PROXYSQL - A PERFECT COMPLEMENT TO YOUR POSTGRESQL DATABASE

May 27, 2026

René Cannaò / Ronald Bradford

ABOUT THE PRESENTERS

RENE' CANNAO'

Author and CEO of ProxySQL

- Author of ProxySQL since 2013
- Over 2 decades as system, network, and database administrator
- Last 20 years focused mainly on MySQL performance

RONALD BRADFORD

ProxySQL CTO

- Data Architect and Data Strategist
- Ingres/Oracle/MySQL/AWS RDS
- 2+ decades open-source contributor
- Author | Speaker | AWS Certified Database Speciality

AGENDA

1. The PostgreSQL middle tier today
2. Where pooling stops being enough
3. ProxySQL for PostgreSQL
4. Evaluating transition from PgBouncer to ProxySQL
5. Benchmark results
6. Demo
7. Announcing new product offerings for PostgreSQL

POSTGRESQL TOOL ECOSYSTEM

CURRENT POSTGRESQL MIDDLE-TIER TOOLS

- **PgBouncer** - focused connection pooling
  - https://www.pgbouncer.org/
- **Pgpool-II** - Connection pooling, load balancing and high availability
  - https://www.pgpool.net/
- **pgcat** - Connection pooling, load balancing, failover, mirroring, sharding
  - https://github.com/postgresml/pgcat
- **Adjacent HA / topology tools** - Patroni, repmgr, pg_auto_failover, managed-service control planes
- **Other proxies / poolers** - Odyssey, Heimdall Data, HAProxy-based compositions

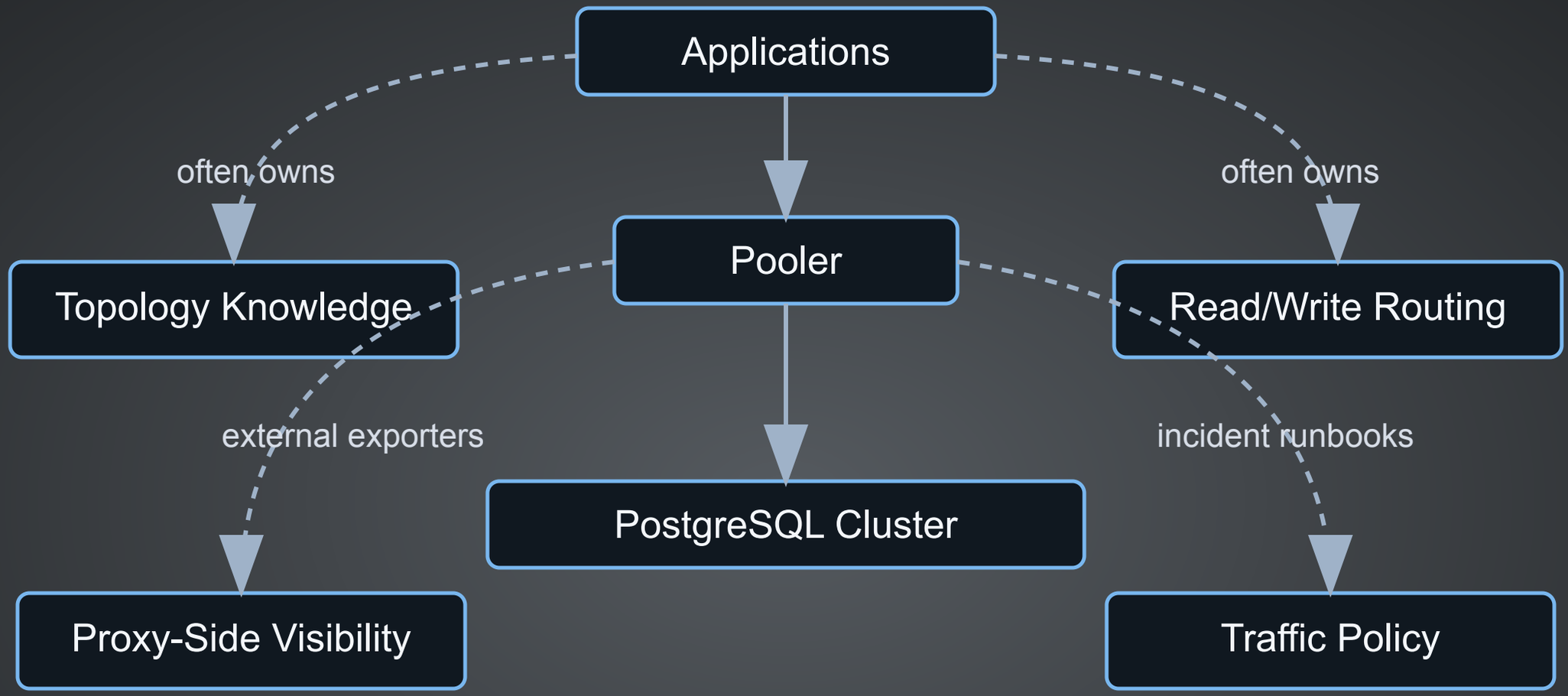
CONNECTION LIFECYCLE CONCEPTS

Concept	PgBouncer	Pgpool-II	pgcat	ProxySQL
Connection pooling	yes	yes	yes	yes
Session pooling	yes	yes	yes	yes
Transaction pooling	yes	partial	yes	yes
Statement pooling	yes	no	no	yes
Multiplexing	yes	yes	yes	yes
Connection queuing	yes	yes	yes	yes
Server connection reuse	yes	yes	yes	yes

ROUTING AND LOAD DISTRIBUTION

Concept	PgBouncer	Pgpool-II	pgcat	ProxySQL
Read/write splitting	no	yes	yes	yes
Replica load balancing	no	yes	yes	yes
Query routing rules	no	basic	basic	yes
Failover detection	no	yes	yes	yes
Sharding	no	no	yes	yes
Database firewall	no	no	no	yes

WHERE POOLING STOPS BEING ENOUGH



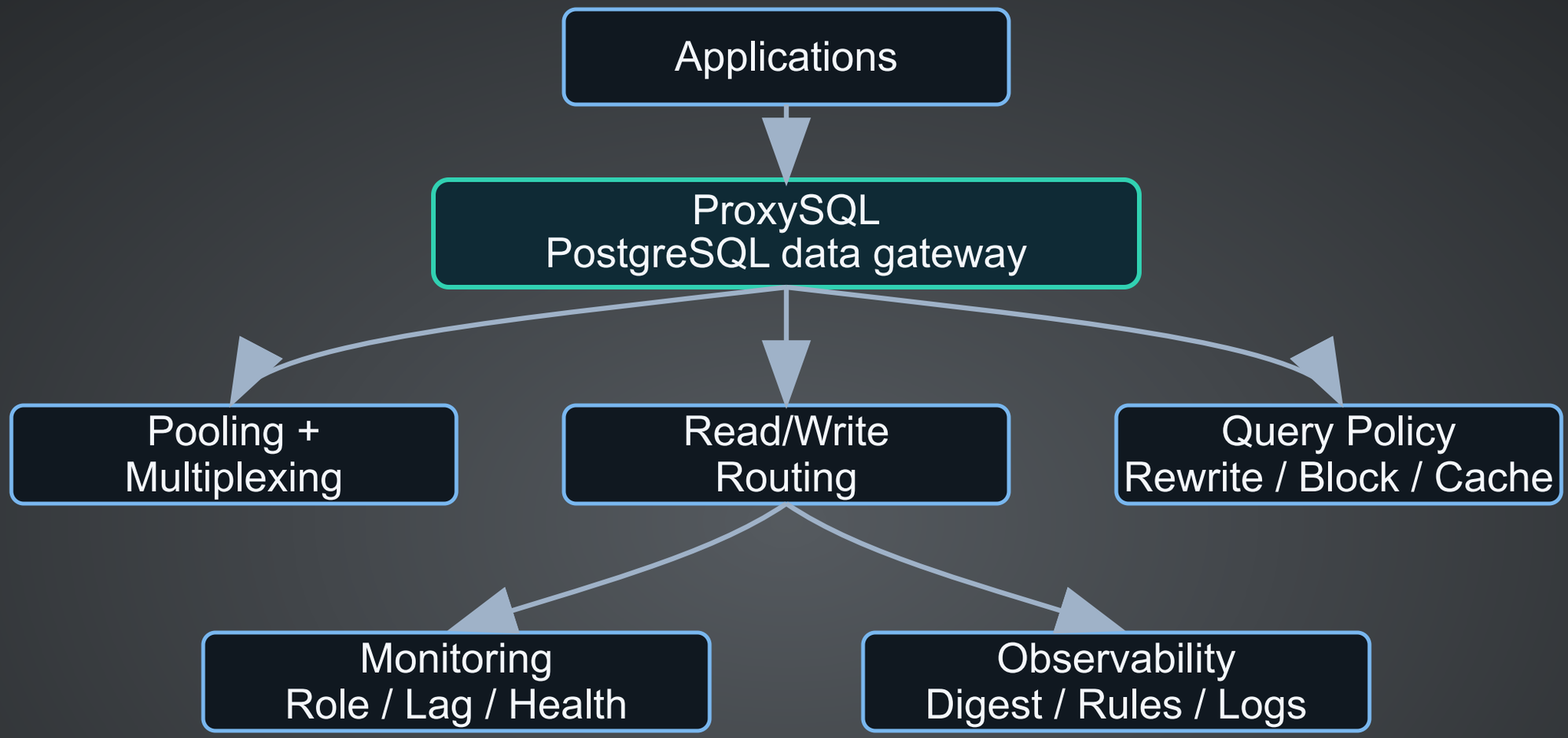
The question is not only "which pooler?" It is where routing, policy, failover reaction, and observability live.

WHAT IS PROXYSQL?

A high-performance, high-availability transparent proxy for PostgreSQL and PostgreSQL wire-compatible products, MySQL and MySQL wire-compatible products.

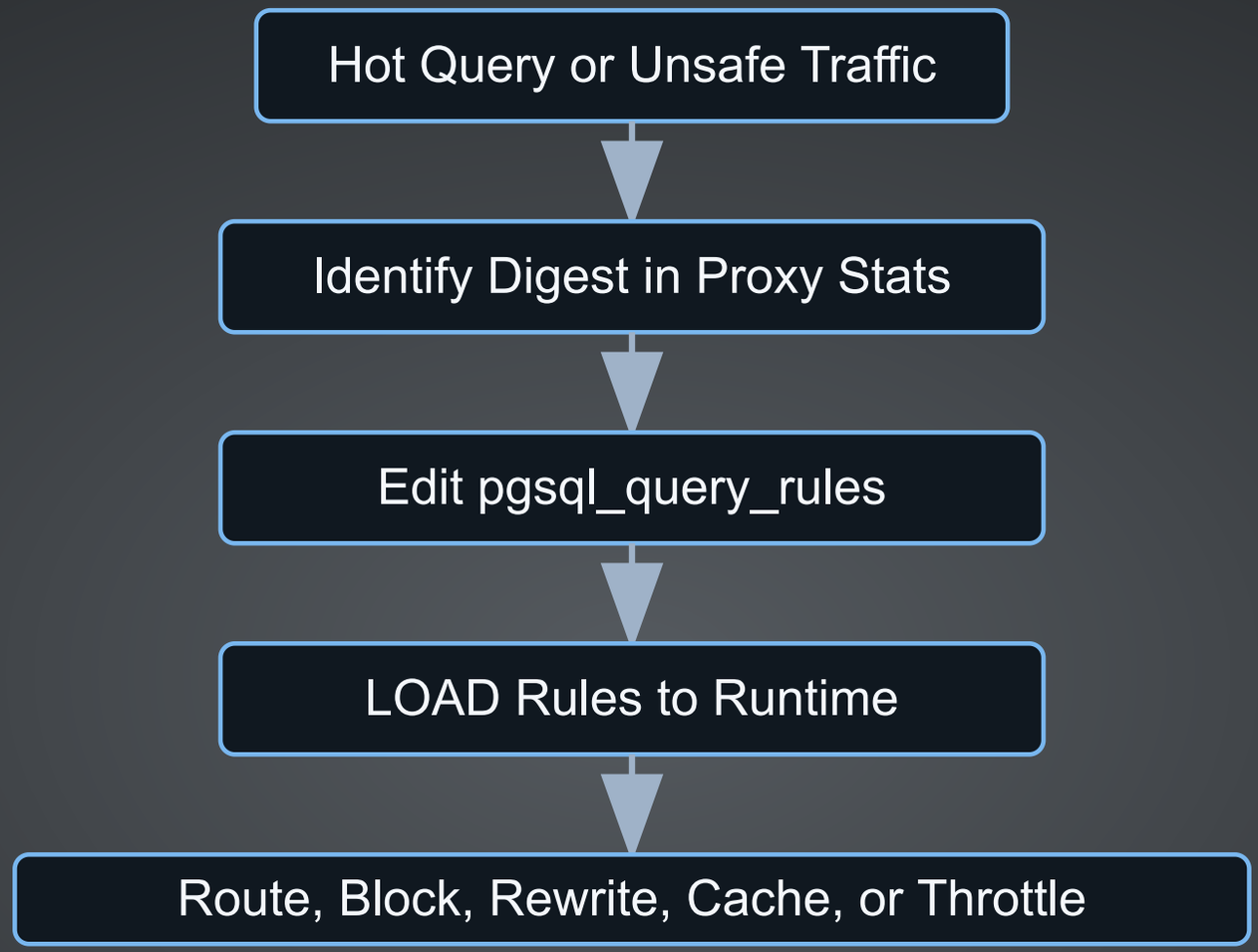
- **Mission:** Maximize performance and scalability at the database layer
- **Operating model:** SQL-native configuration, runtime activation, persistent state
- **Focus:** Reliability engineering for real-world production operations
- **Flexibility:** One data gateway for pooling, routing, policy, and observability

PROXYSQL AS THE POSTGRESQL DATA GATEWAY



The feature list becomes useful when it is attached to operational jobs.

RUNTIME POLICY INSTEAD OF APPLICATION RELEASES



- Change policy while traffic is live
- Verify behavior through runtime and stats tables
- Persist only after the change proves useful

PROXYSQL CONFIG: PGSQL_SERVERS

`pgsql_servers` defines PostgreSQL backend nodes and the hostgroups ProxySQL can route to.

Column	Purpose
`hostgroup_id`	Logical destination for routing rules and users
`hostname`, `port`	PostgreSQL backend address
`status`	Whether the backend is available for traffic
`max_connections`, `weight`	Capacity and balancing controls

```
INSERT INTO pgsql_servers
  (hostgroup_id, hostname, port, status, max_connections, weight)
VALUES
  (10, 'pg-primary', 5432, 'ONLINE', 200, 100),
  (20, 'pg-replica1', 5432, 'ONLINE', 200, 100);

LOAD PGSQL SERVERS TO RUNTIME;
SAVE PGSQL SERVERS TO DISK;
```

PROXYSQL CONFIG: PGSQL_USERS

`pgsql_users` maps client credentials to ProxySQL behavior and the default destination hostgroup.

Column	Purpose
`username`, `password`	Frontend authentication
`default_hostgroup`	Where traffic goes when no rule overrides it
`active`	Enable or disable the account
`max_connections`	Optional frontend connection limit

```
INSERT INTO pgsql_users
  (username, password, default_hostgroup, active)
VALUES
  ('appuser', 'secret', 10, 1);

LOAD PGSQL USERS TO RUNTIME;
SAVE PGSQL USERS TO DISK;
```

PROXYSQL CONFIG: PGSQL_QUERY_RULES

`pgsql_query_rules` turns SQL traffic into runtime policy: route, block, rewrite, cache, or shape traffic.

Column	Purpose
`rule_id`, `active`	Rule identity and enablement
`match_digest` / `match_pattern`	Query matching condition
`destination_hostgroup`	Route matching traffic
`apply`	Stop rule evaluation when matched

```
INSERT INTO pgsql_query_rules
  (rule_id, active, match_digest, destination_hostgroup, apply)
VALUES
  (100, 1, '^SELECT', 20, 1);

LOAD PGSQL QUERY RULES TO RUNTIME;
SAVE PGSQL QUERY RULES TO DISK;
```

PROXYSQL CONFIG: PGSQL-* VARIABLES

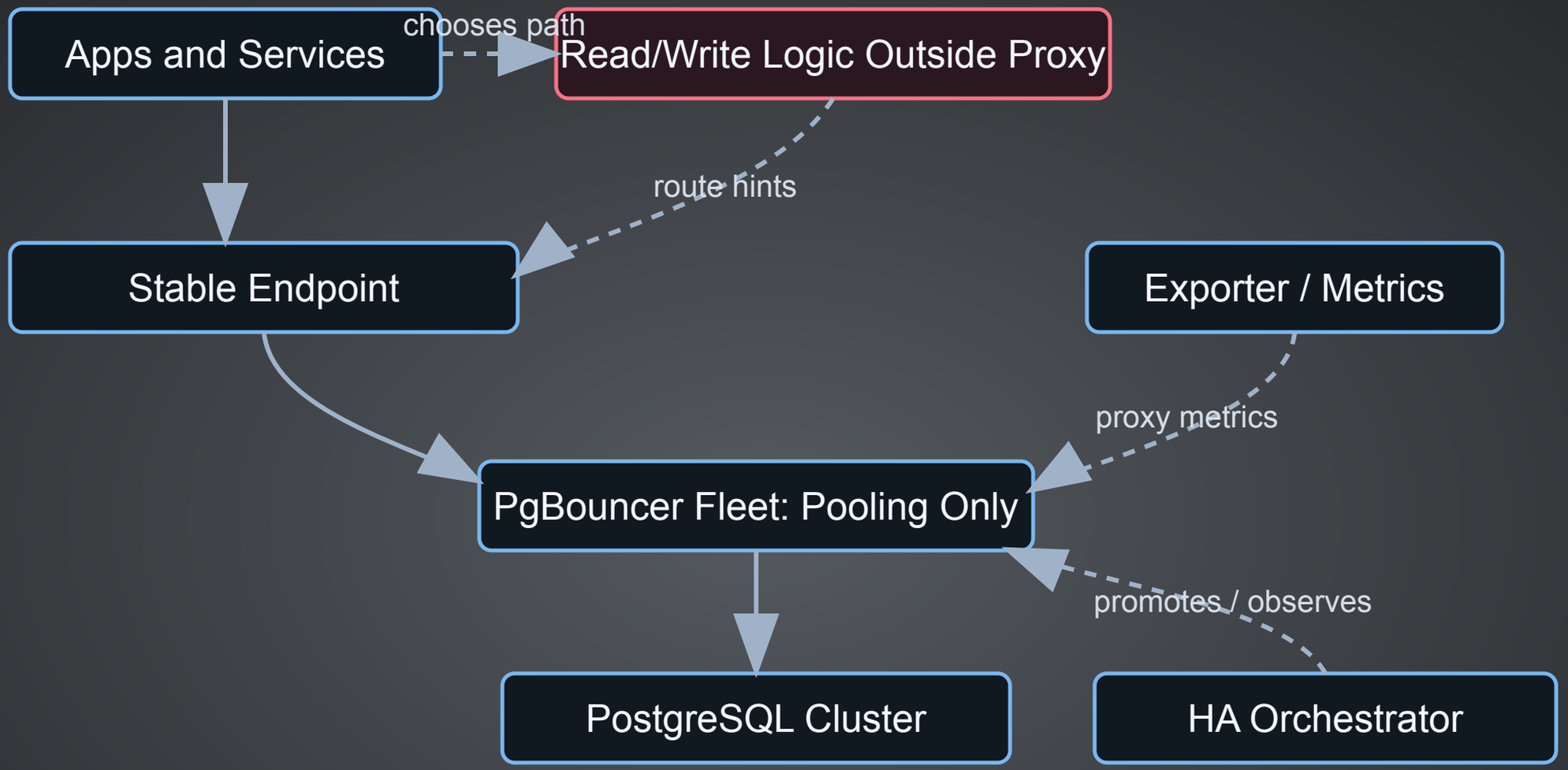
PostgreSQL behavior that is not tied to one user, server, or rule lives in `global_variables`.

Scope	Examples
Listener behavior	`pgsql-interfaces`, connection limits
Monitoring	health checks, intervals, timeouts
Protocol behavior	PostgreSQL frontend/backend behavior
Defaults	operational settings applied globally

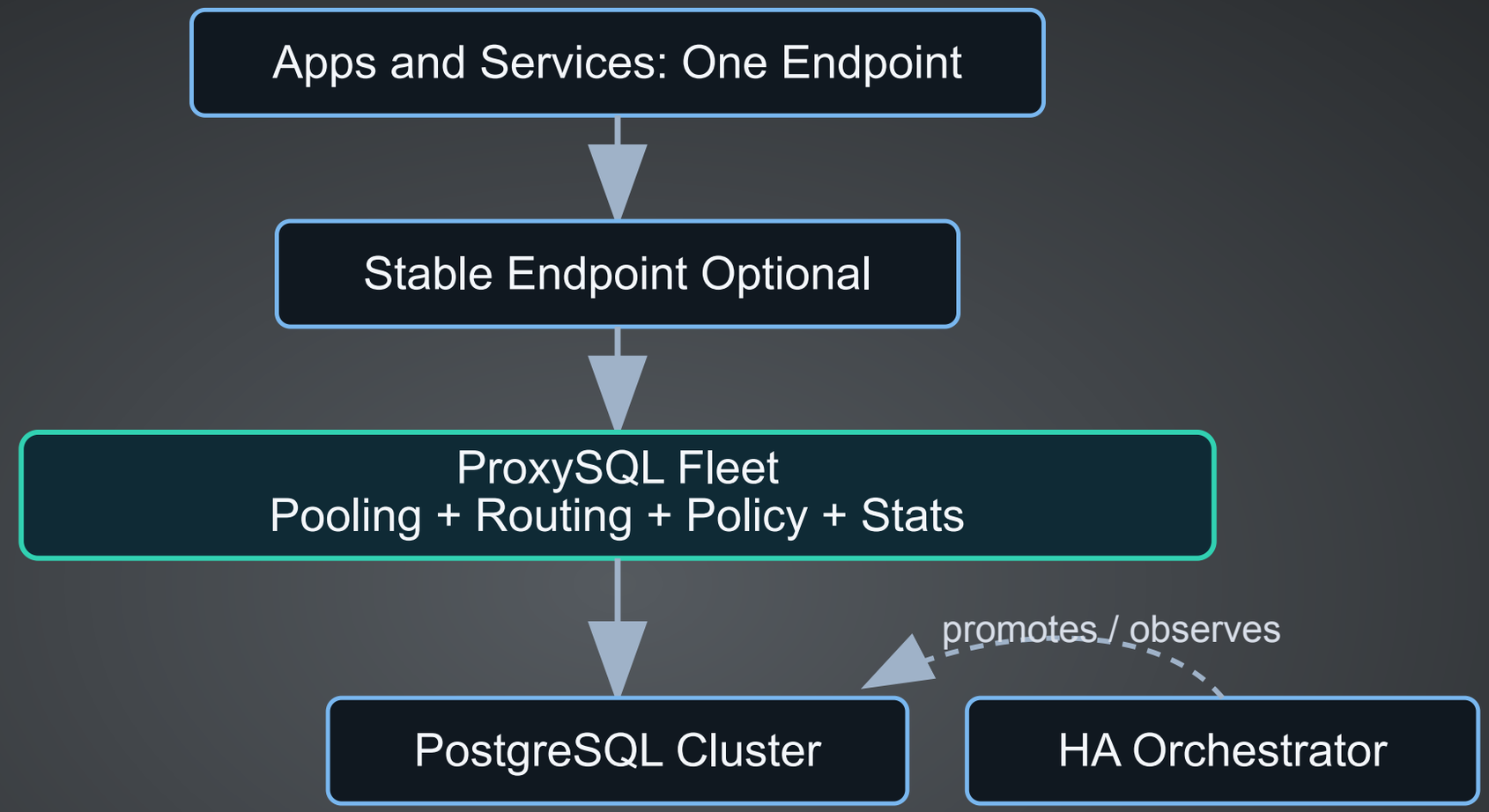
```
SELECT variable_name, variable_value
FROM global_variables
WHERE variable_name LIKE 'pgsql-%'
ORDER BY variable_name;

UPDATE global_variables
SET variable_value='0.0.0.0:6032'
WHERE variable_name='pgsql-interfaces';
LOAD PGSQL VARIABLES TO RUNTIME;
SAVE PGSQL VARIABLES TO DISK;
```

PGBOUNCER-CENTERED MIDDLE TIER

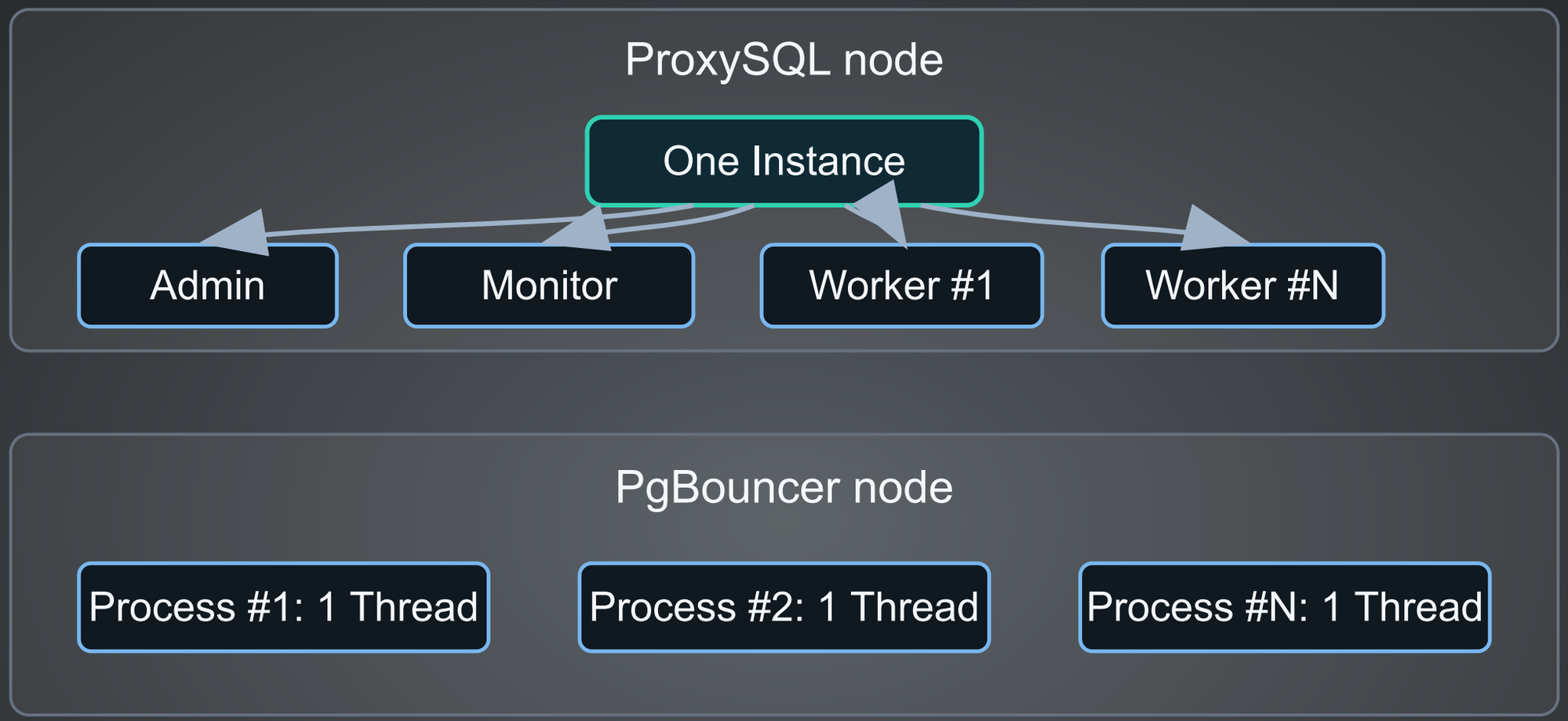


PROXYSQL-CENTERED MIDDLE TIER



Promotion still belongs to the HA authority. ProxySQL reacts to role state and rewires traffic.

PER NODE: PROCESSES VS THREADS

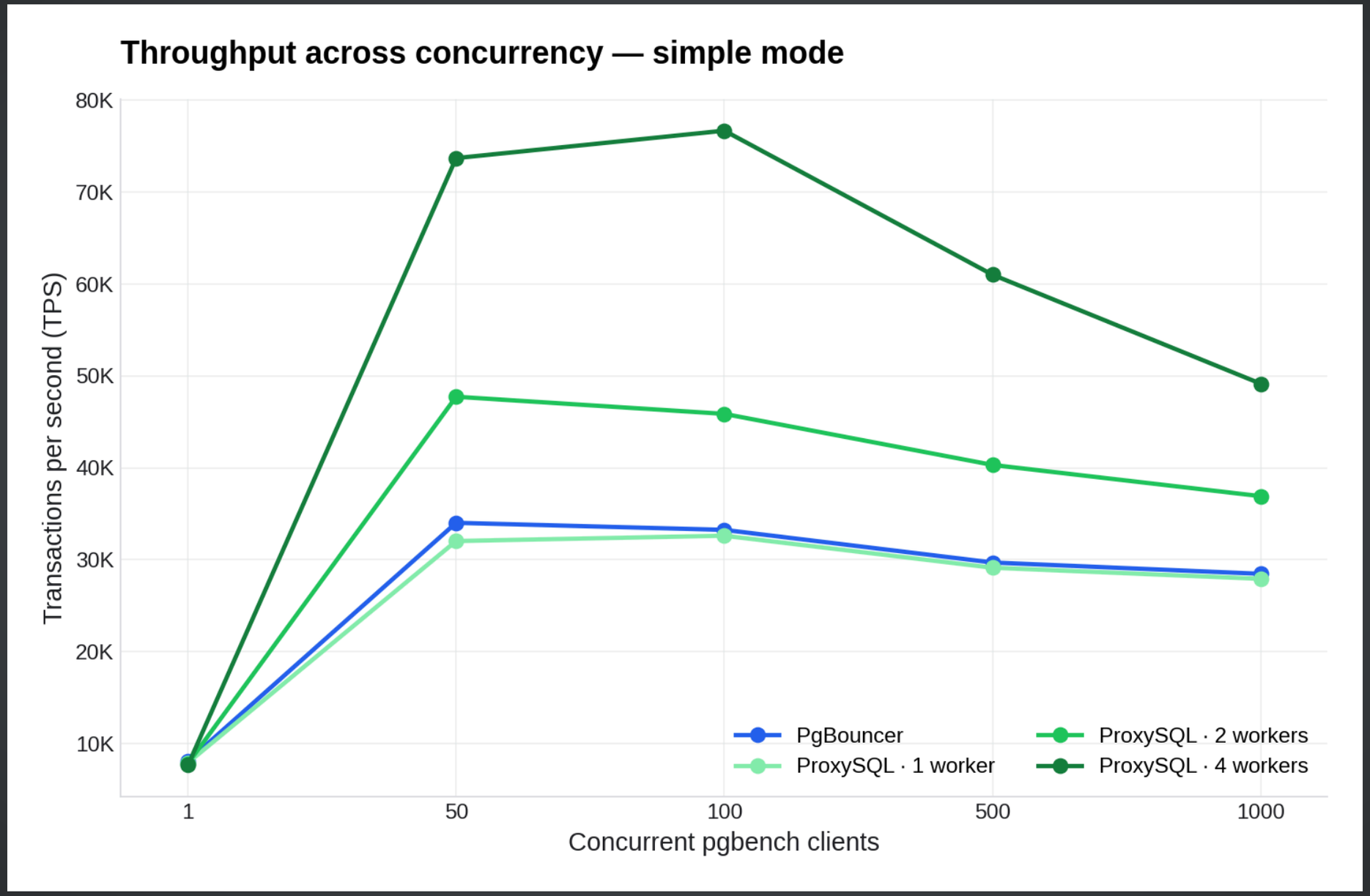


PgBouncer scales per process. ProxySQL scales vertically inside one multi-threaded instance.

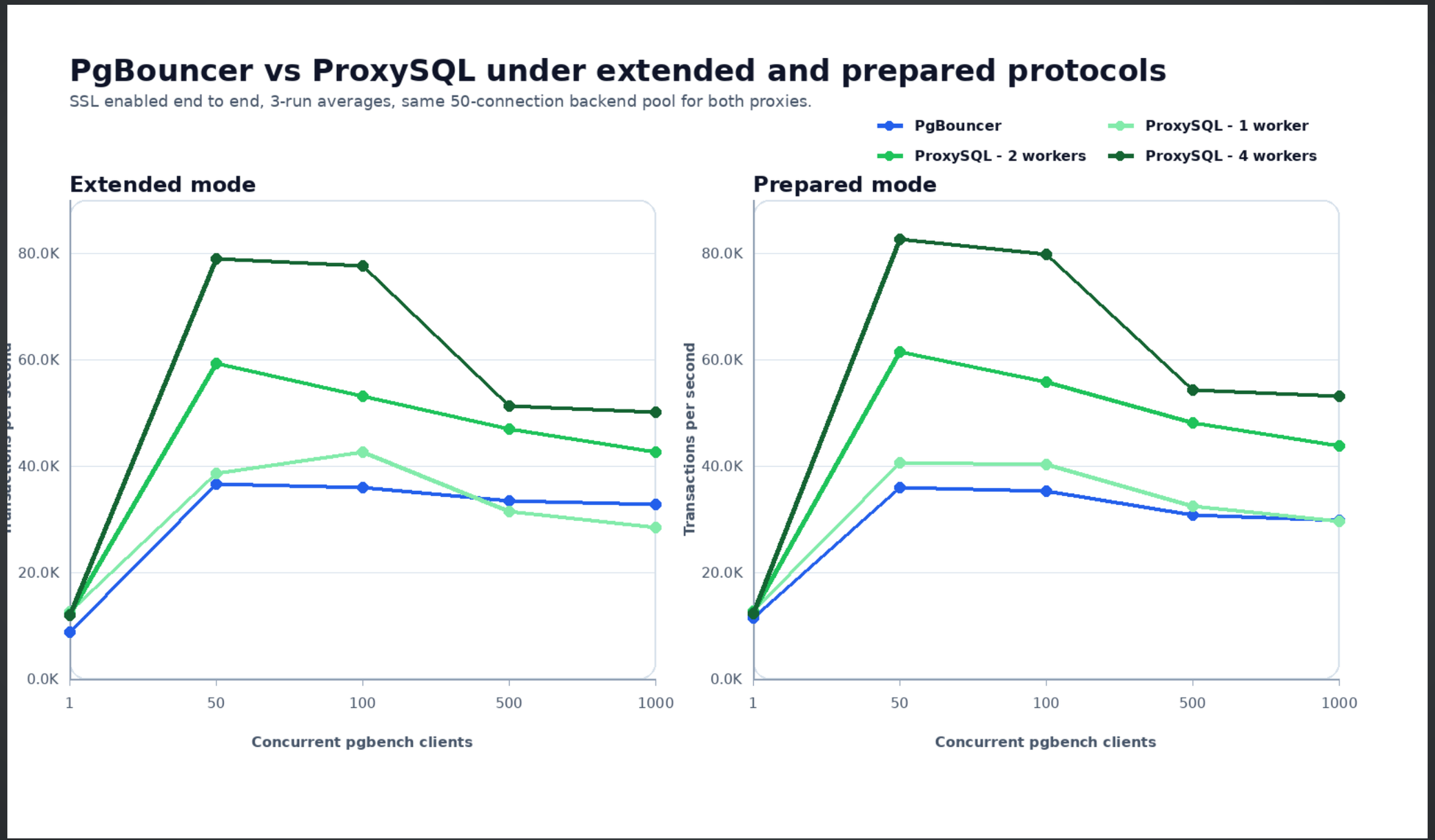
PGBOUNCER VS PROXYSQL: PRACTICAL DIFFERENCE

Capability	PgBouncer	ProxySQL
Pooling	Focused and mature	**Advanced**
Read/write split	Application or external layer	**Query rules + hostgroups**
Backend monitoring	Liveness probe	**Connect, ping, read-only, replication lag**
Live traffic policy	Not a query-rule engine	**Block, route, rewrite, cache, throttle**
Per-query visibility	External / PostgreSQL side	**Proxy-side digest and rule stats**
Config lifecycle	INI + reload	**MEMORY -> RUNTIME -> DISK**

BENCHMARK: SIMPLE MODE



BENCHMARK: EXTENDED AND PREPARED PROTOCOLS



BENCHMARK: PROXYSQL RELATIVE TO PGBOUNCER

How much faster ProxySQL is than PgBouncer

Each cell is ProxySQL TPS divided by PgBouncer TPS at the same mode and client count. PgBouncer is the 1.00x baseline.

Color scale: 0.85x · 1.00x · 1.50x · 2.30x

Extended mode

	c=1	c=50	c=100	c=500	c=1000
ProxySQL - 1 worker	**1.43x** 12.6K TPS	**1.05x** 38.5K TPS	**1.19x** 42.6K TPS	**0.94x** 31.5K TPS	**0.87x** 28.4K TPS
ProxySQL - 2 workers	**1.38x** 12.2K TPS	**1.62x** 59.3K TPS	**1.48x** 53.1K TPS	**1.40x** 46.8K TPS	**1.30x** 42.6K TPS
ProxySQL - 4 workers	**1.35x** 11.9K TPS	**2.16x** 78.9K TPS	**2.16x** 77.6K TPS	**1.53x** 51.3K TPS	**1.53x** 50.1K TPS

Prepared mode

	c=1	c=50	c=100	c=500	c=1000
ProxySQL - 1 worker	**1.12x** 12.8K TPS	**1.13x** 40.6K TPS	**1.14x** 40.2K TPS	**1.06x** 32.5K TPS	**0.99x** 29.5K TPS
ProxySQL - 2 workers	**1.10x** 12.5K TPS	**1.71x** 61.4K TPS	**1.58x** 55.8K TPS	**1.56x** 48.0K TPS	**1.47x** 43.8K TPS
ProxySQL - 4 workers	**1.07x** 12.3K TPS	**2.30x** 82.5K TPS	**2.26x** 79.7K TPS	**1.77x** 54.3K TPS	**1.79x** 53.2K TPS

DEMO

- Deploy PostgreSQL 16 replication topology including ProxySQL
- Deploy PostgreSQL 17 replication topology including ProxySQL

DBDEPLOYER

- Install DbDeployer
- Get Database Binaries & unpack (`~/opt/postgresql`)
- Deploy Sandbox(es) (`~/sandboxes`)
    - Done!

```
# cd primary/replica1/replica2/proxysql
./use
./start
./stop
./restart
./status
./check_replication
./test_replication
```

DEMO NOTES (1)

Blog: What's Coming to dbdeployer

USING UBUNTU 24.04 (VIA ORB)

```
curl https://proxysql.com/get/dbdeployer | bash
sudo mv dbdeployer /usr/local/bin/dbdeployer
dbdeployer --version
```

PREPARE POSTGRESQL SOFTWARE FOR DBDEPLOYER

```
PG_VERSION=16
apt-get download postgresql-${PG_VERSION} postgresql-client-${PG_VERSION}
dbdeployer unpack --provider=postgresql postgresql*${PG_VERSION}.deb
PG_MINOR_VERSION=$(basename $(ls -d ~/opt/postgresql/${PG_VERSION}))
NOTE: There are some pre-requisite issues not shown here.
```

DEMO NOTES (2)

DEPLOY A REPLICATION SANDBOX (1 PRIMARY + 2 REPLICAS) TAKES ~15 SECONDS

```
$ dbdeployer deploy replication ${PG_MINOR_VERSION} --provider=postgresql --with-proxysql --sandbox-directory=demo

  Primary deployed in /home/rbradfor/sandboxes/demo/primary (port: 16614)
  Replica 1 deployed in /home/rbradfor/sandboxes/demo/replica1 (port: 16615)
  Replica 2 deployed in /home/rbradfor/sandboxes/demo/replica2 (port: 16616)
  ProxySQL deployed in /home/rbradfor/sandboxes/demo/proxysql (admin port: 6032, mysql port: 6033)
  postgresql replication sandbox (1 primary + 2 replicas) deployed in /home/rbradfor/sandboxes/demo

$ cd ~/sandboxes/demo
$ ./test_replication
```

DEMO NOTES (3)

CONFIRM PROXYSQL INSTALLED (ADMIN PORT 6132)

```
$ export PATH=~/opt/postgresql/16.14/bin:$PATH
$ psql postgresql://admin:admin@127.0.0.1:6132
psql (16.14 (Ubuntu 16.14-1.pgdg22.04+1), server 16.1)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

admin=# show schemas;
 seq |     name      |                                    file
-----+---------------+-------------------------------------------------------------------------
 0   | main          |
 2   | disk          | /home/rbradfor/sandboxes/rsandbox_8_4_8/proxysql/data/proxysql.db
 3   | stats         |
 4   | monitor       |
 5   | stats_history | /home/rbradfor/sandboxes/rsandbox_8_4_8/proxysql/data/proxysql_stats.db
(5 rows)
```

DBDEPLOYER USE

```
$ cd ~/sandboxes/demo
$ proxysql/use
```

DEMO NOTES (4)

Docs How to Configure ProxySQL for PostgreSQL for the first time

DEFINE SERVER BACKENDS (HOSTGROUPS)

```
INSERT INTO pgsql_servers (hostgroup_id, hostname, port, status, max_connections, weight)
VALUES
 (10, '127.0.0.1', 16614, 'ONLINE', 50, 100),
 (20, '127.0.0.1', 16615, 'ONLINE', 50, 100),
 (20, '127.0.0.1', 16616, 'ONLINE', 50, 100);
LOAD PGSQL SERVERS TO RUNTIME;
SELECT * FROM pgsql_servers;
```

DBDEPLOYER USE

- Configured by default using proxysql.cnf

DEMO NOTES (5)

Docs How to Configure ProxySQL for PostgreSQL for the first time

BACKEND VERIFICATION (MONITORING)

```
admin=# SELECT * FROM monitor.pgsql_server_connect_log ORDER BY time_start_us DESC LIMIT 3;
 hostname  | port  |  time_start_us   | connect_success_time_us | connect_error
-----------+-------+------------------+-------------------------+---------------
 127.0.0.1 | 16616 | 1779228711501777 | 2841                    |
 127.0.0.1 | 16614 | 1779228710928513 | 3344                    |
 127.0.0.1 | 16615 | 1779228710927665 | 3842                    |

admin=# SELECT * FROM monitor.pgsql_server_ping_log ORDER BY time_start_us DESC LIMIT 3;
 hostname  | port  |  time_start_us   | ping_success_time_us | ping_error
-----------+-------+------------------+----------------------+------------
 127.0.0.1 | 16615 | 1779228717452022 | 1254                 |
 127.0.0.1 | 16616 | 1779228716932315 | 2930                 |
 127.0.0.1 | 16614 | 1779228716932207 | 1262                 |
```

DEMO NOTES (6)

Docs How to Configure ProxySQL for PostgreSQL for the first time

DEFINE USERS FOR APPLICATION USAGE

```
INSERT INTO pgsql_users (username, password, default_hostgroup)
VALUES ('appuser', 'perconalive@20', 1);
SELECT * FROM pgsql_users;
LOAD PGSQL USERS TO RUNTIME;
SAVE PGSQL USERS TO DISK;
```

DBDEPLOYER USE

- Configured by default using proxysql.cnf

DEMO NOTES (7)

Docs PostgreSQL Authentication

TEST CONNECTION VIA PROXYSQL

```
$ nc -vz4 127.0.0.1 6133
Connection to 127.0.0.1 6133 port [tcp] succeeded!
$ cat primary/use
$ /home/rbradfor/opt/postgresql/16.14/bin/psql -h 127.0.0.1 -p 6133 -U appuser
Password for user appuser:
demo=> SELECT current_user;
current_user

 appuser
```

DEMO TAKEAWAYS

- dbdeployer creates a reproducible PostgreSQL topology with ProxySQL included
- ProxySQL exposes PostgreSQL admin/configuration through SQL tables
- Backends, users, and monitoring can be verified before application traffic moves
- The same workflow scales from local demo to production operating model

DBDEPLOYER FOR POSTGRESQL

https://github.com/ProxySQL/dbdeployer

- Deploy PostgreSQL and MySQL sandboxes in seconds
- Many topologies (single, replication)
- Run multiple database versions concurrently
- Blog: The Provider Architecture — How dbdeployer Learned to Speak PostgreSQL

ORCHESTRATOR FOR POSTGRESQL

https://github.com/ProxySQL/orchestrator

- HA for PostgreSQL
- Topology Aware discovery and management
- Graceful switchover
- API-Driven
- No external consensus
- Blog: Orchestrator for PostgreSQL: the HA brain, now first-party

MORE ABOUT PROXYSQL

- 3.0.x - Stable Tier
- 3.1.x - Innovation Tier
- 4.0.x - AI/MCP Tier

Supported on RedHat/CentOS/Amazon Linux, Debian/Ubuntu, Almalinux, OpenSUSE, Kubernetes, Docker

PROXYSQL DOCKER TUTORIALS FOR POSTGRESQL

https://github.com/proxysql/tutorials

- Simple Database Tutorial
- Simple Database Benchmark Tutorial
- Simple Database Benchmark using ProxySQL Tutorial
    - Demonstrating connection multiplexing
- HA Database Benchmark using ProxySQL Tutorial
    - Demonstrating read/write splitting with primary/replica
    - Demonstrating read load balancing with multiple replicas
    - Demonstrating advanced query rules for dedicated query redirection

QUESTIONS