is "*pgbouncer*":

```PgSQL
--
-- EXECUTE AS SUPERUSER postgres
--
CREATE ROLE pgbouncer LOGIN WITH PASSSWORD pgbouncer;
```

add_function.sql

This is a user-defined function that ROLE PgBouncer executes in order to obtain the password hash from pg_shadow.

This SQL statement must be executed against each and every database that is to be accessed by any ROLE connection using PgBouncer.

TIP: *Execute this SQL against database template1, and the function call will be included with every newly created database thereafter.*

```PgSQL
--
-- EXECUTE AS SUPERUSER postgres
--
-- execute on each database user accounts will login
--
CREATE FUNCTION public.lookup (
    INOUT p_user     name,
    OUT   p_password text
) RETURNS record
    LANGUAGE sql SECURITY DEFINER SET search_path = pg_catalog AS
$$SELECT usename, passwd FROM pg_shadow WHERE usename = p_user$$;

-- make sure only "pgbouncer" can use the function
REVOKE EXECUTE ON FUNCTION public.lookup(name) FROM PUBLIC;
GRANT EXECUTE ON FUNCTION public.lookup(name) TO pgbouncer;
```

pgbouncer_override

This systemd drop-in file overrides key options in the default PgBouncer unit file and will never be overwritten even after updating the PgBouncer Linux RPM/DEB packages.

As root, execute the following command and paste the contents of the provided file *pgbouncer_override*:

```Shell
systemctl edit pgbouncer
```

```Shell
#
# systemctl edit pgbouncer
# systemctl daemon-reload
#

[Unit]
Requires=pgbouncer.socket
```

pgbouncer.socket

This is the secret sauce; this file enables PgBouncer to listen on all three ports, 6432, 6433, and 6434. You will note that adding or removing ports is a simple matter of adding or removing addresses as per the format shown in the file below.

As root: create this file and perform a *daemon reload*:

```Shell
vi /etc/systemd/system/pgbouncer.socket
```

```INI
```

```
#
# vi /etc/systemd/system/pgbouncer.socket
# systemctl daemon-reload
#

[Unit]
Description=sockets for PgBouncer
PartOf=pgbouncer.service

[Socket]
ListenStream=6432
ListenStream=6433
ListenStream=6434
ListenStream=/var/run/postgresql/.s.PGSQL.6432

ReusePort=true
RemoveOnStop=true

[Install]
WantedBy=sockets.target
```

Shell

```
systemctl daemon-reload
```

Putting it all together

After all the configuration files have been created and edited, this is what we get:

Shell

```
systemctl restart pgbouncer
```

Shell

```
netstat -tlnp
```

Shell

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:5432                :::*                    LISTEN
tcp6       0      0 :::6432                 :::*                    LISTEN
tcp6       0      0 :::6433                 :::*                    LISTEN
tcp6       0      0 :::6434                 :::*                    LISTEN
```

Finally, perform the following:

1. sudo as Postgres
2. create a database db01
3. create a ROLE usr1 accessing database db01
4. update pg_hba.conf and postgresql.conf allowing localhost connections by PgBouncer

Now validate the ports:

Shell

```
for u in 5432 6432 6433 6434
do
    echo "==== port: $u ==="
    export PGHOST=localhost PGPORT=$u PGDATABASE=db01
    psql "user=usr1 password=usr1" -c "select 'hello world' as greetings"
done
```

And here's the output:

Shell

```
==== port: 5432 ===
 greetings
-------------
hello world
(1 row)

==== port: 6432 ===
 greetings
-------------
hello world
(1 row)

==== port: 6433 ===
 greetings
-------------
hello world
(1 row)

==== port: 6434 ===
 greetings
-------------
hello world
(1 row)
```

Conclusion

Because PostgreSQL was designed from the ground up to work in conjunction with the OS, i.e., a UNIX type of operating system, we've been able to solve an interesting problem using a novel approach. And while systemd is ubiquitous, it isn't normally considered part of a solution to a Postgres puzzle. We call this hacking 🙂

Happy Travels!

Percona Distribution for PostgreSQL provides the best and most critical enterprise components from the open-source community, in a single distribution, designed and tested to work together.

Download Percona Distribution for PostgreSQL Today!

About the Author

Robert Bernier

Robert's first working computer was the very user-friendly IBM 360 with an awesome 4MB RAM. After a round of much needed therapy overcoming the trauma of programming with punch cards he discovered the IBM-XT and the miracle of DOS 2.0. Years later, Robert became enamored with Linux and the opensource world and after meeting one of the members of CORE his primary focus had become all things PostgreSQL. Robert has since then worked in mom and pop companies, fortune 50 corporations and a number of very cool environments including the famed Los Alamos National Laboratory in New Mexico, birthplace of the atomic age. Although reluctant to leave the enjoyable experience of California's Silicon Valley commuter life, he returned to the Pacific Northwest and once again experienced real weather. These days, he serves as the PostgreSQL Consultant here at Percona.

January 15, 2026 Radoslaw Szulgo

Announcing Percona ClusterSync for MongoDB: The Open Source Trail to Freedom

Insight for DBAs MongoDB Percona Software

March 30, 2023 Pete Scott

The Ultimate Guide to Open Source Databases

Insight for DBAs

Ready to get started?

Subscribe to our newsletter for updates on enterprise-grade open source software and tools to keep your business running better.

Email

First Name Last Name

Company Name SUBMIT

By submitting my information I agree that Percona may use my personal data in sending communication to me about Percona services. I understand that I can unsubscribe from the communication at any time in accordance with the Percona Privacy Policy. This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.

Featured
MySQL 5.7 End of Life

Learn
Downloads
Resources
Docs

Products
MySQL
MongoDB
PostgreSQL
Cloud Native
Monitoring
Percona Toolkit

Services
Support
Managed Services
Consulting
Policies
Training

Use Cases
Emergency Support
High-Traffic Events
Performance Tuning
Migrations
Upgrades
Security

Discover
Blog
Community
Percona Community Forum
Events

About
About Percona
Partners
In The News
Careers

Follow us for updates