credativ

PgBouncer Everything, Everywhere, All At Once About This Tool

PART 2

Josef Machytka <josef.machytka@credativ.de>

2026-01-27 - P2D2 2026 Workshop

credativ GmbH

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes, Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 33+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 3+ years of practical experience with different LLMs / AI / ML including architecture and principles
- From Czechia, living now 12 years in Berlin

- LinkedIn: linkedin.com/in/josef-machytka
- Medium: medium.com/@josef.machytka
- YouTube: youtube.com/@JosefMachytka

- GitHub: github.com/josmac69/conferences_slides
- ResearchGate: researchgate.net/profile/Josef-Machytka

All My Slides:

Recoded talks:

What We Will Cover in Part 2

- PgBouncer Overview
- Prepared Statements in PgBouncer
- Threads Architecture
- PgBouncer Use Cases
- PgBouncer Resources

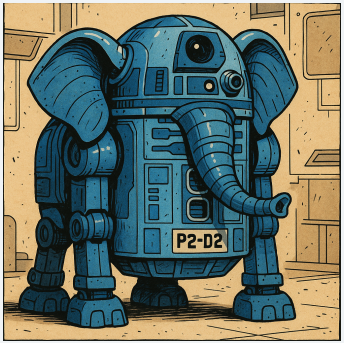


Image created by ChatGPT

credativ

PgBouncer Overview

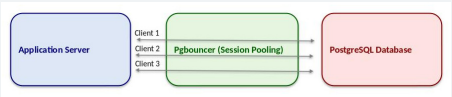
PgBouncer Modes & Methods

credativ

- Supports multiple pooling modes:
  - Session pooling - one client connection per server connection
  - Transaction pooling - server connection assigned per transaction
  - Statement pooling - server connection assigned per statement
- Supports authentication methods:
  - MD5 (deprecated in PostgreSQL 18)
  - SCRAM-SHA-256 (recommended)
  - Plaintext (not recommended)
- Can use auth_query to authenticate against PostgreSQL
- auth_hba_file - file with format like pg_hba.conf for PgBouncer
- Latest v1.25 supports LDAP authentication
- Provides monitoring and statistics via admin console

Session Pooling

- Each client connection gets a dedicated server connection
- Connection remains assigned for entire session
- Pros:
  - Preserves full session state (setting, temp tables, prepared statements)
  - Compatible with all applications
- Cons:
  - PgBouncer cannot reduce total number of server connections
  - Idle connections still consume resources on server
  - No change in overall throughput under high connection counts



Transaction Pooling

credativ

- Server connection assigned per transaction
- After transaction ends (COMMIT/ROLLBACK), connection returned to pool
- Limited support for session state within transactions
- Pros:
  - Better resource utilization, higher throughput
  - Preserves session state within transactions
  - Allows significant reduction in server connections
- Cons:
  - Some application session state not preserved
  - Applications must not rely on session state outside transactions
  - Since v1.25 new setting transaction_timeout on global/user level

Transaction Pooling

credativ

- Applications must avoid:
  - Session-level settings outside transactions (SET / RESET)
  - Temporary tables outside transactions (CREATE TEMP TABLE with ON COMMIT PRESERVE ROWS)
  - Advisory locks outside transactions (pg_advisory_lock / pg_advisory_unlock)
  - Dynamically load extensions libraries into session (LOAD)
  - SQL Cursor WITH HOLD outside transactions (DECLARE CURSOR … WITH HOLD)
- Limited support for prepared statements outside transactions since v1.21
- Suitable for stateless applications
- Common in web applications using ORMs
- Some libraries repeat settings with each transaction
- PgBouncer will accept wrong statements without warnings (!)
- Amazon RDS Proxy is able to detect it and hold the connection

Statement Pooling

credativ

- Server connection assigned per statement
- After statement execution, connection returned to pool
- Forced autocommit mode, no multi-statement transactions
- Pros:
  - Maximum resource utilization, lowest latency
  - Suitable for very high-concurrency workloads
  - Allows drastic reduction in server connections
- Cons:
  - Very limited support for session state
  - Applications must be fully stateless
  - Multi-statement transactions not allowed

What PgBouncer cannot do & Gotchas

credativ

- Does not do High Availability - use HAProxy, etc.
- Does not parse SQL queries, cannot optimize queries
- Does not cache query results
- Does not manage transactions - in statement mode autocommit is forced
- Result of CANCEL command is unpredictable in transaction/statement mode
- Has some dangerous timeouts - be careful when setting them:
  - **query_timeout** - cancels long-running queries (default 0 = disabled)
  - **client_idle_timeout** - disconnects idle clients (default 0 = disabled)
  - **idle_transaction_timeout** - disconnects idle transactions (default 0 = disabled)
  - **transaction_timeout** - cancels long-running transactions (default 0 = disabled)
- Use server side timeouts in PostgreSQL for safety - where possible

How to size PgBouncer connection pool

credativ

- **default_pool_size** -> number of server connections per user/database pair
  - -> we must estimate how many active concurrent connections can server handle
  - -> unknown or mixed workload -> start with 2x number of CPU cores on server
- **max_client_conn** -> maximum number of client connections PgBouncer will accept
  - -> very cheap to allow many client connections - only around 2KB per connection
  - -> can be set to high value (1000s) to allow connection spikes -> but there are limits
  - -> file descriptors are main resource limit - network connection treated like a file
  - -> number of ports - every TCP connection requires a unique port number
  - -> reuse of TCP ports when connection close takes time (TIME_WAIT state) -> for short-lived connections
  - -> there might be limit on firewall/NAT device in front of PgBouncer
- But single-threaded pgbouncer can become bottleneck under very high client connection counts

How PgBouncer handles connection spikes

credativ

- Main configuration parameter - **default_pool_size**
- Number of server connections per user/database pair
- -> Can be changed in the per-database configuration
- Excess client connections are suspended & queued
- -> application sees it as ”hang” or ”blocked” - waiting for slot
- **query_wait_timeout** - max time to wait for a server connection
- -> after timeout client connection is closed with ”query_wait_timeout” error
- **reserve_pool_size** - number of additional connections allowed for emergencies
- **reserve_pool_timeout** - time to wait for a connection from reserve pool
- Used when all regular pool connections are busy for too long
- When not used, pool returns to normal size
- Number of CPUs on PostgreSQL server limits number of concurrent connections (!)
- pool + reserve pool + eventual parallel workers must not overload DB server

PgBouncer Special Users

credativ

- **admin_users** - users allowed to connect to PgBouncer admin console
- **stats_users** - users allowed to view statistics in PgBouncer admin console
- Users can be in both groups
- Admin console listens on same port as PgBouncer
- Connect using special database name "pgbouncer"
- Example: `psql -h pgbouncer_host -p pgbouncer_port -U admin_user pgbouncer`
- Common admin console commands:
  - `SHOW POOLS;` - view connection pool status
  - `SHOW STATS;` - view PgBouncer statistics
  - `RELOAD;` - reload configuration without disconnecting clients
  - `PAUSE;/RESUME;` - pause/resume client connections
  - `KILL <conn_id>;` - terminate specific client connection

PgBouncer Authentication

credativ

- auth_type - authentication method
  - cert - uses client SSL certificates
  - md5 - uses MD5 hashed passwords
  - scram-sha-256 - uses SCRAM-SHA-256 hashed passwords
  - plain - uses plaintext passwords (not recommended)
  - trust - allows all connections without password (not recommended)
  - any - like "trust", but ignores username (not recommended)
  - hba - uses auth_hba_file for authentication rules
  - pam - uses PAM for authentication
  - ldap - uses LDAP for authentication (since v1.25)

PgBouncer Authentication – userlist.txt example

credativ

- Very simple format: ”username” ”password”
- Passwords can be in plaintext, MD5 or SCRAM-SHA-256 format
- Plain text is security risk - avoid if possible

```
"app_user" "md545f2603610af569b6155c45067268c6b"    ; MD5 hash of "AppPassword123"
"pgbouncer" "secretpassword"                        ; plaintext
"monitor_user" "SCRAM-SHA-256$4096:...$...$..."     ; SCRAM secret - retrieved from PostgreSQL
```

PgBouncer Authentication

credativ

- auth_file - path to userlist.txt file with usernames and passwords
- auth_hba_file - path to file with authentication rules (like pg_hba.conf)
- auth_query - SQL query to authenticate users against PostgreSQL
- Example:

```
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
auth_query = SELECT username, password from pgbouncer.get_auth($1)
auth_user = pgbouncer
```

- $1 is placeholder for username provided by client
- Query must return exactly one row with username and password

PgBouncer/PostgreSQL SSL/TLS Configuration

credativ

- PostgreSQL server must be configured to support SSL/TLS connections
- TLS must be enabled on PgBouncer server

```
### postgresql.conf on PostgreSQL server
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'         # private key
ssl_ca_file = 'ca.crt'              # CA that signed client certs (if required)

### pgbouncer.ini on PgBouncer server
client_tls_sslmode = require
client_tls_cert_file = /etc/pgbouncer/pgbouncer.crt
client_tls_key_file  = /etc/pgbouncer/pgbouncer.key
client_tls_ca_file   = /etc/pgbouncer/ca.crt    # CA to validate client certs

server_tls_sslmode = verify-ca
server_tls_ca_file = /etc/pgbouncer/ca.crt          # CA to verify Postgres server cert
server_tls_cert_file = /etc/pgbouncer/pgbouncer_to_pg.crt
server_tls_key_file  = /etc/pgbouncer/pgbouncer_to_pg.key
```

Image from the article

Prepared Statements in Transaction Pooling

- Before v1.21, prepared statements not supported in transaction/statement pooling
- ERROR: prepared statement ”stmt1” does not exist
- PgBouncer 1.21 enabled protocol-level named prepared statements
- Tracks & rewrites prepared statements into new connection
- New setting `max_prepared_statements` max number of PS per-connection
- PgBouncer keeps all prepared statements in an internal cache
- From v1.22 it can handle DISCARD ALL/DEALLOCATE ALL
- Prepared Statements in Transaction Mode for PgBouncer

Find more in my talk About PostgreSQL Prepared Statements

Protocol-Level Prepared Statements

credativ

- Communication between client and PostgreSQL uses ”wire protocol”
- See in PostgreSQL documentation Chapter 54. Frontend/Backend Protocol
- Part 54.2.3. Extended Query describes prepared statements
- Messages used for prepared statements:
  - Parse - create a prepared statement on the server
  - Bind - attach parameters to a prepared statement
  - Execute - execute the prepared statement with parameters
  - Close - deallocate the prepared statement (added in PostgreSQL 17)
- PgBouncer intercepts these messages and puts prepared statements into internal cache
- It cannot track plain SQL commands for prepared statements
- E.g., plain SQL commands PREPARE, EXECUTE, DEALLOCATE must be avoided

Prepared Statements in Transaction Pooling

credativ

- Application must avoid recycling prepared statement names
- If return value or argument types change, PostgreSQL returns an error
- ERROR: cached plan must not change result type
- Typical situation during DDL changes - adding/removing columns, changing types
- In this case we must run RECONNECT in PgBouncer admin console
- This will force re-preparing of all prepared statements
- Feature requires PgBouncer v1.22 or newer + libpq 17 or newer (PG17+)
- Prepared statements must be used via library like JDBC, Psycopg3, etc.

PgBouncer CPU Usage

credativ

- PgBouncer is currently (2025/11) still only single-threaded process
- Basic architecture: events, callbacks, event loop (dispatches events to callbacks)
- Everything is an event - one thread can handle thousands of sockets
- Thread listens to TCP/Unix socket and processes events
- Low CPU and memory overhead per connection
- But limited to single CPU core -> scalability bottleneck

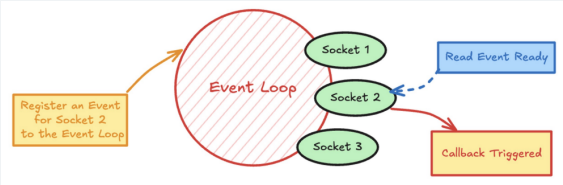


Image from slides PgBouncer in Parallel: Bringing Multithreading to the Trusted Postgres Pooler

Multi-Threaded PgBouncer

- Developers work on multi-threaded version already
- Large pull request Support Multithread Mode #1386 is open and discussed
- PGConf NYC 2025 talk presented a multithreaded prototype
- Main dispatcher thread will listen on the port
- Will distribute new connections to worker threads
- New parameter thread_number to configure number of workers
- Default is 1 (current single-threaded behavior)

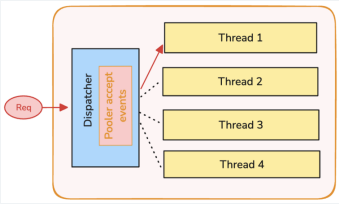
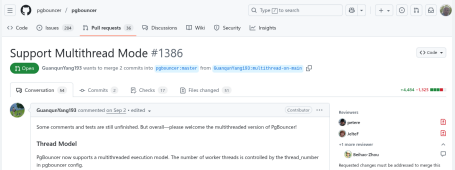


Image from slides PgBouncer in Parallel: Bringing Multithreading to the Trusted Postgres Pooler

Multi-Threaded PgBouncer

- Each worker has its own event loop and subset of connections
- Pool size will be configured per-thread
- Each connection is handled by exactly one worker thread
- *_pool_size settings will be applied on a per-thread basis
- ”Takeover” (restarting PgBouncer without dropping connections) will NOT be supported initially
- Feature is still under development
- PR #1386 suggests work is finishing soon



Current Solution - Multiple PgBouncer Instances

credativ

- With current version we can run multiple PgBouncer instances
- Setting SO_REUSEPORT = 1 allows multiple processes to listen on same port
- Uses templated systemd service file to run multiple instances
- Each instance listens on same port, kernels distributes connections
  - /etc/systemd/system/pgbouncer@.socket - creates TCP socket on port 6432
  - /etc/systemd/system/pgbouncer@.service - uses main socket, creates per instance sockets
  - Instances started as: systemctl start pgbouncer@50001, systemctl start pgbouncer@50002, ...
  - Numbers are the administration ports for each instance
  - Make sure ports are not used by other processes
- Requires PgBouncer 1.14 or newer with support for socket activation
- Example: github.com/josmac69/pgbouncer_workshop/pgbouncer/multiple_instances_example

- Postgres at Scale: Running Multiple PgBouncers
- Running Multiple PgBouncer Instances with systemd

Running Multiple PgBouncer Instances

credativ

- Socket file /etc/systemd/system/pgbouncer@.socket

```
[Unit]
Description=sockets for PgBouncer

[Socket]
ListenStream=6432
ListenStream=%i
ListenStream=/tmp/.s.PGSQL.%i

ReusePort=true

[Install]
WantedBy=sockets.target
```

Example from article Running Multiple PgBouncer Instances with systemd

Running Multiple PgBouncer Instances

credativ

- Per instance sockets /etc/systemd/system/pgbouncer@.service

```
[Unit]
Description=connection pooler for PostgreSQL (%i)
After=network.target
Requires=pgbouncer@%i.socket

[Service]
Type=notify
User=postgres
ExecStart=/usr/local/bin/pgbouncer /etc/pgbouncer/pgbouncer.ini
ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGINT

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

Example from article Running Multiple PgBouncer Instances with systemd

Running Multiple PgBouncer Instances

credativ

```
## Start multiple PgBouncer instances
systemctl start pgbouncer@50001
systemctl start pgbouncer@50002
systemctl start pgbouncer@50003
systemctl start pgbouncer@50004
....

## For production traffic connect to port 6432
psql -h somehost -p 6432 -d database -U user

## For maintenance connect to specific instance port
psql -h somehost -p 50001 -d pgbouncer -U pgbouncer -c 'SHOW STATS;'
```

Example from article Running Multiple PgBouncer Instances with systemd

credativ

Other Use Cases of PgBouncer

Experimental Use Case: Capping User Connections

credativ

- PgBouncer can limit connections per user/database pair
- Prevents a single user from exhausting all connections
- Parameters:
  - max_db_connections limits per database
  - max_user_connections limit per user
  - max_user_client_connections limit per user over all databases
- Example: github.com/josmac69/pgbouncer_workshop/pgbouncer/capping_user_connections

Use Case: PgBouncer Listening on Multile Ports

credativ

- Can happen after consolidation of multiple PostgreSQL servers
- One PgBouncer instance can listen on multiple ports
- Setting `listen_port` allows only one port
- But problem can be solved using systemd .socket unit
- Example: github.com/josmac69/pgbouncer_workshop/pgbouncer/multi_port

```
## /etc/systemd/system/pgbouncer.socket

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

Configuring PgBouncer for Multi-Port Access

Use Case: Dedicated PgBouncer per Tenant/Team

credativ

- Each tenant/team has its own PgBouncer instance
- Isolates authentication and workloads between tenants/teams
- Simplifies setting limits per tenant
- Each tenant/team has its own pgbouncer.ini and userlist.txt files
- Allows usage of specific per tenant/team users for database connections
- Prevents some types of noisy neighbor issues between tenants/teams
- Simplifies user management - one huge pg_hba.conf not needed
- Example: github.com/josmac69/pgbouncer_workshop/pgbouncer/tenant_isolation

- Introducing a PgBouncer authentication layer into our database architecture

Use Case: One PgBouncer Proxying Multiple Databases

credativ

- One PgBouncer instance can proxy connections to multiple databases
- Simplifies connection management for applications
- Allows for shared connection pooling across databases
- Useful for applications with many small databases with low traffic
- Reduces resource consumption compared to multiple PgBouncer instances
- Require user mapping per database to avoid conflicts in user names
- postgres1 = postgres/database1, postgres2 = postgres/database2, ...
- Example: github.com/josmac69/pgbouncer_workshop/pgbouncer/proxy_multiple_dbs

Use Case: Serverless Applications

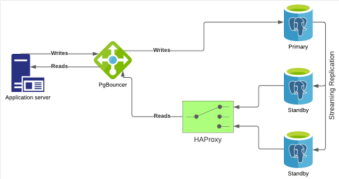
credativ

- Serverless applications often have unpredictable connection patterns
- Can produce connection spikes that overwhelm PostgreSQL server
- PgBouncer can smooth out connection spikes with pooling and queuing
- Reduces connection overhead for short-lived serverless functions
- Improves overall application responsiveness and stability
- Deploy PgBouncer as a lightweight service in the database network
- Configure serverless functions to connect via PgBouncer port

- Serverless Connection Pooling with AWS Lambda and PgBouncer on RDS: A Step-by-Step Guide

Use Case: Read/Write Splitter

- Some applications use separate databases for reads and writes
- Writes go to primary database, reads from replicas
- PgBouncer does not natively support read/write splitting
- It does not parse SQL queries to determine type
- Split must be done at application level via user names
- Solution can further balance reads across replicas using HAProxy



Taking Advantage of Write-Only and Read-Only Connections in PgBouncer with Django

Experimental Use Case: Pooling FDW Connections

credativ

- Foreign Data Wrappers (FDW) allow access to external data sources
- Each FDW connection creates a new PostgreSQL connection
- Can lead to many open connections and resource exhaustion
- PgBouncer can be used to pool FDW connections
- Configure FDW to connect via PgBouncer port
- Reduces number of open connections to external data source

PgBouncer Resources

credativ

- Other useful resources:
- PgBouncer is useful, important, and fraught with peril
- PgBouncer Connection Pooler for Postgres Now Supports More Session Vars
- Prepared Statements in Transaction Mode for PgBouncer
-
-
-
-

Thank you for your attention!

credativ

All my slides

Recorded talks