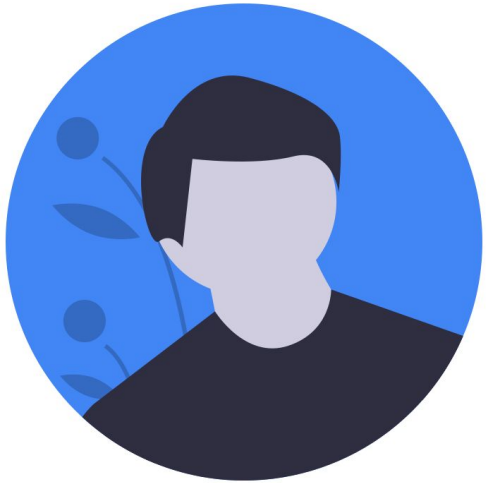
PostgreSQL connections at scale

About me



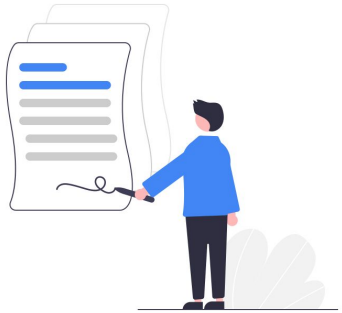
https://bhuvane.sh/

Bhuvanesh (aka The DataGuy)

Data Architect + DBA

- https://linkedin.com/in/rbhuvanesh/
- BhuviTheDataGuy
- BhuviTheDataGuy
- BhuviTheDataGuy (Tech Blog)
- https://thedataguy.in (Tech Blog)

Disclaimer



This presentation is based on my personal experience & research about PostgreSQL and pgbouncer.

No copyright !! Nothing reserved !!!

Icons used - https://undraw.co/

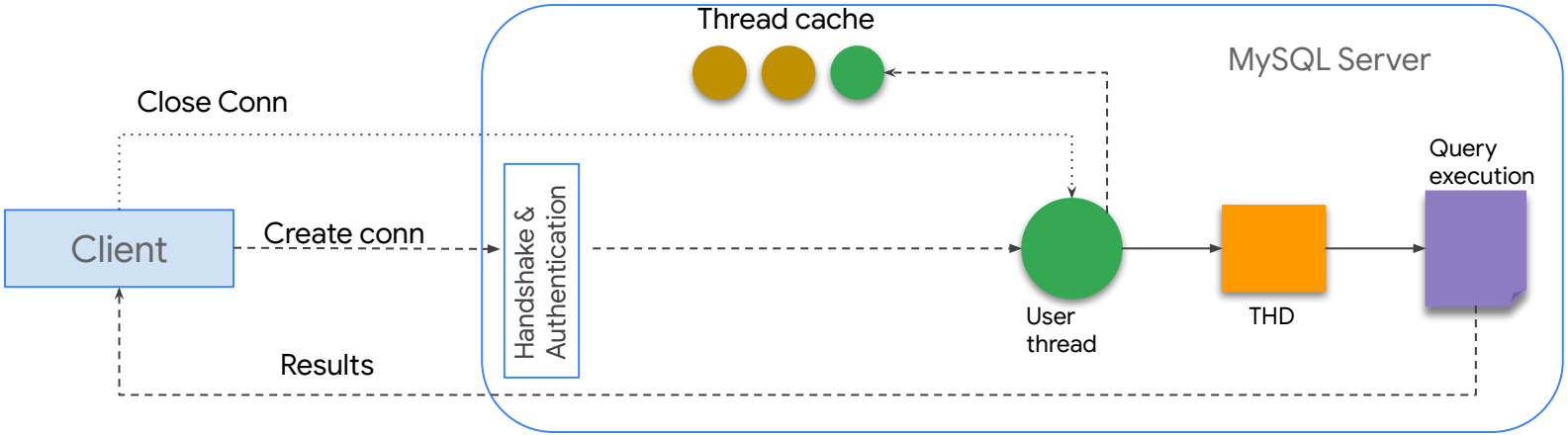
GIFs used - https://giphy.com/

Agenda

1. A database connection and its **Cost**
2. Need for a connection pooler
3. Pgbouncer - Introduction
4. How pgbouncer works and its features
5. Monitoring the pgbouncer
6. Pgbouncer deployment best practices

Why PostgreSQL doesn't scale to a large number of connections?

Life cycle of a connection in MySQL



Reference: *https://mysqlserverteam.com/mysql-connection-handling-and-scaling/*

Cost of a single PostgreSQL connection

- Each connection is a process fork with roughly consume **10 MB** memory (*a research from heroku*)
- Each connection will simultaneously open up to **1000 files** (*default configuration*)

Example:

Let's assume your DB is consuming 400 connections, then

- 10MB * 400 connections = 4GB Memory
- 1000 Files * 400 connections = 4,00,000 Files

What about Idle connections?

States of PostgreSQL connection:

Active	currently running
Idle	Its not doing anything
Idle in transaction	currently not doing anything and could be waiting for an input
Idle in transaction(aborted)	the connections that were idle in the transaction that have since been aborted

"Idle connections are not just idle, they eat the resources in all the ways."

Cost of the idle connections

A benchmark from AWS:

- Open 100 connections.
- Leave the connections idle for 10 minutes.
- Close the connections.

Reference: https://aws.amazon.com/blogs/database/resources-consumed-by-idle-postgresql-connections/

More incoming connections?

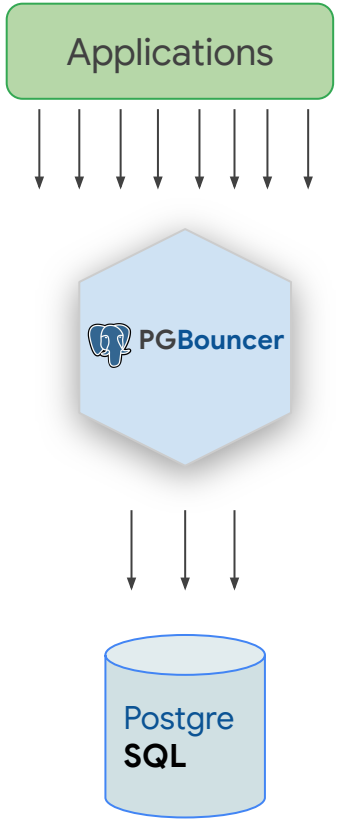
Then your PostgreSQL server is in Danger!

We need a Superhero

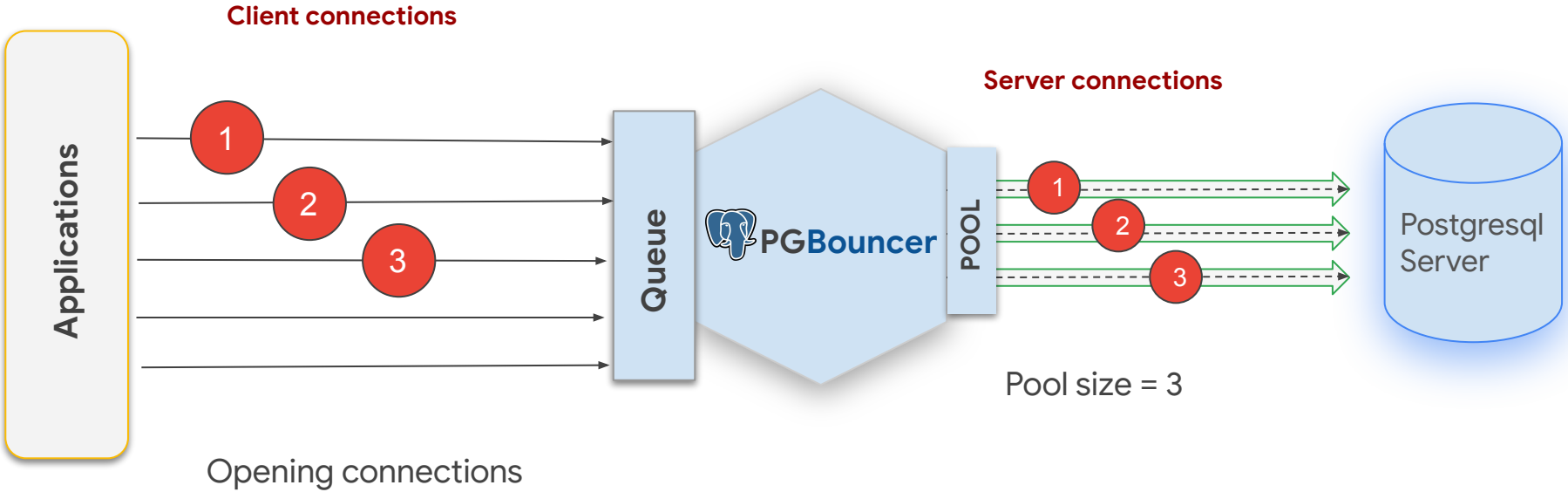
Not the Black Panther,
but
pgbouncer

pgbouncer

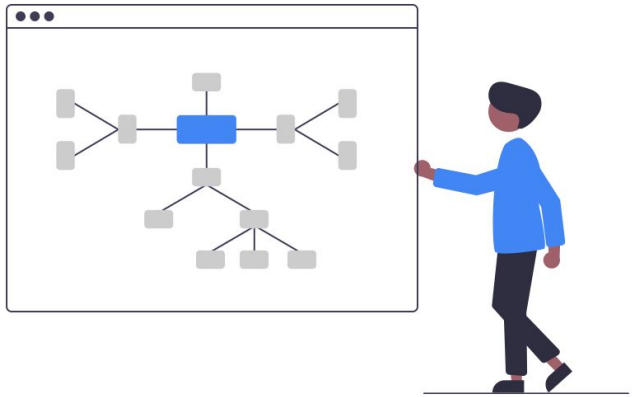
- Lightweight connection pooler
- Reduce the number of backend connections
- Connections economy (creating a connection is a fork and acquire a ProcArrayLock)
- No need for special authentication



What does a connection pooler do?



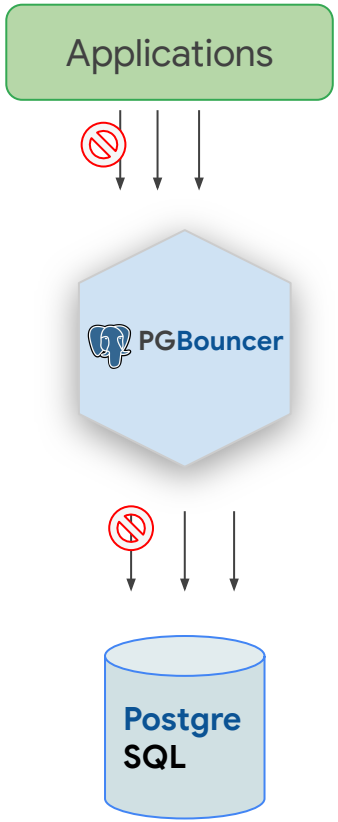
Lets deep dive into pgbouncer



Pgbouncer pooling mode

Session

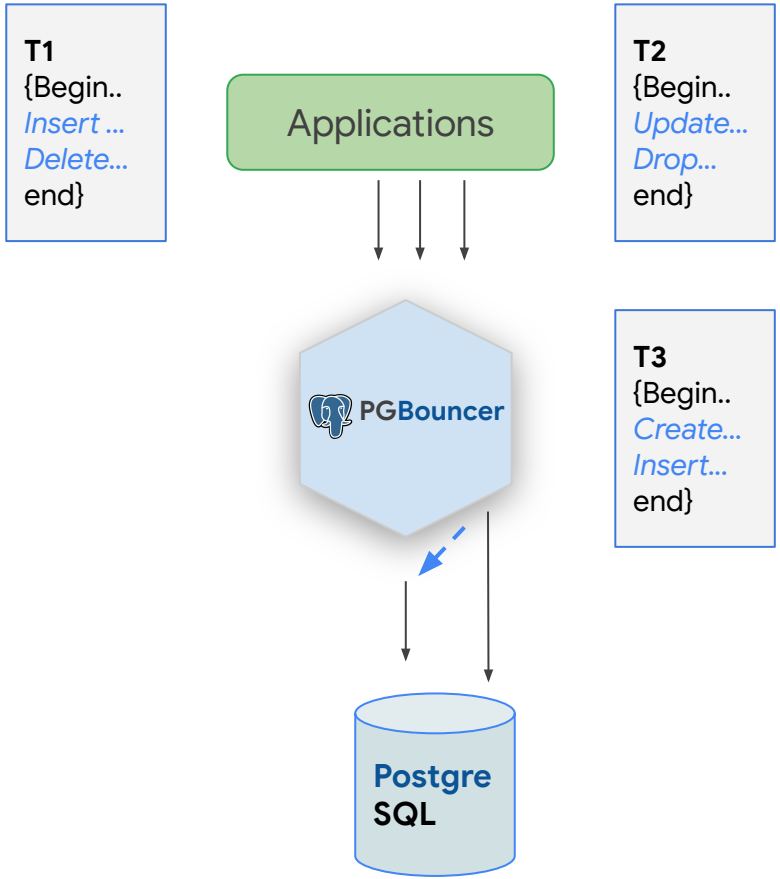
- Each client(app) connection will open backend connections and the connection will remain open until the client closes.
- Client and Server connections are mapped
- It is almost similar to using the database directly without any connection pooler.
- The connection is transparent.



Pgbouncer pooling mode

Transaction

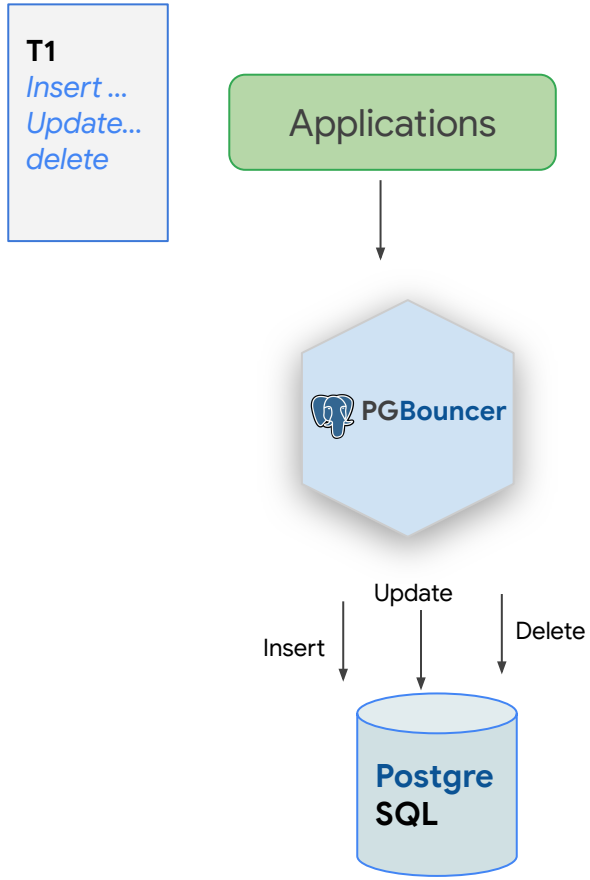
- Client connect to the server connection only during the transaction, after the transaction the same server connection can be used to run another session's transaction.
- All the queries inside the {begin... end} will be executed in one server connection.
- Session variables and prepared statements will not work here.
- The connection is not transparent.



Pgbouncer pooling mode

Statement

- Very aggressive mode.
- The server connection back to the pool when the statement completes.
- Multi statement is not possible here.
- Auto commit is always on in this mode.



Adding the PostgreSQL Server

Like a connection string

```
;; Common patterns
db1 = host=primary dbname=db1
db2 = host=standby dbname=db2
* = host=standby

;; Custom pool size
db2 = host=standby dbname=db2 pool_size=50
reserve_pool=10
```

Pool size value can be override based on users

- Pool_size =10
- No of databases = 10
- No of users = 2
- Pool size can be override to **(no of db * no of users) = 20**

List of parameters:

- dbname
- host
- port
- user
- password
- auth_user
- client_encoding
- datestyle
- timezone
- pool_size
- reserve_pool
- max_db_connections
- pool_mode
- connect_query
- application_name

Authentication

Auth File with auth type

- A **txt file that contains username and the password**
- Password can be plain text of *MD5* hash(recommended)
- Supported auth types:
  - any
  - trust
  - plain
  - md5
  - cert
  - hba
  - pam

pg_hba.conf

- Its very similar to postgresql's hba methold.
- Same postgresql's syntax will work here.
- But LDAP, pam and a few other methods will not work.

Auth_user with query

- Automatically loads the username and password from the target database.
- Just give an user and password(mention it in the **auth_file**) for the authentication(like a dedicated user), then it'll fetch the user, password from the database.
- ***example***:

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

Authentication cont...

```
;; auth file
auth_type = trust
auth_file = /etc/pgbouncer/userlist.txt

;; HBA-style
;auth_hba_file =

;; Auth user with query
;auth_query = SELECT usename, passwd FROM pg_shadow WHERE usename=$1
```

Connections and Pool

Parameter	Description
`max_client_conn`	Maximum number of connection allowed in pgbouncer
`default_pool_size`	How many server connections to allow per user/database pair.
`min_pool_size`	Minimum number of server connections
`reserve_pool_size` `reserve_pool_timeout`	If the pool is full, and a connection is waiting more than the reserve_pool_timeout then it'll use extra connection from this reserve pool. It should be **less in size**.

Example 1: How the connection pool makes the connection

Connection:

```
db1 = host=localhost dbname=db1
```

Pool:

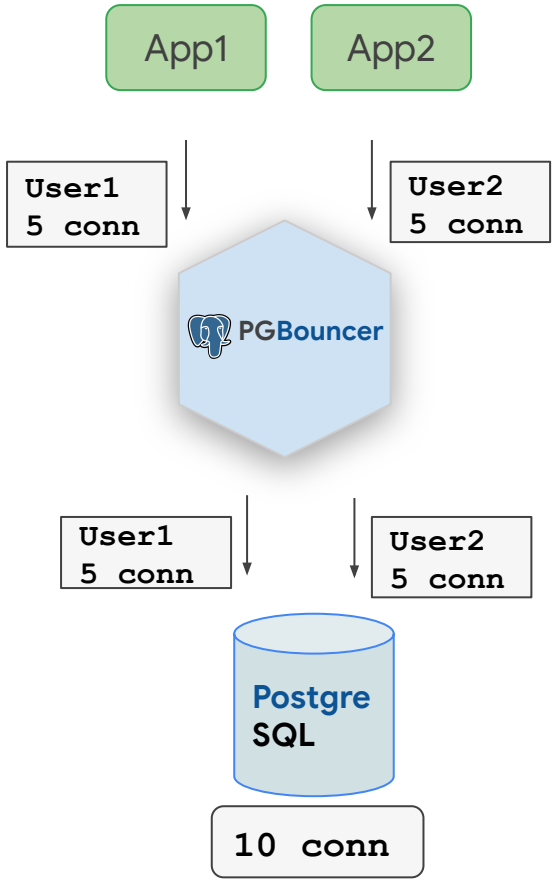
```
default_pool_size = 20
pool_mode = session
```

Auth file:

```
user1:xxxxxx
user2:yyyyy
```

Connection scenario:

- App1 is connecting to the DB via user1 with ***5 connections***
- App2 is connecting to the DB via user2 with ***5 connections***



Example 2: where the pool values override

Connection:

```
db1 = host=localhost dbname=db1
```

Pool:

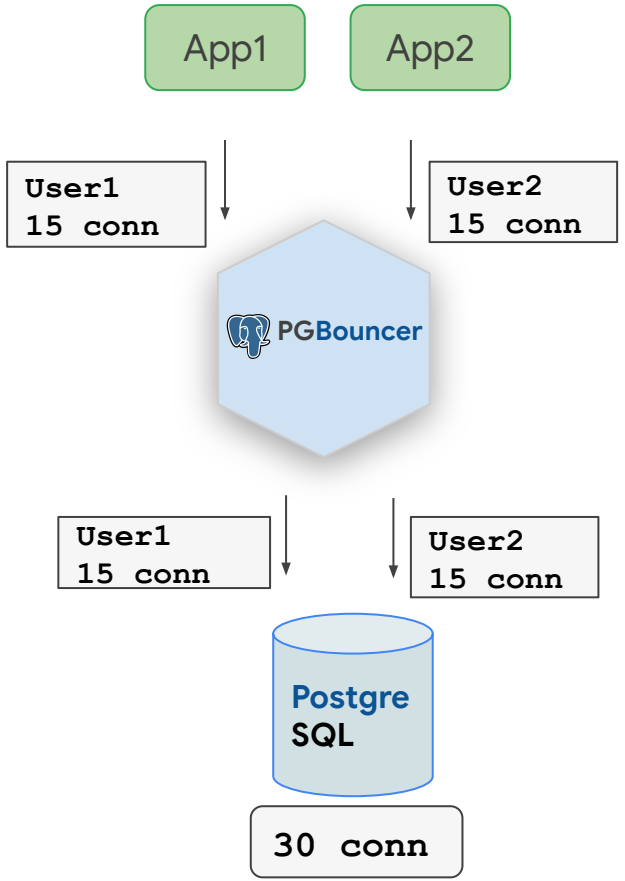
```
default_pool_size = 20
pool_mode = session
```

Auth file:

```
user1:xxxxxx
user2:yyyyy
```

Connection scenario:

- App1 is connecting to the DB via user1 with ***15 connections***
- App2 is connecting to the DB via user2 with 1***5 connections***



Set the Hard limit for the pool size

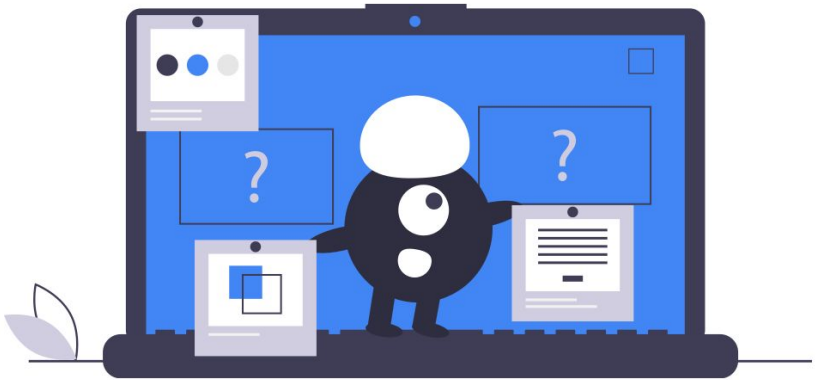
Option 1: Set the limit based on DB or user level

Parameter	Description
max_db_connections	Pgbouncer will not allow more than this value for a database
max_user_connections	Maximum pool for a user

Option 2: Define separate connections for each DB and set the pool size via connection

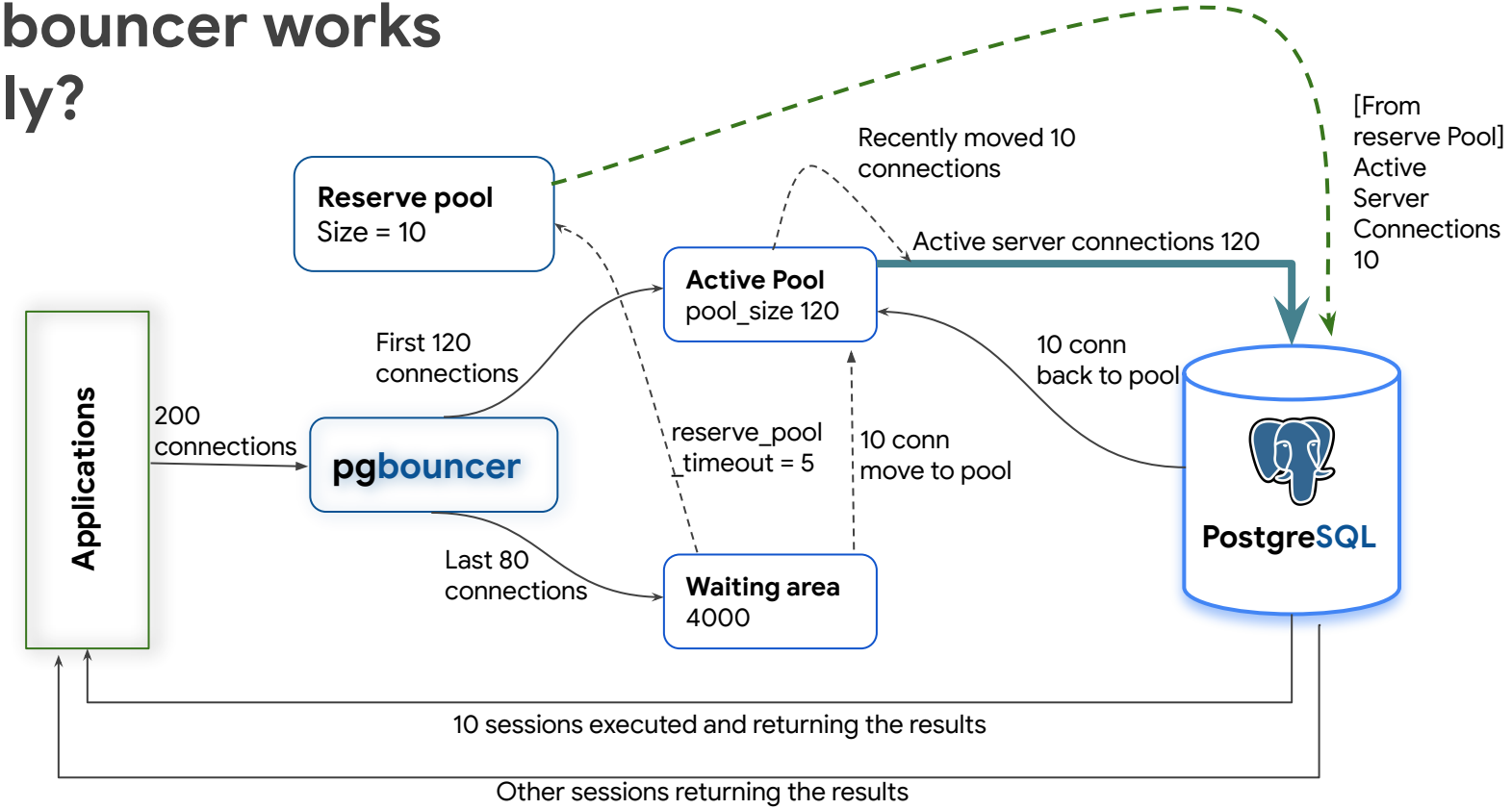
```
db1 = host=34.72.164.39 dbname=db1
pool_size=10
db2 = host=34.72.164.39 dbname=db2
pool_size=10

                [OR]

db1 = host=34.72.164.39 dbname=db1
pool_size=10 max_db_connections=10
db2 = host=34.72.164.39 dbname=db2
pool_size=10 max_user_connections=10
```

Lets see the real world example

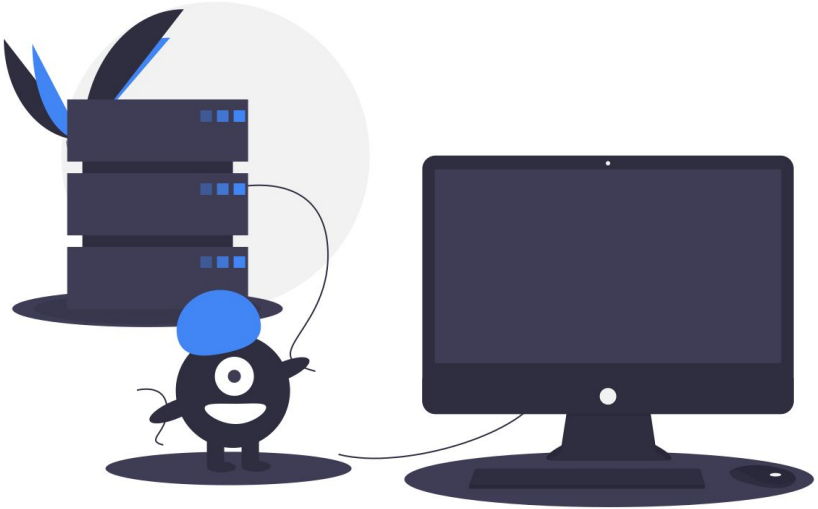


- PostgreSQL max connections = 300
- No of databases = 1
- No of users = 1
- Pool size = 120
- Pool mode = session
- Max client connections = 4k
- Reserve pool size = 10
- Reserve pool timeout = 10s

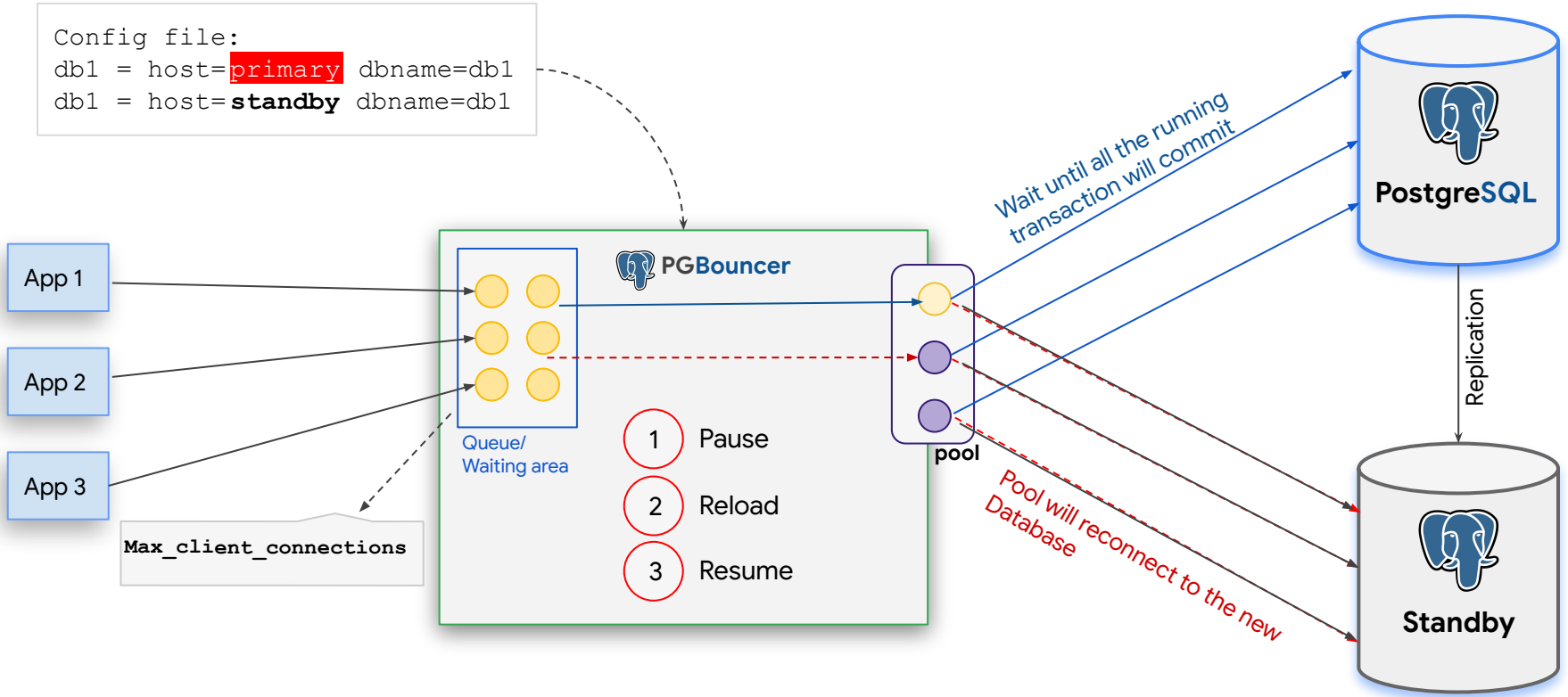
How pgbouncer works internally?



Zero downtime
Maintenance



Seamless switchover to a standby



Restart without an actual restart

`-R`, `--reboot`

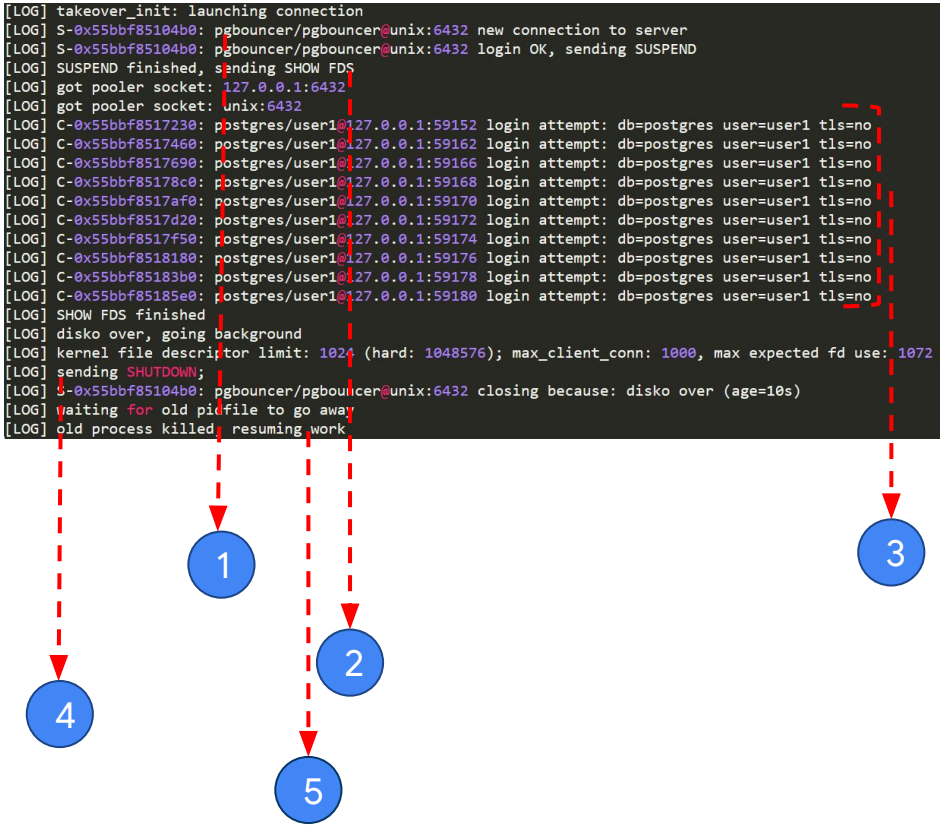
Do an online restart. That means connecting to the running process, loading the open sockets from it, and then using them. If there is no active process, boot normally.

Use this in your systemctl or systemd file:

```
pgbouncer -R /etc/pgbouncer/pgbouncer.ini
```

Online Restart Flow

1. Create new pgbouncer instance, make server connections
2. Send SUSPEND command to the old pgbouncer instance
3. Transfer the pool from old instance to new instance
4. Shutdown the old instance
5. Resume the work on the new instance



Monitoring

CLI and Prometheus exporter

Monitor via CLI

Pgbouncer config:

- Max pool = 300
- Max client = 800

Connect to pgbouncer db:

```
psql -h 127.0.0.1 -p 6432 -U
user1 pgbouncer
```

Run a pgbench test:

```
pgbench -h 127.0.0.1 \
-U user1 -C -c 800 \
-j 2 \
-t 10000 \
-p 6432 \
postgres
```

```
pgbouncer=# show pools;

-[ RECORD 1 ]---------
database   | postgres
user       | user1
cl_active  | 216
cl_waiting | 582
sv_active  | 216
sv_idle    | 0
sv_used    | 0
sv_tested  | 0
sv_login   | 0
maxwait    | 12
maxwait_us | 880893
pool_mode  | session
```

Monitoring with the prometheus exporter and Grafana

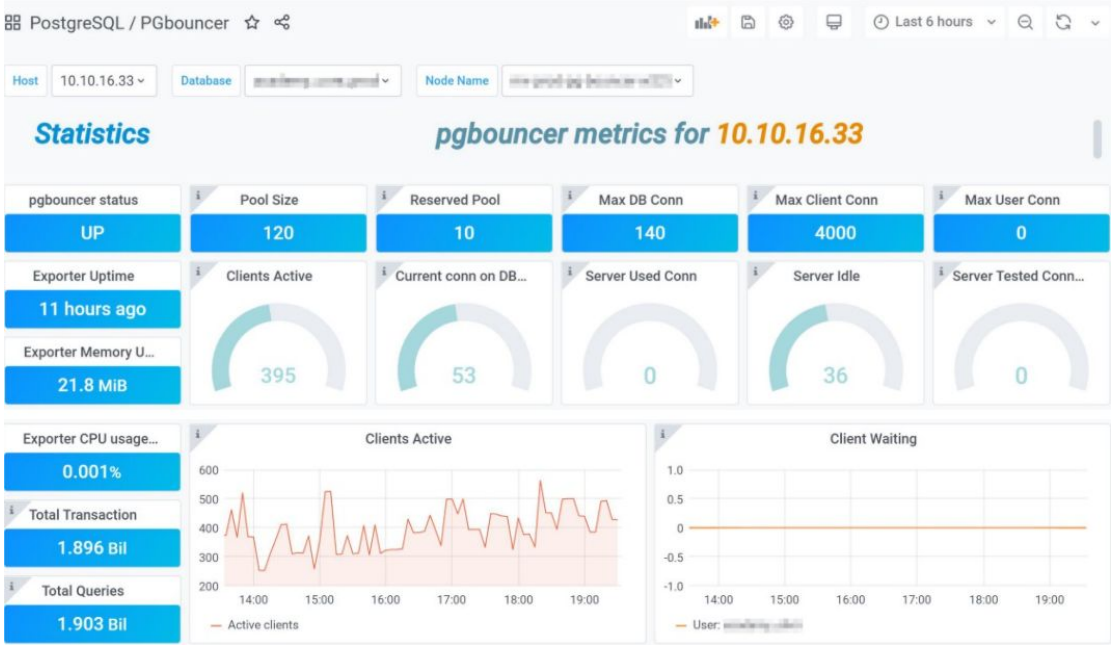
Prometheus exporter:

```
pip3 install
prometheus-pgbouncer-exporter
```

Grafana Dashboard:

The dashboard is open source(*we developed this dashboard*)

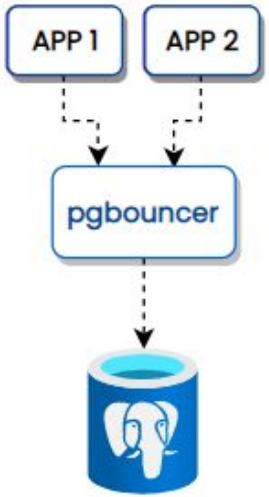
Dashboard ID: **13353**



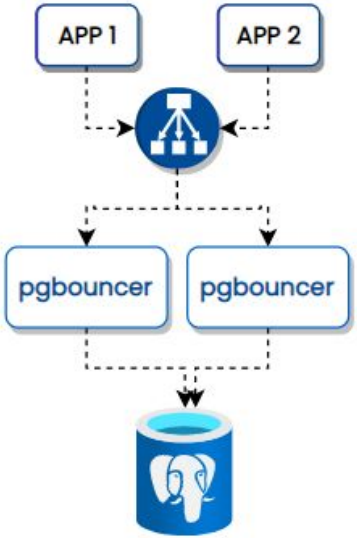
Reference: https://medium.com/searce/grafana-dashboard-for-pgbouncer-and-monitor-with-percona-pmm-3170d3eb4d14

Pgbouncer deployment pattern

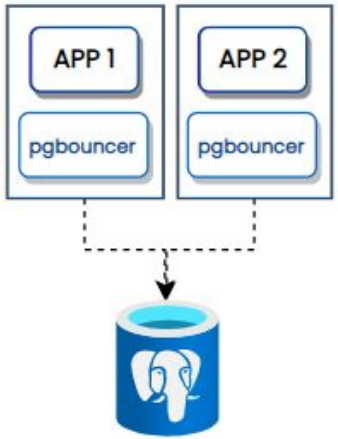
Pattern 1



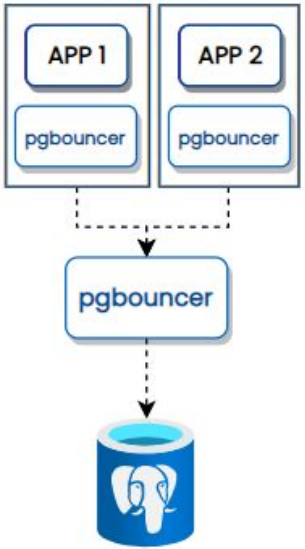
Pattern 2



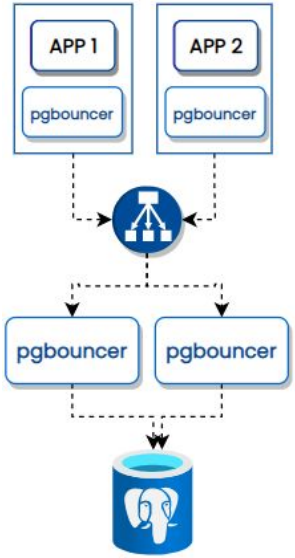
Pattern 3



Pattern 4



Pattern 5



What I didn’t cover?

1. Timeout parameters
2. High availability of pgbouncer
3. TLS
4. server_reset_query - an important parameter
5. And more

Thank you !!!

And any Questions?

Get this deck here:
bit.ly/pgbouncer-deck